

EtherSensor

DeviceLock DLP Suite

Administrator's Guide

Contents

1. DeviceLock EtherSensor.....	1
1.1. Overview of Features.....	1
1.2. DeviceLock EtherSensor Application.....	3
1.3. System requirements.....	5
1.4. Operation Description.....	6
1.5. Administrator's Qualification.....	8
1.6. List of Accompanying Documents.....	8
2. Installation of DeviceLock EtherSensor.....	8
2.1. What Is Included into the Software Distribution Package.....	9
2.2. DeviceLock EtherSensor Ethernet Connection.....	10
2.2.1. Management Interface.....	10
2.2.2. Listening Interface.....	10
2.2.3. Setting up Switches.....	11
2.2.4. Using Third-Party Information Security Tools.....	12
3. Sensor Settings.....	13
3.1. Message Sources.....	13
3.1.1. EtherSensor EtherCAP.....	18
3.1.1.1. Setting up the Configurator.....	22
3.1.1.2. Manual Setup (Config File).....	31
3.1.2. EtherSensor ICAP.....	37
3.1.2.1. Setting up the Configurator.....	39
3.1.2.2. Manual Setup (Config File).....	41
3.1.3. EtherSensor LotusTXN.....	46
3.1.3.1. Setting up the Configurator.....	47
3.1.3.2. Manual Setup (Config File).....	49
3.2. Capture Results Delivery.....	50
3.2.1. Setting up the Configurator.....	54
3.2.1.1. DEVICELOCK Profiles.....	54
3.2.1.2. SMTP Profiles.....	57
3.2.1.3. FTP Profiles.....	59
3.2.1.4. SFTP Profiles.....	62
3.2.1.5. FILEDROP Profiles.....	64
3.2.1.6. IMAP Profiles.....	67
3.2.1.7. SMB Profiles.....	69
3.2.1.8. SYSLOG Profiles.....	71
3.2.1.8.1. Lua Scripts.....	73
3.2.1.9. GROUP Profiles.....	73
3.2.2. Manual Setup (Config File).....	74
3.2.2.1. DEVICELOCK Profiles.....	77
3.2.2.2. SMTP Profiles.....	78

3.2.2.3. FTP Profiles	80
3.2.2.4. SFTP Profiles	82
3.2.2.5. FILEDROP Profiles	83
3.2.2.6. IMAP Profiles	85
3.2.2.7. SMB Profiles	86
3.2.2.8. SYSLOG Profiles	88
3.2.2.9. GROUP Profiles	89
3.3. Logging	89
3.3.1. Setting up the Configurator	90
3.3.2. Manual Setup (Config File)	92
3.4. EtherSensor Agent	103
3.4.1. Agent Operation Conditions	103
3.4.2. Agent Installation	104
3.4.3. Agent Files	105
3.4.4. Agent Logical Modules	106
3.4.5. Data Transferred to EtherStat	106
3.4.6. Data Transferred to EtherSensor	110
3.4.7. Working with the Agent	110
3.4.7.1. Possible Agent Operation Methods	112
3.4.7.2. Configuring the Service EtherSensor Agent	114
3.4.7.3. Agent Operation Logging	118
3.4.7.4. Troubleshooting	120
4. Event and Object Analysis	121
4.1. Setting up the Configurator	124
4.2. Manual Setup (Config File)	131
4.3. Messages Created	137
4.4. Capture Results Filtering	144
4.4.1. Filtration Basics	144
4.4.1.1. Filter Configuration	146
4.4.1.2. Tables	146
4.4.1.3. Rules	147
4.4.1.3.1. Criteria and Conditions	148
4.4.1.3.1.1. ALL, * Condition	151
4.4.1.3.1.2. DETECTOR Condition	152
4.4.1.3.1.3. PROTOCOL Condition	152
4.4.1.3.1.4. MSG-SIZE, TOTAL-SIZE Condition	153
4.4.1.3.1.5. CHECK-MD5 Condition	155
4.4.1.3.1.6. CHECK-MESSAGE-ID Condition	156
4.4.1.3.1.7. HOSTNAME Condition	157
4.4.1.3.1.8. IP Condition	158
4.4.1.3.1.9. HEADER Condition	160
4.4.1.3.1.10. ATTACH-NAME Condition	163

4.4.1.3.1.11. ATTACH-EXIST Condition.....	164
4.4.1.3.1.12. TAG Condition.....	165
4.4.1.3.1.13. FROM, TO, CC, BCC, ADDRESS, SUBJECT Condition.....	167
4.4.1.3.1.14. TEXT Condition.....	170
4.4.1.3.2. Actions	173
4.4.1.3.2.1. ACCEPT Action.....	174
4.4.1.3.2.2. DROP Action.....	174
4.4.1.3.2.3. JUMP Action.....	175
4.4.1.3.2.4. RETURN Action.....	177
4.4.1.3.2.5. LABEL Action.....	179
4.4.1.3.2.6. TAG Action.....	181
4.4.1.3.2.7. DATETIME Action.....	183
4.4.1.3.2.8. DNS Action.....	184
4.4.1.3.2.9. DNSBL-LH, DNSBL-RH Action.....	185
4.4.1.3.2.10. SAVE RAW DATA Action.....	188
4.4.1.3.2.11. TRANSPORT Action.....	189
4.4.1.3.2.12. HEADER Action.....	190
4.4.1.3.2.13. HEADER_EX Action.....	191
4.4.1.3.2.14. LOG Action.....	193
4.4.1.4. Short rules for developing filters	196
4.4.1.5. Tips.....	197
4.4.2. Prefiltering HTTP Requests	198
4.4.2.1. Conditions.....	199
4.4.2.1.1. ALL, * Condition.....	199
4.4.2.1.2. METHOD Condition.....	200
4.4.2.1.3. IP Condition.....	201
4.4.2.1.4. REQ-SIZE, RESP-SIZE, SIZE Condition.....	203
4.4.2.1.5. REQ-HEADER, RESP-HEADER Condition.....	205
4.4.2.1.6. URL Condition.....	208
4.4.2.1.7. TAG Condition.....	209
4.4.2.2. Actions	211
4.4.2.2.1. ACCEPT Action.....	211
4.4.2.2.2. DROP Action.....	212
4.4.2.2.3. JUMP Action.....	213
4.4.2.2.4. RETURN Action.....	214
4.4.2.2.5. COPY Action.....	216
4.4.2.2.6. ACCESS-LOG Action.....	217
4.4.2.2.7. TAG Action.....	218
4.4.2.2.8. LABEL Action.....	220
4.4.3. Filter Use Cases	222
4.4.3.1. Adding a Host Name.....	222

4.4.3.2. Host Filtering.....	225
4.4.3.3. URL Filtering.....	227
4.4.3.4. Filtering by HTTP+DNSBL.....	228
4.4.3.5. Filtering Large HTTP Objects.....	231
5. EtherSensor Updater Service.....	232
5.1. Setting up the Configurator.....	234
5.2. Manual Setup (Config File).....	238
6. Sensor Routine Maintenance.....	241
6.1. Questions on the Sensor Maintenance.....	241
7. Licensing EtherSensor.....	242
7.1. The License File.....	243
7.2. Runtime Environment UHID (HardwareID).....	246
7.3. How the Licensing System Works.....	247
8. What to Do in Case of Emergency.....	249
8.1. Hardware Failure.....	249
8.2. Unauthorized Access to the Software or OS.....	250
9. GUI Localization.....	250
9.1. Language Files.....	252
9.2. Editing GUI Elements.....	253
10. DeviceLock EtherSensor Changelog.....	256

1. DeviceLock EtherSensor

DeviceLock EtherSensor, version 5.1.0.13519
Administrator's Guide.

© 2001 - 2018, Microolap Technologies, Chernogolovka, Russian Federation.

This document describes administrative tasks required to manage the operation of DeviceLock EtherSensor on your system. This document is intended for a technical audience, specifically the system administrators and information security officers in charge of DeviceLock EtherSensor.

Reproduction, adaptation or translation without written permission of Microolap Technologies is prohibited. The information contained in this document is subject to change without notice.

1.1. Overview of Features

DeviceLock EtherSensor is a high-performance real-time network event and message extraction platform with the following features:

- Access to a significant number of Internet services known to **DeviceLock EtherSensor** (several thousand).
- High performance: streamed processing over 20GBps+ links.
- Message, event and metadata delivery to any SOC subsystems (DLP, SIEM, eDiscovery, etc.).
- Prolonged continuous unattended operation.
- Support of off-the-shelf equipment with low footprint.

DeviceLock EtherSensor consists of several Windows services which interoperate to intercept and analyze application-level messages and metadata (normally messages exchanged by network users). The resulting messages, message metadata or data extracted from them are delivered to consumer systems.

The key feature and fundamental operating principle of **DeviceLock EtherSensor** is its non-participation in the traffic delivery of the monitored network, which results in network reliability independent of the service. That said, **DeviceLock EtherSensor** ensures full control over traffic in networks up to to 20GBps networks by detecting messages from several thousand of Internet services.

Traffic filtering methods at each level (either IP packets or reconstructed application-level objects) ensure minimum resource consumption with the desired level of control over network communications.

Expandability enables **DeviceLock EtherSensor** both to accept data from external sources (SPAN/TAP traffic, ICAP clients, Lotus Notes transaction log, PCAP files) and delivers reconstructed messages to external consumer systems.

Independence of **DeviceLock EtherSensor** services from the logging service (the **EtherSensor Watcher** service can be used to configure complex "logging level/logging direction" combinations for each data type of messages without impairing performance).

The features **DeviceLock EtherSensor** are described below, grouped by functional units.

Webmail (WM)

Processes traffic to extract outgoing messages from the following webmail services: Mail.RU, Yandex.RU, Pochta.RU, GMail, etc. (over 40 domains), as well as all services based on the SquirrelMail core. To process messages sent over an encrypted channel (HTTPS protocol) you will also need an ICAP server or **SSLsplitter**.

Social networks (SN)

Extracts various types of data (authentication credentials, text messages, comments, etc.) from the traffic in the following social networks:

- Social networks, including Vk.com, Facebook, LinkedIn and Mamba.ru.
- phpbb-, ipb-, vbulletin- and mybb-based forums.
- SMS/MMS messaging services (including over 500 domains).

To process messages sent over an encrypted channel (HTTPS protocol) you will also need an ICAP server or **SSLsplitter**.

Email (EM)

Processes traffic to extract email messages sent over SMTP and POP3 protocols.

ICAP server (IS)

Can be used to extract messages from HTTP- and FTP traffic delivered over the ICAP protocol by external systems, such as SQUID, Blue Coat Proxy SG, Cisco WSA, Webwasher, Websense, McAfee Web Gateway, FortiGate, Entensys UserGate, etc.

Instant messages (IM)

Processes traffic to extract messages sent and received via instant messaging services over such protocols as Skype, XMPP/Jabber, IRC, MSN, Yahoo and OSCAR.

Lotus Notes (LN)

Processes traffic to extract **Lotus Notes** events, including messages, calendar events, etc. For encrypted traffic, messages are extracted from the **Lotus Notes Transaction Log**. These methods do not affect the operation of **Lotus Notes**.

File transfer (FT)

Processes traffic to extract files transmitted over the SMB, HTTP, FTP and WebDAV protocols. To process messages sent over an encrypted channel (HTTPS protocol) you will also need an ICAP server or **SSLsplitter**.

Curriculum vitae (CV)

Processes traffic to extract events (registration, authentication, replies to job posts, updates to CVs) from job seeking sites, e.g. HH.ru, SuperJob.ru, Job.ru, etc. (over 150 domains). You would additionally need **SSLsplitter** or ICAP server to extract messages sent over an encrypted channel (HTTPS protocol).

EtherSensor Agent (AG)

EtherSensor Agent is installed on workstations in case the installation of a full end-point DLP-solution is impossible for some reason.

Is used to map TCP connections created by local processes at workstations to user names and host names in a corporate environment with NAT, terminal servers, etc.

Additionally **EtherSensor Agent** can be used to track changes (equipment, processes, etc.) on a workstation and to transmit that data to **DeviceLock EtherSensor** and **EtherStat** servers.

1.2. DeviceLock EtherSensor Application

DeviceLock EtherSensor is used to analyze L2-L7 level traffic in the OSI model and to extract the content and metadata of messages and message-related events.

The **DeviceLock EtherSensor** platform was developed based on the following requirements:

Adaptability to a consumer system:

DeviceLock EtherSensor must be able to transmit both data and metadata of a discovered object to any consumer system in any required format and over the transport recognized by the system. The type and purpose of the consumer system do not matter: it may be SIEM, DLP, eDiscovery, UEBA, Enterprise Search, file systems, user systems, etc.

Concurrent interaction with multiple sources and consumers:

There must not be a limit to the number of consumer systems with which **DeviceLock EtherSensor** is capable of communicating simultaneously.

Similarly, there must not be a limit to the number of data sources with which **DeviceLock EtherSensor** is capable of communicating simultaneously.

Total control:

DeviceLock EtherSensor must detect any application-level object transmitted over the network, regardless of its type. Processing results of such objects are to be transmitted to specialized consuming systems.

Real time:

DeviceLock EtherSensor must operate in real time over the fastest enterprise data links available. The current version of **DeviceLock EtherSensor** readily supports data stream processing in 20Gbps networks using off-the-shelf equipment.

Separability:

DeviceLock EtherSensor must be deployed "out of the box" and must not require constant attention from the developer or the customer. Ideally it will be a completely unattended data infrastructure element.

The common use of **DeviceLock EtherSensor** is in the following tasks:

Message archiving systems (Compliance archiving):

Extracting documents and other objects of email, messengers, social media, blogs, forums and other popular communication environments.

Examples of consumers:

- Bloomberg Vault
- Global Relay
- Google Vault (help)
- IBM Content Collector
- Mailarchiva
- Smarsh
- Somansa Mail-i
- Veritas eDiscovery Platform (former Clearwell);
- Veritas Enterprise Vault.

SIEM systems, UEBA, log analyzers

Extracting application-level object metadata and object-related events from traffic (including real time extraction from an object content) for recording to SIEM systems.

Examples of consumers:

- AlienVault
- ArcSight Enterprise Security Manager (ESM) (ранее HP ArcSight)
- Elastic Stack (ранее ELK Stack: Elasticsearch, Logstash, Kibana)
- EventTracker
- FortiSIEM (ранее AccelOps)
- Graylog
- IBM QRadar
- LogRhythm
- ManageEngine EventLog Analyzer
- McAfee Enterprise Security Manager (ESM)
- Micro Focus Sentinel (ранее NetIQ)
- RuSIEM
- SolarWinds Log & Event Management (LEM)
- Splunk
- SQLstream

- Trustwave SIEM
- Any software which can accept data by SYSLOG, NETFLOW, or other TCP/UDP protocol.

To prevent leaks of confidential data (DLP systems).

To extract application-level message content from the traffic (OSI level 7) of external and internal communication services and forward it to a consumer system: a DLP system, an email archiving system, a document-processing system, a local search system, or any other system with document archiving functions.

Examples of consumers:

- DeviceLock DLP
- Forcepoint DLP
- Proofpoint Email DLP
- Symantec DLP
- Trustwave DLP.

1.3. System requirements

Below are the minimum system requirements for a physical server in order to run **DeviceLock EtherSensor**:

Channel	50 Mbps	150 Mbps	250 Mbps	500 Mbps	750 Mbps	1.5 Gbps	2.5 Gbps	5 Gbps	10 Gbps	15 Gbps	20 Gbps
Users	100	300	500	1000	1500	3000	5000	10000	20000	30000	40000
CPU	1 CPU x 2 Cores	1 CPU x 4 Cores	1 CPU x 4 Cores	1 CPU x 4 Cores	1 CPU x 4 Cores	1 CPU x 6 Cores	1 CPU x 8 Cores	2 CPU x 6 Cores	2 CPU x 10 Cores	4 CPU x 10 Cores	4 CPU x 12 Cores
RAM	1 GB	4 GB	4 GB	8 GB	8 GB	32 GB	64 GB	128 GB			
HDD	75 GB SATA	2x75 GB SATA RAID1	2x75 GB SATA RAID1	2x150 GB SATA RAID1	2x150 GB SATA RAID1	2x146 GB SAS RAID1	4x300 GB SSD RAID10	4x500 GB SSD RAID10	6x500 GB SSD RAID10	8x500 GB SSD RAID10	8x900 GB SSD RAID10
NIC	1 x 1Gbps	1 x 1Gbps	2 x 1Gbps	4 x 1Gbps	4 x 1Gbps	1 x 10Gbps	2 x 10Gbps	4 x 10Gbps	4 x 10Gbps	6 x 10Gbps	6 x 10Gbps

The disk subsystem is used only when the consumer system is **temporarily** unavailable to cache network objects and to store analysis results when the consumer system is not available. That is why the required size of the disk subsystem should be calculated on an individual basis based on the expected downtime of the consumer system.

Windows Server 2008, Windows Server 2012 or Windows Server 2016 x64 is used as an operating system, only NTFS file system is allowed. Windows Server 2012 or Windows Server 2016 on the x64 platform is recommended for maximum performance.

The latest version of Windows, fully updated, is required for proper operation of **DeviceLock EtherSensor**. The availability of all **DeviceLock EtherSensor** features cannot be guaranteed when the OS update service is disabled.

1.4. Operation Description

DeviceLock EtherSensor consists of a set of services running on a dedicated hardware or a virtual server (the "sensor").

The sensor is connected either to the Ethernet port of an active network appliance where network traffic duplication from specific ports is configured (mirroring, rx and tx packets), or to a network tap. The Packet Sniffer SDK Microolap Technologies proprietary technology is used to intercept traffic with zero loss of packets with a 20Gbps+ load.

The sensor server has several network interfaces, one of which is an administrative interface, while the others are used to accept tapped network traffic. The OS network stack on the sensor is only configured on the administrative network interface which is also used to transmit intercepted messages to external consumers over the protocols configured in transport profiles.

The policy of the sensor is stored in its configuration files and contains rules which define the interception of IP packets, analysis of the contents of application-level intercepted messages and, depending on the result, the format, method and direction of transmission of intercepted results.

The following services run on the sensor:

Services which operate on data sources.

Service for extracting application-level messages from Ethernet traffic (EtherSensor EtherCAP, ethersensor_ethercap.exe)

Is used for passive traffic interception on one or more Ethernet adapters and for traffic processing from PCAP files. The **EtherSensor EtherCAP** service extracts application-level (according to the OSI) messages from the processed traffic model and transmits these messages for further processing to the **EtherSensor Analyser** service.

Service for extracting application-level messages from data provided by ICAP clients (EtherSensor ICAP, ethersensor_icap.exe)

Receives traffic over the ICAP protocol in the REQMOD+RESPMOD mode from any ICAP clients (Squid, Blue Coat Proxy SG, Cisco WSA, etc.). Forwards objects received from ICAP clients to the **EtherSensor Analyser** service for further analysis.

Service for extracting messages from Lotus Notes Transaction Log (EtherSensor LotusTXN, ethersensor_lotustxn.exe)

The **EtherSensor LotusTXN** service is used to monitor and reconstruct messages from the **Lotus Notes** system by extracting them from the **Lotus Notes Transaction Log**. The service extracts

messages from Lotus Notes Transaction Log files and forwards these messages for further analysis to the **EtherSensor Analyser** service.

Service for analyzing messages extracted from Ethernet traffic (EtherSensor Analyser, ethersensor_analyser.exe)

The **EtherSensor Analyser** service is used to detect, analyze and filter intercepted messages and network events. The service analyses OSI model application-level protocol objects received from the **EtherSensor EtherCAP**, **EtherSensor ICAP** and **EtherSensor LotusTXN** services in order to detect messages and network events transmitted over the network.

When that is done, the filter engine of the service takes one of the following decisions, based on user-defined rules:

1. Stop message processing.
2. Transmit the message to the consumer system (DLP, UEBA, archive, eDiscovery system, Enterprise Search, etc.).
3. Generate an arbitrary string based on data extracted from the message (usually a syslog-string for the SIEM system).

The EtherSensor Analyser service also interacts with EtherSensor Agent instances, which mark sessions to associate them with the workstation when NAT, terminal services, etc. are used.

Traffic analysis results delivery service (EtherSensor Transfer, ethersensor_transfer.exe)

EtherSensor Transfer is a **DeviceLock EtherSensor** subsystem used to transmit interception results (either intercepted messages themselves or the pre-configured syslog string) to various consumers over any of a number of protocols based on pre-defined profiles.

EtherSensor Logging Service (EtherSensor Watcher, ethersensor_watcher.exe)

EtherSensor Watcher is a **DeviceLock EtherSensor** subsystem used to work with logs and monitor the current state of **DeviceLock EtherSensor**.

The DeviceLock EtherSensor update service (EtherSensor Updater, ethersensor_updater.exe)

Is used to download and install **DeviceLock EtherSensor** updates and license files whenever new versions and/or patches are released.

You can use **EtherSensor Configuration Editor (ethersensor_console.exe)** from the **DeviceLock EtherSensor** installation directory to view sensor statistics.

You can also use **EtherSensor Configuration Editor** to manage sensor policy or edit sensor service configuration files in the `[INSTALLDIR]\config` directory.

1.5. Administrator's Qualification

A **DeviceLock EtherSensor** system administrator must:

- Have basic knowledge of the TCP/IP protocol stack, specifically SMTP, HTTP application protocols.
- Have basic knowledge of network interface and system service administration for the Window Server OS family.
- Have the requisite service administration skills and knowledge for running **DeviceLock EtherSensor** and be able to use them to implement a corporate security policy regarding the use of external communication services.

1.6. List of Accompanying Documents

In order to be able to operate and administer **DeviceLock EtherSensor** and to understand the basics of **DeviceLock EtherSensor** commissioning, the administrator must review the following documents:

- This Guide.
- DeviceLock documentation (<https://www.device-lock.com/support/docs.html>);
- Documentation for external consumers of messages and/or message-related events (depending on the purpose of **DeviceLock EtherSensor**).

2. Installation of DeviceLock EtherSensor

To install **DeviceLock EtherSensor**, start the installer from the bundle. This will also install the following software (necessary for the proper functioning of the system):

- Microsoft .Net Framework 4.0.
- Microsoft Visual C++ 2015 redistributable runtime.

Warning!

To correctly install **DeviceLock EtherSensor**, you need to have administrator's rights (as they are set right after Windows installation). Any limitations to administrator rights may cause **DeviceLock EtherSensor** to behavior incorrectly.

During **DeviceLock EtherSensor** installation, the following services will be installed:

EtherSensor EtherCAP (ethersensor_ethercap.exe process)

This service passively intercepts traffic on network adapters, processes traffic from PCAP files, and extracts messages.

EtherSensor ICAP (ethersensor_icap.exe process)

This service receives and processes ICAP network traffic from ICAP clients and extracts messages.

EtherSensor LotusTXN (ethersensor_lotustxn.exe process)

This service receives **Lotus Notes** messages from the transaction log (*Lotus Notes Transaction Log*).

EtherSensor Analyser (ethersensor_analyser.exe process)

This service detects, analyses and filters intercepted messages.

EtherSensor Transfer (ethersensor_transfer.exe process)

This service delivers analysis results for objects extracted from traffic to receiving systems.

EtherSensor Watcher (ethersensor_watcher.exe process)

The **DeviceLock EtherSensor** logging service. This is the first service to start; all other **DeviceLock EtherSensor** services depend on it.

EtherSensor Updater Service (ethersensor_updater.exe process)

This is an automatic update service for **DeviceLock EtherSensor**.

In addition to services, the following applications will be installed:

EtherSensor Configuration Editor (ethersensor_console.exe process)

A tool to configure **DeviceLock EtherSensor**, monitor its performance, and collect information on how **DeviceLock EtherSensor** works, to create diagnostic reports.

When the sensor setup is finished, set up the installed services.

2.1. What Is Included into the Software Distribution Package

To prepare **DeviceLock EtherSensor** for work, you need to have the following distribution packages:

- Windows Server 2008, Windows Server 2012, Windows Server 2012 x64 or Windows Server 2016. Windows Server 2016 is recommended.
You can also install **DeviceLock EtherSensor** with Windows x64 versions for end users, but performance may be degraded, given the limitations of 64 bit operation systems.
- The **DeviceLock EtherSensor** distribution package.

The **DeviceLock EtherSensor** distribution package includes

- **DeviceLock EtherSensor** installation programs for Windows x86 and x64
- **EtherSensor Updater** installation programs for Windows x86 and x64
- This Guide.

The **DeviceLock EtherSensor** installation program contains all dependencies and installers it requires to work on the relevant platforms. Use it for the product first-time installation to a new server: this

will ensure there are no problems related to dependencies on other software or to other prerequisites.

The update service won't install the latest version of the software until the basic version has been installed. You need to install the basic version irrespective; after that the update service will be able to install the latest version.

2.2. DeviceLock EtherSensor Ethernet Connection

To connect the sensor to the organization network, perform the following steps:

1. Connect the network control interface.
2. Connect network interfaces for traffic interception.
3. Set up switches.
4. Ensure compatibility with third-party information security tools.

2.2.1. Management Interface

DeviceLock EtherSensor requires a connection to a network of network control and interception interfaces.

The control interface is a standard network interface with a set TCP/IP address, subnet mask and the default router address, if required.

When setting up **DeviceLock EtherSensor**, it is recommended to specify DNS servers that support reverse name resolution (from IP address to the hostname) of the LAN hosts. This allows detection of names of the hosts participated in the connection, host TCP/IP addresses are assigned using DHCP.

Data received from DNS servers are cached, which decreases any added DNS server workload. Caching settings are set up using **EtherSensor Configuration Editor** utility (ethersensor_console.exe).

The control interface connection speed should be enough to pass all intercepted messages without accumulating them on the sensor. You can use 100/1000/10000 Mbit/s network interfaces.

When configuring the network control interface, account for **DeviceLock EtherSensor** special settings, which apply when filtering recognized data sent through the network control interface. For more information on these settings, refer to "EtherSensor EtherCAP Service" section.

2.2.2. Listening Interface

Network traffic interception doesn't require setting up the TCP/IP protocol stack and other protocols. This network interface captures packets to analyze their content. The traffic may come to this interface from a router mirror port, a hub or a network tap.

For normal work, it is required that the traffic on the interception interface contain all network packets, both from the server (RX) and the client (TX).

Warning!

For **DeviceLock EtherSensor** it doesn't matter whether traffic has VLAN tags or not. The network adapter driver removes these tags either way.

The interception interface bandwidth should exceed the incoming stream by at least 20%. This guarantees that the packets won't drop when captured on the network hardware, and that interception quality remains high.

The sensor can handle more than one interception interface on one server. The maximum number of these interfaces in the **DeviceLock EtherSensor** runtime environment is not limited; however, it depends on the performance of network interfaces and on the overall performance of the **DeviceLock EtherSensor** runtime environment (e.g., RAM size and CPU characteristics, etc.).

2.2.3. Setting up Switches

These are sample settings of the mirror port (SPAN) for Cisco Catalyst switches (these switches make it possible to set up two active SPAN sessions at once):

```
monitor session 1 source interface gigabitEthernet 1/0/24 both
monitor session 1 destination interface gigabitEthernet 1/0/23
```

"1/0/24" is the switch port from which traffic should be sent to **DeviceLock EtherSensor** for analysis, and "1/0/23" is the switch port to which the sensor is connected.

This is a typical sensor connection scheme in a LAN:

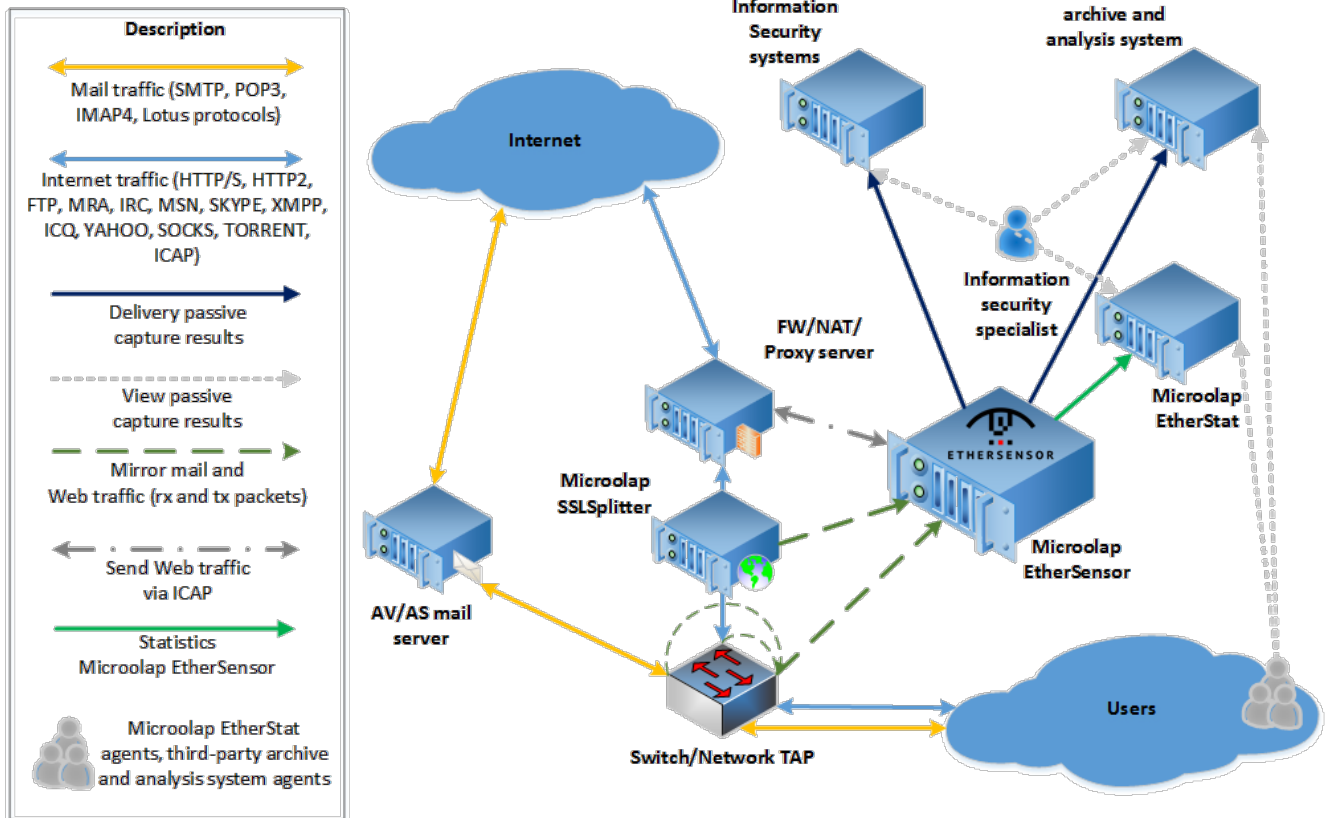


Fig. 1. Typical DeviceLock EtherSensor connection scheme in a LAN.

2.2.4. Using Third-Party Information Security Tools

To ensure smooth operation of **DeviceLock EtherSensor** it is necessary to account for compatibility with third-party information security tools and other infrastructure components:

- By default, **DeviceLock EtherSensor** is installed in the **C:\Program Files\Microolap EtherSensor** folder. Administrators can choose to install to another folder. The folder where **DeviceLock EtherSensor** is installed contains work subfolders. Exclude this folder and all its subfolders from the paths monitored by antivirus software, search indexers and integrity control tools. No software should block creating, deleting, moving or modifying files in these folders.
- DeviceLock EtherSensor** includes the following Windows services: **EtherSensor EtherCAP**, **EtherSensor ICAP**, **EtherSensor LotusTXN**, **EtherSensor Analyser**, **EtherSensor Transfer**, **EtherSensor Watcher**, and **EtherSensor Updater**, which is the update service. These services should be running with local system rights, as they require access to kernel functions.
- DeviceLock EtherSensor** needs to send SMTP, FTP, SMB, SYSLOG or IMAP messages to the remote server; otherwise, it may work incorrectly. Information security tools should not check, modify or limit connections to the server to which **DeviceLock EtherSensor** sends the data. Similarly, information security tools should not prevent the **EtherSensor Transfer** service from opening connections to ports used to send messages.

- **DeviceLock EtherSensor** requires a connection to an NDIS system module. Third-party information security tools should not prevent the **EtherSensor EtherCAP** service from using functions to access this module.
- During installation and regular work, **DeviceLock EtherSensor** processes use calls that require special privileges. The OS security policy should allow **DeviceLock EtherSensor** to perform operations with drivers, control processes and access network interfaces.
- During interception, **DeviceLock EtherSensor** can manipulate large amounts of data in work folders. The file system of the **DeviceLock EtherSensor** runtime environment should have enough free space to write and store the data.
- When working with large network streams , we recommend mounting the **[INSTALLDIR]\data** folder as a separate partition on a RAID controller optimized for access speed and write operations (*raid10*).

3. Sensor Settings

Before running the sensor, configure the following settings:

1. Make sure the license file is present in the **DeviceLock EtherSensor** installation directory.
2. Set up data sources.
3. Set up capture results delivery.
4. Set up the logging service.
5. Set up the message analysis service.
6. Set up the HTTP object filter.
7. Set up the update service.

3.1. Message Sources

In the current **DeviceLock EtherSensor** version (5.1.0.13519), the sensor runs the following services to work with data sources: **EtherSensor EtherCAP**, **EtherSensor ICAP** and **EtherSensor LotusTXN**.

The following diagram shows how the services generally work:

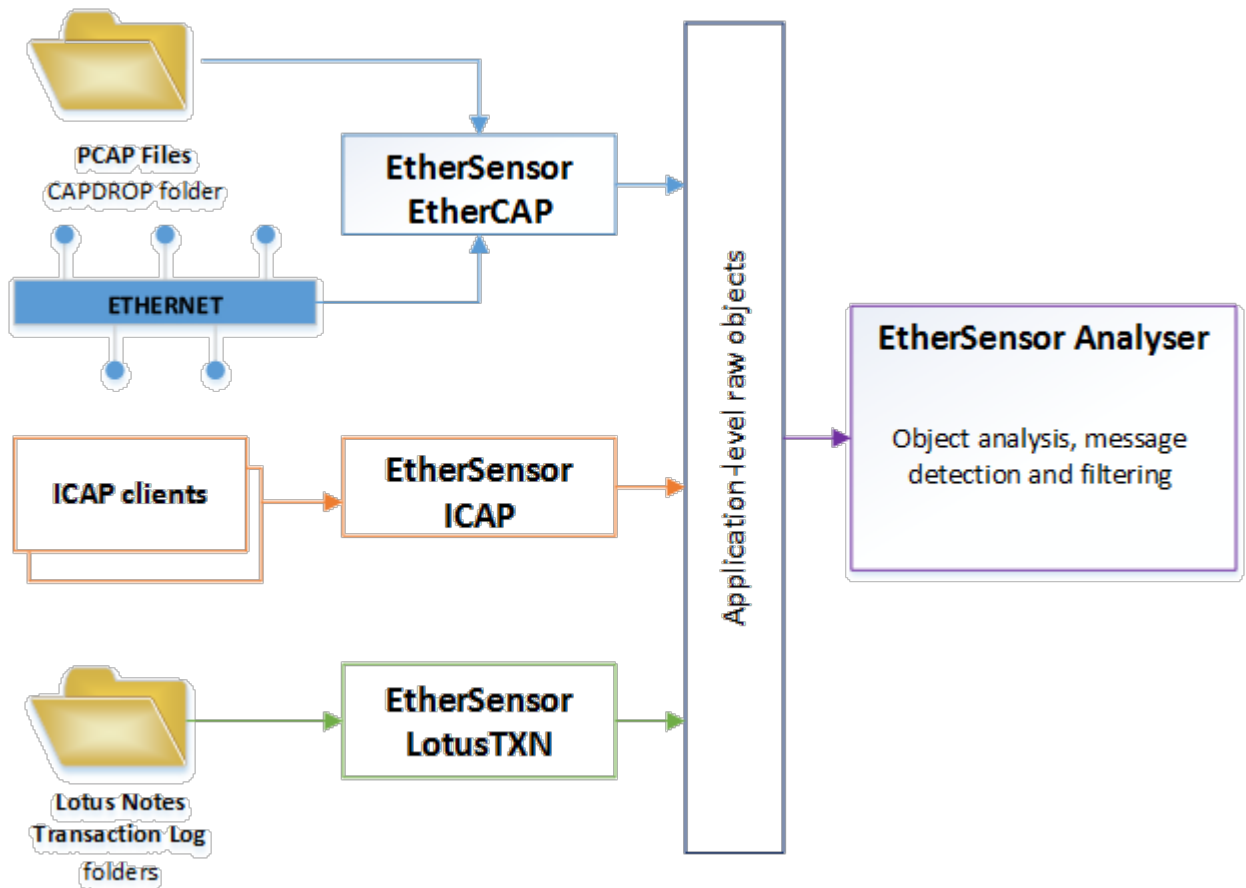


Fig. 2. EtherSensor EtherCAP, EtherSensor ICAP, EtherSensor LotusTXN services operation diagram.

The **EtherSensor EtherCAP** service is responsible for passive traffic capture at network adapters, processing the traffic from PCAP files, and the reconstruction of application-level protocol sessions:

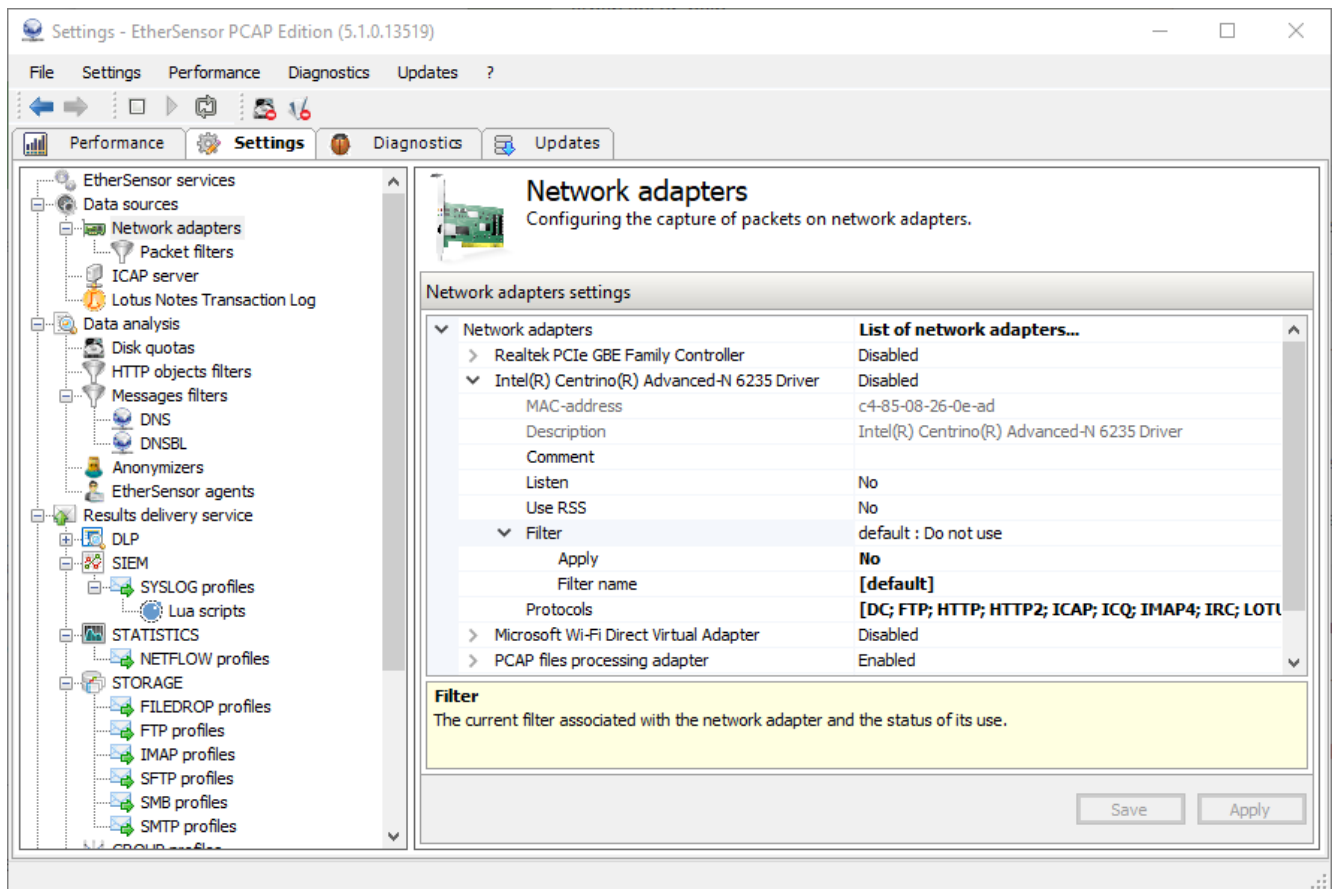


Fig. 3. EtherSensor EtherCAP service configuration window.

The **EtherSensor ICAP** service is responsible for obtaining traffic via ICAP from any ICAP clients and the subsequent delivery of received objects to the **EtherSensor Analyser** service:

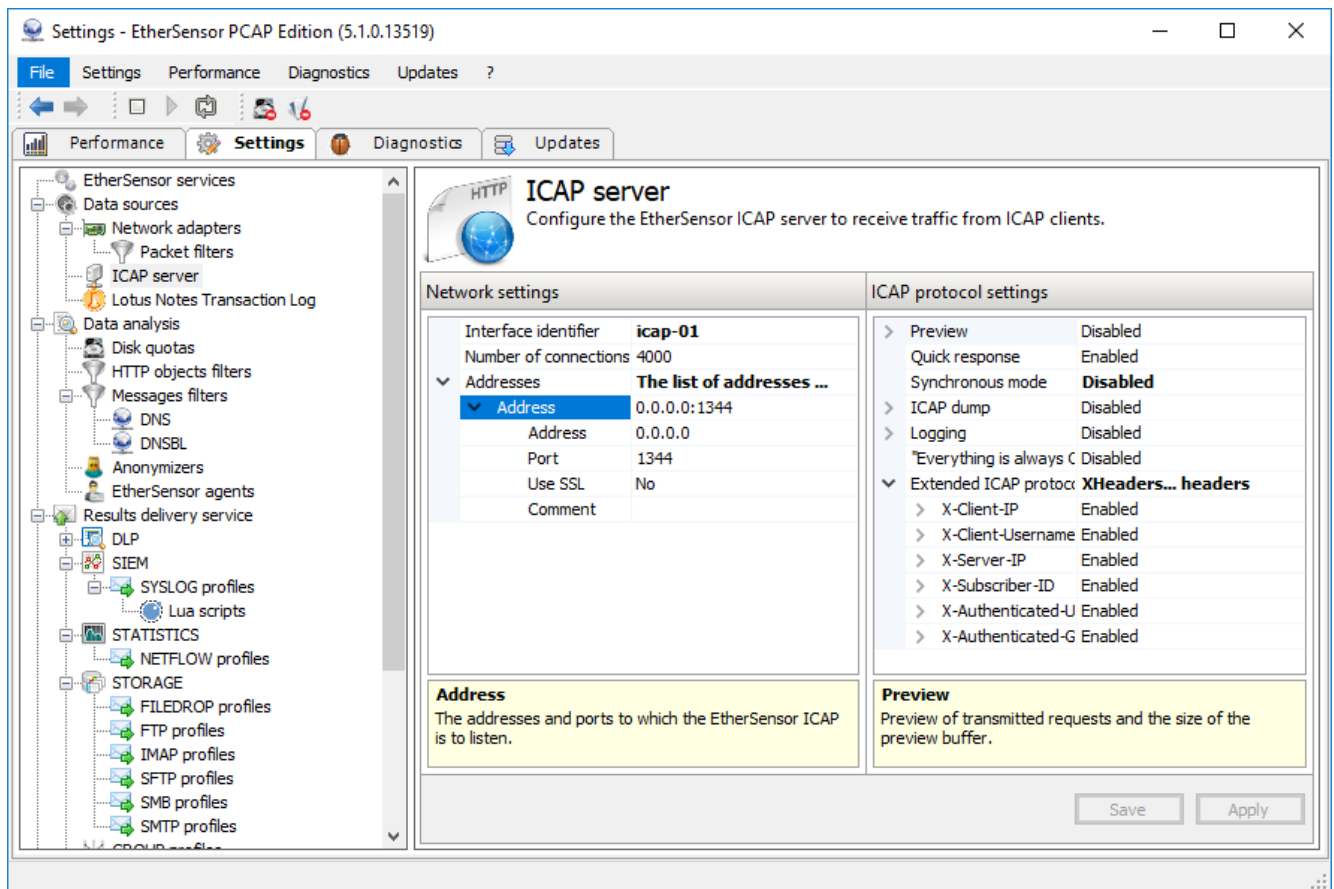


Fig. 4. EtherSensor ICAP service configuration window.

The **EtherSensor LotusTXN** service is responsible for extracting messages from **Lotus Notes Transaction Log** files:

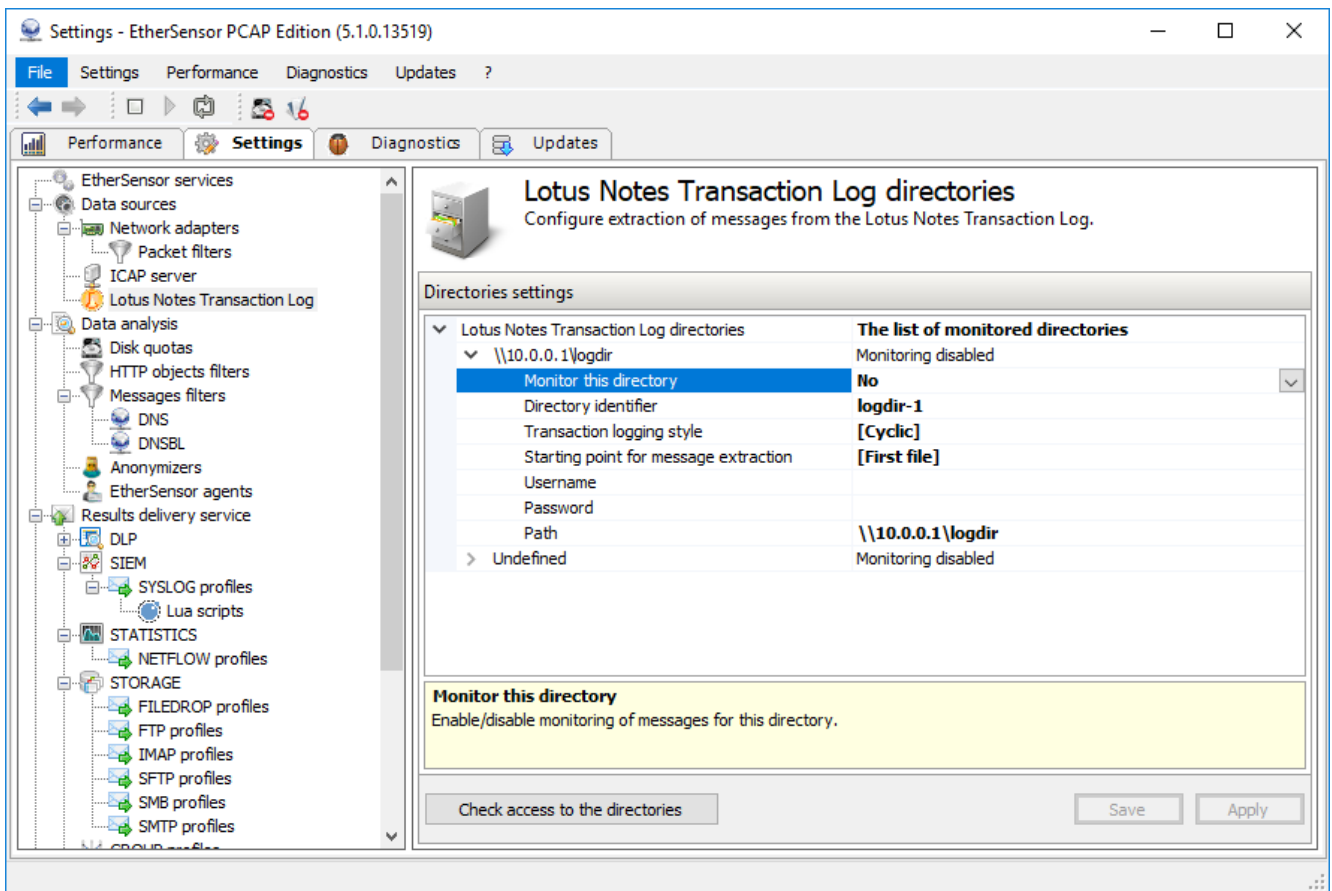


Fig. 5. EtherSensor LotustXN service configuration window.

The output of all these services is reconstructed objects delivered to the result analysis service, **EtherSensor Analyser**.

To start and stop **EtherSensor EtherCAP**, **EtherSensor ICAP** and **EtherSensor LotusTXN** services, you can use both the standard Windows **Services** framework and **EtherSensor Configuration Editor** (**ethersensor_console.exe**) from the installation directory **DeviceLock EtherSensor**:

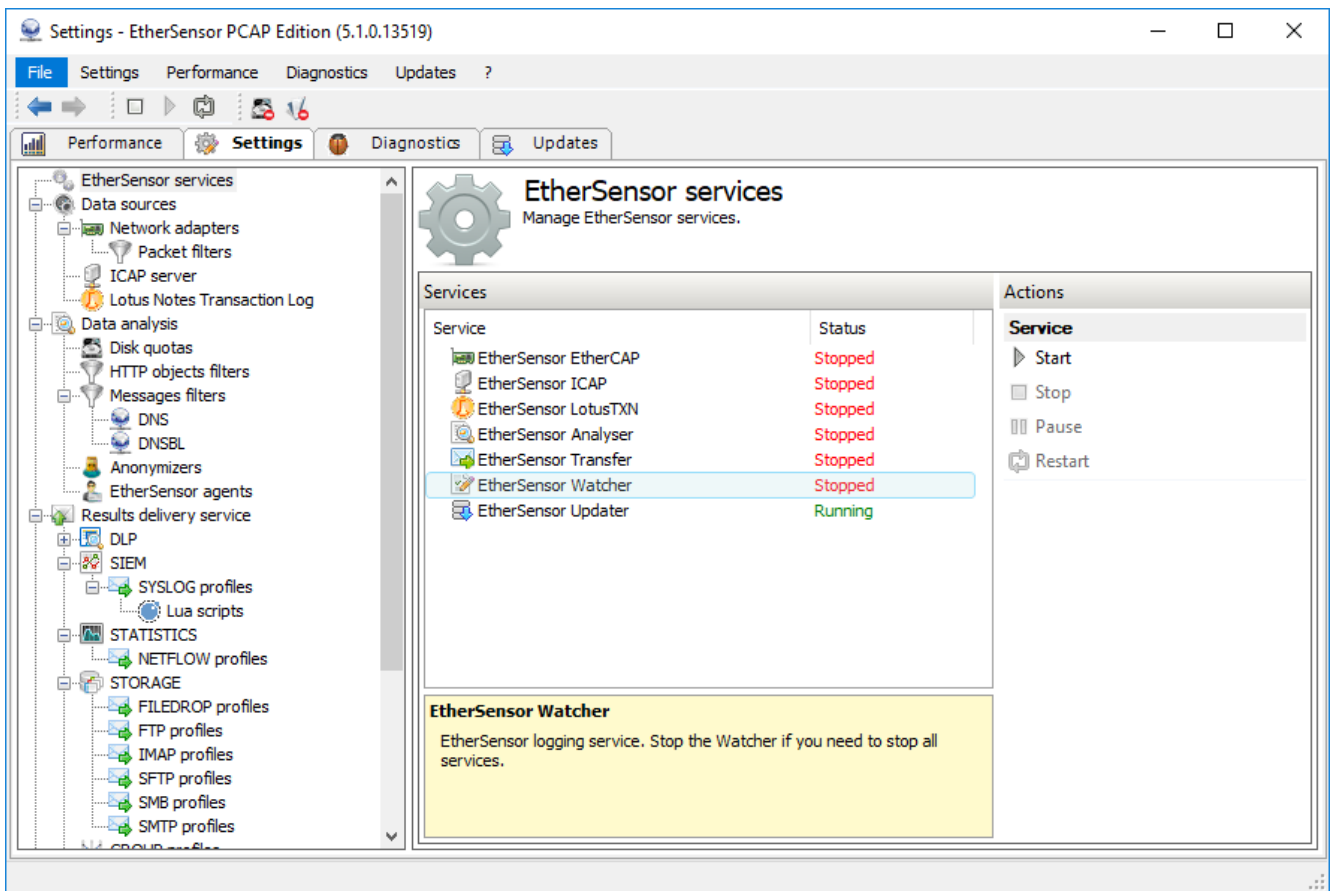


Fig. 6. Data source services start/stop.

3.1.1. EtherSensor EtherCAP

The **EtherSensor EtherCAP** service is responsible for passive traffic capture on network adapters and for processing the traffic from PCAP files in the following formats: **tcpdump/libpcap/pcapng**.

The service extracts application-level (L7 of the OSI model) objects and passes them to the **EtherSensor Analyser** service for further processing.

EtherSensor EtherCAP is capable of processing the following third-level OSI model protocols:

IP:

Regular traffic

GRE:

Tunneled traffic, e.g. unencrypted Ethernet over GRE or IP-over-IP connections

IPv6-over-IPv4:

Encapsulated IPv6 protocol (normally used in large networks and trunk channels).

In any case, **DeviceLock EtherSensor** processes chiefly TCP and UDP streams.

In the current version of **DeviceLock EtherSensor** (5.1.0.13519), **EtherSensor EtherCAP** is capable of recognizing and processing the following popular Internet protocols used to transfer application-level data:

HTTPv1/HTTPv2:

All types of requests, messages and files

SMTP:

Outgoing email messages

POP3:

Incoming email messages

IMAP4:

Incoming and outgoing email messages

ICQ:

Outgoing and incoming instant messages, contact lists and files

DC:

Instant messages over NMDC and ADC (DC++) protocols

LOTUS:

Mail messages, calendars and **Lotus Notes** tasks

MRA:

Instant messages, contact list and files via Mail.Ru Agent

MSN:

Instant messages, contact list and files via MSN

XMPP:

Instant messages, contact list and files via XMPP clients (Google Talk, etc.)

IRC:

Instant messages and files

SKYPE:

Detection of skype client usage and message extraction

SSL:

Detection of SSL usage

TORRENT:

Detection of Torrent client usage

FTP:

File transfer

Yahoo:

Instant messages and files

ICAP:

Capture of data transferred over ICAP protocol

SOCKS:

Capture of data transferred over SOCKS protocol

WEBSOCKET:

Capture of data transferred over WEBSOCKET protocol

The following diagram shows how the **EtherSensor EtherCAP** service generally works:

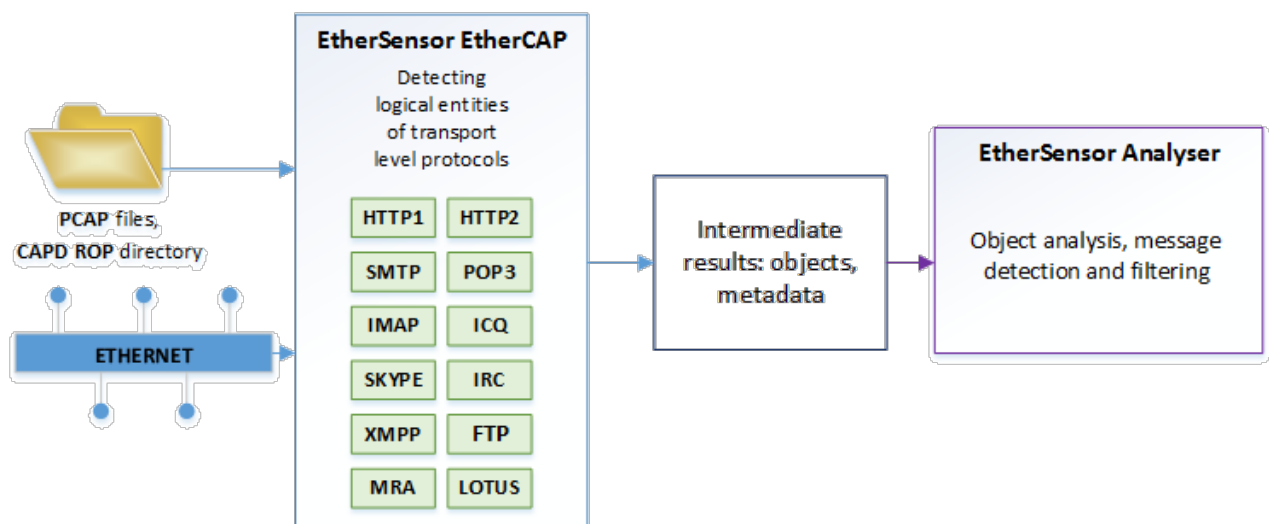


Fig. 7. EtherSensor EtherCAP service operation diagram.

The **EtherSensor EtherCAP** service enables capture of traffic from all available Ethernet interfaces, together with monitoring the local directory to process PCAP files put into it.

Defining the required hardware resources per captured network interface, keep in mind that the average amount of RAM required to process one TCP connection is about 40 Kbyte:

Network interface bandwidth	Amount of RAM required to cache packets being processed
10,000 Mbit	2,000 MB
5,000 Mbit	1,000 MB
1,000 Mbit	200 MB
100 Mbit	50 MB
10 Mbit	10 MB

Therefore, to simultaneously monitor 10,000 TCP sessions through a 1 Gbps network interface, **DeviceLock EtherSensor** needs the following amount of available RAM:

$$200 \text{ MB} + 10000 * 40 \text{ KB} = \text{about } 600 \text{ MB}$$

To optimize processing, the **EtherSensor EtherCAP** service lets you assign a packet filter to each captured network interface, as well as network protocols required for monitoring.

Ways to set up the service operation:

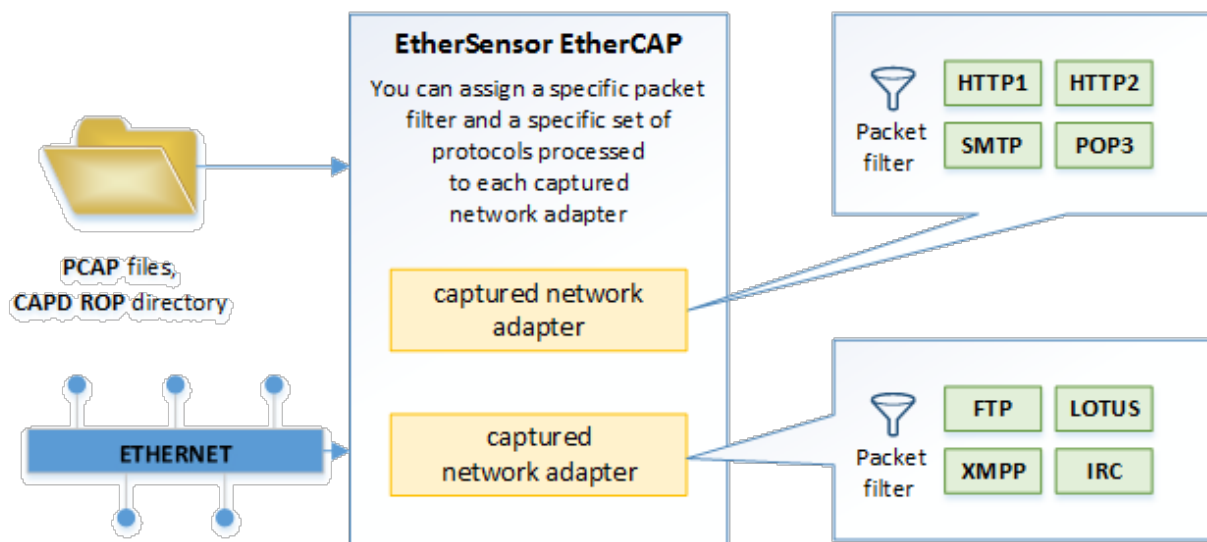


Fig. 8. Ways to set up the EtherSensor EtherCAP service operation.

To process PCAP files, the service simulates work with the network interface, which can be set up using the service configuration file. In the configuration file, a unique name is assigned to the network interface: **capdrop**. In the current version of **DeviceLock EtherSensor** (5.1.0.13519), the **EtherSensor EtherCAP** service supports only the two most popular PCAP file formats: **tcpdump/libpcap** and **pcapng**.

Command Line Parameters

The Windows **EtherSensor EtherCAP** service is set up to start automatically during **DeviceLock EtherSensor** installation. However, you can start the **ethersensor_ethercap.exe** process as a Windows application using the following command line parameters:

/process

Starts the **ethersensor_ethercap.exe** process as a regular Win32 process (may be helpful for debugging)

/service

Starts as a Windows service

/config

Saves the service default configuration

3.1.1.1. Setting up the Configurator

The **EtherSensor EtherCAP** service is capable of capturing both hardware network interfaces installed in the runtime environment of **DeviceLock EtherSensor** and a special virtual device, **PCAP files processing adapter**, intended to process the traffic previously gathered in the **tcpdump/libpcap pcap** or **pcapng** formats.

Hardware Network Interface Settings

The information on the network interface settings and packet filters is stored in the **ethcap.xml** file, located in the **[INSTALLDIR]\config** directory. You can edit the file either in the configurator or in a text editor.

In the configurator (**ethersensor_console.exe**), set up the **EtherSensor EtherCAP** service as follows:

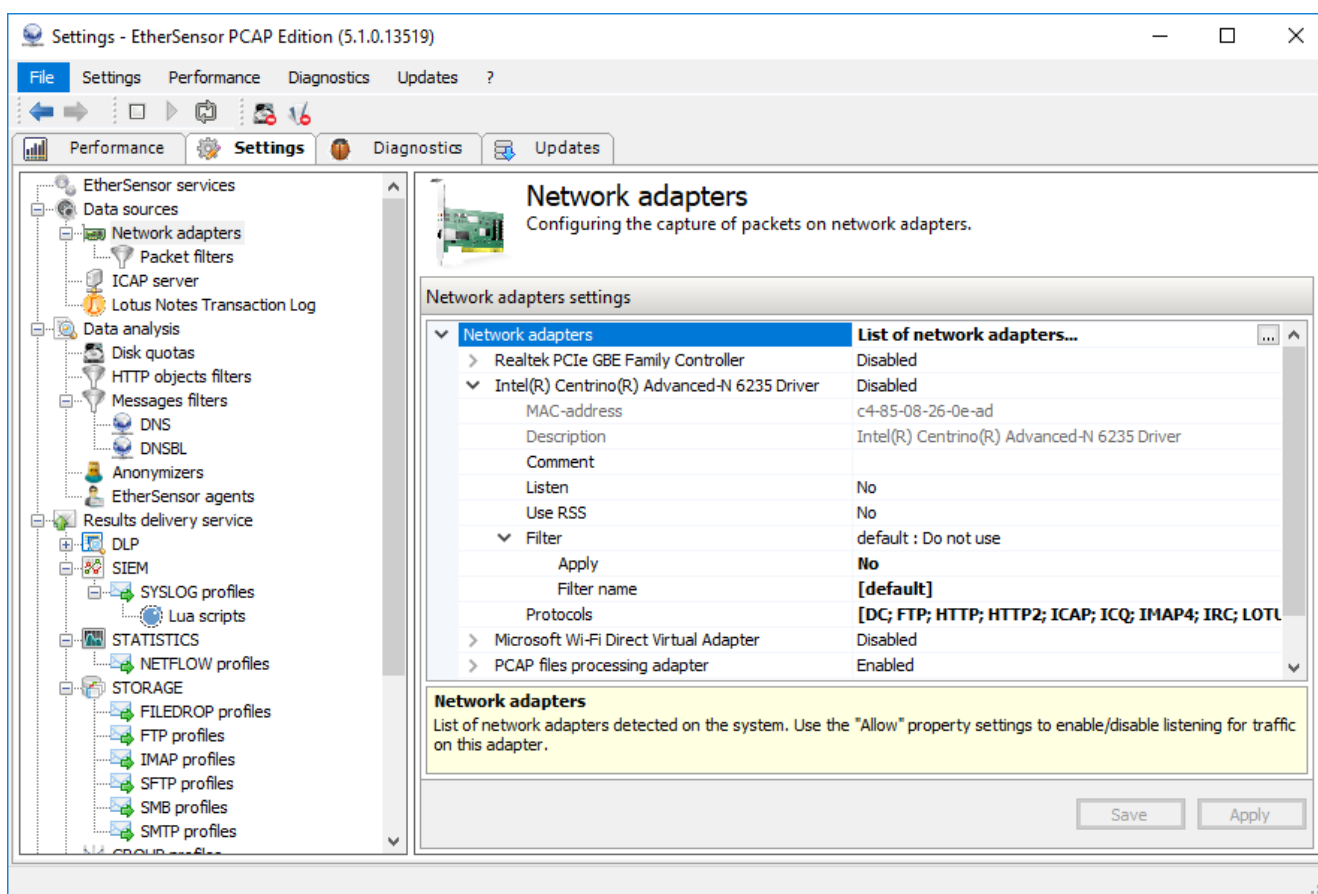


Fig. 9. Setting up EtherSensor EtherCAP.

MAC-address:

The MAC address of the network interface being configured. The MAC address is used in all messages sent by **DeviceLock EtherSensor** to the consumer system.

Listen:

Enables/disables capturing this adapter by the **EtherSensor EtherCAP** service. It can capture multiple network adapters simultaneously. The **EtherSensor EtherCAP** service will capture all interfaces that are not configured.

To capture an interface configured for the OS, set the value for the **Listen** parameter to **Yes**. To avoid capturing interfaces where the OS stack is not installed, set the **Listen** parameter for those interfaces to **No**.

Use RSS:

Enables/disables the use of RSS technology. Receive Side Scaling (RSS) is a technology that equally distributes the network packet processing workload among the CPU cores, optimizing the performance.

Filter:

Displays the name of the filter associated with this adapter and its usage status (**Use/Do not use**). You can have as many pre-prepared filters as you want, but only one of them can be used on a given network adapter at any time.

Filter name:

Selects the packet filter. Filters are stored in the **[INSTALLDIR]\configethcap.xml** file.

Apply:

Enables/disables the specified filter on network interface.

Protocols:

Allows you to save resources by disabling unused protocol filters. For example, if you don't need to work with instant messages, disable filters for the following protocols: **DC, ICQ, IRC, MRA, MSN, SKYPE** and **YAHOO**.

PCAP Files Processing Adapter

The **PCAP files processing adapter** is a special virtual device intended to process traffic previously saved in PCAP files (e.g., received in another network segment). The **PCAP files processing adapter** settings are identical to those for hardware adapters:

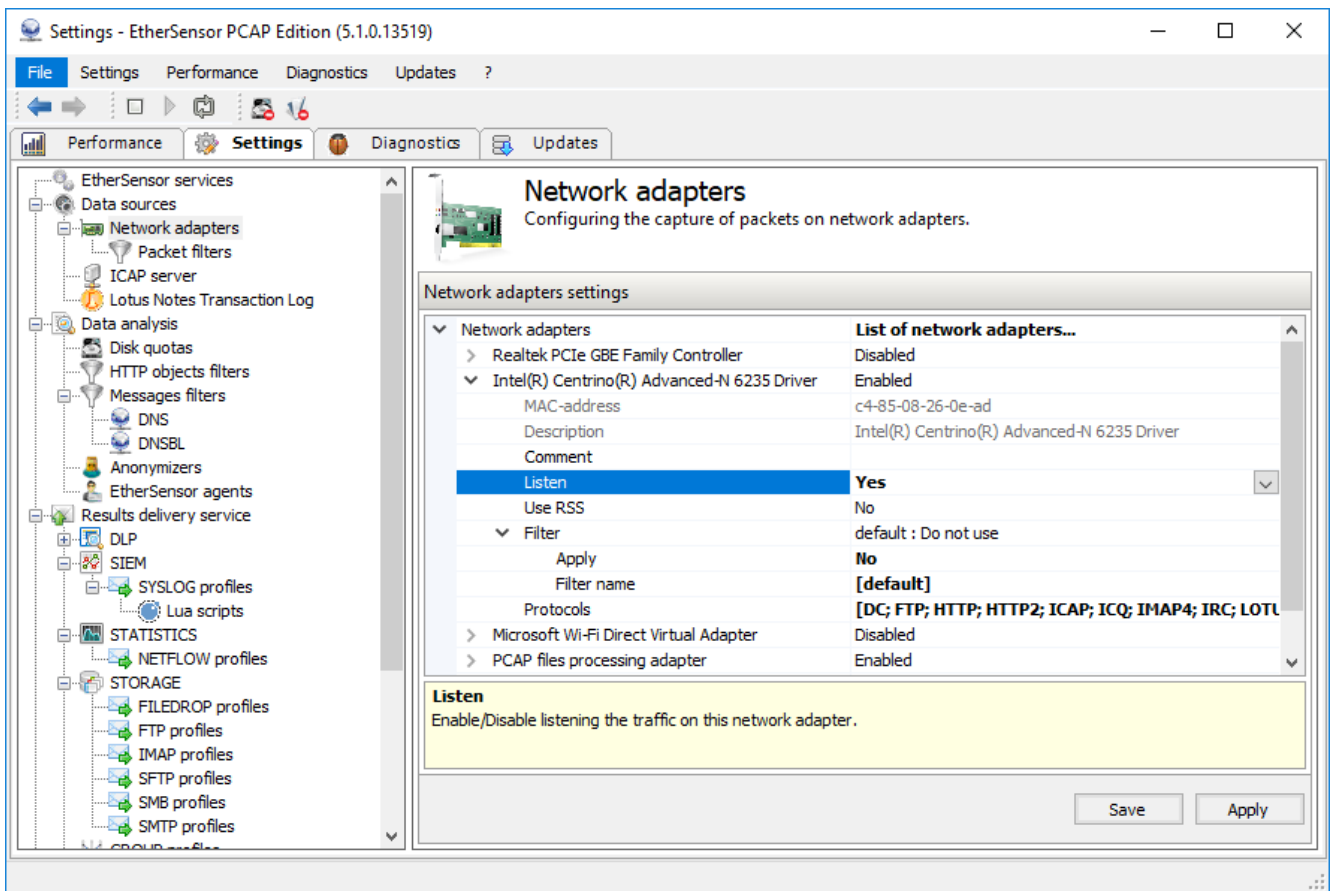


Fig. 10. PCAP files processing adapter settings.

To process PCAP files, copy them to the `[INSTALLDIR]\data\capdrop` directory and then use the configurator to allow the **PCAP files processing adapter** to capture them. PCAP files are captured in FIFO order: the first file moved to the `[INSTALLDIR]\data\capdrop` directory (i.e., one that has an earlier modification date in the file system) will be "captured" first, the second file will be "captured" second, etc.

If the **PCAP files processing adapter** is **Enabled**, the `[INSTALLDIR]\data\capdrop` directory was empty, and a PCAP file has been moved into it, this file will be captured immediately.

If the **PCAP files processing adapter** is **Disabled**, and there were PCAP files in the `[INSTALLDIR]\data\capdrop` directory, then file capture starts immediately after the adapter is **Enabled**.

Processed PCAP files are moved to the `[INSTALLDIR]\data\capdrop\processed` directory without any changes, so you can re-process them as many times as you want (for example, to debug filters).

Messages received by the **EtherSensor EtherCAP** service from PCAP files are then processed in the usual way.

Creating and Configuring Packet Filters

To create and edit packet filters, use the configurator form in the **Packet filters** section:

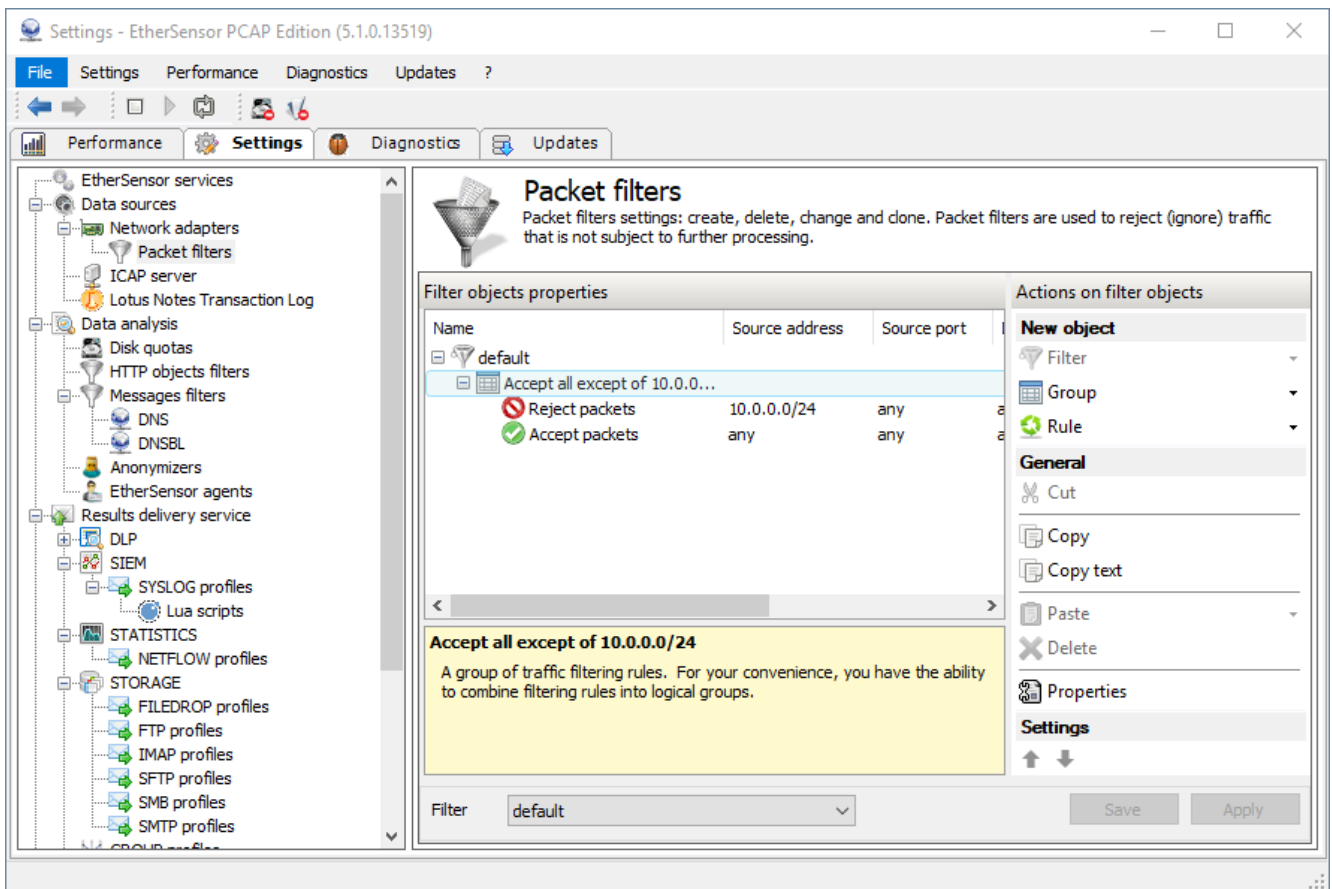


Fig. 11. Packet filter settings.

Actions on filter objects panel:

Allows for the creation, cloning and deletion of filters and their objects: groups and rules.

Filter objects properties panel:

Enables editing of filter, group and rule properties.

To open the object property editing panel, double-click the respective property: a filter, a rule group or a rule:

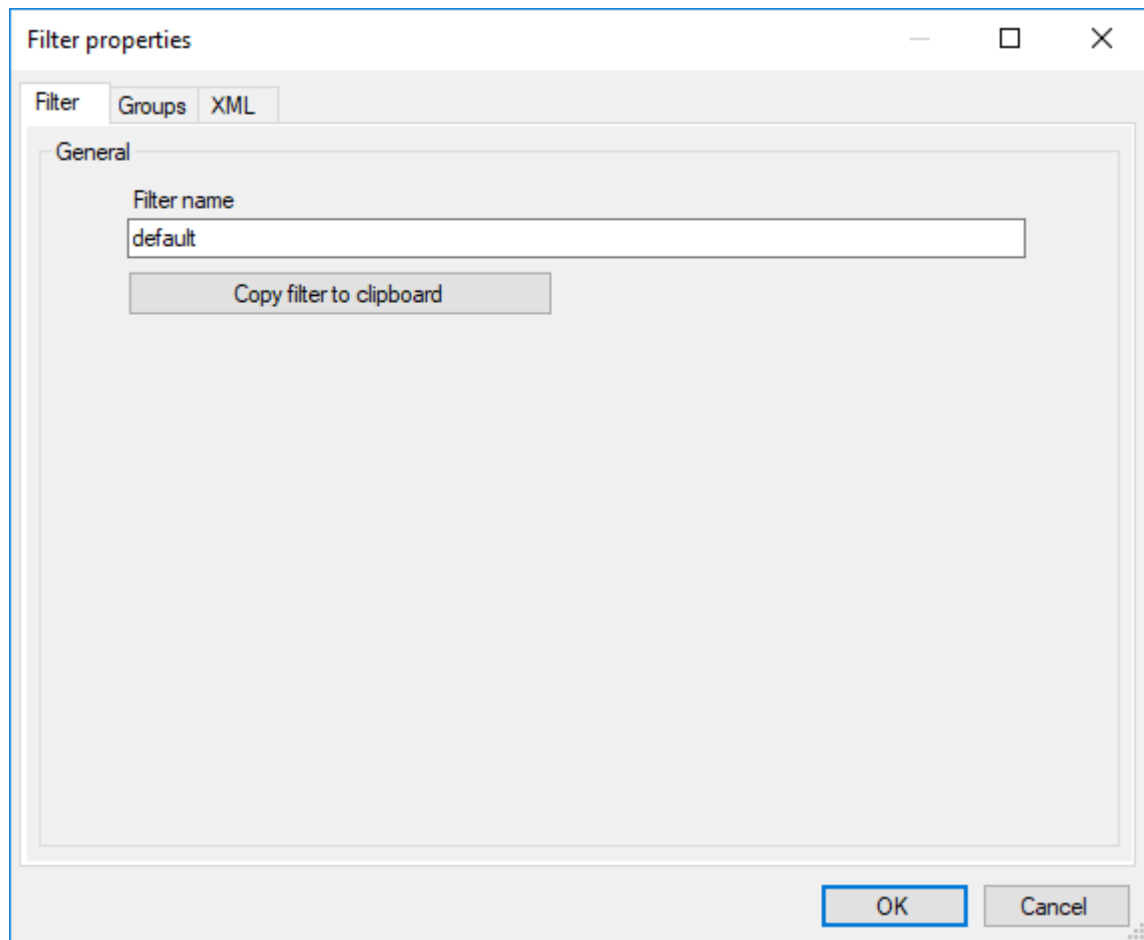


Fig. 12. Filter property editing panel.

Filter name:

A filter name (admins can assign any name).

Filter rule group tab:

BPF program imposes no length limits and can accommodate a large number of rules. For your convenience, you can group rules and assign intuitive names to groups.

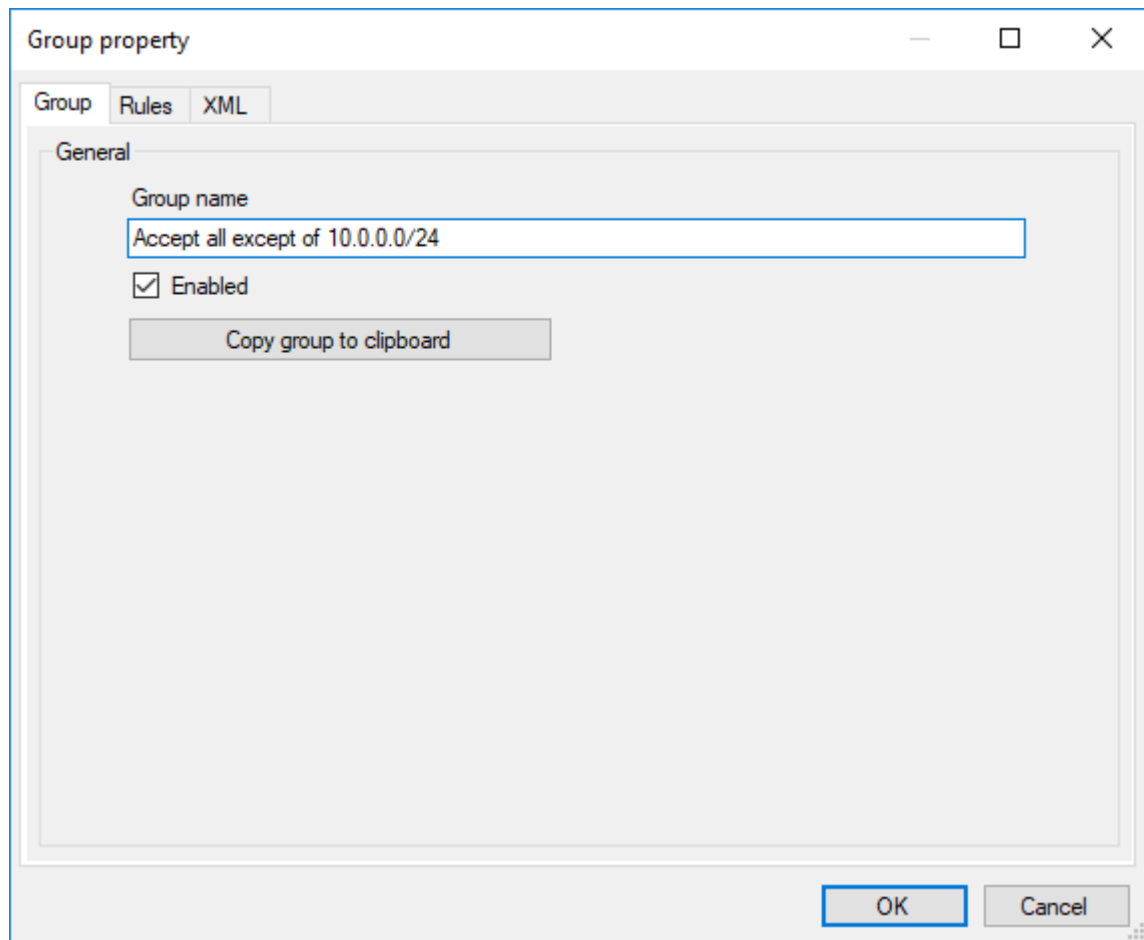


Fig. 13. Rule group property editing panel.

Group name:

Rule group name (admins can assign any name).

Enabled checkbox:

Enables/disables a rule group.

Rules tab:

You can edit rules from both the filter level and the **Rules** tab of the rule group properties.

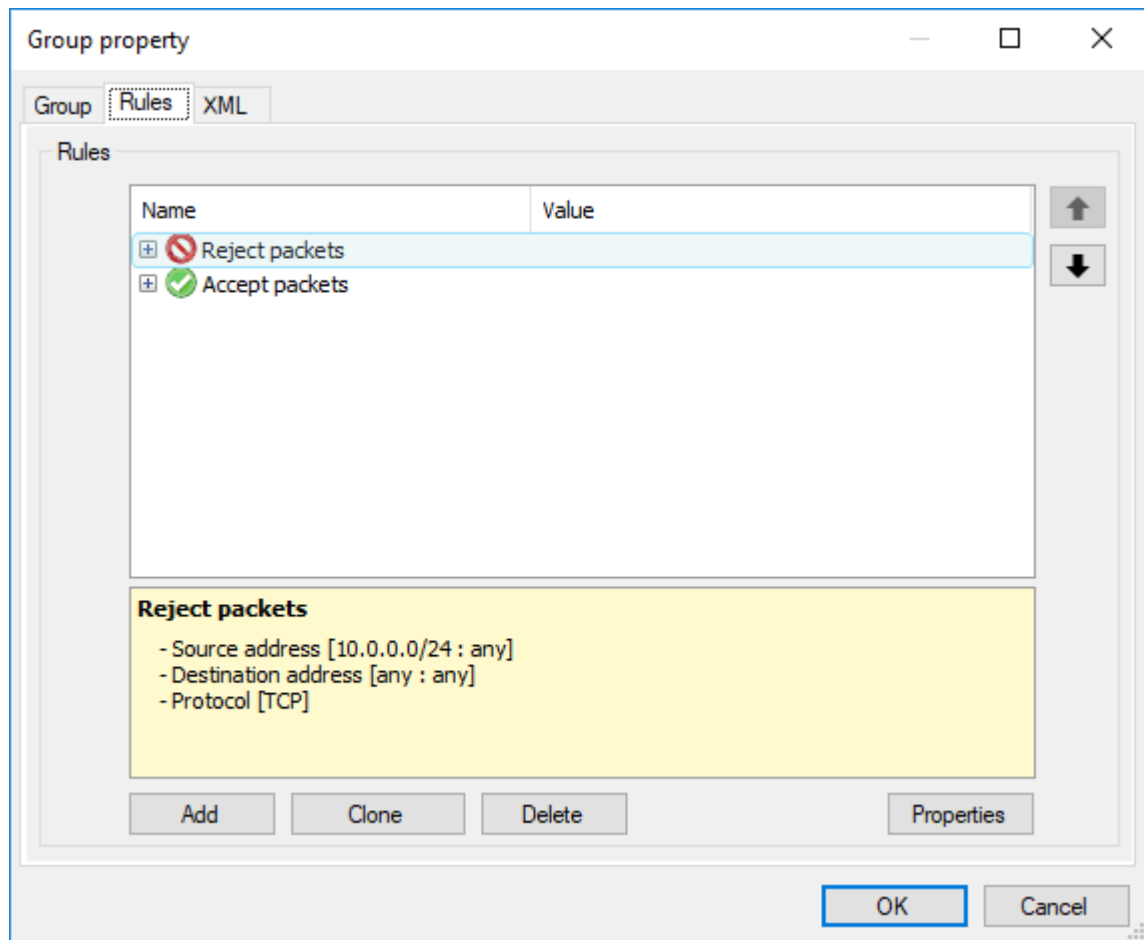


Fig. 14. Edit rule panel.

To open the edit rule action and conditions panel, double-click the rule name.

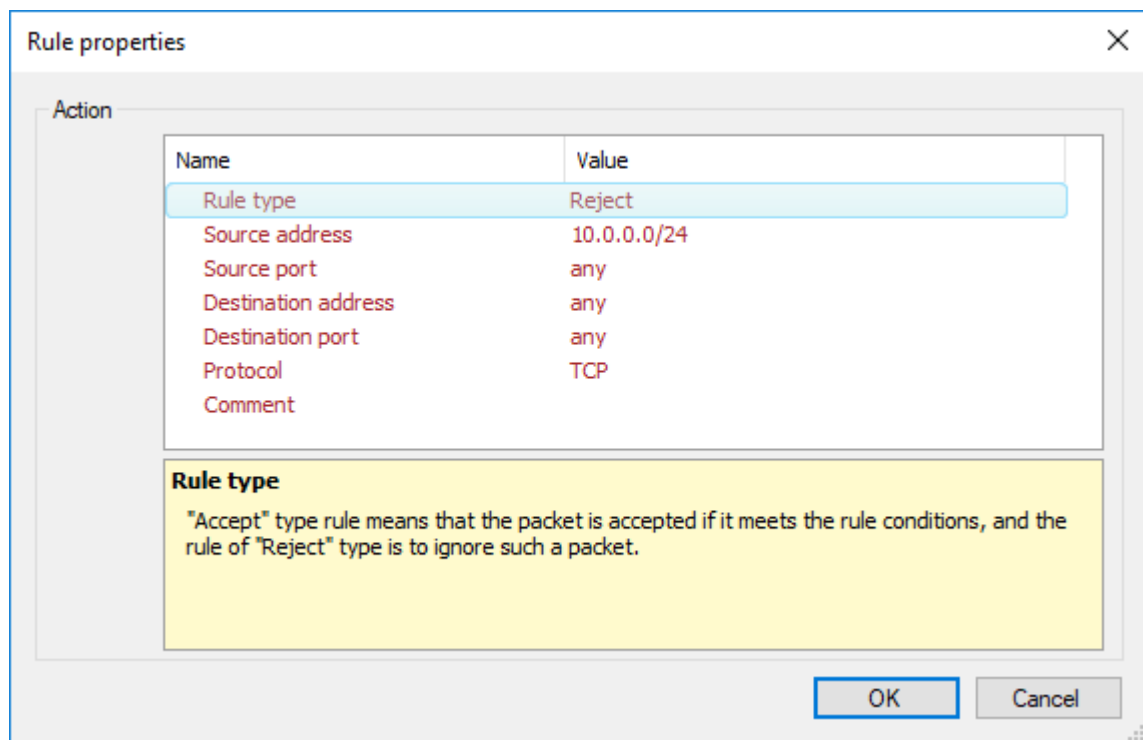


Fig. 15. Edit rule panel.

Rule type:

There are two rule types: **Accept** and **Reject**. **Accept** means packets that meet the rule conditions are accepted. **Reject** means such packets are ignored.

Warning!

Filter rules are always applied in sequence. Changing the order of the rules in a filter may radically change the results.

Source address:

The source address. For example: **any** or **10.0.0.0/24** or **10.0.0.0-10.0.0.255** or a comma-separated list, like **100.100.100.1-100.100.100.255, 192.168.0.0/8**.

Source port:

The source port, for example: **any** or **80** or **80-8080** (80 through 8080), or **80-8000, 9000-10000** (80 through 8000 and 9000 through 10000).

Destination address:

The destination address. For example: **any** or **10.0.0.0/24** or **10.0.0.0-10.0.0.255** or a comma-separated list, like **100.100.100.1-100.100.100.255, 192.168.0.0/8**.

Destination port:

The destination port, for example: **any** or **80** or **80-8080** (80 through 8080), or a comma-separated list, like **80-8000, 9000-10000** (80 through 8000 and 9000 through 10000).

Protocol:

One of **TCP, UDP, GRE, IP6** or **any**.

Comment

Your comment on this filter rule.

For more information on **EtherSensor EtherCAP** service settings, refer to the "Manual Setup (Config File)" section.

3.1.1.2. Manual Setup (Config File)

The **EtherSensor EtherCAP** service configuration is stored in the **ethcap.xml** file located in the **DeviceLock EtherSensor common configuration directory**. **[INSTALLDIR]\config**

```
<?xml version="1.0" encoding="utf-8"?>
<EtherCapConfig version="4.2"
  flow_count="16"
  flow_buff_count="512"
  flow_buff_size="524288">
  <NetworkAdapters>
    <NetworkAdapter enabled="true" rss="true" mac="00-1F-C6-2D-EA-40"
      description="Marvell Yukon 88E8056 PCI-E Gigabit Ethernet Controller">
      <Filter enabled="true" name="internet" />
      <Protocol enabled="true" name="dc" />
      <Protocol enabled="true" name="ftp" />
      <Protocol enabled="true" name="http" />
      <Protocol enabled="true" name="icq" />
      <Protocol enabled="true" name="imap4" />
      <Protocol enabled="true" name="irc" />
      <Protocol enabled="true" name="lotus" />
      <Protocol enabled="true" name="mra" />
      <Protocol enabled="true" name="msn" />
      <Protocol enabled="true" name="pop3" />
      <Protocol enabled="true" name="skype" />
      <Protocol enabled="true" name="smtp" />
      <Protocol enabled="true" name="ssl" />
      <Protocol enabled="true" name="xmpp" />
      <Protocol enabled="true" name="yahoo" />
    </NetworkAdapter>
    <NetworkAdapter enabled="true" mac="capdrop"
      description="PCAP files processing adapter">
      <Filter enabled="true" name="default" />
      <Protocol enabled="true" name="dc" />
      <Protocol enabled="true" name="ftp" />
      <Protocol enabled="true" name="http" />
      <Protocol enabled="true" name="icq" />
      <Protocol enabled="true" name="imap4" />
      <Protocol enabled="true" name="irc" />
      <Protocol enabled="true" name="lotus" />
      <Protocol enabled="true" name="mra" />
      <Protocol enabled="true" name="msn" />
      <Protocol enabled="true" name="pop3" />
      <Protocol enabled="true" name="skype" />
      <Protocol enabled="true" name="smtp" />
      <Protocol enabled="true" name="ssl" />
      <Protocol enabled="true" name="xmpp" />
      <Protocol enabled="true" name="yahoo" />
    </NetworkAdapter>
  </NetworkAdapters>

  <Filters>

  <Filter name="default">
    <RuleGroup enabled="true" name="">
      <Rule
        type="accept"
        src="any"
        srcport="any"
        dst="any"
        dstport="any"
        proto="tcp"
        comment="Comment to the rule" />
    </RuleGroup>
  </Filter>

  <Filter name="internet">
    <RuleGroup enabled="true" name="">
      <Rule
```

```
    type="reject"
    src="192.168.0.1"
    srcport="any"
    dst="any"
    dstport="any"
    proto="tcp"
    comment="Comment to the rule" />
<Rule
  type="reject"
  src "*"
  srcport="any"
  dst="192.168.0.1"
  dstport="any"
  proto="tcp" />
<Rule
  type="accept"
  src="any"
  srcport="any"
  dst="any"
  dstport="any"
  proto="tcp" />
</RuleGroup>
</Filter>

</Filters>
</EtherCapConfig>
```

EtherCapConfig tag

This is the root tag of the service configuration. The **version** attribute specifies the configuration files version.

The **flow_count** attribute is used to specify the number of simultaneously processed threads. All captured traffic is equally distributed among the processing threads. The traffic distribution occurs at the operation system kernel level; this ensures parallel data processing.

The **flow_buff_count** attribute is used to specify the number of data buffers in the traffic processing thread. Together with the **flow_buff_size** attribute, it specifies the amount of static memory to be used for a single processing thread.

The **flow_buff_size** attribute specifies the size of the data buffer in the traffic processing thread. Provide a value in bytes. Together with the **flow_buff_count** attribute, it specifies the amount of static memory to be used for a single processing thread. The amount of memory for a single thread will be (**flow_buff_count * flow_buff_size**) = (512 * 524288 = 256 MB)

Warning!

The above-listed attributes are used for fine configuration of **DeviceLock EtherSensor** and require a sophisticated understanding of how the product operates. Don't experiment with them in a live system!

NetworkAdapters tag

Defines the settings of the network interfaces being captured.

NetworkAdapter tag

The **NetworkAdapter** tag is nested within the **NetworkAdapters** tag. It specifies the description of settings for each particular network interface. The **enabled** attribute specifies the network interface activity status. If it is set to **false**, the network interface will not be used in data processing (i.e., the traffic from this interface will be ignored).

The **rss** attribute is used to use Receive Side Scaling (RSS) technology. This technology equally distributes the network packet processing workload among the CPU cores, optimizing the performance.

The **mac** attribute is used to specify the name of the network interface. This attribute is read-only. The **description** attribute contains a description of the network interface. Admins can provide any value for this attribute.

The Filter Tag

The **Filter** tag is nested within the **NetworkAdapter** tag. It specifies a description for the IP filter used for a particular network adapter. The **enabled** attribute specifies the IP filter usage status. If it is set to **false**, no IP filter will be used to process data from this network interface. The **name** attribute specifies the name of the IP filter profile. IP filter profiles are specified in the **Filters** tag.

The Protocol Tag

The **Protocol** tag is nested within the **NetworkAdapter** tag. It specifies a description for the Internet protocol used to process data. The **enabled** attribute specifies the Internet protocol usage status. If it is set to **false**, this Internet protocol will be ignored for this network interface. The **name** attribute specifies the Internet protocol name. This attribute is read-only.

Example 1:

Network interface settings to capture a client's DC, ICQ, IRC, MRA, MSN, XMPP/Jabber and/or Yahoo messages:

```
<NetworkAdapter
  enabled="true"
  mac="capdrop"
  description="PCAP files processing adapter">
  <Filter enabled="true" name="default" />
    <Protocol enabled="true" name="dc" />
    <Protocol enabled="false" name="ftp" />
    <Protocol enabled="false" name="http" />
    <Protocol enabled="true" name="icq" />
    <Protocol enabled="false" name="imap4" />
    <Protocol enabled="true" name="irc" />
    <Protocol enabled="false" name="lotus" />
    <Protocol enabled="true" name="mra" />
    <Protocol enabled="true" name="msn" />
    <Protocol enabled="false" name="pop3" />
    <Protocol enabled="false" name="skype" />
    <Protocol enabled="false" name="smtp" />
    <Protocol enabled="false" name="ssl" />
    <Protocol enabled="true" name="xmpp" />
    <Protocol enabled="true" name="yahoo" />
</NetworkAdapter>
```

Filters tag

Defines IP filter profile settings.

Filter tag

The **Filter** tag is nested within the **Filters** tag. It specifies a description of the IP filter settings. The **name** attribute specifies the name of the IP filter profile. This attribute value may be used as the **NetworkAdapter/Filter/name** attribute value to specify the IP filter for the network interface.

The RuleGroup Tag

The **RuleGroup** tag is nested within the **Filter** tag. It is used to group filter rules related to a particular traffic filtering problem. The **name** attribute defines the name (description) of the filter rule group. The value for this attribute may be empty.

Rule tag

The **Rule** tag is nested within the **RuleGroup** tag. It specifies a description for the network traffic filter rule. The **type** attribute specifies the rule type. If it is set to **accept**, then network packets that match the rule will be passed for further processing. If it is set to **reject**, then network packets that match the rule will be rejected. The **src**, **dst** attributes specify an IP address, a range of IP addresses or network parameters to filter necessary IP addresses that match the specified value. In the **comment** attribute, you can provide a comment for the packet filter rule.

Example 2:

Reject packets passing between 10.1.5.10, 10.1.5.15-10.1.5.59 machines and 10.1.6.0/255.255.255.0 network:

```
<Rule
  type="reject"
  src="10.1.5.10, 10.1.5.15-10.1.5.59"
  dst="10.1.6.0/255.255.255.0"
  proto="tcp"
  comment="" />

<Rule type="reject"
  src="10.1.6.0/255.255.255.0"
  dst="10.1.5.10, 10.1.5.15-10.1.5.59"
  proto="tcp" />
```

The **srcport**, **dstport** attributes specify TCP ports or a range of TCP ports for filtering.

Example 3:

Reject packets passing between 10.1.5.10, 10.1.5.15-10.1.5.59 machines and 10.1.6.0/255.255.255.0 network on ports 80, 443-1024:


```
<Rule
  type="reject"
  src="10.1.5.10, 10.1.5.15-10.1.5.59"
  srcport="80, 443-1024"
  dst="10.1.6.0/255.255.255.0"
  proto="tcp" />

<Rule
  type="reject"
  src="10.1.6.0/255.255.255.0"
  dst="10.1.5.10, 10.1.5.15-10.1.5.59"
  dstport="80, 443-1024"
  proto="tcp" />
```

The rules are applied in a linear fashion, i.e. from top to bottom. The top line is the first filter instruction, and the bottom line is the last one. Each line rejects or accepts only the type of packets it describes. Therefore, to reject a connection between two hosts or a group of hosts, you need to reject the traffic in both directions.

Example:

```
<Rule
  type="reject"
  src="10.31.5.212"
  dst="10.31.5.57"
  dstport="1025"
  proto="tcp" />

<Rule
  type="reject"
  src="10.31.5.57"
  srcport="1025"
  dst="10.31.5.212"
  proto="tcp" />
```

Also, remember that, if no filter rule is defined, all traffic will be accepted. Conversely, if filter rules exist, only the traffic defined by these rules will be processed.

Example:

1. Receive all connections only to/from 10.31.5.57

```
<Rule
  type="accept"
  src="10.31.5.57"
  srcport="*"
  dst="*"
  dstport="*"
  proto="tcp" />

<Rule
  type="accept"
  src="*"
  srcport="*"
  dst="10.31.5.57"
  dstport="*"
  proto="tcp" />
```

2. To reject a group of hosts, you need first to reject this group's packets, and then to accept all other packets; otherwise, all traffic will be passed without being analyzed:

```
<Rule
  type="reject"
  src="10.31.5.212"
  dst="10.31.5.57"
  dstport="1025"
  proto="tcp" />

<Rule
  type="reject"
  src="10.31.5.57"
  srcport="1025"
  dst="10.31.5.212"
  proto="tcp" />

<Rule
  type="accept"
  src "*"
  srcport "*"
  dst "*"
  dstport "*"
  proto="tcp" />
```

This allow you to reject traffic from one side of a proxy server.

3. If you define rules that accept traffic only from certain hosts and reject the remaining traffic, only traffic from these hosts will be processed:

```
<Rule
  type="accept"
  src="10.31.5.212"
  dst="10.31.5.57"
  dstport="1025"
  proto="tcp" />

<Rule type="accept"
  src="10.31.5.57"
  srcport="1025"
  dst="10.31.5.212"
  proto="tcp" />
```

3.1.2. EtherSensor ICAP

The **EtherSensor ICAP** service is an ICAP server designed to receive the ICAP network traffic from any ICAP clients in REQMOD mode.

ICAP (Internet Content Adaptation Protocol) is designed to work only with HTTP and is used to filter the content and detect any harmful content (such as viruses, spyware/malware).

The ICAP client is the system that transfers HTTP traffic. It may include various HTTP proxies that support ICAP (e.g., SQUID, Blue Coat Proxy SG, Cisco IronPort S or Webwasher). Upon receipt of client data, some ICAP servers may process and modify them, if necessary.

Then the data are returned to the ICAP client that sends them on to a server or a client, depending on where the data have been directed.

Because the **DeviceLock EtherSensor** ICAP server only analyzes the traffic received via ICAP clients, the traffic always returns to the ICAP client without change. The system architecture using ICAP is shown below:

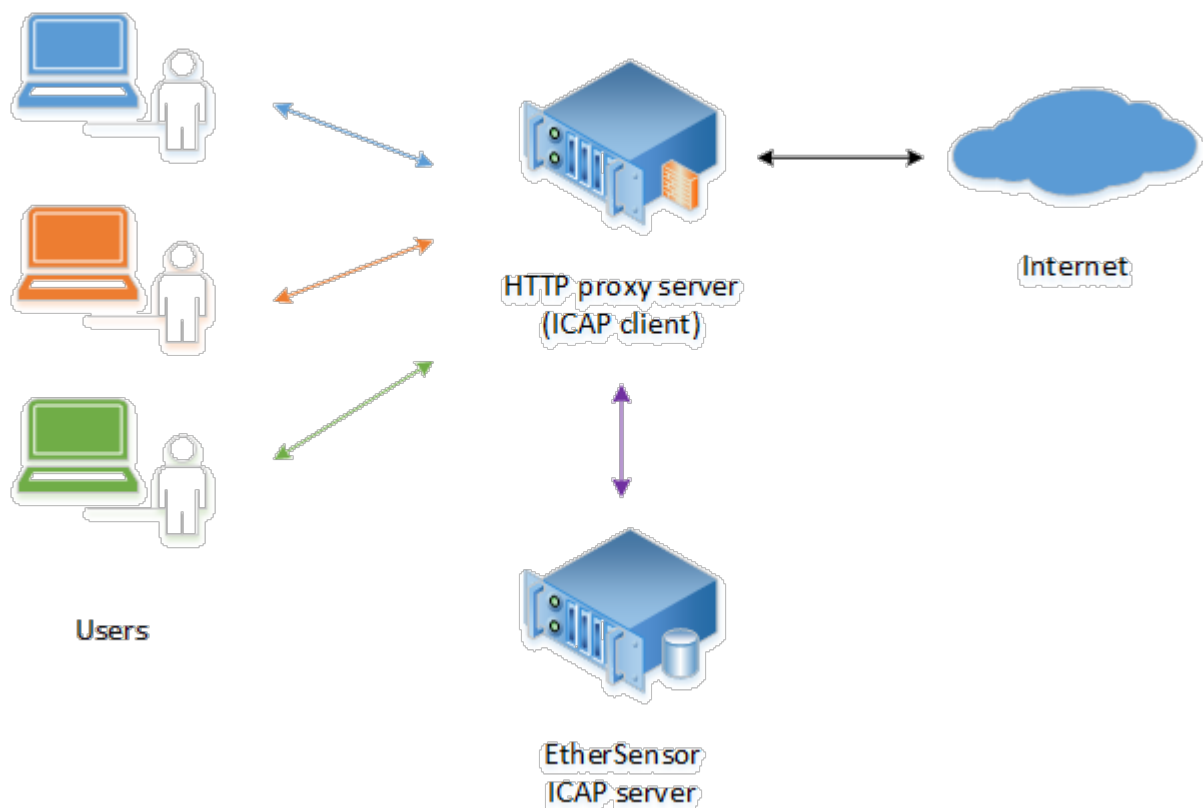


Fig. 16. EtherSensor ICAP service and ICAP client communication diagram.

Some ICAP clients use header extensions, which allows them to send information on users authorized on a proxy server to the ICAP server. This information is taken into account in further message processing in **DeviceLock EtherSensor**.

Command Line Parameters

The Windows **EtherSensor ICAP** service is set up to start automatically during **DeviceLock EtherSensor** installation. However, you can start the **ethersensor_icap.exe** process as a Windows application using the following command line parameters:

/process

Starts the **ethersensor_icap.exe** process as a regular Win32 process (may be helpful for debugging)

/service

Starts as a Windows service

/config

Saves the service default configuration

3.1.2.1. Setting up the Configurator

The **EtherSensor ICAP** service is an ICAP server to work with ICAP clients. Its settings are typical for this server type:

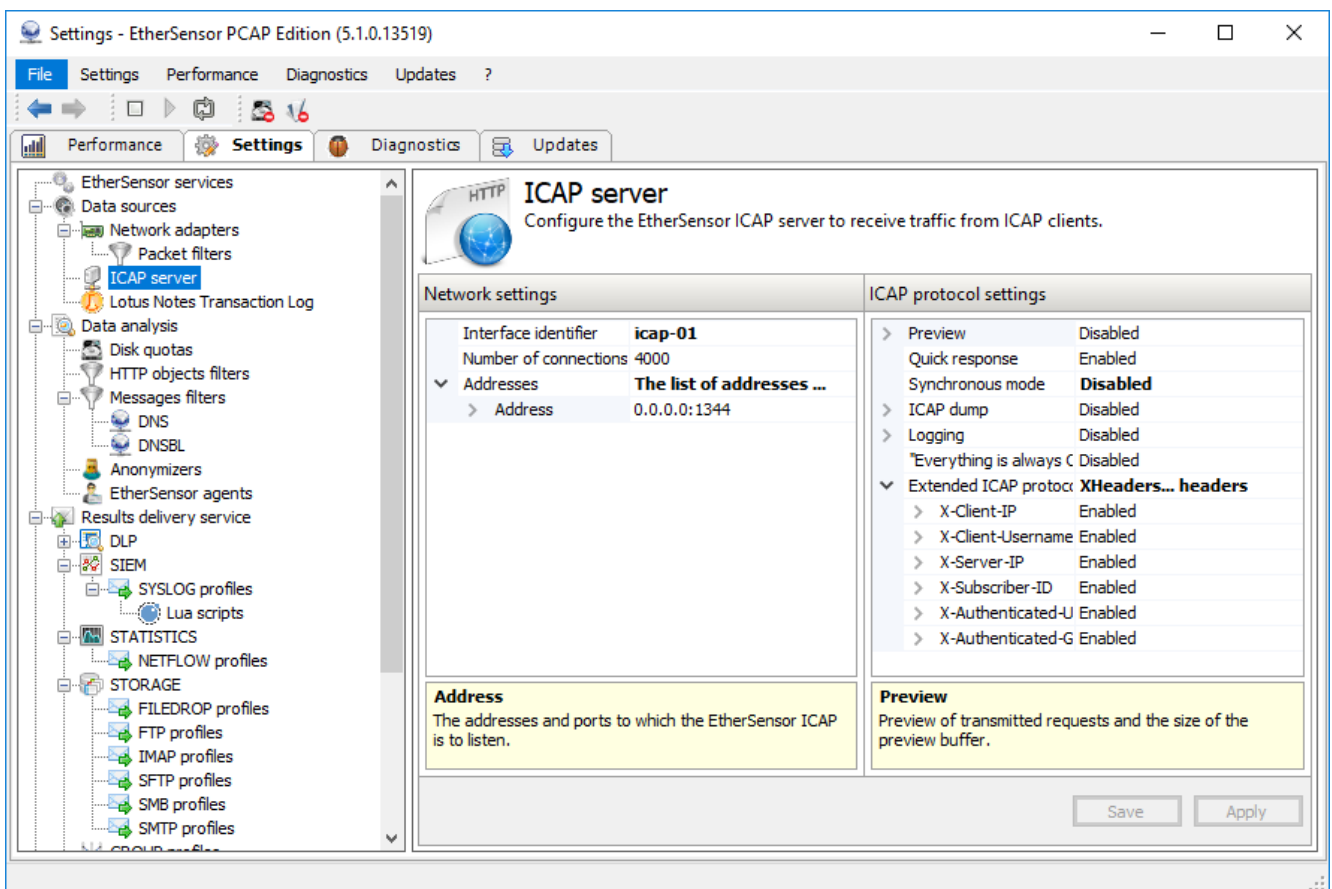


Fig. 17. EtherSensor ICAP service settings.

Number of connections:

The maximum number of ICAP server connections to clients. The amount of RAM required for one connection is approximately 8 KB.

Addresses:

IP addresses and ports listened by the ICAP server.

Preview:

Enables/disables the preview and buffer size.

Quick response:

Enables/disables the fast short response: **Allow 204 (No modifications)**.

Synchronous mode:

Enables/disables the synchronous operating mode for the ICAP server. In synchronous mode, the ICAP server first receives the entire request, and only then sends it back to the ICAP client, even if the request is very large, such as an ISO image of a disk, etc. Some ICAP client settings (e.g., Blue Coat, Ironport, etc.) may require the synchronous mode.

ICAP dump:

The dump of all data exchanged between the ICAP client and server. It is used only to debug communication with a third-party software.

Logging:

Logging HTTP requests by the ICAP server for the **EtherSensor Watcher** service.

"Everything is always OK" mode:

In this mode, even if the ICAP server detects errors from the client in ICAP, it does not send an error code in response, but responds with the **Allow 204 (No modifications)** code.

Extended ICAP protocol headers.:

Enables notification of ICAP clients the ICAP server support for specified headers.

Supported ICAP header names are:

X-Client-IP

X-Server-IP

X-Client-Username

X-Subscriber-ID

X-Authenticated-User

X-Authenticated-Groups

If a message signature is detected when processing ICAP traffic, these headers are translated into the following when sending an email with the recognized data:

X-Sensor-Icap-Client-Username

X-Sensor-Icap-Subscriber-ID

X-Sensor-Icap-Authenticated-User

X-Sensor-Icap-Authenticated-Groups

The values of **X-Client-IP** and **X-Server-IP** headers are saved in the **X-Sensor-Src-Address** and **X-Sensor-Dst-Address** headers.

For more information on the **EtherSensor ICAP** service settings, refer to the "ICAP Manual Setup (Config File)" section.

3.1.2.2. Manual Setup (Config File)

EtherSensor ICAP service configuration is stored in the **icap.xml**, file located in the **DeviceLock EtherSensor [INSTALLDIR]\config** common configuration directory.

A sample **icap.xml** configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<IcapConfig version="3.0">
  <SensorId>icap-01</SensorId>
  <Network max_connections="4000">
    <ListenAddress
      address="0.0.0.0:1344"
      ssl="false" />
    <ListenAddress
      address="0.0.0.0:1345"
      ssl="true" />
  </Network>
  <Icap>
    <XHeader
      enabled="true"
      name="X-Client-IP" />
    <XHeader
      enabled="true"
      name="X-Server-IP" />
    <XHeader
      enabled="true"
      name="X-Client-Username" />
    <XHeader
      enabled="true"
      name="X-Subscriber-ID" />
    <XHeader
      enabled="true"
      name="X-Authenticated-User" />
    <XHeader
      enabled="true"
      name="X-Authenticated-Groups" />
    <AlwaysOk
      enabled="true" />
    <Preview
      enabled="false"
      size="4096" />
    <Allow204
      enabled="true" />
    <SyncMode
      enabled="false" />
    <RawLog
      enabled="false"
      path=".\\raw-log" />
    <RequestLog
      enabled="false"
      http_enabled="true"
      channel="ICAP-REQUEST" />
  </Icap>
</IcapConfig>
```

IcapConfig tag

This is the root tag of the service configuration. The **version** attribute specifies the configuration version.

SensorId tag

Defines the interface ID. It is used in traffic analysis to determine the host from which messages have been received. Although the **DeviceLock EtherSensor** ICAP server's performance is very high, it makes sense to distribute the processing among several **DeviceLock EtherSensor** servers if there are too many connections.

Network tag

Defines the TCP/IP network settings for the service's server. The **max_connections** attribute specifies the maximum number of simultaneous connections from an ICAP client that the server can process. The default value of the **max_connections** attribute is 4000. For a server with a 2xDualCore Intel Xeon 2,3GHz CPU and 8GB RAM, the ICAP server can process up to 10,000 simultaneous connections.

For improved performance, use more CPU cores, a faster disk system or more RAM.

ListenAddress tag

The **ListenAddress** tag is nested within the **Network** tag. It specifies the network address (the IP address and port) that accepts new connections from ICAP clients. You can have between 1 and 60 **ListenAddress** tags with distinct network addresses where the server should be running. If the server should run on all addresses set up on a machine, specify only one tag with just one IP address: 0.0.0.0. The **ssl** attribute defines whether SSL will be used to protect data over this connection.

For example, the following configuration

```
<Network max_connections="4000">
  <ListenAddress
    address="0.0.0.0:1344"
    ssl="false" />
</Network>
```

instructs the system to run the service on all server IP addresses set up on port 1344.

```
<Network max_connections="4000">
  <ListenAddress
    address="1.2.3.4:1344"
    ssl="false" />
  <ListenAddress
    address="1.2.3.5:1345"
    ssl="true" />
</Network>
```

instructs the system to run the service on network interfaces with these IP addresses: 1.2.3.4:1344 and 1.2.3.5:1345.

The Icap Tag

Groups settings related to the ICAP protocol and its processing.

Preview tag

The **Preview** tag is nested within the **Icap** tag. It specifies the parameters for the **preview** mode in ICAP. The **preview** mode in ICAP allows you to send, initially, only a part of the data. Then, after this part is analyzed, it can either send the remaining data or stop processing.

It doesn't make sense to use the preview mode in **DeviceLock EtherSensor** because the service always receives all traffic for analysis. This mode is supported because it is required by certain ICAP clients. It also ensures compatibility with the ICAP specification as per **RFC 3507**.

The **enabled** attribute for the **Preview** tag determines whether the mode is active:

enabled="true" - the preview mode is on

enabled="false" - the preview mode is off

The **size** attribute of the **Preview** tag defines the size of the data sent for the preview, in bytes. We recommend selecting 4096 (4 Kbytes). 0 means that only HTTP request or response headers will be sent for preview.

We recommend turning this mode off whenever possible (**enabled="false"**), turning it on only when an ICAP client cannot work any other way. This will allow you to significantly reduce traffic between the ICAP client and the server.

Allow204 tag

The **Allow204** tag is nested within the **Icap** tag. It defines whether the subsystem ICAP server can or cannot use a short response with a **204** code. The short response means that if the ICAP server of the service receives any data from an ICAP client, it doesn't send the data back, but only sends a response with a **204** code. This allows you to reduce traffic sent from the ICAP server to ICAP clients and lower the workload to the service, improving performance.

The **enabled** attribute of the **Allow204** tag specifies whether the server is allowed to use a short response:

enabled="true" - 204 code is allowed

enabled="false" - 204 code is not allowed

We recommend turning this mode on whenever possible (**enabled="true"**), turning it off only when an ICAP client doesn't support processing responses with a 204 code.

RawLog tag

The **RawLog** tag is nested within the **Icap** tag. It sets parameters for use of the **RawLog** mode for ICAP requests/responses of the server. **RawLog** is a mode for logging the "raw" data exchanged between the ICAP client and ICAP server. We recommend using this mode only if there is no other way to determine why communication between the ICAP client and ICAP server has been interrupted.

The **enabled** attribute of the **RawLog** tag specifies if the **RawLog** mode is active or not:

enabled="false" - the raw data log mode is off

The **path** attribute of the **RawLog** tag specifies the directory path to which the raw data of the TCP sessions between the ICAP client and ICAP server is to be saved.

We recommend turning this mode off whenever possible (**enabled="false"**) because, with a heavy workload, service performance may decrease significantly, generating a heavier workload for the **DeviceLock EtherSensor** runtime environment disk system.

If you are unable to determine why an ICAP client cannot communicate with an ICAP server, please send the following to the **DeviceLock EtherSensor** manufacturer: **RawLog** data files, a detailed description of the ICAP client's and ICAP server's behavior and messages that the ICAP client saved to its standard logs.

RequestLog tag

The **RequestLog** tag is nested within the **Icap** tag. It establishes the parameters of the log mode for logging ICAP and HTTP errors, as well as HTTP requests processed by the ICAP server. This log is used to save errors that may appear in the communication between an ICAP client and a server and are related to incorrect formats of data sent by an ICAP client, incorrect usage of ICAP, etc.

The **enabled** attribute of the **RequestLog** tag determines whether if the log mode for ICAP and HTTP errors is on or off:

enabled="true" - the log mode is on

enabled="false" - the log mode is off

The **http_enabled** attribute of the **RequestLog** tag determines whether if the log mode for HTTP requests is on or off:

http_enabled="true" - the log mode for HTTP requests is on

http_enabled="false" - the log mode for HTTP requests is off

By default (if the attribute is missing), this mode is on.

The **channel** attribute of the **RequestLog** tag specifies the internal system name for the logging channel for HTTP requests. This parameter should always be set to **channel="ICAP-REQUEST"** and you should only change it if directly instructed to by the **DeviceLock EtherSensor** manufacturer.

AlwaysOk tag

The **AlwaysOk** tag is nested within the **Icap** tag. It determines whether if the "Everything is always OK" mode is on or off. In this mode, if the ICAP server detects errors from the client in ICAP, it doesn't send an error code in response, but rather responds with the **204 (No modifications)** code.

The **enabled** attribute of the **AlwaysOk** tag determines whether the "Everything is always OK" mode is on or off.

enabled="true" - the mode is on

enabled="false" - the mode is off

If the tag is missing, this mode is **off** by default.

If the mode is active, the **204** response code is sent in return to errors even if it is prohibited by the **Allow204** tag.

We recommend turning this mode on only when absolutely necessary because it sometimes may cause the ICAP client that sends the traffic to unexpectedly fail.

SyncMode tag

The **SyncMode** tag is nested within the **Icap** tag. It enables the "ICAP synchronous work" mode. In this mode, the ICAP server sends a response to an ICAP client only after the entire request has been received. This will happen even for large requests (download/upload ISO images or other large files). If the synchronous mode is disabled, the ICAP server works in streaming mode. This means that the server doesn't wait for the client to send the entire request, but starts responding immediately. The user doesn't see any lags in file transfers. This is especially important if the multimedia traffic is sent via ICAP (such as video streaming). The user doesn't have to wait for the entire video stream to be downloaded first to the ICAP proxy, then to the ICAP server, then returned by the ICAP server and only after that sent out from the ICAP proxy to the user.

The synchronous mode may, however, be required by certain ICAP clients largely for the purpose of using ICAP to work with antivirus software. Antivirus software usually needs to receive the entire object first (no matter what the object's size is) to decide what its response to the ICAP client will be.

The **enabled** attribute of the **SyncMode** tag determines whether the "ICAP synchronous work" mode is on or off.

enabled="true" - the mode is on

enabled="false" - the mode is off

If the tag is missing, this mode is **on** by default.

XHeader tag

The **XHeader** tag manages extended ICAP headers. The tag allows for notification of the ICAP client of whether the server supports the header or not, thus allowing or forbidding the ICAP client to send this header to the ICAP server.

The **enabled** attribute of the **XHeader** tag determines whether the header is supported:

enabled="true" - the header support is on

enabled="false" - the header support is off

If the tag is missing, the header is **supported** by default.

The **name** attribute of the **XHeader** tag specifies the name of the extended header.

Supported ICAP header names are:

X-Client-IP

X-Server-IP

X-Client-Username

X-Subscriber-ID

X-Authenticated-User

X-Authenticated-Groups

If these headers appear when processing the ICAP traffic, they are translated into the following headers:

X-Sensor-Icap-Client-Username

X-Sensor-Icap-Subscriber-ID

X-Sensor-Icap-Authenticated-User

X-Sensor-Icap-Authenticated-Groups

The values of **X-Client-IP** and **X-Server-IP** headers are saved in the **X-Sensor-Src-Address** and **X-Sensor-Dst-Address** headers.

By default (if **XHeader** tags are not specified in the configuration file), all headers are considered to be supported.

3.1.3. EtherSensor LotusTXN

The **EtherSensor LotusTXN** service extracts **Lotus Notes** messages from the transaction log (*Lotus Notes Transaction Log*).

This service extracts messages from the **Lotus Notes Transaction Log** and passes them on to the **EtherSensor Analyser** service for further processing.

In the current **DeviceLock EtherSensor** version (5.1.0.13519), the service can monitor multiple **Lotus Notes Transaction Log** directories simultaneously. Thus, the administrator can use a single **DeviceLock EtherSensor** installation to monitor several **Lotus Notes** systems. The **Lotus Notes Transaction Log** directories may be local or remote.

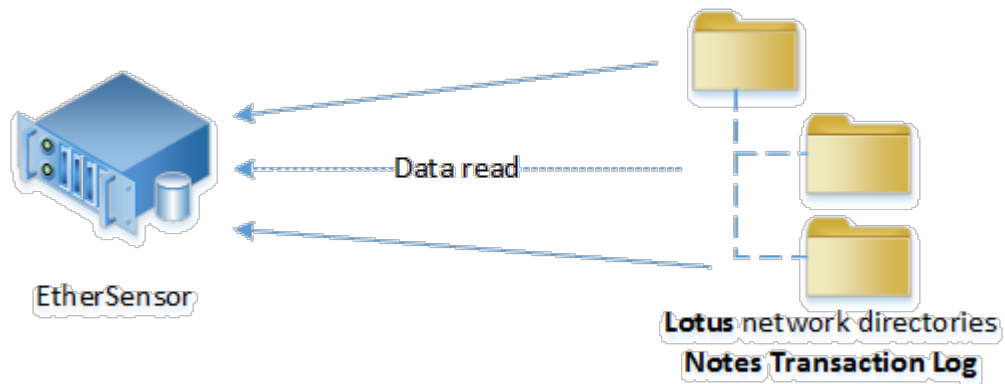


Fig. 18. EtherSensor LotusTXN service operation diagram.

Command Line Parameters

The Windows **EtherSensor LotusTXN** service is set up to start automatically during **DeviceLock EtherSensor** installation. However, you can start the **ethersensor_lotustxn.exe** process as a Windows application using the following command line parameters:

/process

Starts the **ethersensor_lotustxn.exe** process as a regular Win32 process (may be helpful for debugging).

/service

Starts as a Windows service.

/config

Saves the service default configuration.

3.1.3.1. Setting up the Configurator

The **EtherSensor LotusTXN** service allows for monitoring and reconstruction of **Lotus Notes** messages by extracting them from the **Lotus Notes Transaction Log**.

Please note:

If **Lotus Notes** doesn't use any encryption, the **EtherSensor EtherCAP** service is used to extract **Lotus Notes** messages from the traffic, as well as from SMTP/POP3/IMAP4.

To set up the **EtherSensor LotusTXN** service to monitor messages, specify the **Lotus Notes** transaction log directories to monitor.

Information on **Lotus Notes Transaction Log** directory settings is stored in the **lotustxn.xml** file located in the **[INSTALLDIR]\config** directory. You can edit it directly in the configuration file using any text editor.

In the configurator (**ethersensor_console.exe**), set up the **EtherSensor LotusTXN** service as follows:

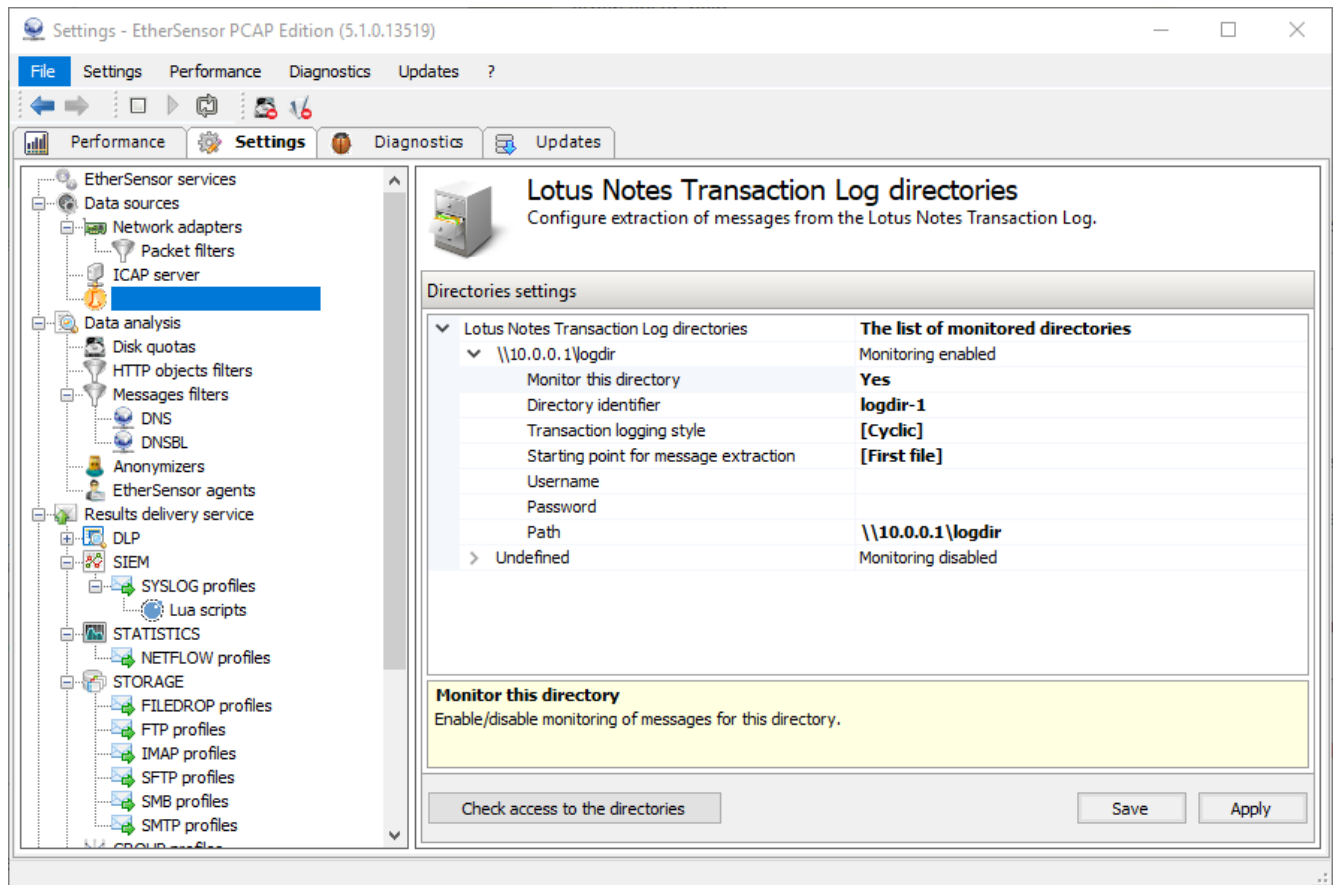


Fig. 19. EtherSensor LotusTXN service settings.

Directory identifier:

A **Lotus Notes Transaction Log** directory ID used to identify the data source in **DeviceLock EtherSensor**.

Transaction logging style:

Specifies the transaction logging style to be used by **Lotus Notes** for this directory.

Starting point for message extraction:

The message extraction starting point is used to point **DeviceLock EtherSensor** to the file in the **Lotus Notes Transaction Log** directory from which extraction should begin.

Username:

Defines the Windows username to access the transaction directory. The value should be in UPN (user principal name) format. Example: **administrator@example.com**, where **administrator** is

the username and **example.com** is the user's domain. If this parameter is not specified, the account from which **EtherSensor LotusTXN** service is started is used to access the transaction log files. Note that, when **DeviceLock EtherSensor** is installed, all services are started with the SYSTEM user privileges.

Password:

Defines the user password for access to the **Lotus Notes** transaction directory.

Path:

The full path to the directory where the **Lotus Notes** transaction logs are stored.

3.1.3.2. Manual Setup (Config File)

The **EtherSensor LotusTXN** service configuration is defined in the **lotustxn.xml** file, located in the common configuration directory, **[INSTALLDIR]\config**.

A sample **lotustxn.xml** configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<LotusTXNConfig version="1.0">
  <Capture>

    <Directory
      enabled="true"
      logstyle="linear"
      start="currentfile"
      id="logdir-1"
      path="\\10.31.5.16\logdir">
      <UserName>XFW50UpKPj4=</UserName>
      <UserPassword>OTkLCzg4DAw=</UserPassword>
    </Directory>

    <Directory
      enabled="true"
      logstyle="circular"
      start="firstfile"
      id="logdir-2"
      path="D:\lotus\data\logdir">
      <UserName>Cg46PgIGPjoHAzs/fno4PHx4SEwOCjQwchQ=</UserName>
      <UserPassword>VlcxMVZWMzNGRiAg</UserPassword>
    </Directory>

  </Capture>
</LotusTXNConfig>
```

LotusTXNConfig tag

This is the root tag of the service configuration. The **version** attribute specifies the configuration version.

Capture tag

Defines a block indicating the directories being monitored.

Directory tag

This tag is nested within the **Capture** tag. It specifies the settings of the directory being monitored. The **enabled** attribute specifies the directory activity status. If it is set to **false**, then the directory is not used in data processing.

The **logstyle** attribute specifies the style for **Lotus Notes** transaction logging. Possible values of this attribute are:

- **linear** - linear logging;
- **circular** - circular logging.

For more information, please refer to the documentation on **Lotus Notes**.

The **start** attribute specifies the starting point for message capture. If it is set to **firstfile**, then the messages will be extracted from the first **Lotus Notes Transaction Log** file. If it is set to **currentfile**, then the messages will be extracted from the current **Lotus Notes Transaction Log** file.

The **id** attribute defines the ID of the directory being monitored. It is used in message analysis to determine the directory from which messages have been received.

The **path** attribute defines the full path to the directory where **Lotus Notes Transaction Log** files are stored.

UserName tag

This tag is nested within the **Directory** tag. It defines the Windows username for access to the transaction directory. The value should be in UPN (user principal name) format. Example: **administrator@example.com**, where **administrator** is the username and **example.com** is the user's domain. If this parameter is not specified, the account from which **EtherSensor LotusTXN** service is started is used to access the transaction log files. Note that when **DeviceLock EtherSensor** is installed, all services are started with the SYSTEM user privileges.

UserPassword tag

This tag is nested within the **Directory** tag. It defines the user password for access to the transaction directory.

3.2. Capture Results Delivery

The **EtherSensor Transfer** service is responsible for the delivery of the **EtherSensor Analyser** work results to external consumer systems.

The delivery of the results of analysis of objects extracted from traffic is performed according to a delivery profile, of which there are the following types:

Universal

Deliver the content of application-level messages to external consumer systems: DLP systems, eDiscovery, Enterprise Archiving, Enterprise Search, etc. Include SMTP/SMTPS, FTP/FTPS, SFTP, IMAP, FILEDROP (local file system), or SMB/CIFS (network directory).

Proprietary

Deliver the content of application-level messages to external consumer systems. The current version (5.1.0.13519) supports delivery profiles that work with the following proprietary protocols, e.g. DeviceLock Enterprise Server.

Group

Include the usual delivery profiles with preassigned weights, designed to balance the load between the external consumer systems.

SYSLOG profiles

Used to transport results to consumer systems that receive event data via a SYSLOG server (usually SIEM systems). A SYSLOG line may be formed as a result of filter work, as well as by processing an object extracted from traffic by a pre-prepared Lua script. This allows for real-time data preparation in an arbitrary format specific to the particular consumer system (without so-called "connectors").

The delivery profile is assigned to a message using the message filter while the message data are analyzed. If the message analysis discovers that no delivery profile was assigned to the message, then it is delivered using the default profile. After successful delivery, the message is deleted from the message cache, and all available information about the message is destroyed.

The same object or its processing results *using multiple methods* may be delivered to *multiple consumers using multiple delivery profiles*.

For example:

Email content is delivered to the corporate DLP system and to the eDiscovery system simultaneously, and at the same time the metadata in CEF format are delivered to a SIEM system that works in an external SOC on the MSSP side.

The main format used by **DeviceLock EtherSensor** to deliver the content of the reconstructed objects to the consumer systems is the EML envelope. The **EtherSensor Transfer** service can also deliver data in its internal XML and/or JSON formats if the EML envelope is not a required component of the delivery protocol (copying data to directories or the use of FTP).

The system architecture when insecure data delivery protocols are used:

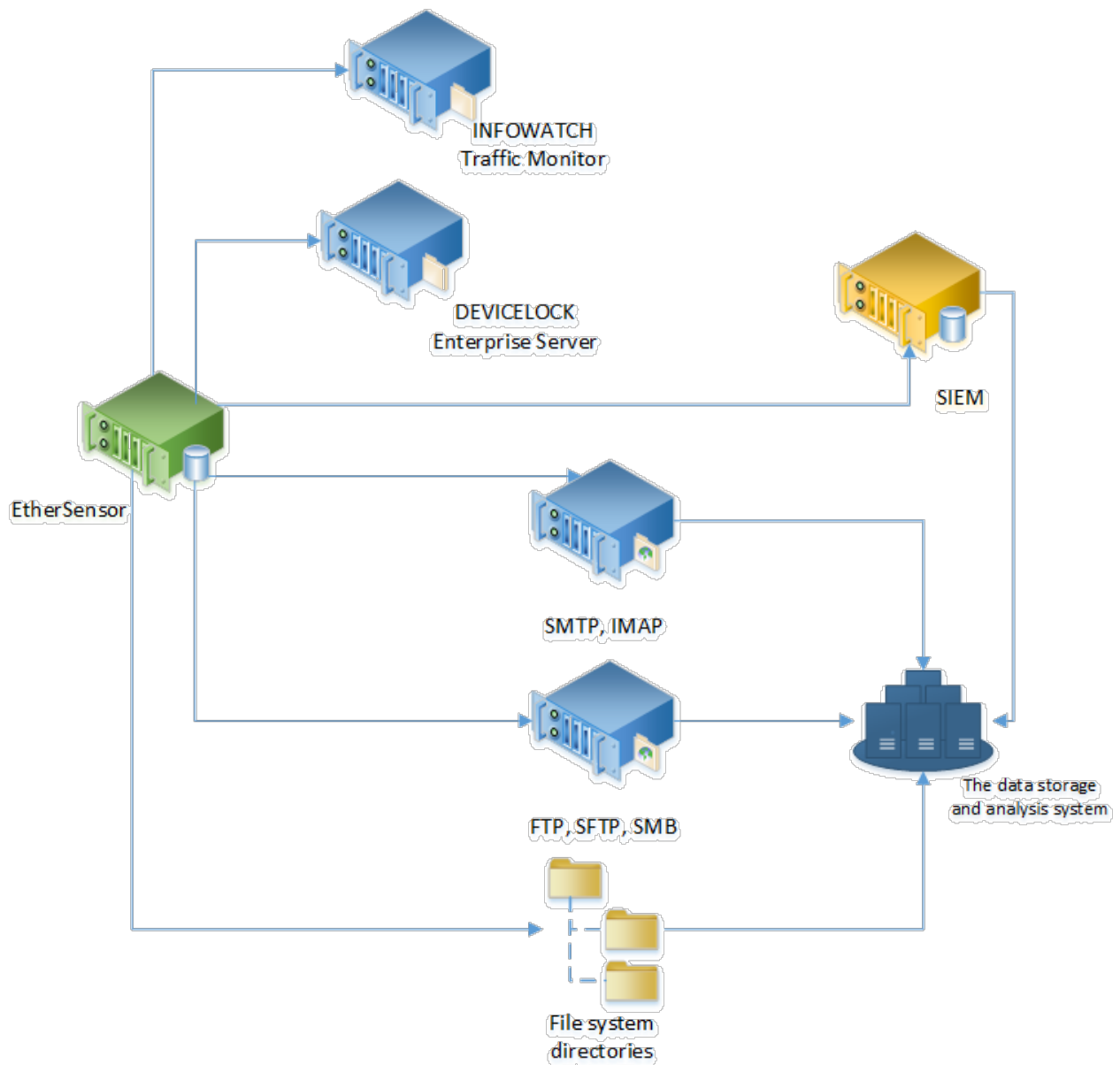


Fig. 20. EtherSensor Transfer service operation diagram.

The system architecture when secure data delivery protocols are used:

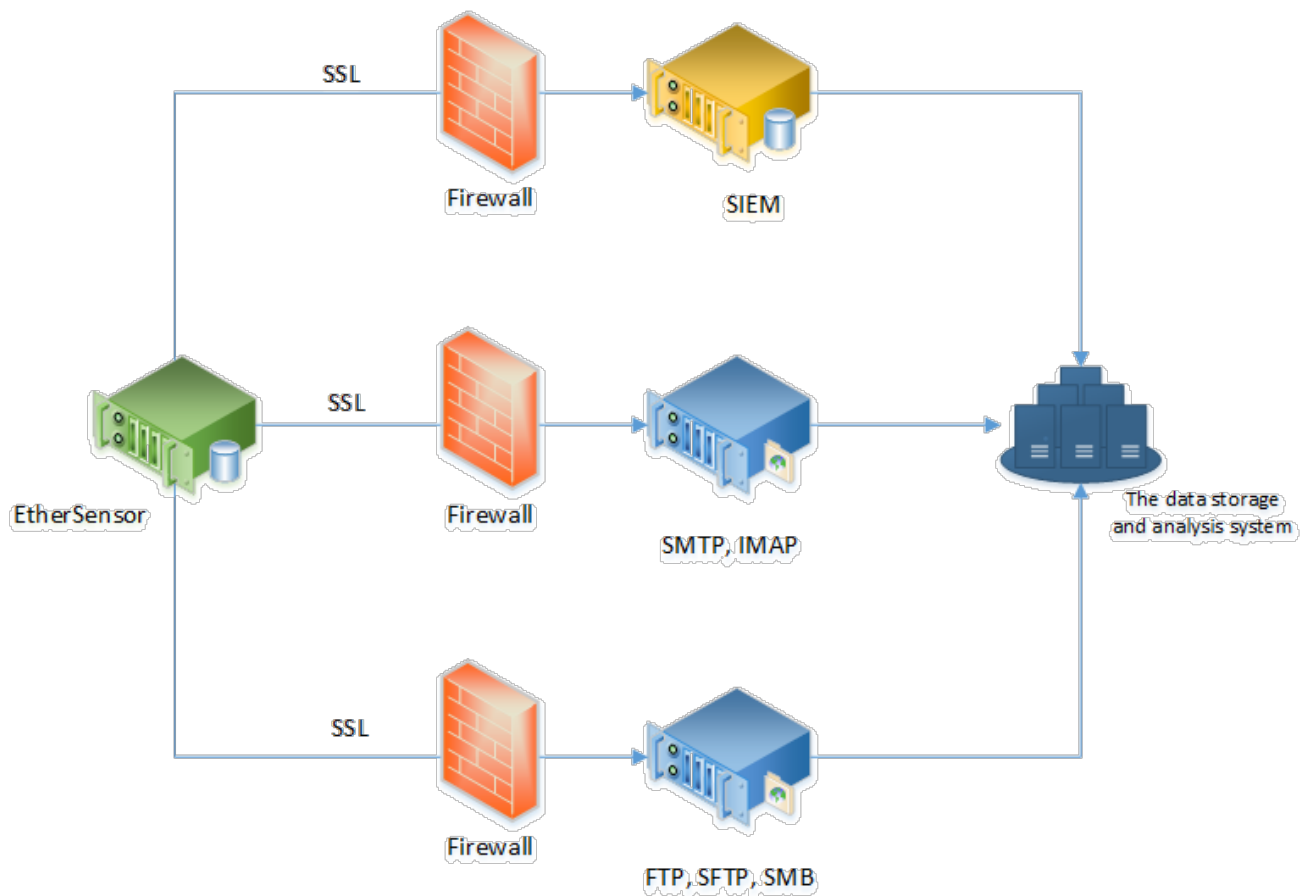


Fig. 21. How EtherSensor Transfer service works with secure protocols.

To deliver the capture data, the **EtherSensor Transfer** service monitors local directory spools and delivers the data as soon as they appear in a spool. To do this, the **EtherSensor Transfer** service uses delivery profiles that specify actions to be performed with the data being delivered. A delivery profile may be assigned by the **EtherSensor Analyser** service during data analysis. Then the delivery profile information is specified in the capture results metadata. If there is no profile specified in the capture results metadata, then the default profile is used:

The process of assigning a delivery profile and delivering the results.

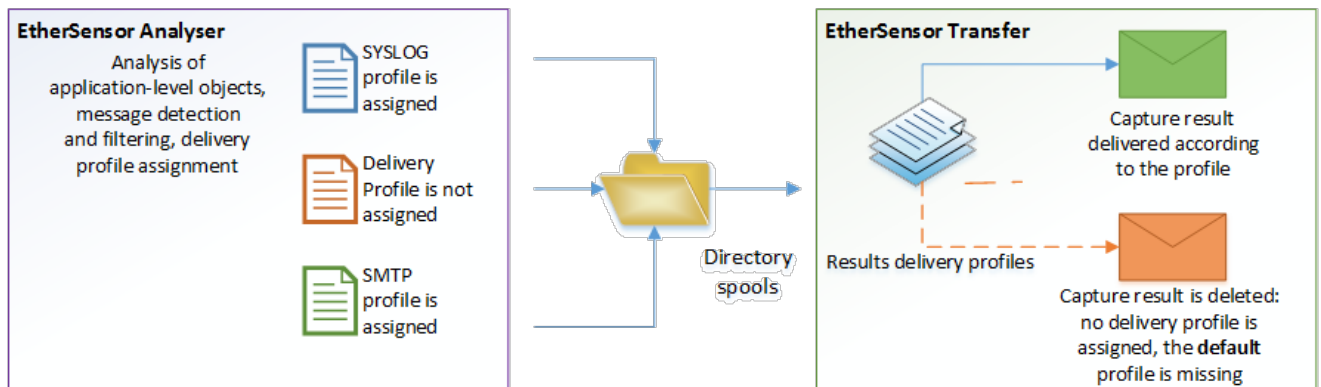


Fig. 22. The process of assigning a result delivery profile

Command Line Parameters

The **EtherSensor Transfer** service, during **DeviceLock EtherSensor** installation, is installed as a Windows service set to start automatically. However, you can also start it as **ethersensor_transfer.exe**, a Windows application with the following command line parameters:

/process

Starts the **ethersensor_transfer.exe** process as a regular Win32 process (may be helpful for debugging).

/service

Starts as a Windows service.

/config

Saves the service default configuration.

3.2.1. Setting up the Configurator

The main idea of the **EtherSensor Transfer** service is the concept of a predefined delivery profile.

If the captured messages consumer is a server, the profile contains data about this server and the authentication to access it. For profiles of FILEDROP, SMB, FTP or SFTP type, the profile contains the local directory path as well as the requirements for objects being saved.

If necessary, you can instantly replace a profile with another one in the filter rules of the **EtherSensor Analyser** service. Because profiles are created and carefully checked by admins, the likelihood of a mistake in these settings is extremely low.

3.2.1.1. DEVICELOCK Profiles

Setting up DEVICELOCK Profiles

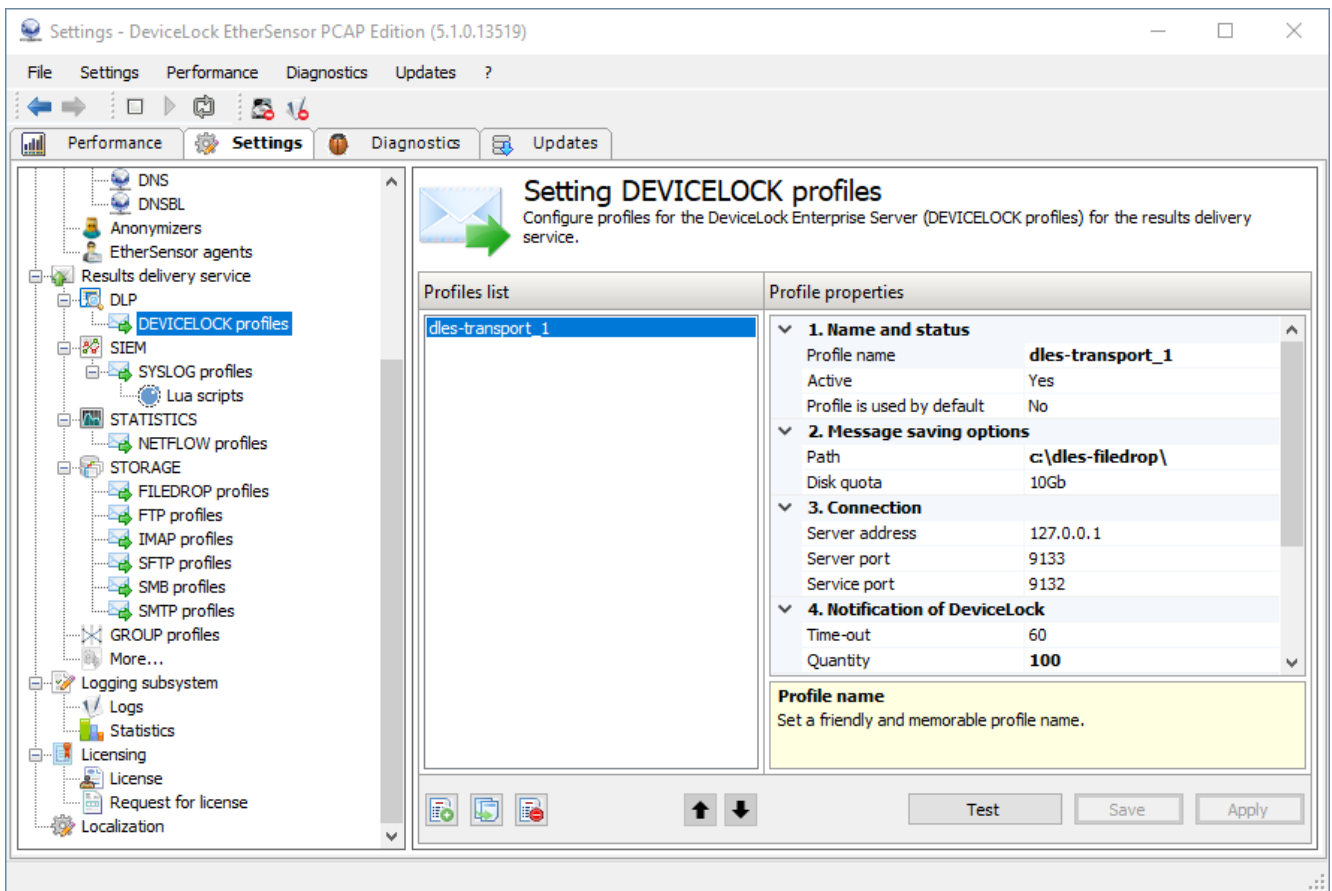


Fig. 23. DEVICELOCK profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. Message saving options

Path:

The path to the directory where captured messages will be saved.

This directory is used as an intermediate storage for the results being sent to the DEVICELOCK Enterprise Server.

Disk quota:

The size of the disk quota for storing messages. Example: 10Gb or 500Mb or 100Kb or 10,000. If the quota is exhausted, files will no longer be saved until the quota again allows this. To resume saving files, either free up space or increase the quota.

3. Connection

Server address:

IP-address or name of the DLES (DeviceLock Enterprise Server) server for results delivery.

Server port:

The port used to notify the DEVICELOCK Enterprise Server about messages (events).

Service port:

The port through which the DEVICELOCK Enterprise Server "grabs" the results (messages/events).

4. Notification of DeviceLock

Time-out:

Sets the timeout for DLES (DeviceLock Enterprise Server) notification in seconds. If there are unsent messages after the timeout has elapsed, the server will be notified about the availability of results.

Quantity:

Sets the number of accumulated messages upon which the notification of DLES will be performed.

5. Message format

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

6. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

7. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

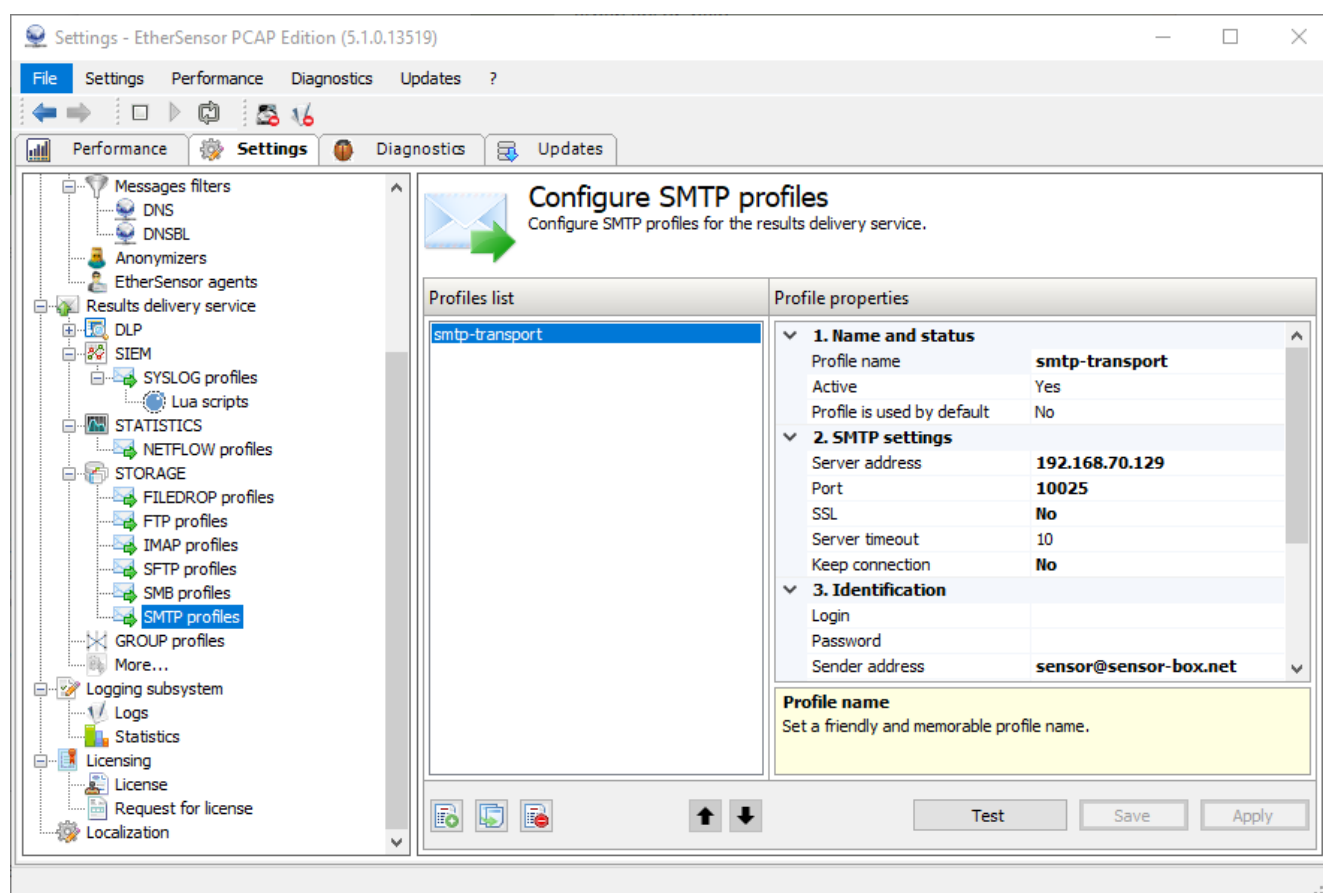
3.2.1.2. SMTP Profiles**Setting up SMTP Profiles**

Fig. 24. SMTP profile settings.

1. Name and status**Profile name:**

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. SMTP settings

Server address:

IP address or name of the SMTP server for results delivery.

Port:

SMTP server port for sending messages.

SSL:

Enables/disables the use of SSL encryption when sending messages.

Server timeout:

Timeout (response timeout) of the SMTP server in seconds.

Keep connection:

Send all messages in the same connection with the server. If this option is disabled, then each message will be sent in a separate TCP connection.

3. Identification

Login:

Login to access the SMTP server.

Password:

Password for access to the SMTP server.

Sender address:

The address of the sender of the message for the consumer system.

Admins can set this to any value; it is recommended, however, to account for possible address checks in the mail system.

Recipient address:

A working email address of the consumer system (for example, the message archiving system).

4. Message format

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

5. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

6. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.1.3. FTP Profiles

Setting up FTP Profiles

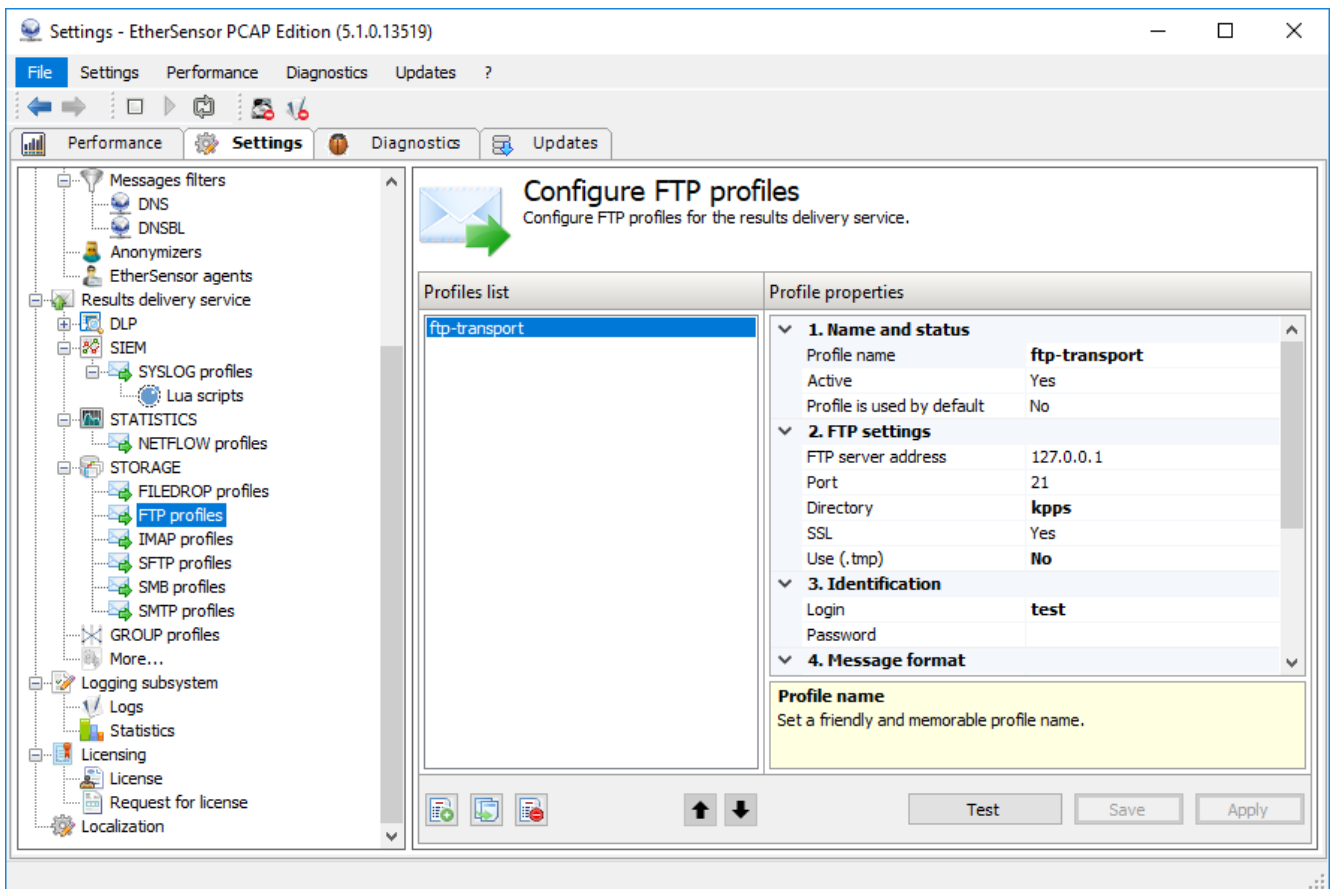


Fig. 25. FTP profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. FTP settings

FTP server address:

The IP address or name of the FTP server for sending messages.

Port:

FTP-server port to send messages.

Directory:

The directory for storing messages.

SSL:

Enables/disables the use of SSL encryption when sending messages.

Use (.tmp):

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

3. Identification

Login:

Login to access the FTP-server.

Password:

Password for access to the FTP server.

4. Message format

Message format:

Stored or sent messages format (EML, XML, JSON ...)

Save as ZIP:

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

ZIP file compression rate:

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

5. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

6. For GROUP profiles**Weight:**

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

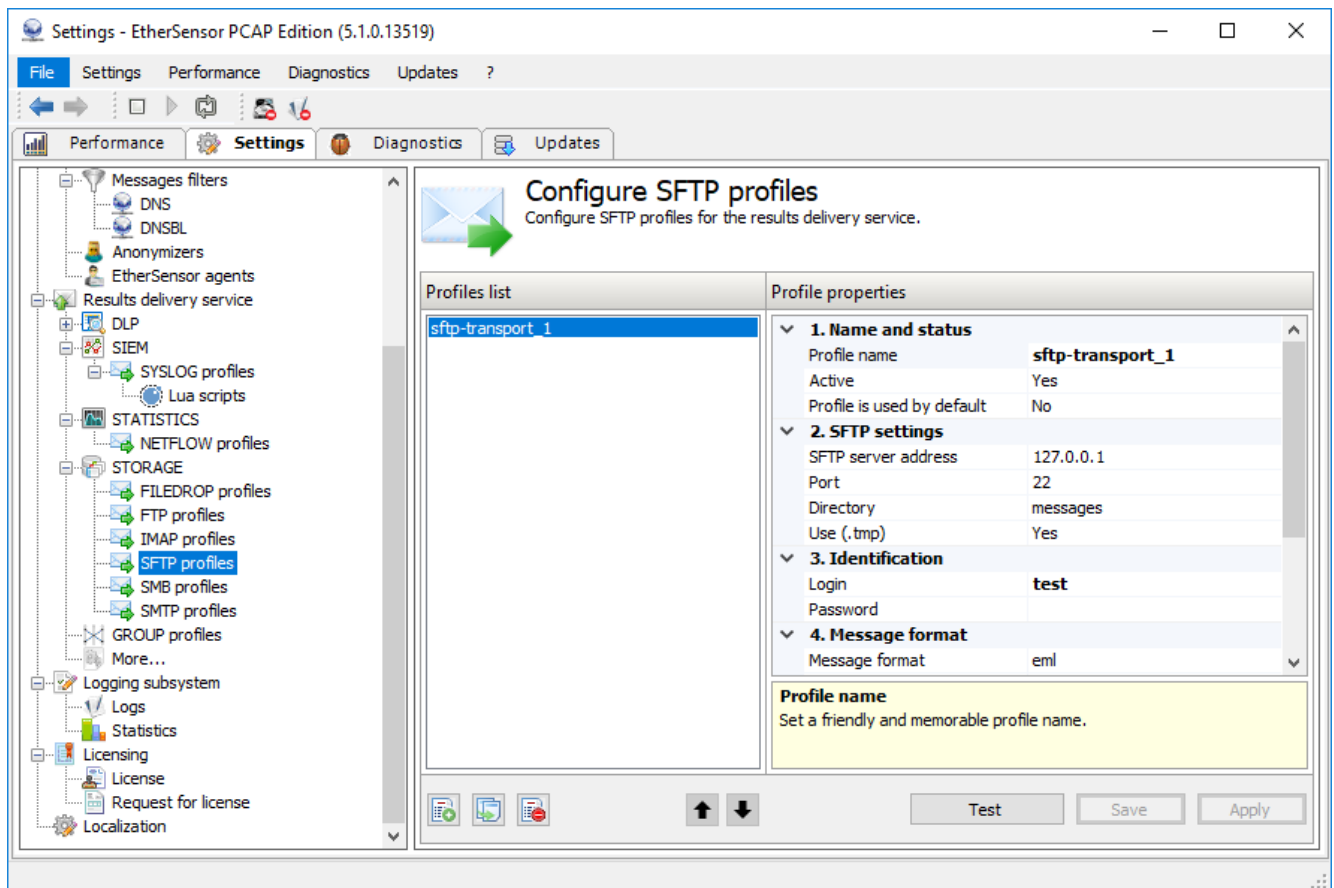
3.2.1.4. SFTP Profiles**Setting up SFTP Profiles**

Fig. 26. SFTP profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. SFTP settings

SFTP server address:

The IP address or name of the SFTP server for results delivery.

Port:

Port SFTP-server for sending messages.

Directory:

The directory for storing messages.

Use (.tmp):

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

3. Identification

Login:

Login for access to the SFTP server.

Password:

Password for access to the SFTP-server.

4. Message format

Message format:

Stored or sent messages format (EML, XML, JSON ...)

Save as ZIP:

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed

in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

ZIP file compression rate:

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

5. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

6. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.1.5. FILEDROP Profiles

Setting up FILEDROP Profiles

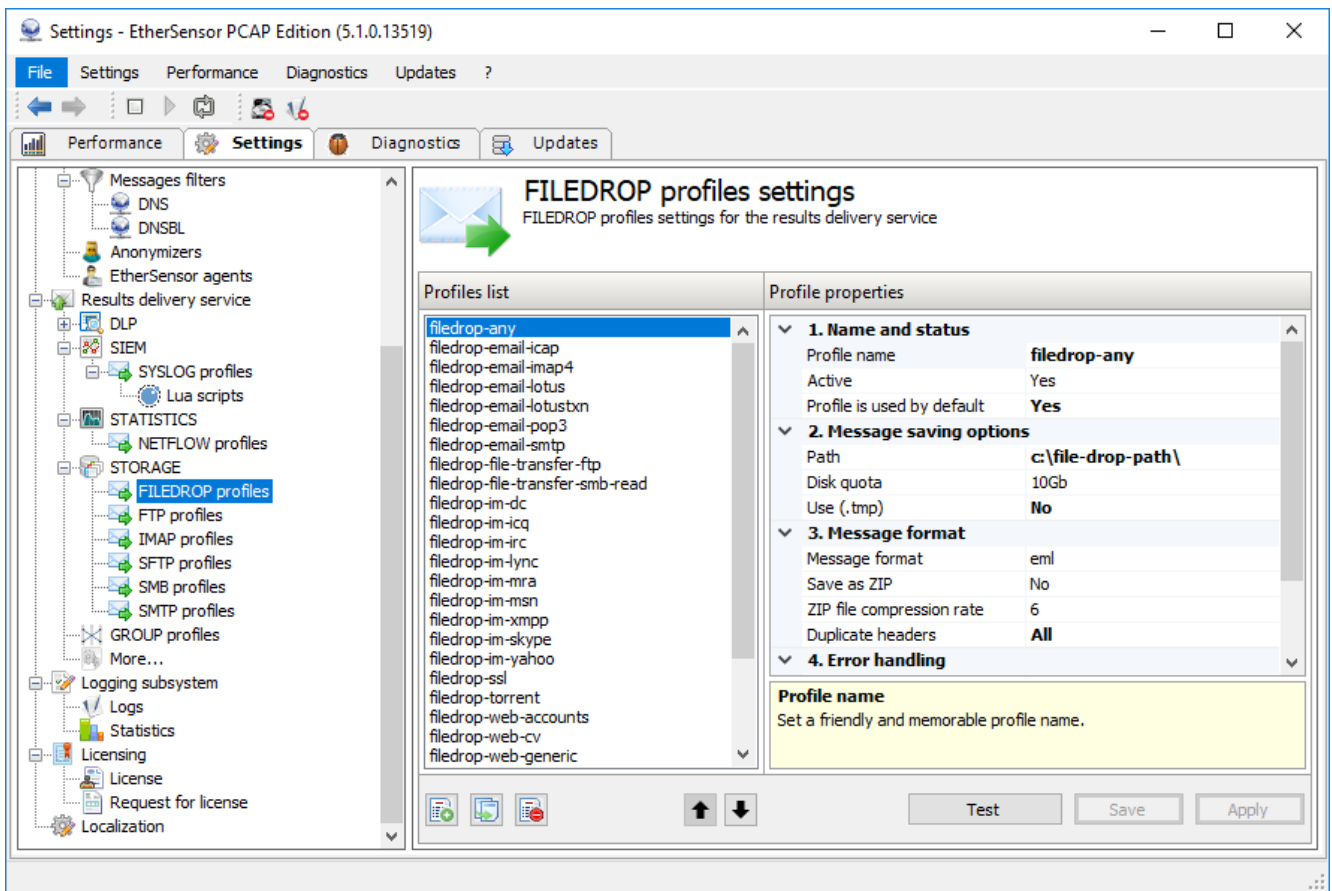


Fig. 27. FILEDROPPROFILE profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. Message saving options

Path:

The path to the directory where captured messages will be saved.

Disk quota:

The size of the disk quota for storing messages. Example: 10Gb or 500Mb or 100Kb or 10,000. If the quota is exhausted, files will no longer be saved until the quota again allows this. To resume saving files, either free up space or increase the quota.

Use (.tmp):

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

3. Message format

Message format:

Stored or sent messages format (EML, XML, JSON ...)

Save as ZIP:

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

ZIP file compression rate:

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

4. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

5. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.1.6. IMAP Profiles

Setting up IMAP Profiles

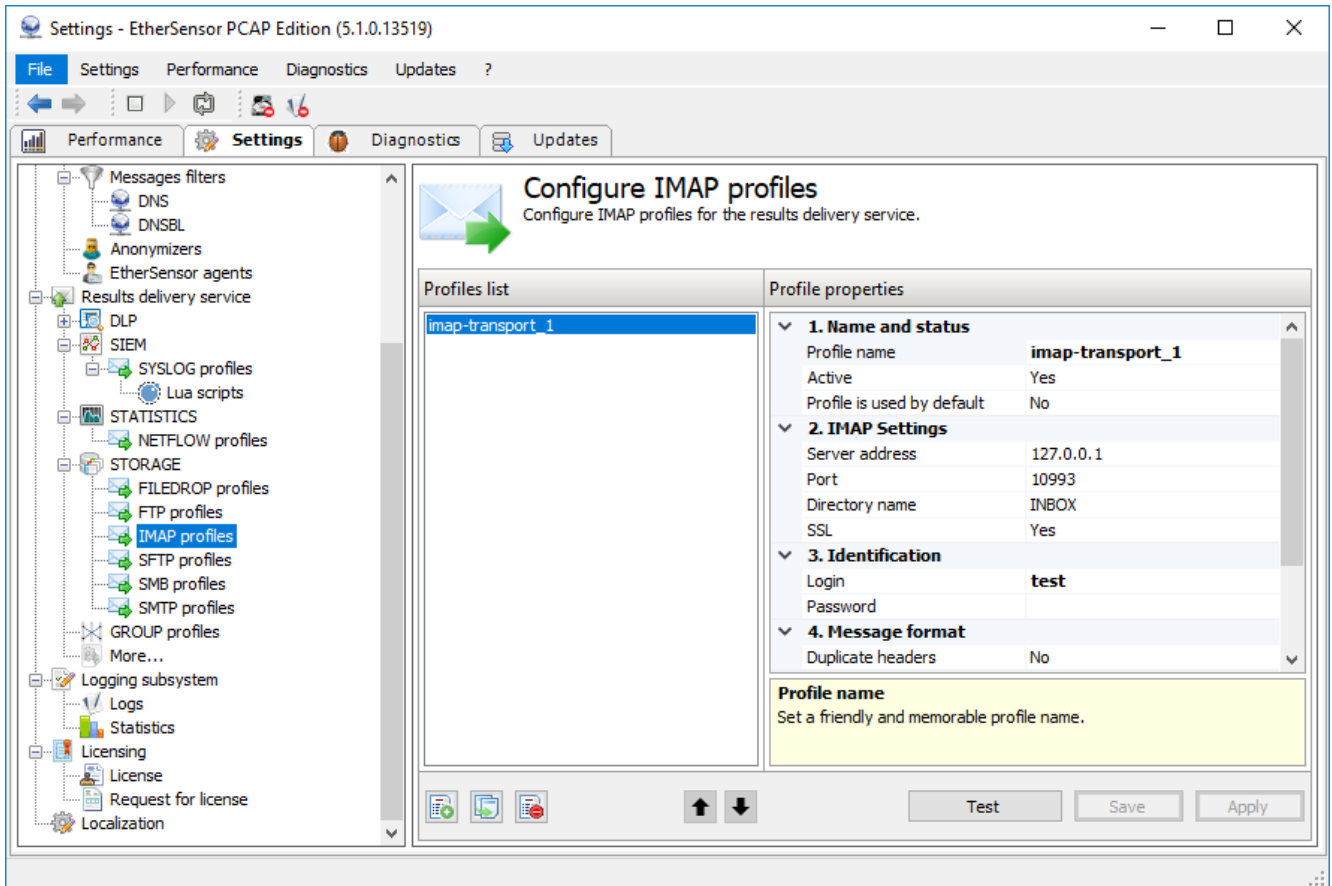


Fig. 28. IMAP profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. IMAP Settings

Server address:

The IP address or name of the IMAP server for results delivery.

Port:

IMAP server port for messages delivery.

Directory name:

The name of the IMAP directory to which the message will be delivered.

SSL:

Enables/disables the use of SSL encryption when sending messages.

3. Identification

Login:

Login to access the IMAP server.

Password:

Password to access the IMAP server.

4. Message format

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microoolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

5. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

6. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.1.7. SMB Profiles

Setting up SMB Profiles

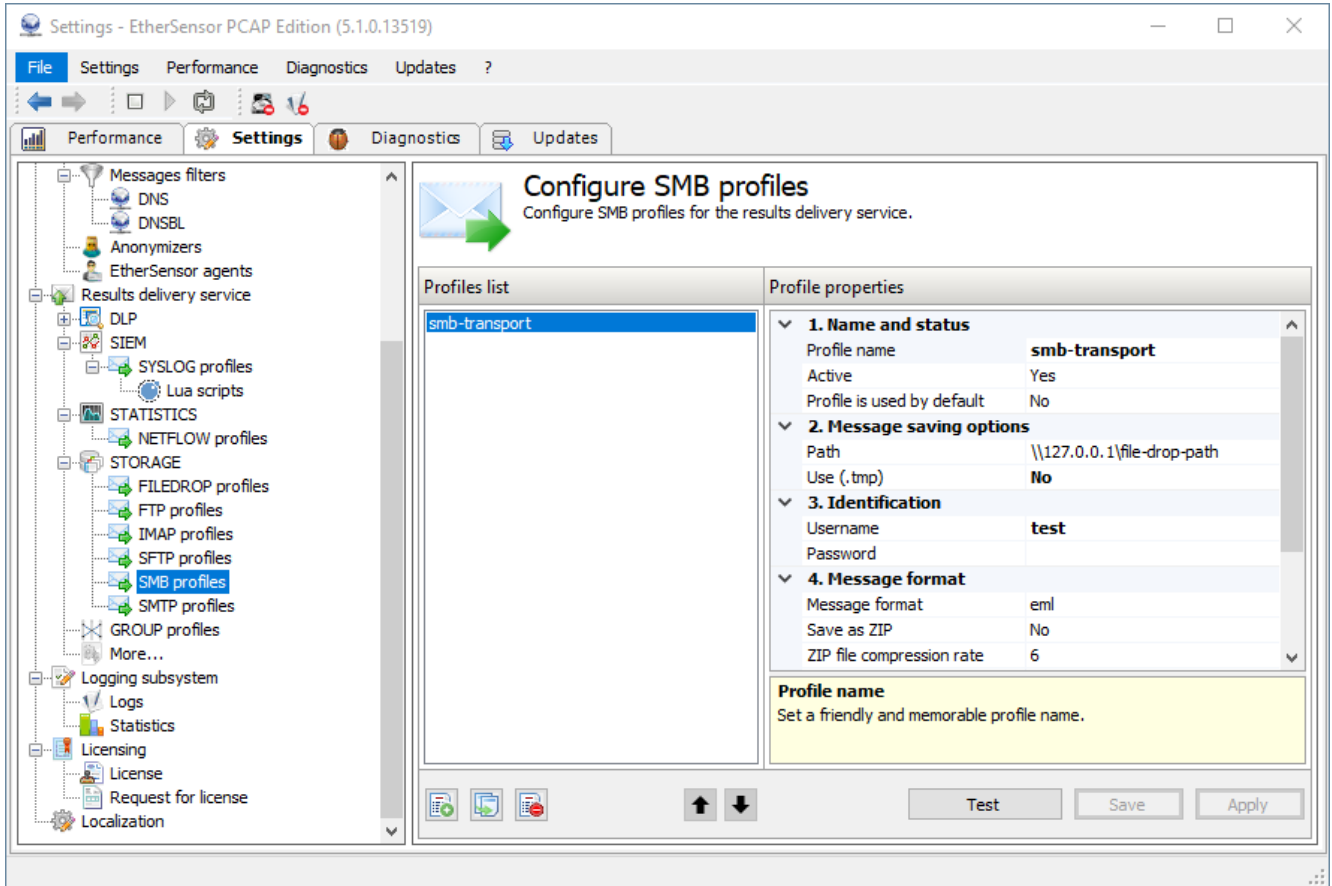


Fig. 29. SMB profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. Message saving options

Path:

Path to the SMB/CIFS directory to save messages. Example: \\ARCHSERVER1\Messages

Use (.tmp):

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

3. Identification

Username:

User name for accessing the SMB/CIFS directory to save messages.

Password:

Password for accessing the SMB/CIFS directory to save messages.

4. Message format

Message format:

Stored or sent messages format (EML, XML, JSON ...)

Save as ZIP:

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

ZIP file compression rate:

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

Duplicate headers:

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

5. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

6. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

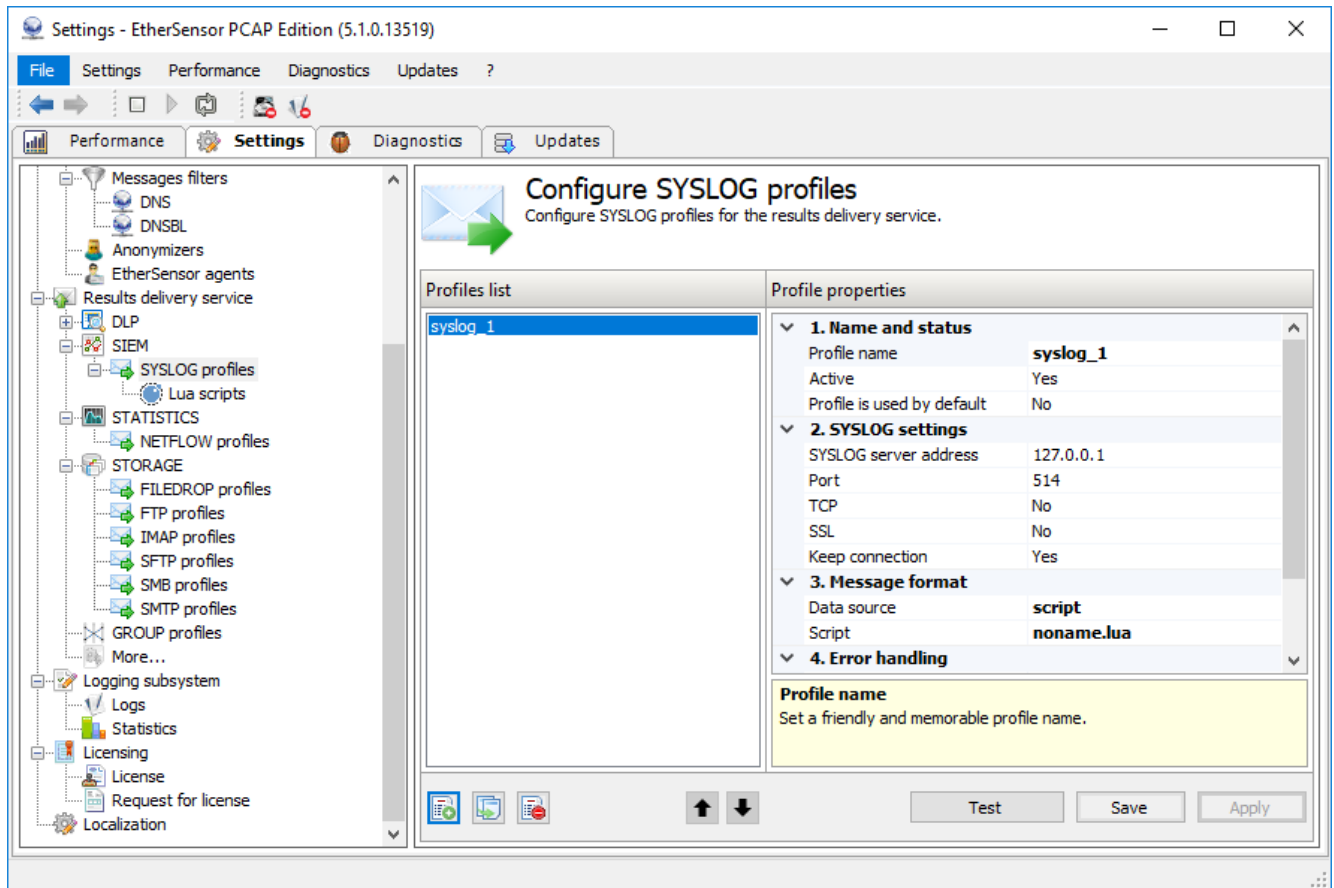
3.2.1.8. SYSLOG Profiles**Setting up SYSLOG Profiles**

Fig. 30. SYSLOG profile settings.

1. Name and status**Profile name:**

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. SYSLOG settings

SYSLOG server address:

The IP-address or name of the SYSLOG server for results delivery.

Port:

SYSLOG server port for sending messages.

TCP:

Allows use of the TCP protocol to send messages to the SYSLOG server. This is necessary if SSL encryption is used when sending messages to the consumer system.

Warning!

You can use SSL to send messages to the SYSLOG server only if TCP is enabled.

SSL:

Enables/disables the use of SSL encryption when sending messages.

Keep connection:

Send all messages in the same connection with the server. If this option is disabled, then each message will be sent in a separate TCP connection.

3. Message format

Script:

The name of the Lua script file that will be used to generate the message to be sent to the SYSLOG server. The script file must be in the \scripts subfolder.

4. Error handling

Timeout after failure:

Timeout in seconds to retry message delivery in the event of receiver rejection.

5. For GROUP profiles

Weight:

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

Reserve profile:

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.1.8.1. Lua Scripts

TBD

3.2.1.9. GROUP Profiles

Setting up GROUP Profiles

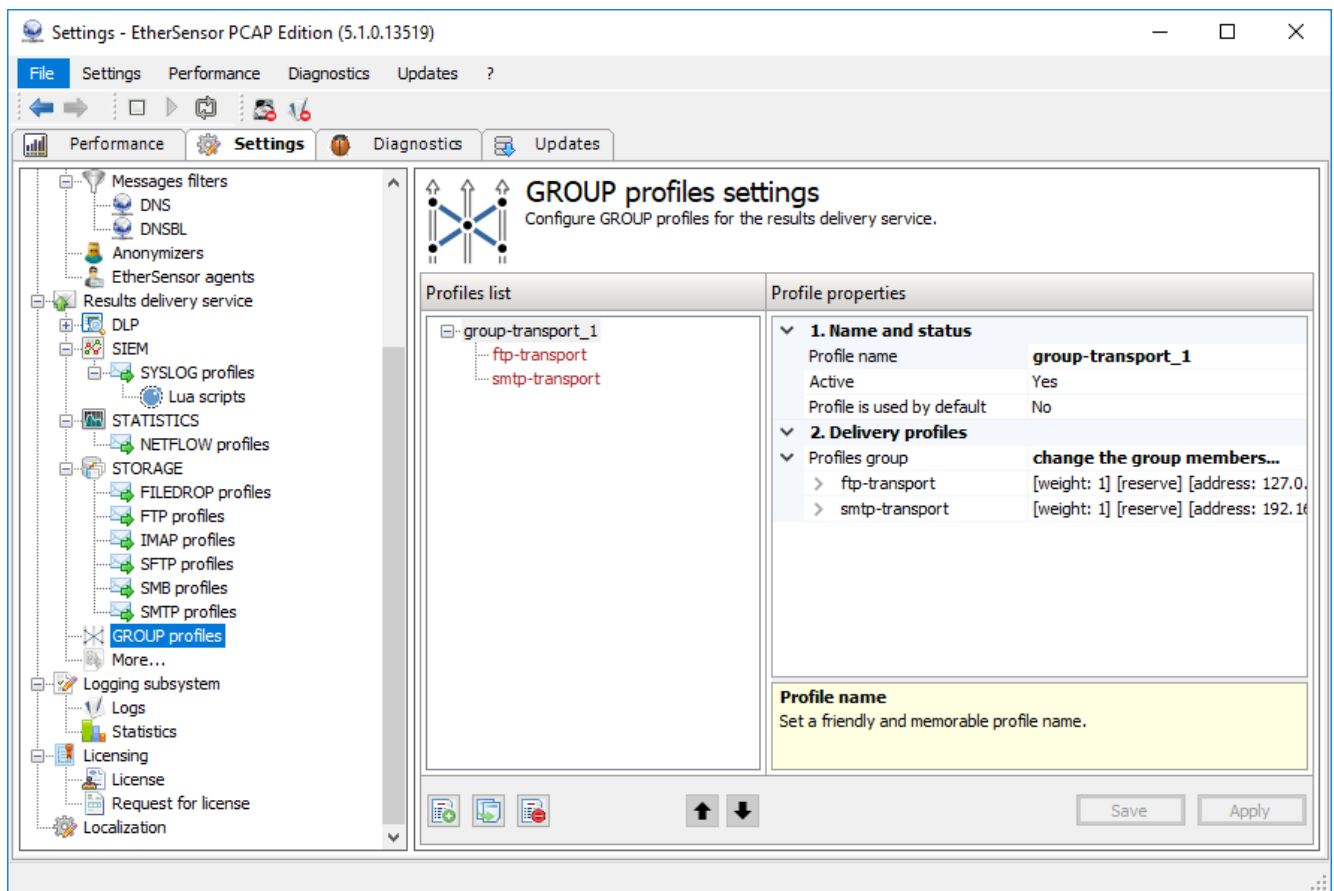


Fig. 31. GROUP profile settings.

1. Name and status

Profile name:

Administrators can select any profile name that is helpful, meaningful and easy to remember.

Profile is used by default:

"Yes" means that this delivery profile is used by default.

2. Delivery profiles

Profiles group:

A list of pre-existing delivery profiles used in the GROUP profile.

How Group Delivery Profiles Work

Basic Concepts

You can assign a group delivery profile as a default profile in the same manner in which any standard delivery profile is assigned. You can also assign it using the message filter rules.

While a standard delivery profile only contains settings to fine-tune the message delivery according to the message delivery method being used, a group profile contains only names of pre-created standard delivery profiles.

Standard profiles in a group profile may be divided into main and reserve profiles. Main profiles are used to deliver messages in the standard operation mode. Reserve profiles are used to deliver messages when no standard profile can deliver them.

Additionally, you can assign weights to standard profiles in the group profile. These weights will be used to distribute message delivery to the consumers among profiles. You can assign weights to both main and reserve profiles.

Message Delivery Using a Group Profile

Let's consider an example where messages are to be delivered using a group profile containing 3 main standard delivery profiles (SMTP 1, SMTP 2, SMTP 3) with equal weights and one reserve standard delivery profile (FILEDROP 1).

In this example, in standard operation mode, messages are equally distributed for delivery to the receivers.

If one of the message receivers described by a standard delivery profile cannot accept the messages, then the acceptance workload will move to the remaining delivery profiles.

If all main delivery profiles (SMTP 1, SMTP 2, SMTP 3) cannot accept messages, a reserve delivery profile (FILEDROP 1) will be used for message delivery until at least one of the main delivery profile is able to send messages.

3.2.2. Manual Setup (Config File)

The **EtherSensor Transfer** service configuration is stored in the **transfer.xml** file, located in the common configuration directory, **DeviceLock EtherSensor [INSTALLDIR]\config**.

A sample **transfer.xml** configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<TransferConfig version="4.1">
  <SensorID>0AFE95E7-DC45-4993-B9FC-5E002D9B1BCA</SensorID>
  <TransportThreads>4</TransportThreads>

  <Profile name="smtp-main" enabled="true" default="false">
    <Option name="protocol" value="smtp" />
    <Option name="smtp-server" value="smtp.domain.com" />
    <Option name="smtp-port" value="465" />
    <Option name="enable-ssl" value="true" />
    <Option name="mail-from" value="sensor@internal.net" />
    <Option name="mail-to" value="archive@internal.net" />
    <Option name="tcp-timeout" value="10" />
    <Option name="save-headers" value="all" />
    <Option name="keep-connection" value="true" />
    <Option name="profile-fail-timeout" value="10" />
    <Option name="go-profile-weight" value="1" />
    <Option name="go-profile-reserve" value="false" />
  </Profile>

  <Profile name="imap-transport" enabled="true" default="false">
    <Option name="protocol" value="imap" />
    <Option name="imap-server" value="imap.domain.com" />
    <Option name="imap-port" value="10993" />
    <Option name="file-drop-path" value="INBOX" />
    <Option name="enable-ssl" value="true" />
    <Option name="username" value="administrator@domain.com" />
    <Option name="password" value="test123" />
    <Option name="tcp-timeout" value="10" />
    <Option name="save-headers" value="none" />
  </Profile>

  <Profile name="local-dir" enabled="true" default="true">
    <Option name="protocol" value="filedrop" />
    <Option name="file-drop-path" value="C:\FileDropPath\" />
    <Option name="enable-tmp" value="false" />
    <Option name="save-passport" value="false" />
    <Option name="save-eml" value="true" />
    <Option name="save-zip" value="false" />
    <Option name="zip-level" value="0" />
    <Option name="quota" value="10Gb" />
    <Option name="save-headers" value="xsensor, xkpps" />
    <Option name="profile-fail-timeout" value="10" />
    <Option name="go-profile-weight" value="1" />
    <Option name="go-profile-reserve" value="false" />
  </Profile>

  <Profile name="largefile-ftp" enabled="true" default="false">
    <Option name="protocol" value="ftp" />
    <Option name="file-drop-path" value="ftp://127.0.0.1/" />
    <Option name="enable-ssl" value="true" />
    <Option name="user-name" value="test" />
    <Option name="password" value="" />
    <Option name="enable-tmp" value="false" />
    <Option name="save-passport" value="true" />
    <Option name="save-eml" value="true" />
    <Option name="save-zip" value="false" />
    <Option name="zip-level" value="6" />
    <Option name="save-headers" value="none" />
    <Option name="profile-fail-timeout" value="10" />
    <Option name="go-profile-weight" value="1" />
    <Option name="go-profile-reserve" value="false" />
  </Profile>
</TransferConfig>
```



```
<Profile name="net_share_1" enabled="true" default="false">
  <Option name="protocol" value="smb" />
  <Option name="file-drop-path" value="\\ARCHSERVER1\Messages" />
  <Option name="user-name" value="test" />
  <Option name="password" value="" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-passport" value="false" />
  <Option name="save-eml" value="true" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="6" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="true" />
</Profile>

<Profile name="sftp-transport_1" enabled="true" default="false">
  <Option name="protocol" value="sftp" />
  <Option name="sftp-server" value="10.101.100.125" />
  <Option name="sftp-port" value="22" />
  <Option name="file-drop-path" value="messages" />
  <Option name="user-name" value="test" />
  <Option name="password" value="" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-format" value="eml" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="6" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="true" />
</Profile>

<Profile name="iwthm-transport_1" enabled="true" default="false">
  <Option name="protocol" value="iwthrift" />
  <Option name="iwthrift-server" value="10.101.100.196" />
  <Option name="iwthrift-port" value="9101" />
  <Option name="iwthrift-company" value="microolap" />
  <Option name="iwthrift-token" value="" />
  <Option name="capture-server-ip" value="10.100.101.68" />
  <Option name="capture-server-host" value="sensor.chg" />
  <Option name="keep-connection" value="true" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="true" />
</Profile>

<Profile name="group-transport" enabled="true" default="false">
  <Option name="protocol" value="gprofile" />
  <Option name="profilename" value="smtp-main" />
  <Option name="profilename" value="net_share_1" />
</Profile>
</TransferConfig>
```

TransferConfig tag

This is the root tag of the service configuration. The **version** attribute specifies the configuration version.

SensorID tag

Defines the sensor ID. It is in the traffic analysis to determine the physical source from which data have been received.

TransportThreads tag

Defines the number of threads sending messages simultaneously. The maximum number of sending threads cannot be greater than the current number of logical **CPU cores*2** and cannot be less than **1**.

Profile tag

The **Profile** tag provides the description for a data delivery profile. The **name** attribute specifies the name of the profile. The **enabled** attribute specifies the profile activity status. If it is set to **false**, then the profile is not used in data delivery. The **default** attribute is used to specify the default profile status. This attribute may be **true** only for one profile of the entire profile set.

Option tag

The **Option** tag is nested within the **Profile** tag. It specifies the description of a particular option of the data delivery profile. The **name** attribute specifies the name of the option. The **value** attribute defines the option value.

3.2.2.1. DEVICELOCK Profiles

A sample setup of a delivery profile using DEVICELOCK protocol is provided below:

```
<Profile name="dles-transport_1" enabled="true" default="false">
  <Option name="protocol" value="dlesrpc" />
  <Option name="dlesrpc-server-addr" value="127.0.0.1" />
  <Option name="dlesrpc-server-port" value="9133" />
  <Option name="dlesrpc-client-port" value="9132" />
  <Option name="file-drop-path" value="c:\dles-filedrop\" />
  <Option name="quota" value="10Gb" />
  <Option name="alert-timeout" value="60" />
  <Option name="alert-rescount" value="100" />
  <Option name="save-headers" value="none" />
  <Option name="go-profile-weight" value="1" />
</Profile>
```

Description of DEVICELOCK profile options:

protocol

Defines the data transfer protocol. In this case, it is **dlesrpc**

dlesrpc-server-addr

IP-address or name of the DLES (DeviceLock Enterprise Server) server for results delivery.

dlesrpc-server-port

The port through which the DEVICELOCK Enterprise Server "grabs" the results (messages/events).

dlesrpc-client-port

The port used to notify the DEVICELock Enterprise Server about messages (events).

file-drop-path

The path to the directory where captured messages will be saved.

quota

The size of the disk quota for storing messages. Example: 10Gb or 500Mb or 100Kb or 10,000. If the quota is exhausted, files will no longer be saved until the quota again allows this. To resume saving files, either free up space or increase the quota.

alert-timeout

Sets the timeout for DLES (DeviceLock Enterprise Server) notification in seconds. If there are unsent messages after the timeout has elapsed, the server will be notified about the availability of results.

alert-rescount

Sets the number of accumulated messages upon which the notification of DLES will be performed.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

3.2.2.2. SMTP Profiles

A sample setup of a delivery profile using SMTP is provided below:

```
<Profile name="smtp-main" enabled="true" default="false">
  <Option name="protocol" value="smtp" />
  <Option name="smtp-server" value="smtp.domain.com" />
  <Option name="smtp-port" value="465" />
  <Option name="enable-ssl" value="true" />
  <Option name="mail-from" value="sensor@internal.net" />
  <Option name="mail-to" value="archive@internal.net" />
  <Option name="tcp-timeout" value="10" />
  <Option name="save-headers" value="all" />
  <Option name="keep-connection" value="false" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="false" />
</Profile>
```

Description of SMTP profile options:

protocol

Defines the data transfer protocol. In this case, it is **smtp**.

smtp-server

IP address or name of the SMTP server for results delivery.

smtp-port

SMTP server port for sending messages.

enable-ssl

Enables/disables the use of SSL encryption when sending messages.

mail-from

The address of the sender of the message for the consumer system.

mail-to

A working email address of the consumer system (for example, the message archiving system).

tcp-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

keep-connection

Send all messages in the same connection with the server. If this option is disabled, then each message will be sent in a separate TCP connection.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

profile-fail-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

go-profile-reserve

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.2.3. FTP Profiles

A sample setup of a delivery profile using FTP is provided below:

```
<Profile name="largefile-ftp" enabled="true" default="false">
  <Option name="protocol" value="ftp" />
  <Option name="file-drop-path" value="ftp://127.0.0.1/test" />
  <Option name="enable-ssl" value="true" />
  <Option name="user-name" value="test" />
  <Option name="password" value="" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-passport" value="true" />
  <Option name="save-eml" value="true" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="0" />
  <Option name="save-headers" value="xsensor, xkpps" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="false" />
</Profile>
```

Description of FTP profile options:

protocol

Defines the data transfer protocol. In this case, it is **ftp**

file-drop-path

Defines the path to the directory on the FTP server to which the result is to be saved. Example: **ftp://127.0.0.1/test**

enable-ssl

Enables/disables the use of SSL encryption when sending messages.

user-name

Login to access the FTP-server.

password

Password for access to the FTP server.

enable-tmp

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

save-passport

Enables saving of data to the FTP server in the **DeviceLock EtherSensor** internal format. Values: **true** or **false**

save-eml

Enables saving of data to the FTP server in the EML envelope format. Values: **true** or **false**

Please note:

1. The **save-eml** option suppresses the **save-passport** option: if both **save-eml** and **save-passport** are set to **true**, data will only be saved in **EML** format.
2. If both **save-eml** and **save-passport** are set to **false**, the delivery profile will work, but the data will not be saved.

save-zip

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

zip-level

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

profile-fail-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

go-profile-reserve

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.2.4. SFTP Profiles

A sample setup of a delivery profile using SFTP is provided below:

```
<Profile name="sftp-transport_1" enabled="true" default="false">
  <Option name="protocol" value="sftp" />
  <Option name="sftp-server" value="10.101.100.125" />
  <Option name="sftp-port" value="22" />
  <Option name="file-drop-path" value="messages" />
  <Option name="user-name" value="test" />
  <Option name="password" value="" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-format" value="eml" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="6" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="true" />
</Profile>
```

Description of SFTP profile options:

protocol

Defines the data transfer protocol. In this case, it is the **sftp** protocol

sftp-server

The IP address or name of the SFTP server for results delivery.

sftp-port

Port SFTP-server for sending messages.

file-drop-path

The directory for storing messages.

user-name

Login for access to the SFTP server.

password

Password for access to the SFTP-server.

enable-tmp

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

save-format

Stored or sent messages format (EML, XML, JSON ...)

save-zip

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

zip-level

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

profile-fail-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

go-profile-reserve

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.2.5. FILEDROP Profiles

A sample setup of a FILEDROP delivery profile is provided below:

```
<Profile name="local-dir" enabled="true" default="true">
  <Option name="protocol" value="filedrop" />
  <Option name="file-drop-path" value=" C:\FileDropPath\" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-passport" value="false" />
  <Option name="save-eml" value="true" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="6" />
  <Option name="quota" value="10Gb" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="1" />
  <Option name="go-profile-reserve" value="false" />
</Profile>
```


Description of FILEDROP profile options:

protocol

Defines the data transfer protocol. In this case, it is **filedrop**.

file-drop-path

The path to the directory where captured messages will be saved.

enable-tmp

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

save-passport

Defines saving of data in the **DeviceLock EtherSensor** internal format. Values: **true** or **false**

save-eml

Defines saving of data in the EML envelope format. Values: **true** or **false**

Please note:

- 1.** The **save-eml** option suppresses the **save-passport** option: if both **save-eml** and **save-passport** are set to **true**, data will only be saved in **EML** format.
- 2.** If both **save-eml** and **save-passport** are set to **false**, the delivery profile will work, but the data won't be saved.

save-zip

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

zip-level

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

quota

The size of the disk quota for storing messages. Example: 10Gb or 500Mb or 100Kb or 10,000. If the quota is exhausted, files will no longer be saved until the quota again allows this. To resume saving files, either free up space or increase the quota.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

profile-fail-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

go-profile-reserve

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.2.6. IMAP Profiles

A sample setup of a delivery profile using IMAP is provided below:

```
<Profile name="imap-transport" enabled="true" default="false">
  <Option name="protocol" value="imap" />
  <Option name="imap-server" value="imap.domain.com" />
  <Option name="imap-port" value="10993" />
  <Option name="file-drop-path" value="INBOX" />
  <Option name="enable-ssl" value="true" />
  <Option name="username" value="administrator@domain.com" />
  <Option name="password" value="test123" />
  <Option name="tcp-timeout" value="10" />
  <Option name="save-headers" value="none" />
</Profile>
```

Description of IMAP profile options:

protocol

Defines the data transfer protocol. In this case, it is **imap**.

imap-server

The IP address or name of the IMAP server for results delivery.

imap-port

IMAP server port for messages delivery.

file-drop-path

The name of the IMAP directory to which the message will be delivered.

enable-ssl

Enables/disables the use of SSL encryption when sending messages.

username

Login to access the IMAP server.

password

Password to access the IMAP server.

tcp-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

3.2.2.7. SMB Profiles

A sample setup of a delivery profile that uses SMB/CIFS protocol is provided below:

```
<Profile name="net_share_1" enabled="true" default="false">
  <Option name="protocol" value="smb" />
  <Option name="file-drop-path" value="\\ARCHSERVER1\Messages" />
  <Option name="user-name" value="test" />
  <Option name="password" value="" />
  <Option name="enable-tmp" value="false" />
  <Option name="save-passport" value="false" />
  <Option name="save-eml" value="true" />
  <Option name="save-zip" value="false" />
  <Option name="zip-level" value="6" />
  <Option name="save-headers" value="none" />
  <Option name="profile-fail-timeout" value="10" />
  <Option name="go-profile-weight" value="2" />
  <Option name="go-profile-reserve" value="true" />
</Profile>
```

Description of SMB/CIFS profile options:

protocol

Defines the data transfer protocol. In this case, it is **smb**

file-drop-path

Path to the SMB/CIFS directory to save messages. Example: \\ARCHSERVER1\Messages

user-name

User name for accessing the SMB/CIFS directory to save messages.

password

Password for accessing the SMB/CIFS directory to save messages.

enable-tmp

Enable/disable the use of 2-step moving of files to avoid collisions: 1) The file is moved with a temporary extension, for example, 2012-01-08-15-29-48-586.7.m.zip.tmp, then 2) After the transfer is complete, ".tmp" is deleted from the file name, the file is renamed in 2012-01-08-15-29-48-586.7.m.zip. May be helpful if a process is waiting for new files to appear in the directory (renaming is an atomic operation).[]

save-passport

Enables saving of data to the file server in the **DeviceLock EtherSensor** internal format. Values: **true** or **false**

save-eml

Enables saving of data to the file server in the EML envelope format. Values: **true** or **false**

Warning!

1. save-eml option suppresses **save-passport** option: if both **save-eml** and **save-passport** are set to **true**, data will only be saved in **EML** format.

2. If both save-eml and save-passport are set to **false**, the delivery profile will work, but the data won't be saved.

save-zip

Whether to pack the saved messages in a ZIP archive (a file with a .zip extension). If this setting is enabled in conjunction with the "Save as EML" setting, EML message envelopes will be packed in a ZIP archives. If this option is enabled and the "Save as EML" setting is disabled, messages in the internal format will be packed into the ZIP archive.

zip-level

ZIP archive compression rate [0-9], where 0 means no compression, 1 means best compression speed, and 9 means best compression algorithm.

save-headers

Enables/disables the saving of standard message headers, as well as of "X-Sensor" headers, additionally, in a separate message attachment - the "microolap_msis_headers.txt" file. Available options: "all" - saves all message headers; "xsensor" - save headers with the "X-Sensor" prefix; "other" - the standard "From", "To", "Cc", "Bcc", and other headers that do not fall under the flag "xsensor"; "none" - disables saving message headers in a separate attachment.

profile-fail-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

go-profile-weight

The weight of the profile (from 1 to 10) specifies a proportion for the distribution of messages among profiles and is used only if this profile is included in a GROUP profile.

go-profile-reserve

Enable/disable the use of the profile as a reserve one. If this setting is enabled and the main (non-reserve) delivery profiles fail, this profile will be used for message delivery. The setting is valid only when used in a GROUP profile.

3.2.2.8. SYSLOG Profiles

A sample setup of a delivery profile using SYSLOG protocol is provided below:

```
<Profile name="syslog_1" enabled="true" default="false">
  <Option name="protocol" value="syslog" />
  <Option name="syslog-server" value="127.0.0.1" />
  <Option name="syslog-port" value="514" />
  <Option name="syslog-tcp" value="false" />
  <Option name="enable-ssl" value="false" />
  <Option name="keep-connection" value="true" />
  <Option name="script-name" value="noname.lua" />
  <Option name="tcp-timeout" value="10" />
</Profile>
```

protocol

Defines the data transfer protocol. In this case, it is **syslog**.

syslog-server

The IP-address or name of the SYSLOG server for results delivery.

syslog-port

SYSLOG server port for sending messages.

syslog-tcp

Allows use of the TCP protocol to send messages to the SYSLOG server. This is necessary if SSL encryption is used when sending messages to the consumer system.

enable-ssl

Enables/disables the use of SSL encryption when sending messages.

keep-connection

Keep connection

script-name

The name of the Lua script file that will be used to generate the message to be sent to the SYSLOG server. The script file must be in the \scripts subfolder.

tcp-timeout

Timeout in seconds to retry message delivery in the event of receiver rejection.

3.2.2.9. GROUP Profiles

A sample setup of a GROUP delivery profile is provided below (a group profile):

```
<Profile name="group-transport" enabled="true" default="false">
  <Option name="protocol" value="gprofile" />
  <Option name="profilename" value="smtp-main" />
  <Option name="profilename" value="net_share_1" />
</Profile>
```

Description of GROUP profile options:

protocol

Defines the data transfer protocol. In this case, it is **gprofile**

profilename

Defines the name of a predefined delivery profile that is a part of the group.

3.3. Logging

The **EtherSensor Watcher** service collects system messages from other **DeviceLock EtherSensor** services and writes them to files and syslog servers assigned by admins, depending on the message channel, logging level and other criteria.

Command line parameters

The **EtherSensor Watcher** service, during **DeviceLock EtherSensor** installation, is installed as a Windows service set to start automatically. However, you can also start it as a Windows **ethersensor_watcher.exe** application with the following command line parameters:

/process

Starts the **ethersensor_watcher.exe** process as a regular Win32 process (may be helpful for debugging).

/service

Starts as a Windows service

/config

Saves the service default configuration

3.3.1. Setting up the Configurator

The **EtherSensor Watcher** service is responsible for collecting system messages from other **DeviceLock EtherSensor** services and writing them to files and syslog servers assigned by admins, depending on the message channel, logging level and other criteria.

Event Logs

The window below allows you to edit the existing rules for **DeviceLock EtherSensor** event logging.

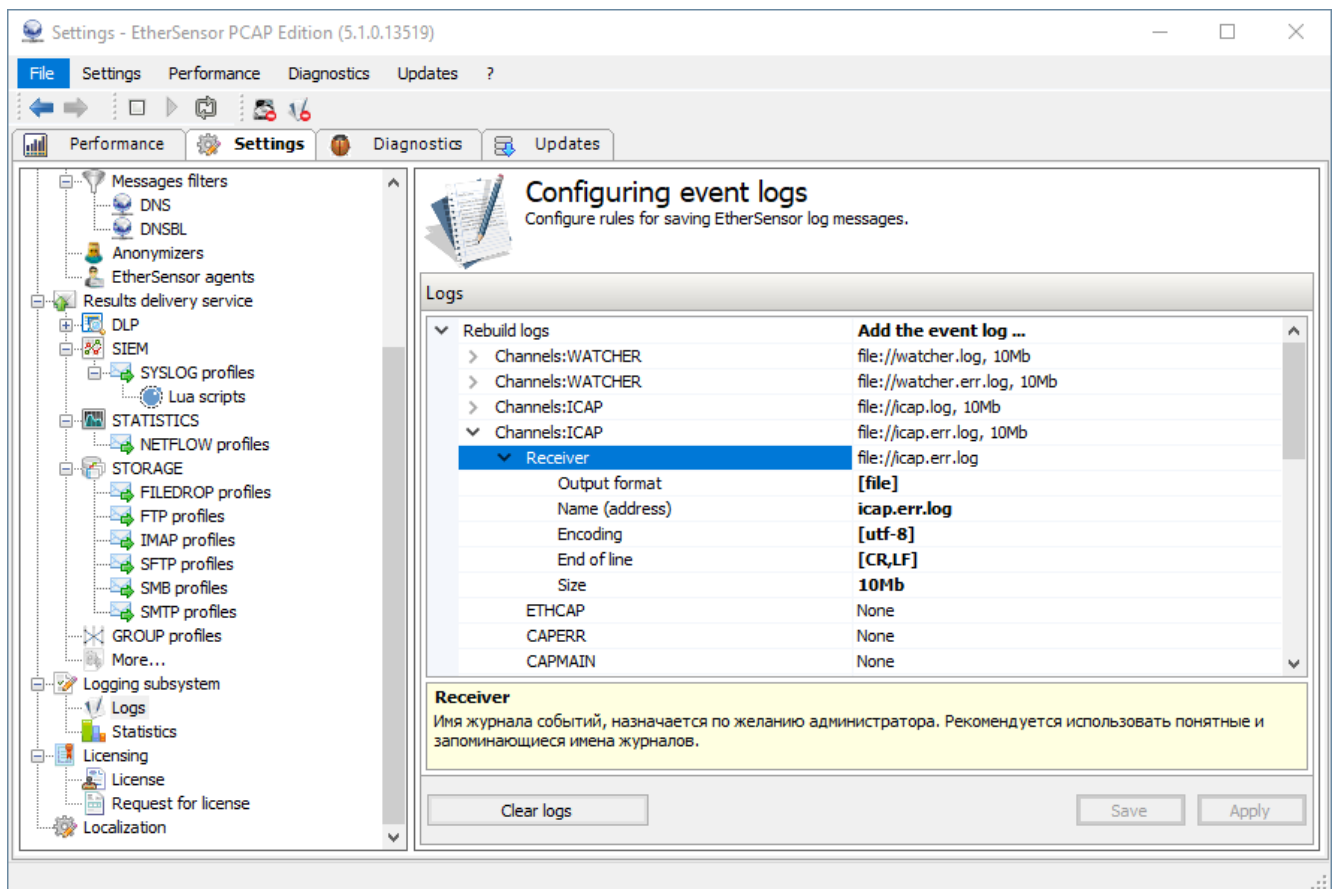


Fig. 32. EtherSensor Watcher service settings.

If you need to define additional log files or change the order in which messages are written to the log files, click the button to the right of "Add the event log ...":

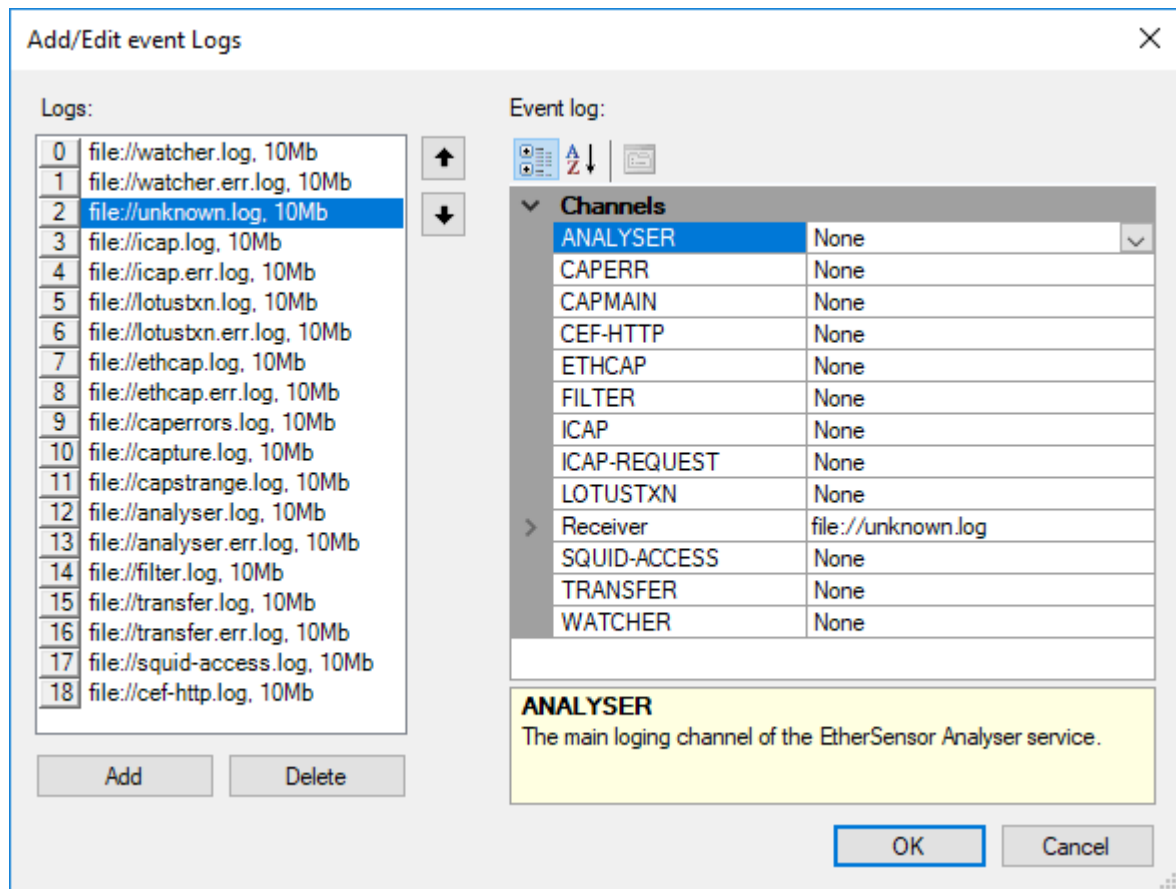


Fig. 33. Fine-tuning the logging rules.

For more information on log channels and levels, refer to "EtherSensor Watcher Services Manual Setup (Config File)" section.

Statistics

When detecting messages, **DeviceLock EtherSensor** allows to gather a variety of statistical data. For example, TCP connection statistics (the MAC address of the interface where the connection has been captured, the connection creation and termination time, connection IP-addresses and ports, the volume of data sent from the client to the server and vice versa, the application-level protocol used (HTTP, ICQ, SMTP, POP3...), etc.).

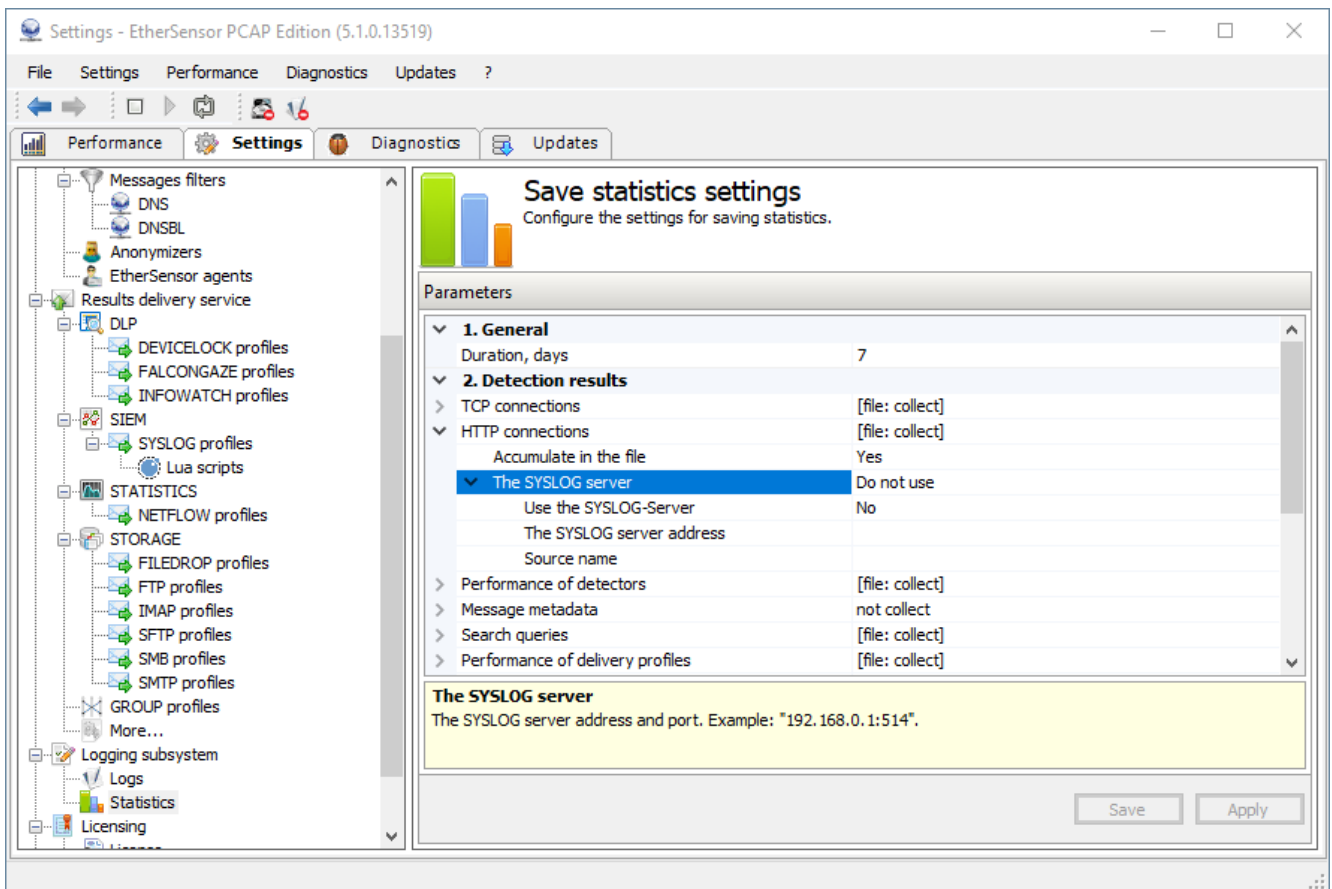


Fig. 34. Configure settings for saving statistics.

The statistics are either gathered in specific local directories or sent to the syslog server.

3.3.2. Manual Setup (Config File)

The **EtherSensor Watcher** service configuration is stored in the **watcher.xml** file, located in the common configuration directory, DeviceLock EtherSensor **[INSTALLDIR]\config**.

A sample **watcher.xml** configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<WatcherConfig version="4.1">
  <Syslog>
    <LogRule
      output="file://icap.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel
        name="ICAP"
        loglevels="all" />
    </LogRule>
    <LogRule
      output="file://icap.err.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel
        name="ICAP"
        loglevels="error, warning, criterr" />
    </LogRule>
    <LogRule output="file://ethcap.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="ETHCAP"
        loglevels="all" />
    </LogRule>
    <LogRule output="file://ethcap.err.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="ETHCAP"
        loglevels="error, warning, criterr" />
    </LogRule>
    <LogRule output="file://caperrors.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="CAPERR"
        loglevels="all" />
    </LogRule>
    <LogRule output="file://capture.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="CAPMAIN"
        loglevels="all" />
    </LogRule>
    <LogRule output="file://analyser.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="ANALYSER"
        loglevels="all" />
    </LogRule>
    <LogRule output="file://analyser.err.log"
      maxsize="10Mb"
      encoding="utf-8"
      endlines="CR,LF">
      <Channel name="ANALYSER"
        loglevels="error, warning, criterr" />
    </LogRule>
    <LogRule output="file://filter.log"
```

```
        maxsize="10Mb"
        encoding="utf-8"
        newline="CR,LF">
    <Channel name="FILTER"
        loglevels="all" />
</LogRule>
<LogRule output="file://transfer.log"
        maxsize="10Mb"
        encoding="utf-8"
        newline="CR,LF">
    <Channel name="TRANSFER"
        loglevels="all" />
</LogRule>
<LogRule output="file://transfer.err.log"
        maxsize="10Mb"
        encoding="utf-8"
        newline="CR,LF">
    <Channel name="TRANSFER"
        loglevels="error, warning, criterr" />
</LogRule>
<LogRule output="file://watcher.log"
        maxsize="10Mb"
        encoding="utf-8" newline="CR,LF">
    <Channel name="WATCHER"
        loglevels="all" />
</LogRule>
<LogRule output="file://watcher.err.log"
        maxsize="10Mb"
        encoding="utf-8"
        newline="CR,LF">
    <Channel name="WATCHER"
        loglevels="error, warning, criterr" />
</LogRule>
<LogRule output="file://squid-access.log"
        maxsize="10Mb"
        encoding="us-ascii"
        newline="LF">
    <Channel name="SQUID-ACCESS"
        loglevels="all" />
</LogRule>
</Syslog>
<Statistics daysnumber="7">
    <Detection>
        <Sessions>true</Sessions>
        <Hosts>true</Hosts>
        <Detectors>true</Detectors>
        <Messages>false</Messages>
        <Squeries>false</Squeries>
        <TransportProfiles>true</TransportProfiles>
    </Detection>
    <Counters>
        <Performance>true</Performance>
        <Icap>true</Icap>
        <Interfaces>true</Interfaces>
        <Parsers>true</Parsers>
        <Cache>true</Cache>
        <Analysers>true</Analysers>
        <Filters>true</Filters>
        <Quotas>true</Quotas>
        <Transports>true</Transports>
    </Counters>
</Statistics>
</WatcherConfig>
```

Description of tags used in the **watcher.xml** configuration file:

WatcherConfig tag

This is the root tag of the service configuration. The **version** attribute specifies the configuration version.

Syslog tag

This is the root tag for configuring access to syslog servers.

LogRule tag

Defines log file configuration. The **output** attribute specifies the log file name, and the **maxsize** attribute specifies the maximum file size. When the maximum file size is reached, the log file is moved to the **backup** directory and a new log file with the same name and the same parameters is created. The **encoding** attribute specifies the encoding in which messages are to be saved, and the **endline** attribute specifies the format for defining the end of a line in a message.

Channel tag

The **Channel** tag is nested within the **LogRule** tag. It contains the name of the service message channel, messages from which are to be saved in a file named **output**.

The channel is a label that indicates a message belongs to a service. Sometimes it also provides details on the service module that was running when the message appeared.

The **name** attribute of the **Channel** tag specifies the name of the channel. In the current version of **DeviceLock EtherSensor** (5.1.0.13519), the following channel names are available:

For the EtherSensor EtherCAP service:

ETHCAP

The main message channel for the service

CAPERR

The channel for traffic analysis error messages

CAPMAIN

The channel for messages from the connections being processed.

For the EtherSensor ICAP service:

ICAP

The main message channel for the service

ICAP-REQUEST

The channel for logging HTTP requests (responses) received from ICAP clients

For the EtherSensor LotusTXN service:

LOTUSTXN

The main message channel for the service

For the EtherSensor Analyser service:**ANALYSER**

The main message channel for the service

FILTER

The channel for messages from the service data filtering service **EtherSensor Analyser**

For the EtherSensor Transfer service:**TRANSFER**

The main message channel for the service

For the EtherSensor Watcher service:**WATCHER**

The main message channel for the service

If **EtherSensor Watcher** has received a message with a channel name that is not defined in the configuration, the message will be saved to **unknown.log**.

The **loglevel** attribute specifies levels for messages that may be logged in the file.

Available values:

all

All messages are logged

criterr

Critical errors

error

Non-critical errors/external resource (a file, a connection) unavailability

warning

Warnings: invalid data formats, exceeding quotas, etc.

info

Regular messages issued during normal operation

debug

Debug messages

detdebug

Detailed debugging messages

DbChangeInterval tag

Defines the interval (in minutes) for the rotation of the database files of **DeviceLock EtherSensor** work counters.

Statistics tag

This is the root tag to configure the gathering of statistics. The **daysnumber** attribute of this tag specifies a number of days over which statistics will be gathered. Possible values are between 0 and 62 days. 0 means that the gathering of statistics has been suspended.

Detection tag

The **Detection** tag is nested within the **Statistics** tag. It contains the settings for gathering statistics on message detection results.

Sessions tag

The **Sessions** tag is nested within the **Statistics** tag. It contains the flag for gathering statistics on TCP connections. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\sessions** directory and includes the following parameters for the connections being monitored:

- the MAC address of the interface where the connection has been captured;
- the connection creation and termination time;
- IP addresses and ports of the connection;
- the amount of data send from the client to the server and vice versa, the protocol used over TCP/IP (HTTP, ICQ, SMTP, POP3).

Hosts tag

The **Hosts** tag is nested within the **Statistics** tag. It contains the flag for gathering statistics on HTTP connections. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\hosts** directories and includes the following parameters for the connections being monitored:

- the MAC address of the interface where the connection has been captured;
- IP addresses and ports of the connection;
- the DNS name of the server to which the connection has been established.

Detectors tag

The **Detectors** tag is nested within the **Statistics** tag. It contains the flag for gathering statistics on the results of message detectors operation. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\detectors** directories and includes the following parameters:

- Event timestamp.
- Detector name.
- Detection status.
- The number of messages detected.

Messages tag

The **Messages** tag is nested within the **Statistics** tag. It contains the flag for gathering message metadata. Such statistics are gathered in an XML file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\messages** directories and includes the captured messages metadata:

- Message service headers sent according to the protocol used.
- Metadata created by **DeviceLock EtherSensor** during message processing (**X-Sensor-...** headers).
- **From, To, Cc, Bcc, Subject** headers.

Squeries tag

The **Squeries** tag is nested within the **Statistics** tag. It contains the flag for gathering search requests. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\squeries** directories and includes the following parameters:

- Event timestamp.
- IP address and port of the search request sender.
- the DNS name of the search service.
- Search request phrase.

TransportProfiles tag

The **TransportProfiles** tag is nested within the **Statistics** tag. It contains the flag for gathering statistics on the results of delivery profiles operation. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\transports** directories and includes the following parameters:

- Event timestamp.
- Delivery profile name.
- Protocol used to send the message.
- Message sending status.

Counters tag

The **Counters** tag is nested within the **Statistics** tag. It contains the settings for gathering **DeviceLock EtherSensor** performance counter values.

Performance tag

The **Performance** tag is nested within the **Counters** tag. It contains the flag for gathering the machine resource usage counter values. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\performance** directories and includes the following parameters:

- Event timestamp.
- CPU usage (current, average and peak values).
- Memory usage (current, average and peak values).
- System thread usage.
- System object (files, events, etc.) descriptor usage.

- General OS load indicators (CPU, memory).

Icap tag

The **Icap** tag is nested within the **Counters** tag. It contains the flag for gathering ICAP server counter values. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\icap** directories and includes the values of the **EtherSensor ICAP** service counters:

- Event timestamp.
- The number of connections to the ICAP server.
- The number of requests (GET, POST, PUT) processed.

Interfaces tag

The **Interfaces** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for traffic processing on interface adapters. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\interfaces** directories and includes the following parameters:

- Event timestamp.
- Processed packet counter.
- Counters of processed TCP connections.

Parsers tag

The **Parsers** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for detected connections by protocol. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\parsers** directories and includes the following parameters:

- Event timestamp.
- Counters of processed TCP connections by protocol (SMTP, POP3, HTTP, FTP).

Cache tag

The **Cache** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for the analyzer cache. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\cache** directories and includes the following parameters:

- Event timestamp.
- Processed analyzer cache object counters.

Analysers tag

The **Analysers** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for detected messages. Such statistics are gathered in a CSV file located in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\analyser** directories and includes the following parameters:

- Event timestamp.
- Analyzer message detection counters.

Filters tag

The **Filters** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for the HTTP request filter and the message filter. Such statistics are gathered in a CSV file located in the `[INSTALLDIR]\data\statistics\YYYY-MM-DD\filters` directories and includes the following parameters:

- Event timestamp.
- Analyzer filters counters (HTTP request RAW filter, message filter).

Quotas tag

The **Quotas** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for disk quotas. Such statistics are gathered in a CSV file located in the `[INSTALLDIR]\data\statistics\YYYY-MM-DD\quotas` directories and includes the following parameters:

- Event timestamp.
- Disk quota usage counters.

Transports tag

The **Transports** tag is nested within the **Counters** tag. It contains the flag for gathering the counter values for delivered messages. Such statistics are gathered in a CSV file located in the `[INSTALLDIR]\data\statistics\YYYY-MM-DD\transports` directories and includes the following parameters:

- Event timestamp.
- Messages delivery counters.

Sample Configurations of EtherSensor Watcher

Example 1:

```
<LogRule output="file://example.log"
  maxsize="10Mb"
  encoding="utf-8"
  endlines="CR,LF">
  <Channel name="ETHCAP"
    loglevels="criterr" />
  <Channel name="ICAP"
    loglevels="error" />
  <Channel name="ANALYSER"
    loglevels="warning" />
  <Channel name="TRANSFER"
    loglevels="warning" />
  <Channel name="WATCHER"
    loglevels="all" />
</LogRule>
```

This configuration will create a file named **example.log** to which messages will be saved for the following channels:

- The **EtherSensor EtherCAP** service - messages from the main **ETHCAP** channel with the following logging level: critical errors.
- The **EtherSensor ICAP** service - messages from the main **ICAP** channel with the following logging level: non-critical service errors.
- The **EtherSensor Analyser** service - messages from the main **ANALYSER** channel with the following logging level: warnings.
- The **EtherSensor Watcher** service - messages from the main **WATCHER** channel with the following logging level: all messages.

Example 2:

```
<LogRule output="file://error.log"
  maxsize="10Mb"
  encoding="utf-8"
  endlines="CR,LF">

  <Channel name="ETHCAP"
    loglevels="error" />
  <Channel name="ICAP"
    loglevels="error" />
  <Channel name="ANALYSER"
    loglevels="error" />
  <Channel name="TRANSFER"
    loglevels="error" />
  <Channel name="WATCHER"
    loglevels="error" />
</LogRule>
```

This configuration will create a file named **error.log** to save messages from all **DeviceLock EtherSensor** services with the logging level of "non-critical errors".

Example 3:

```
<LogRule output="file://ethercap.log"
  maxsize="10Mb"
  encoding="utf-8"
  endlines="CR,LF">

  <Channel name="ETHCAP"
    loglevels="all" />
  <Channel name="CAPERR"
    loglevels="error" />
  <Channel name="CAPMAIN"
    loglevels="error" />
</LogRule>
```

In this case, the file will include messages with the following channel tags: **ETHCAP** - all messages from the service, **CAPERR** - only capture errors, **CAPMAIN** - only message processing errors.

Log File Format

Log files are XML files that look approximately as follows:

```
<Message time="2010-07-12T15:22:27.5390000" level="info">
  <Client channelname="WATCHER"
    processname="watcher.exe"
    modulename="watcher.exe"
    processId="5252" />
  <Text>Start of the application work.</Text>
</Message>
<Message time="2010-07-12T15:22:27.5390000" level="info">
  <Client channelname="WATCHER"
    processname="watcher.exe"
    modulename="watcher.exe"
    processId="5252" />
</Message>
<Message time="2010-07-12T15:28:52.4670000" level="info">
  <Client channelname="WATCHER"
    processname="watcher.exe"
    modulename="watcher.exe"
    processId="5252" />
  <Text>Finish of the application work.</Text>
</Message>
```

Message tag

It is the root directory of a message stored in a log file. It has the following attributes: **time** - specifies the time when the message has been sent, **level** - the priority with which the message was sent (for example, **info** means an information message, **error** means an error message).

Client tag

This tag describes the message sender. It has the following attributes: **channelname** - the message channel name, **processname** - the sender process name, **modulename** - the module name within the process that created the message, **processId** - the process ID in the runtime environment OS **DeviceLock EtherSensor**.

Text tag

The text of the message.

Additionally, in critical situations, the following tags may be used:

AdditionalDotNet tag

Describes **dotNet** error messages (including the information about exceptions thrown during **DeviceLock EtherSensor** work in .NET Framework).

AdditionalWin32 tag

Describes Win32 error messages (including information about exceptions thrown during **DeviceLock EtherSensor** in the native code).

3.4. EtherSensor Agent

EtherSensor Agent is a Windows service installed on workstations. **EtherSensor Agent** does two things:

- The Agent sends details of the process that creates external TCP connections to the **DeviceLock EtherSensor** server (over the UDP protocol). This can be used to map TCP sessions to specific workstations when users work in terminal sessions, are located behind a NAT, or in other, similar situations.
- The Agent sends to the **EtherStat** server (over the TCP protocol) to the port and address specified in the configuration information on the events on the workstation. Data that cannot be sent at the moment (no connection, etc.) are accumulated in a local database. As soon as the Agent detects that the information can be transmitted, it splits the data into 64Kb packets and sends them to the server.

3.4.1. Agent Operation Conditions

All the specialized components of **EtherSensor Agent** run on workstations with the following system requirements:

- OS (Windows 7, Windows 2008, Windows 8, Windows 8.1, Windows 2012, Windows 10) 32/64 bits.
- At least 10 MB of free disk space.
- Requirements for third party ISTs must be followed.

Interacting with third party ISTs

Stable operation of **EtherSensor Agent** requires compatibility with third party information security tools (ISTs) and other infrastructure components:

- **EtherSensor Agent** is installed in the **C:\Program Files\Microolap EtherSensor Agent** directory by default. It can also be installed in other directories specified by the administrator. The installation directory of **EtherSensor Agent** contains other work directories. This path and all its subdirectories should be excluded from monitoring by such tools as antivirus software, search indexers and file change monitoring tools. Such software must not lock files in this directory or its subdirectories from creation, removal, moving or modification.
- **EtherSensor Agent** contains the **ethersensor_agent.exe** service, which must be allowed to run and to operate with local system privileges.
- The **EtherSensor Agent** service must be able to use the UDP and TCP protocols to communicate with the remote server. ISTs must not monitor, modify or restrict connections to the server to which **EtherSensor Agent** sends data. Similarly, ISTs must not prevent the **EtherSensor Agent** service from opening connections to ports used to send data to the **DeviceLock EtherSensor** server.
- **EtherSensor Agent** requires registering **Layered Service Provider** of the **ethersensor_lsp.dll** module. Third party ISTs must not restrict registration of **ethersensor_lsp.dll** and its operation.

- When the software is installed and operated the **EtherSensor Agent** processes use high-privileged calls. OS security policy must allow operations with drivers, process management and access to network interfaces for **EtherSensor Agent**.

3.4.2. Agent Installation

The **EtherSensor Agent** can be installed manually on each workstation or automatically via Active Directory group policies (GPO).

Run the supplied MSI installer (32 or 64 bit) in order to install the Agent manually.

Please note:

Correct installation of **EtherSensor Agent** required default administrator privileges installed with the Windows operating system. Using limited administrator privileges may result in the software operating incorrectly.

The **EtherSensor Agent** service (the **ethersensor_agent.exe** process) is installed with the Agent, and the **Layered Service Provider** (**ethersensor_lsp.dll** module) is registered in the system.

The Agent is installed to the **C:\Program Files\Microolap EtherSensor Agent** directory by default.

Please note:

Correct operation of installed **EtherSensor Agent** instances requires the DNS server in the company network to be configured so that the server name specified in **EtherSensor Agent** settings to point to the **DeviceLock EtherSensor** server IP address.

You can also install the Agent by running the Windows Installer **msiexec.exe** utility in the command line with administrator privileges and the following parameters:

INSTALLDIR:

Directory path where **EtherSensor Agent** is to be installed.

ETHERSTATSERVER:

IP address and port of the **EtherStat** server in the "address:port" format.

KEY:

ZeroMQ protocol key for secure connection with the **EtherStat** server; must contain 40 characters.

ETHERSENSORSERVER:

IP address and port of the DeviceLock EtherSensor server in the "address:port" format.

Example:

```
msiexec /i [path to the EtherSensor Agent MSI package] INSTALLDIR="C:\Program Files\Microolap EtherSensor Agent" ETHERSTATSERVER="etherstat:44445" KEY="0123456789ABCDEFabcdefg!@#$%^&*<>?-+=^" ETHERSENSORSERVER="ethersens:44444"
```

3.4.3. Agent Files

Files in the EtherSensor Agent installation package:

[INSTALLDIR]\config\agent.xml

Text configuration XML-file used to specify **EtherStat** and **DeviceLock EtherSensor** server connection properties, polling period in the workstation system and the filter of applications for which TCP connections are not to be sent to the **DeviceLock EtherSensor** server.

[INSTALLDIR]\syslog.dll

The library required to create and maintain all the *.log files **EtherSensor Agent**.

[INSTALLDIR]\ethersensor_agent.exe

Executable file of the **EtherSensor Agent**, implements the main Agent features. It must be run with one of the following command line parameters:

/service

Run as a Windows service

/process

Run as a Windows process

/install

Install as an OS service

/remove

Remove from OS services

If none of these parameters are specified an incorrect command line error is recorded to the **[INSTALLDIR]\log\svcagent.log** file with the corresponding hint and the service stops.

Files generated in the course of operation:

[INSTALLDIR]\log\svcagent.log

A text XML file where main actions and errors of **EtherSensor Agent** are logged.

[INSTALLDIR]\log\ethersensor_agent.exe.log

A text file which stored details of events generated in the logging subsystem of **EtherSensor Agent**.

[INSTALLDIR]\log\processinfo.log

A text while with details of the current processes tracked by **EtherSensor Agent**.

[INSTALLDIR]\data.db

A database file with details of events which were not sent to the **EtherStat** server over a TCP connection. When a new connection to the server is made the data are sent again and removed from the database on successful delivery.

3.4.4. Agent Logical Modules

HTTP connection tracking and marking module (ethersensor_lsp.dll)

This module is implemented as **Layered Service Provider**. This means that the installed module embeds itself into the network application stack and transparently proxies (and tracks) all TCP connections created by local processes. The settings of the module are stored in the Windows registry.

- UDP port of the **EtherSensor EtherCAP** service. The default port is **44444**.
- HTTP traffic marking flag, default value **1**. If the flag is **1** then each HTTP query is marked with a **X-Sensor-UID** header in the following format: 554E4B4E-4F57-4E20-5555-494400000000, where 554E4B4E-4F57-4E20-5555-494400000000 is the unique user ID associated with a specific computer and specific user of this computer and is global throughout the company network.

The **ethersensor_lsp.dll** module uses the **UDP** protocol to communicate locally with the second main **EtherSensor Agent** module - the **EtherSensor Agent** service (the **ethersensor_agent.exe** process) and to forward to it details of the processes which create TCP connections.

The DeviceLock EtherSensor server communication module (EtherSensor Agent service).

This module is implemented as a Windows system service with the following functions:

- Collection and transfer of data about workstation events over an encrypted TCP connection to the **EtherStat** monitoring and statistics server. If the data cannot be sent the **EtherSensor Agent** service saves them to the local database: **[INSTALLDIR]\data.db**.
- Transfer of data about TCP connections of workstation processes to the **DeviceLock EtherSensor** server over the **UDP** protocol. You can use settings to specify the list of processes to be excluded from tracking.
- Saving service logs to a file: **[INSTALLDIR]\log\svcagent.log**.

The **ethersensor_agent.exe** module extract configuration settings from the **[INSTALLDIR]\config\agent.xml** file. The **EtherSensor Agent** service must be restarted for any change in settings to come into force.

3.4.5. Data Transferred to EtherStat

The agent delivers the data to the **EtherStat** server in the following two cases:

1. Regularly, based on the configuration.
2. In case of an event, which should be reported by **EtherSensor Agent** according to the configuration.

Data sent regularly:

The list of equipment

It is only sent when the **EtherSensor Agent** is started; after that, only changes are delivered (triggered by a change event). This list contains data structures for the devices installed on the workstation. All equipment details are extracted from device **Properties** in Windows **Device Manager** using the corresponding **WinAPI** in the following format:

- Device name.
- Device description.
- Manufacturer name.
- Corresponding **Device Id**.
- **GUID Class** of the device in the {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx} format.
- List of **Hardware Ids** of the device.
- Name of the service with which the device communicates.

The list of installed applications and services

It is only sent when the **EtherSensor Agent** is started; after that, the data are only delivered in case of a change event. This list contains data structures for the software installed on the workstation. All software details are extracted from the Windows registry using **WinAPI** in the following format:

- Product name.
- Manufacturer name.
- Current product version.
- Product installation path.

The following registry keys are used to extract information:

- HKML\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
- HKML\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall
- HKEY_USERS\<Search in each registry>\Software\Microsoft\Windows\CurrentVersion\Uninstall

OS details

Is sent based on the **OSMonitor** configuration tag. These are the details of the Windows operating system installed on the workstation. They are extracted using the **WinAPI**:

- Name, current version, type and status of the operating system.
- Serial number in the XXXXX-XXXXX-XXXXX-XXXXX format.
- Architecture (x86 or x64).
- System and/or domain computer name.
- Physical disk partition and directory of the operating system.

- Date and time of the last system restart, and date of the last system update (OS installation date if no restart has been made yet). The current workstation time is also sent.
- Model and manufacturer of the system board.
- Number of physical and logical processes running.
- Size of the operating system paging file.

Network adapter details

Is sent based on the **NETMonitor** configuration tag. A separate message containing description and settings is created for each adapter. Network adapter details and configuration are extracted using the **WinAPI** and have the following format:

- Adapter name.
- Manufacturer name.
- The MAC address, IP addresses, subnet mask, DNS address settings, etc.
- Network and/or domain computer name.
- DHCP enabled flag.
- Network adapter **GUID**.
- Maximum data transfer rate of the network adapter (in bits per second).

Current load of the computer

These data are extracted using **WinAPI** and contain the following:

- Current CPU load in percent.
- Current RAM usage in percent.
- Current HDD usage in percent.
- Free HDD space.

Data sent on an event:

Changes in the list of installed software

This event is triggered when newly installed applications are detected or existing applications are removed from the workstation. Information on the software installed or removed is extracted from the Windows registry using **WinAPI** and has the following format:

- Product name.
- Manufacturer name.
- Current product version.
- Product installation path.

Changes in the list of installed equipment

This event is triggered when a newly installed or removed device is detected on the workstation. All

the equipment details are extracted from device **Properties** in Windows **Device manager** using the corresponding **WinAPI** and have the following format:

- Device name.
- Device description.
- Manufacturer name.
- Corresponding **Device Id**
- **GUID Class** of the device in the {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx} format.
- List of **Hardware Ids** of the device.
- Name of the service with which the device communicates.

Process starts and stops, associated with the TCP session

This event is triggered when a new process is detected or stopped. When a new process is detected the corresponding process data structure is sent with the following details:

- Process name.
- Command line with the arguments of the process.
- Directory path of the process.
- Process use time by the user.
- The **ProcessID**, **SessionID** and **ParentID** identifiers.

When the process is stopped, the **ProcessID** and **SessionID** identifiers of the session created by the process are sent.

Data sent to the server when a user performs actions in the system:

When a user logs into the system:

- Domain name to which the user belongs.
- User account name.
- **SID** - unique user ID in the operating system.
- **SessionID** - session number on the computer.
- Name of the mode of operation of the user at the workstation: **console** or **rdp**.
- Date and time when the user logged into the system.

When a user logs out of the system:

- **SID** - unique user ID in the operating system.
- **SessionID** - session number on the computer.
- Date and time when the user logged out of the system.

When a user account is locked or unlocked:

- **SID** - unique user ID in the operating system.
- **SessionID** - session number on the computer.

When the active window is changed:

- Current window title.
- ID of the owner process of the window.

3.4.6. Data Transferred to EtherSensor

EtherSensor Agent delivers the obtained data to the **DeviceLock EtherSensor** server over the UDP protocol.

The agent provides the **DeviceLock EtherSensor** server with the details of the process which creates external TCP connections:

- Process ID.
- Process name.
- User name used to run the process.
- Computer name.
- User ID.

along with the details of the TCP connection itself:

- Process ID.
- User ID.
- Connection properties.

The data delivered by the agent are used by the **DeviceLock EtherSensor** server to assign the following properties to reconstructed traffic objects depending on the configuration:

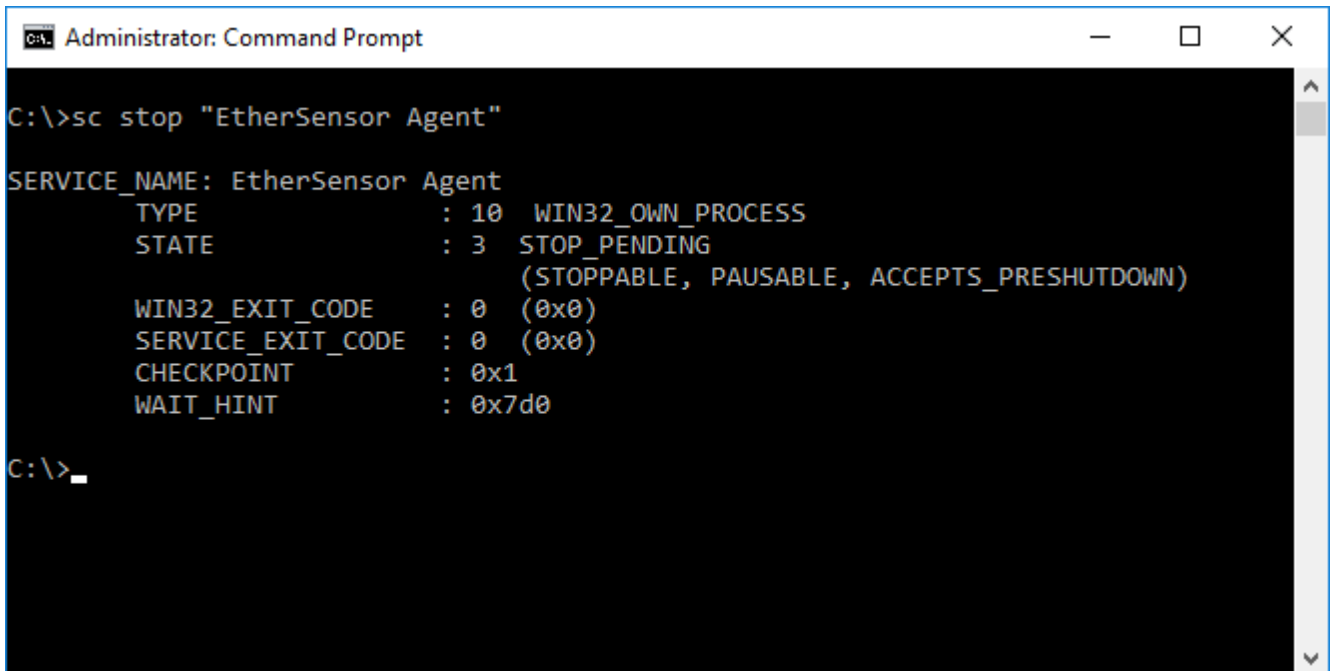
- User name used to run the process.
- Computer name.
- User ID in the **X-Sensor-UID** format: 554E4B4E-4F57-4E20-5555-494400000000.

3.4.7. Working with the Agent

Prior to using the **EtherSensor Agent** some setup is required:

1. Use any text editor to configure the required settings in the **[INSTALLDIR]\config\agent.xml** file.
2. Run **cmd.exe** with Administrator privileges.

3. Stop the **EtherSensor Agent** service by running **sc stop "EtherSensor Agent"**.



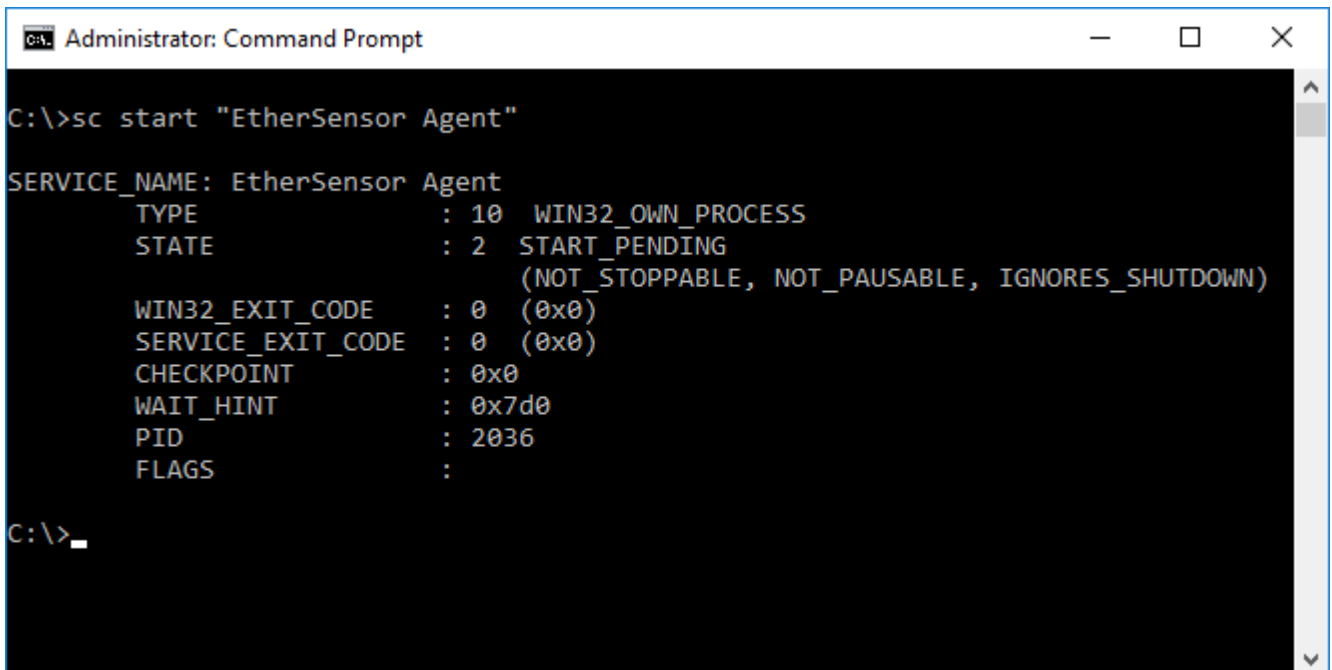
```
C:\>sc stop "EtherSensor Agent"

SERVICE_NAME: EtherSensor Agent
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 3   STOP_PENDING
                        (STOPPABLE, PAUSABLE, ACCEPTS_PRESHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE    : 0   (0x0)
        CHECKPOINT           : 0x1
        WAIT_HINT            : 0x7d0

C:\>_
```

Figure 35. Stopping "EtherSensor Agent" service.

4. Start the **EtherSensor Agent** service by running **sc start "EtherSensor Agent"**.



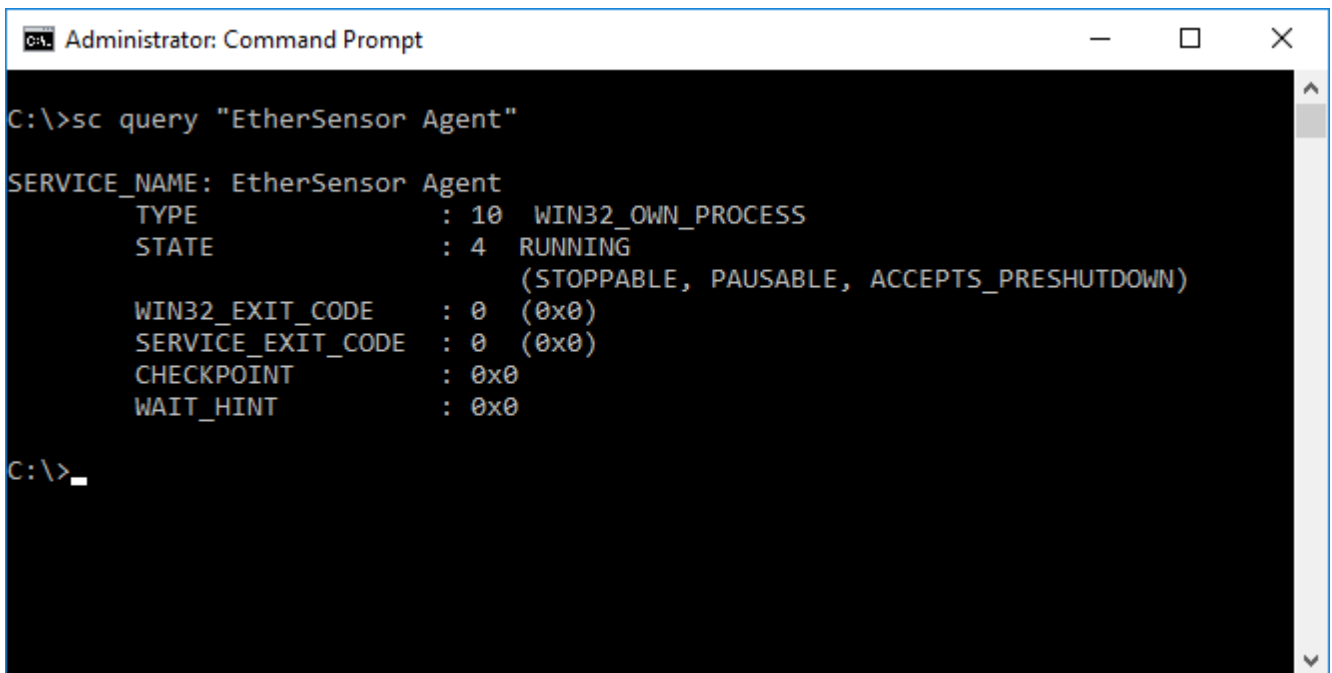
```
C:\>sc start "EtherSensor Agent"

SERVICE_NAME: EtherSensor Agent
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 2   START_PENDING
                        (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE    : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x7d0
        PID                 : 2036
        FLAGS                 :

C:\>_
```

Figure 36. Starting "EtherSensor Agent" service.

5. Make sure the service is started by running **sc query "EtherSensor Agent"**. The service must be in the **RUNNING** state.



```
C:\>sc query "EtherSensor Agent"

SERVICE_NAME: EtherSensor Agent
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                        (STOPPABLE, PAUSABLE, ACCEPTS_PRESHUTDOWN)
        WIN32_EXIT_CODE       : 0    (0x0)
        SERVICE_EXIT_CODE   : 0    (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

C:\>_
```

Figure 37. Checking "EtherSensor Agent" service

3.4.7.1. Possible Agent Operation Methods

Communication with EtherStat

Communication with the **EtherStat** server requires that any workstations where **EtherSensor Agent** is installed belong to the same local network. The **EtherStat** server uses encrypted TCP connection to collect information from workstations and then analysis it. The Agent generates unique UHIDs which identify workstations and sends them in the messages to the **EtherStat** server.

Transparent proxying without traffic marking with DeviceLock EtherSensor

In this mode, the Agent proxies transparently connections of applications which run on the user computer. If an application establishes a successful connection the Agent provides the **DeviceLock EtherSensor** server with information about the connection established by the specific application run by a specific network user.

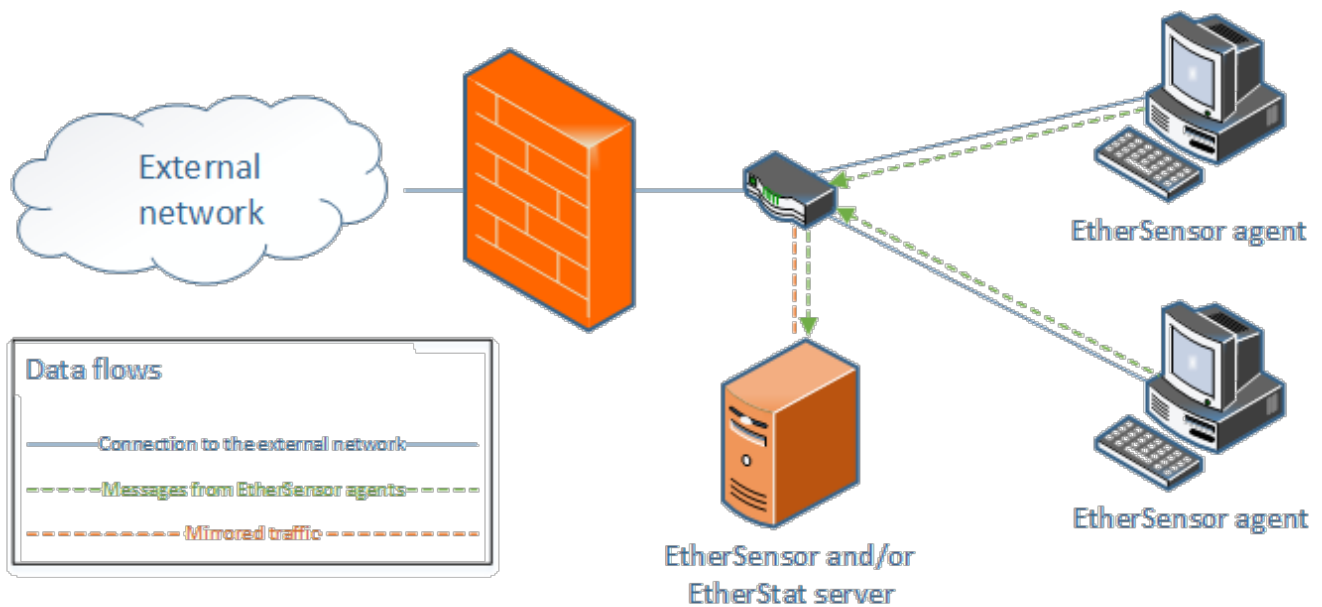


Figure 38. Transparent proxying without traffic marking

Thus, **DeviceLock EtherSensor** can reconstruct the message to completely identify the user who sent this message over any of the currently supported protocols (ICQ, MSN, MRA, IRC, XMPP, SMTP, POP3, LOTUS, HTTP, FTP, etc.).

The analyzed traffic must be turned readdressed to **DeviceLock EtherSensor** before any changes are made to connection parameters.

Examples:

- To the proxy server.
- To NAT.
- To the network firewall.

Transparent proxying with HTTP traffic marking with DeviceLock EtherSensor

This operating mode of the Agent is only different from the non-marking mode in that the Agent modifies HTTP queries sent by applications on the client workstation by adding the **X-Sensor-UID: <GUID>** header to them where **<GUID>** is the unique user ID of the user of a specific computer within the local network. These actions are performed in strict accordance to the HTTP protocol without any violations of it.

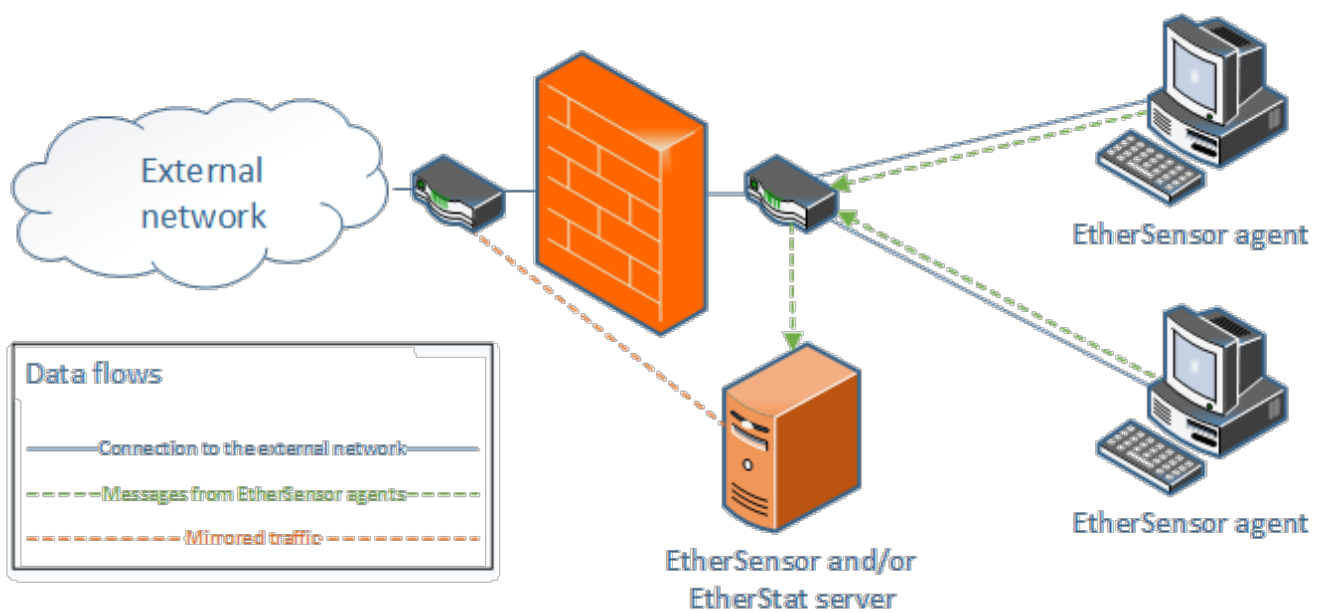


Figure 39. Proxying with traffic marking

The operating mode may be used when **DeviceLock EtherSensor** receives the traffic readdressed for analysis after connection parameters are modified. E.g. when connections pass through a proxy server, NAT or a network firewall.

In this case **DeviceLock EtherSensor** reconstructs the message to completely identify the network user who sent this message over the HTTP protocol.

3.4.7.2. Configuring the Service EtherSensor Agent

EtherSensor Agent is configured by editing the configuration file: **EtherSensor Agent [INSTALLDIR]\config\agent.xml**.

Service configuration data **EtherSensor Agent** are stored in XML format in the **agent.xml** file in the **[INSTALLDIR]\config** directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config version="1.1">
  <Local port="44444" markhttp="true" />

  <EtherSensor protocol="2">
    <server address="ethersensor.server1:44444" transport="udp" />
  </EtherSensor>
  <Filter>
    <Excludes>
      <application name="ethersensor_agent.exe" />
      <application name="mstsc.exe" />
      <application name="wmplayer.exe" />
      <application name="uTorrent.exe" />
      <application name="skype.exe" />
      <application name="wmpnetwk.exe" />
      <application name="winlogon.exe" />
      <application name="svchost.exe" />
      <application name="spoolsv.exe" />
      <application name="nissrv.exe" />
    </Excludes>
  </Filter>

  <EtherStat address="127.0.0.1:44445" ZMQKEY="" />
  <DataCollectionSetup>
    <Hardware duration_ms="10000" />
    <Software duration_ms="30000" />
    <OperatingSystem duration_ms="60000" />
    <Processes duration_ms="1000" />
    <Performance duration_ms="60000" />
    <Network duration_ms="60000" />
    <UserMonitor duration_ms="1000" />
    <DatabaseStore size="1" />
  </DataCollectionSetup>
</Config>
```

Config tag

It is the main configuration tag. The **version** attribute inside the **Config** tag defines the configuration version.

Local tag

The **Local** tag is nested within the **Config** tag and defines settings of the connection tracking module (**ethersensor_lsp.dll**). After the configuration is loaded the **EtherSensor Agent** service saves the settings to the Windows registry.

The **port** attribute defines the local UDP port the **ethersensor_lsp.dll** module uses to communicate with the **EtherSensor Agent** service.

The **markhttp** flag attribute allows or blocks HTTP traffic marking by the **ethersensor_lsp.dll** module.

The **Local** tag is nested within the **Config** tag and defines settings of the connection tracking module (**ethersensor_lsp.dll**). After the configuration is loaded the **ethersensor_agent.exe** service saves the settings to the Windows registry.

The **port** attribute defines the local UDP port the **ethersensor_lsp.dll** module uses to communicate with the **EtherSensor Agent** service.

The **markhttp** flag attribute allows or blocks HTTP traffic marking by the **ethersensor_lsp.dll** module.

The **Local** tag is nested within the **Config** tag and defines settings of the connection tracking module (**ethersensor_lsp.dll**). After the configuration is loaded the **ethersensor_agent.exe** service saves the settings to the Windows registry.

The **port** attribute defines the local UDP port the **ethersensor_lsp.dll** module uses to communicate with the **EtherSensor Agent** service.

The **markhttp** flag attribute allows or blocks HTTP traffic marking by the **ethersensor_lsp.dll** module.

EtherSensor tag

The **EtherSensor** tag is nested within the **Config** tag and defines the list of **DeviceLock EtherSensor** servers with which **EtherSensor Agent** exchanges information (over the UDP protocol) about the processes which create TCP connections in order to associate the workstation to the TCP session.

The **protocol** attribute defines the maximum version of the protocol used by **EtherSensor Agent** to send messages to the **DeviceLock EtherSensor** server. Version **3** of the protocol is the latest one. Support of this protocol requires **DeviceLock EtherSensor** version **4.3.3** or later. For compatibility with previous **DeviceLock EtherSensor** versions, set this field to **2**.

server tag

The **server** tag is nested within the **EtherSensor** tag and defines the address and transport protocol of the **DeviceLock EtherSensor** server.

The **address** attribute defines the address and port used to communicate with the **DeviceLock EtherSensor** server. Possible addresses are **IP:Port** or **DNSNAME:Port**.

The **transport** attribute defines the transport protocol type used to communicate with the **DeviceLock EtherSensor** server. Possible options: **udp**.

Filter tag

The **Filter** tag is nested within the **Config** tag and defines filtering settings for messages sent to the **DeviceLock EtherSensor** server.

Excludes tag

The **Excludes** tag is nested within the **Filter** tag and defines the list of applications for which TCP connection data should not be sent to the **DeviceLock EtherSensor** server.

application tag

The **application** tag is nested within the **Excludes** tag and defines the application for which TCP connection data will not be sent to the **DeviceLock EtherSensor** server.

The **name** attribute defines the exact name of the tracked process.

Thus, **EtherSensor Agent** notifies the **DeviceLock EtherSensor** server only about TCP connections created for communication with other workstations and servers in the local network and on the Internet, and you can configure the settings to exclude certain processes from tracking.

EtherStat tag

Is nested within the **Config** tag and defines settings used to connect to the **EtherStat** monitoring and statistics system.

The **address** attribute defines the server address in the "**IP address:port**" format.

The **ZMQKEY** attribute must contain the key for the encrypted connection operation mode.

DataCollectionSetup tag

Is nested within the **Config** tag and defines timer settings for **EtherSensor Agent** service polling.

Hardware tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to poll existing equipment.

Software tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to poll installed software.

OperatingSystem tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to poll OS data.

Processes tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to monitor current processes.

Network tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to poll network adapter data and configuration.

Performance tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to poll OS performance data.

Users tag

Is nested within the **DataCollectionSetup** tag and uses the **duration_ms** attribute to define the timer (in seconds) to monitor user actions.

DatabaseSetup tag

Is nested within the **DataCollectionSetup** tag and defines the database size limit as percentage of free space on the HDD where **EtherSensor Agent** is installed.

3.4.7.3. Agent Operation Logging

EtherSensor Agent logs its activity to the **[INSTALLDIR]\log** directory in the following files:

- The **svcagent.log** file stores information about the main actions performed by the **EtherSensor Agent** service.
- The **ethersensor_agent.exe.log** file stores information about the events generated inside the **EtherSensor Agent** logging service.
- The **processinfo.log** file stored information about the current processes tracked by **EtherSensor Agent**.

Log files (**svcagent.log**, **ethersensor_agent.exe.log**) are XML files with the following contents:

```
<Message time="2012-03-23T17:47:48.148+04:00" level="information">
  <Client channelname="MICROOLAPAGENT"
    processname="ethersensor_agent.exe"
    modulename="ethersensor_agent.exe" />
  <Text>Start of the application.</Text>
</Message>
```

Message tag

Is the root tag of the message saved to a log file. The **time** attribute stores the message sending time, the **level** attribute defines the message sending priority (e.g. **information** for an information message, **error** for an error message).

Client tag

This tag describes the message sender. It has the following attributes: **channelname** - name of the message channel, **processname** - name of the sender process, **modulename** - name of the module within the process which created the message.

Text tag

Message text.

The **processinfo.log** file is an XML file with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<Processes>

  <Process pid="4136" name="chrome.exe">
    <User uuid="32014294-5bbf-11e1-b8f5-005056c00808"
      name="Home-PC\Home"/>
    <Sessions local="0" remote="311"/>
  </Process>

  <Process pid="636" name="svchost.exe">
    <User uuid="3a45de5b-5be6-11e1-b8f5-005056c00808"
      name="HOME\HOME-PC$"/>
    <Sessions local="0" remote="3"/>
  </Process>

  <Process pid="948" name="firefox.exe">
    <User uuid="32014294-5bbf-11e1-b8f5-005056c00808"
      name="Home-PC\Home"/>
    <Sessions local="2" remote="741"/>
  </Process>

  <Process pid="1584" name="googletalk.exe">
    <User uuid="32014294-5bbf-11e1-b8f5-005056c00808"
      name="Home-PC\Home"/>
    <Sessions local="0" remote="89"/>
  </Process>

  <Process pid="2860" name="uTorrent.exe">
    <User uuid="32014294-5bbf-11e1-b8f5-005056c00808"
      name="Home-PC\Home"/>
    <Sessions local="100" remote="27084"/>
  </Process>

  <Process pid="3076" name="vmware.exe">
    <User uuid="32014294-5bbf-11e1-b8f5-005056c00808"
      name="Home-PC\Home"/>
    <Sessions local="4" remote="1"/>
  </Process>

  <Process pid="3908" name="NisSrv.exe">
    <User uuid="fb91c5e7-5eec-11e1-b226-005056c00808"
      name="NT AUTHORITY\LOCAL SERVICE"/>
    <Sessions local="0" remote="1"/>
  </Process>
</Processes>
```

Processes tag

Is the main tag of the displayed list of tracked processes.

Process tag

The **Process** tag is nested within the **Processes** tag. This tag describes the tracked process. It has the following attributes: **pid** - process ID in the execution environment, **name** - name of the tracked process.

User tag

The **User** tag is nested within the **Process** tag. This tag describes the local system user the credentials of which are used to run the tracked process. It has the following attributes: **uuid** - user ID, **name** - user name.

Sessions tag

The **Sessions** tag is nested within the **Process** tag. This tag describes tracked connections of the process. It has the following attributes: **local** - the number of local connections within the process or with other processes, **remote** - the number of remote connections made by this process.

3.4.7.4. Troubleshooting

EtherSensor Agent service does not start.

- Check the **EtherSensor Agent** configuration for port settings for the **DeviceLock EtherSensor** and **EtherStat** servers, and the local port value: the **Local**, **EtherSensor** and **EtherStat** tags. The ports must not be the same. Otherwise the service may operate incorrectly and stop its process.
- Check the log files (**svcagent.log**, **ethersensor_agent.exe.log**) for any error messages of **EtherSensor Agent**.
- Check the Windows logs for any error messages of **EtherSensor Agent**.
- Report the incidents to support.

No tracked process displayed in the processinfo.log file after EtherSensor Agent is started.

- The network card is not connected, or the TCP/IP stack is disabled. Check whether the system creates TCP connections (e.g. by opening a browser and loading any page). After that the browser process should appear in the **processinfo.log** file.
- This computer belongs to a domain. If so, the **EtherSensor Agent** modules loaded into processes creating TCP connections attempt to get the name of the user who runs the process. Correct DNS settings of the tracked OS are very important in this case because the system API which provides information on the domain user uses these DNS settings.

No application in the system is able to create a remote connection, while there were no such issues before EtherSensor Agent was installed in the system.

- Open the **EtherSensor Agent** installation directory and run the following command: **ethersensor_instlsp.exe -p > log.txt**. This will save the list of installed providers of the OS network stack to the **log.txt** file.
- Analyze the **log.txt** file yourself and send it to support if necessary.
- Run the following command: **ethersensor_instlsp.exe -f -c b**. This command disables the **EtherSensor Agent ethersensor_lsp.dll** tracking module. Start a new browser instance and open a remote page. If the page opens then there is a problem with the **ethersensor_lsp.dll** tracking module. Otherwise the problem is not related to the **EtherSensor Agent** tracking module.

DeviceLock EtherSensor does not recognize the user of an intercepted message

- Check the **EtherSensor Agent** configuration: the **EtherSensor** tag, then the **server** tag. The **address** attribute of the **server** tag must contain the correct DNS name of the **DeviceLock EtherSensor** service which must be correctly resolved on this workstation. If a DNS address is specified instead, check the availability of the **DeviceLock EtherSensor** server IP address (e.g. using the **ping** utility).

The EtherStat server does not receive messages from EtherSensor Agent:

- Check the **svcagent.log** file for any error messages of **EtherSensor Agent**.
- Check the configuration of connection with the **EtherStat** server: the **EtherStat** tag. The **address** attribute of this tag must contain the IP address to which the connection is established. The **key** attribute contains the public key of the encrypted connection. This key must be identical to the public key of the **EtherStat** server.
- Check the availability of the **EtherStat** server IP address (e.g. using the **ping** utility).
- Check the information processing time configuration in the following tags: **OSMonitor**, **HWMonitor**, **SWMonitor**, **UserMonitor**, **NetMonitor** and **ProcMonitor**. If the **timer** attribute is set to **0**, the messages for corresponding events are not processed and thus are not sent to the **EtherStat** server.

4. Event and Object Analysis

The **EtherSensor Analyser** service is used to detect, filter and analyse objects extracted from traffic.

The service analyses OSI model application-level protocol objects received from the **EtherSensor EtherCAP**, **EtherSensor ICAP** and **EtherSensor LotusTXN** to detect messages sent by network users and attribute them to certain Internet services. The messages extracted from the traffic are analyzed to check the following parameters:

- Network addresses of communication parties.
- Domain addresses of communication parties.
- E-mail addresses (**from**, **to**, **cc**, **bcc**).
- IDs of instant messenger users (**ICQ**, **MRA**, **MSN**, **IRC**, **XMPP**).
- IDs of social network users.
- Text field contents of messages (**subject**, **body**).
- Names of transmitted attachments.
- Message sizes.

The service filtering mechanism takes one of the following decisions based on pre-configured filter rules logic:

1. Stop message processing.
2. Transmit the message to the consumer system (DLP, UEBA, archive, eDiscovery system, Enterprise Search, etc.).

3. Generate an arbitrary string based on data extracted from the message (usually a syslog string for a SIEM system).

General principles of service operation **EtherSensor Analyser**:

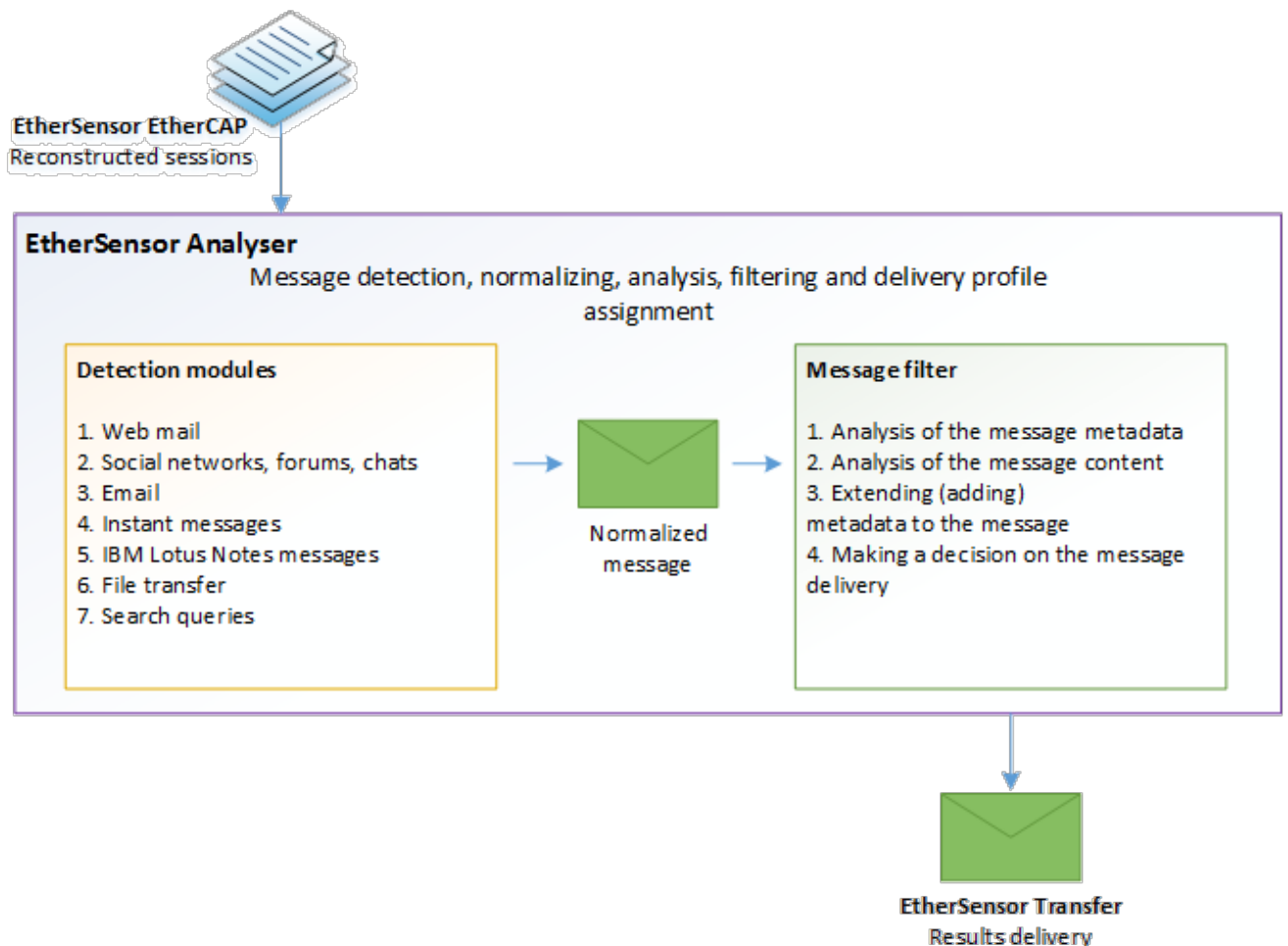


Figure 40. Principles of the EtherSensor Analyser service operation.

The service filtering mechanism is configured with a separate configuration file where logic processing logic is described for recognized messages. The message processing concept is based on chains of rules created and then combined into tables.

The message is checked against rules which may modify its contents, metadata or processing depending on whether the message is affected by the rule. This concept is similar to filter rules used in the **iptables** UNIX-based utility.

General diagram of message processing by the filtering mechanism:

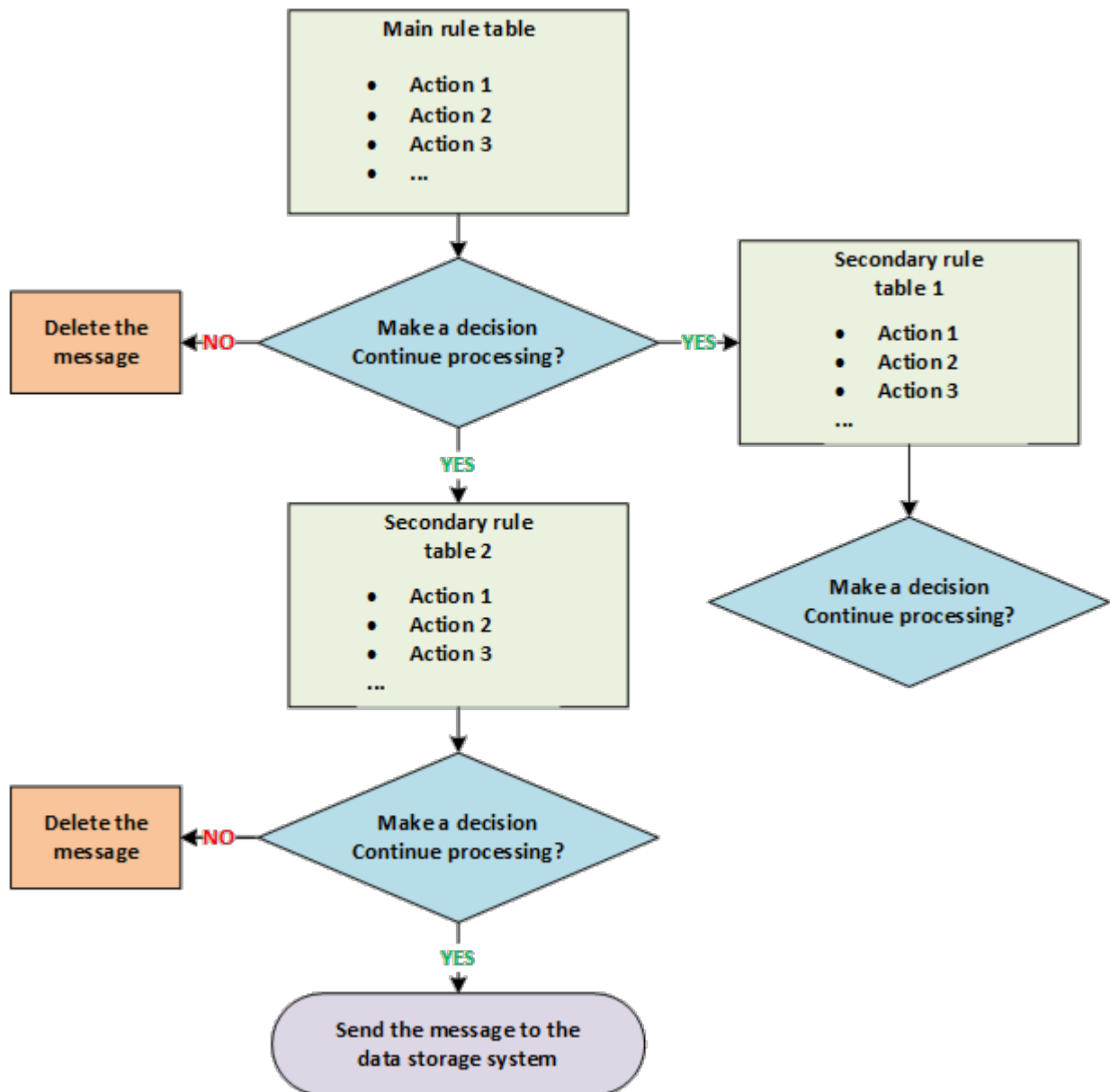


Figure 41. The diagram of message processing by the EtherSensor Analyser service.

Command line parameter

The **EtherSensor Analyser** Windows service is set up for automatic start when **DeviceLock EtherSensor** is installed. But you can also start the **ethersensor_analyser.exe** process as a Windows application with the following command line parameter:

/process

Starts the **ethersensor_analyser.exe** process as a common Windows Win32 process (can be used for debugging).

/service

Starts as a Windows service.

/config

Saves the default service configuration.

4.1. Setting up the Configurator

In addition to editing of the **EtherSensor Analyser** configuration file directly, you can configure it in the graphic interface of the **DeviceLock EtherSensor** utility (the **ethersensor_console.exe** provided with the software).

Disk quotas for the storage of intercepted objects

Intercepted objects may be saved by the **EtherSensor Analyser** service in the file system for further analysis. The form in the left configurator window can be used to configure disk quotas for the storage of such objects (the **ethersensor_console.exe** provided with **DeviceLock EtherSensor**):

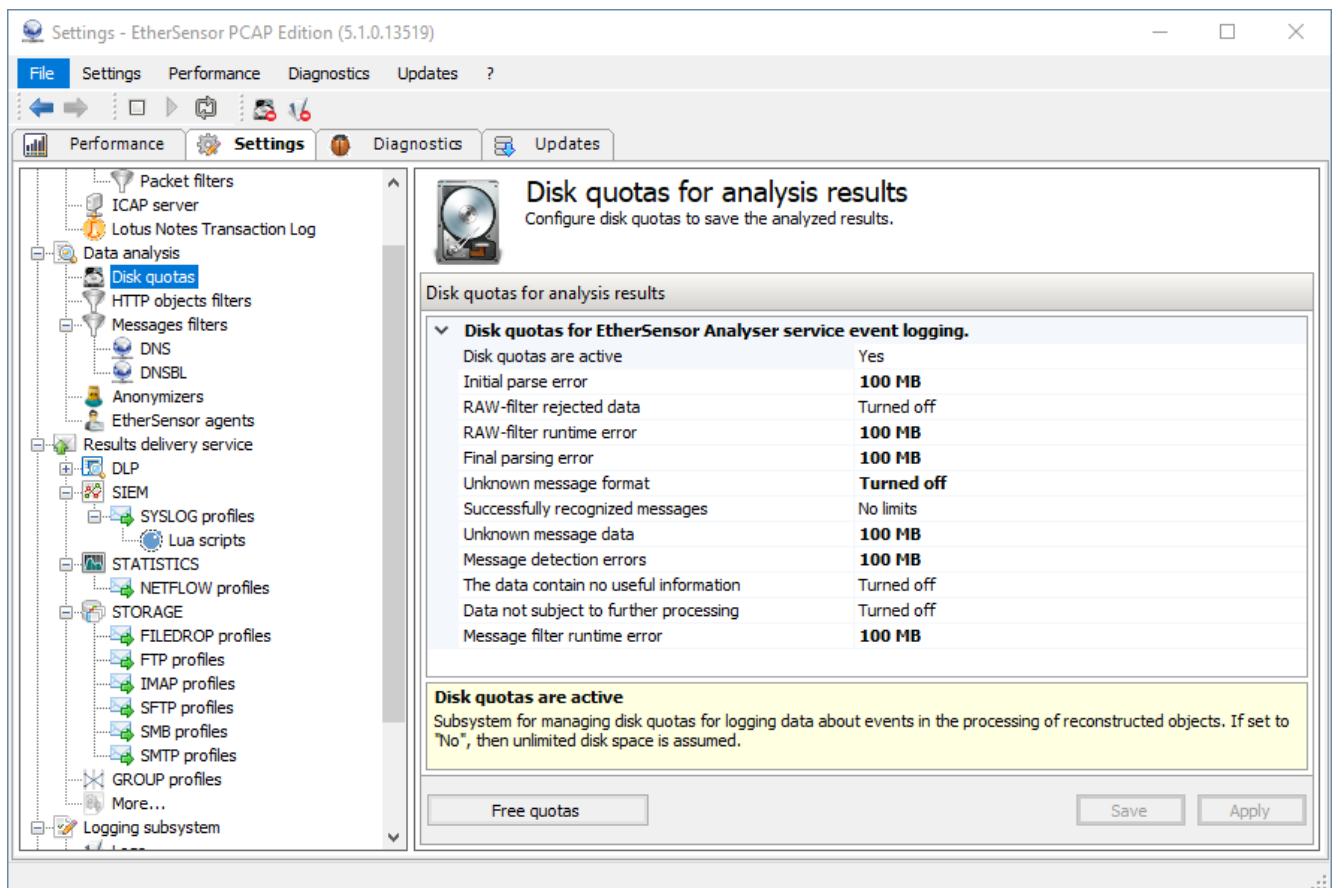


Figure 42. EtherSensor Analyser service settings.

Disk quotas are active:

Enabling/disabling quota usage. If it is setup in **No**, unlimited disk space is assumed as available.

Initial parse error:

Logging of pre-parsing errors of extracted objects: TCP session was reconstructed successfully, but when the object was being parsed an error occurred, usually caused by protocol violations (for example, a browser add-on sets the **00** end-of-line byte in the POST request instead of **0DOA**). The objects are saved to the **[INSTALLDIR]\data\results\errors\preparse** directory.

RAW-filter rejected data

Logging of data rejected by the RAW filter is used to debug or test the logic of the RAW filter. The objects are saved to the **[INSTALLDIR]\data\results\errors\rawfiltered** directory.

RAW-filter runtime error

Logging of RAW filter runtime errors. The objects are saved to the **[INSTALLDIR]\data\results\errors\rawfilter** directory.

Final parsing error

Logging of parsing errors of extracted objects: The TCP session was reconstructed successfully, but when the object was being parsed an error occurred, usually caused by protocol violations (for example, a browser add-on sets the **00** end-of-line byte in the POST request instead of **0DOA**). The objects are saved to the **[INSTALLDIR]\data\results\errors\parse** directory.

Unknown message format

Logging of errors caused by the fact that the system does not recognize the format of the object: TCP session data have been reconstructed successfully, but there is no detector for this message type in the current version. The objects are saved to the **[INSTALLDIR]\data\results\unknown** directory.

Successfully recognized messages

Logging events for successfully recognized objects: >TCP session data has been reconstructed successfully, the message is recognized and passed to the **EtherSensor Transfer** service for delivery to the consumer system. The objects are saved to the **[INSTALLDIR]\data\results\detect\ok** directory.

Unknown message data

Logging errors related to objects that were recognized but contain unknown data: the TCP session has been reconstructed successfully, the corresponding detector for this message type has been triggered, but it is unable to extract all the data - most likely the message format has changed. The objects are saved to the **[INSTALLDIR]\data\results\detect\unknown** directory.

Message detection errors

Logging messages detection errors. The objects are saved to the **[INSTALLDIR]\data\results\errors\detect** directory.

The data contain no useful information

Logging data that have been successfully recognized but contain no useful data for further processing. The objects are saved to the **[INSTALLDIR]\data\results\detect\nobjects** directory.

Data not subject to further processing

Logging data that initially carry no useful information, since these are service data transmitted during the interaction of the user with the Internet service. Such data are not intended for further processing. The objects are saved to the `[INSTALLDIR]\data\results\detect\filtered` directory.

Message filter runtime error

Logging of message filtering errors. The objects are saved to the `[INSTALLDIR]\data\results\errors\msgfilter` directory.

Prefiltering of HTTP objects

Filtering of intercepted HTTP queries can be configured in the **HTTP objects filters** configurator window:

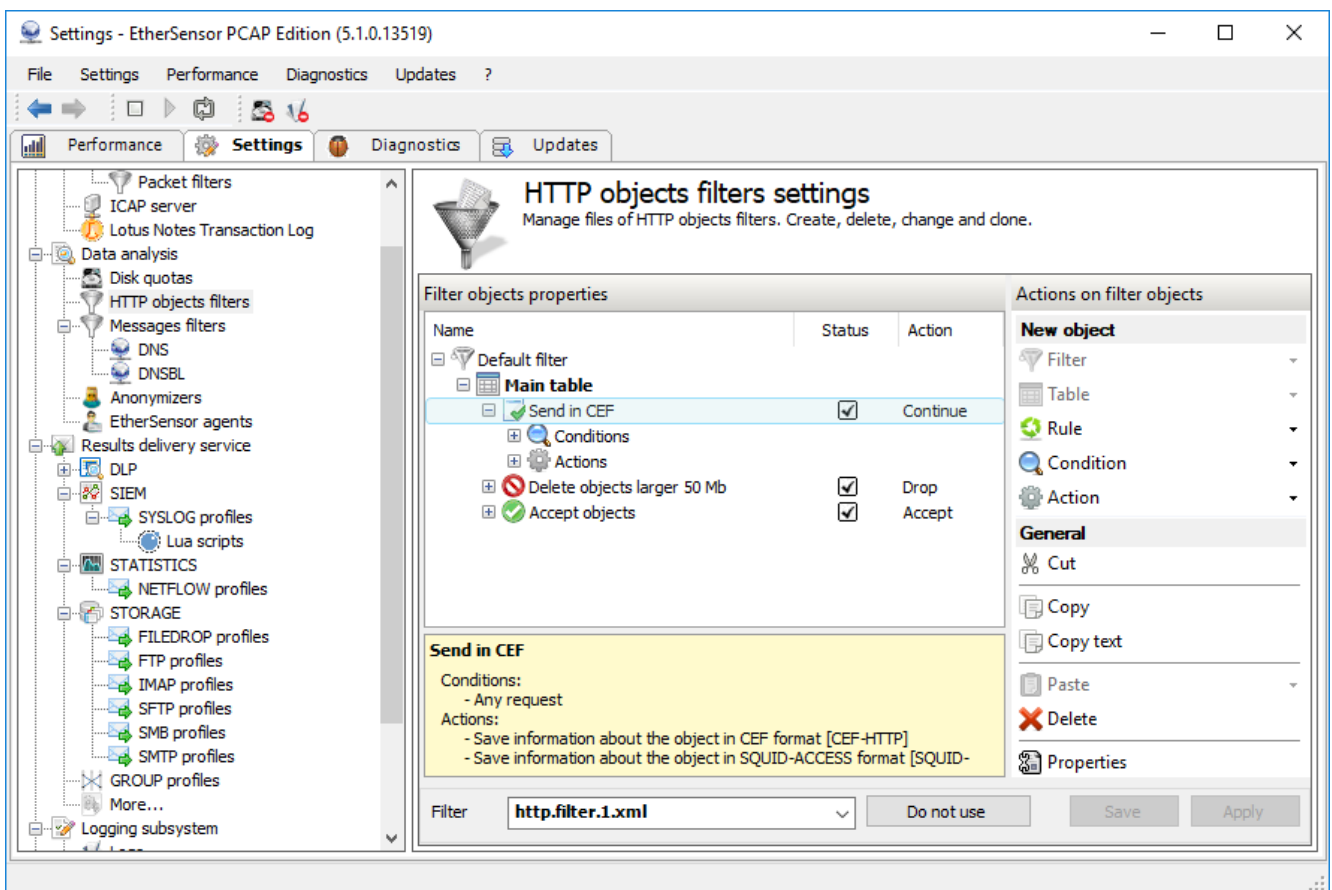


Figure 43. HTTP object filtering management.

Prefiltering of HTTP objects is mainly used to reduce the load on the sensor. This feature is described in more detail in the "Prefiltering of HTTP queries" section.

Filtering of reconstructed messages

Filtering of intercepted messages can be configured in the **Messages filters** configurator window. Filter rules for intercepted objects are stored in XML files in the **[INSTALLDIR]\data\statistics\YYYY-MM-DD\filters** directory.

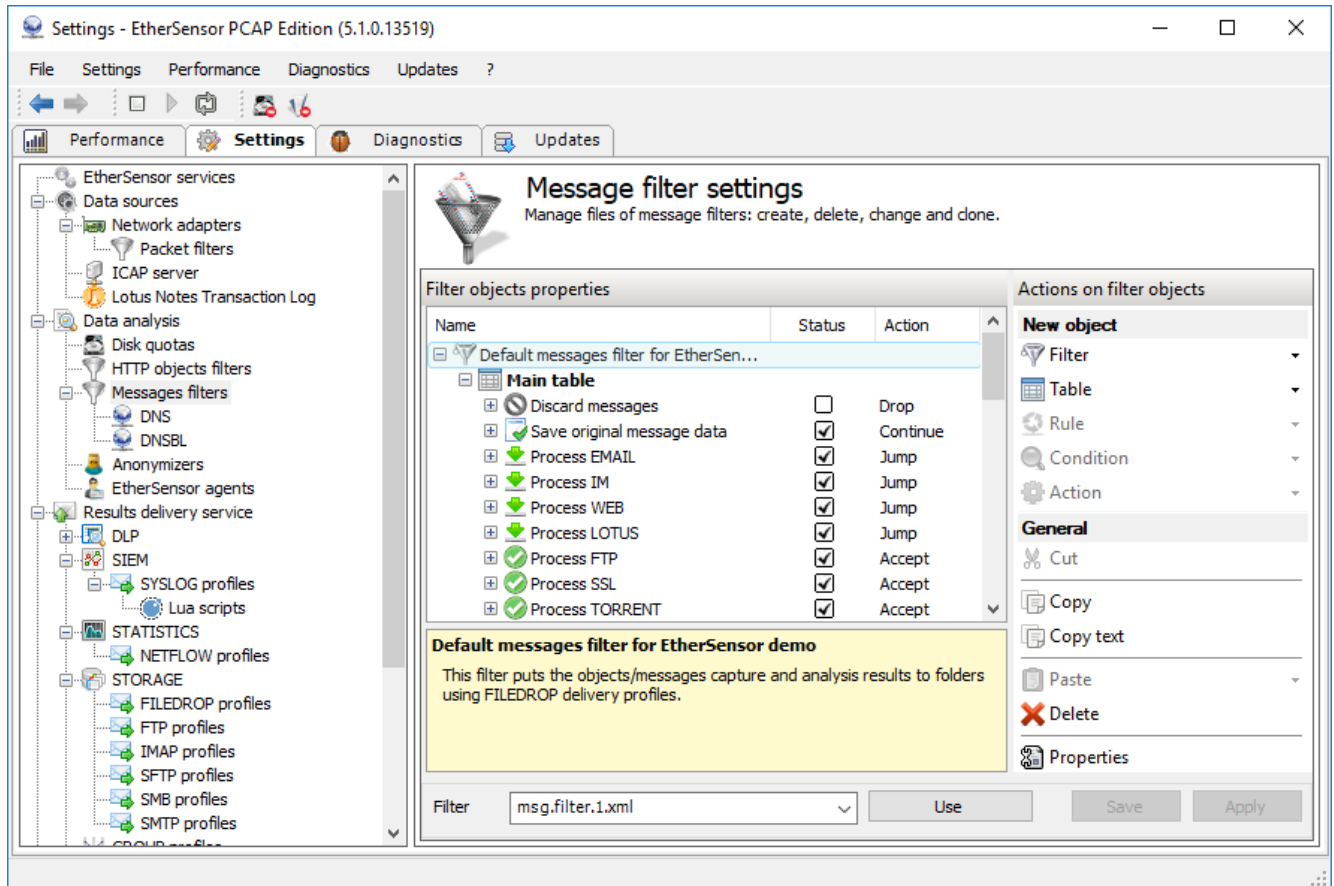


Figure 44. Message filtering management.

The filter is edited in the right window; you can also edit the active filter. In order for the **EtherSensor Analyser** service to start using a modified filter you need to make this filter active and restart the service.

Creating filters is described in more detail in the "Messages filtering" section.

DNS

You may need to use domain names in filter rules, while only IP addresses are available in the attributes of intercepted objects.

In order to be able to use this feature in real time during filtering (dynamic DHCP assignment of IP addresses should also be taken into account), you need to define and configure available DNS servers in the **DNS** section.

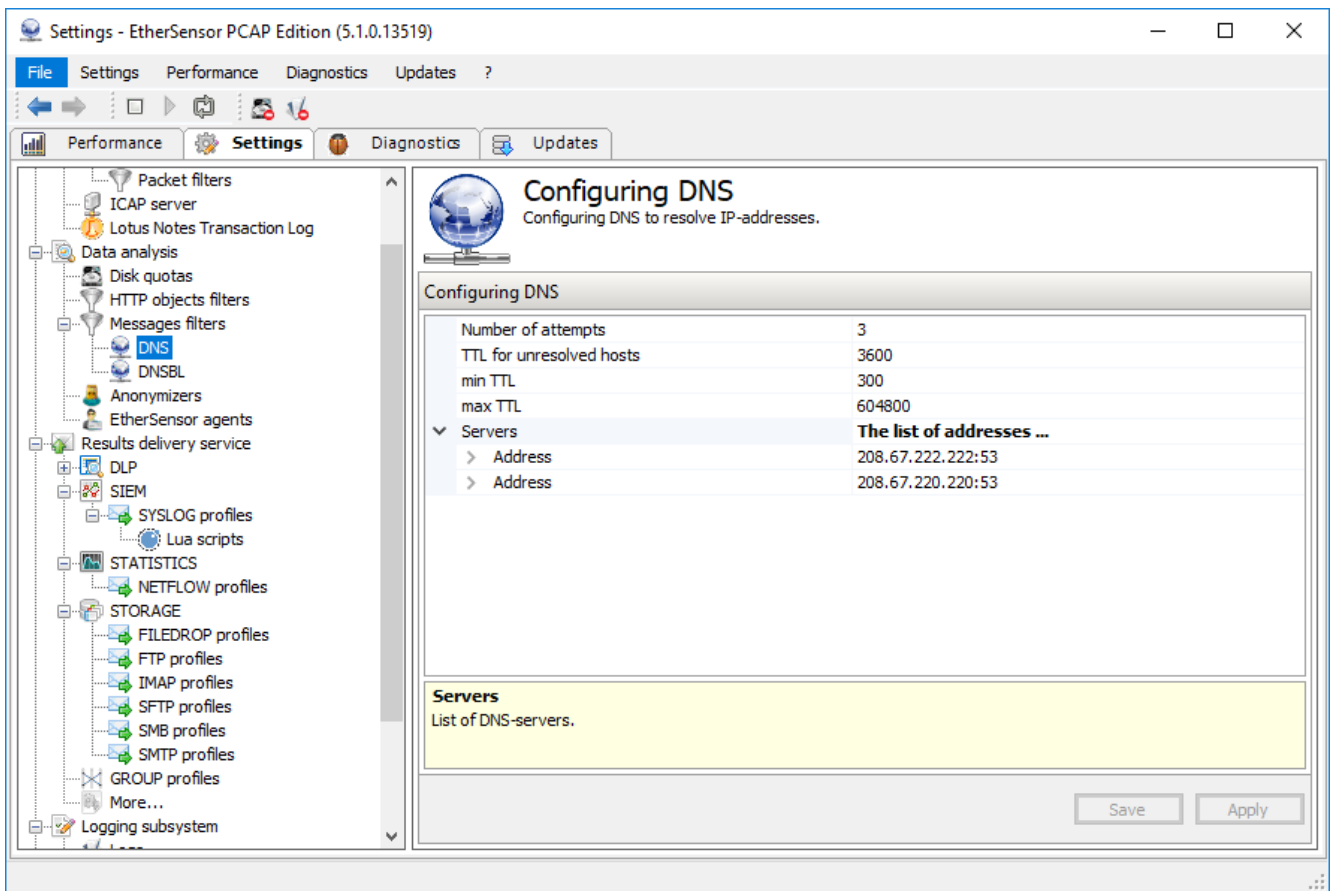


Figure 45. DNS configuration.

In order to add, delete or edit DNS server properties or change the order in which they are used, click the button to the right of **The list of addresses ...**:

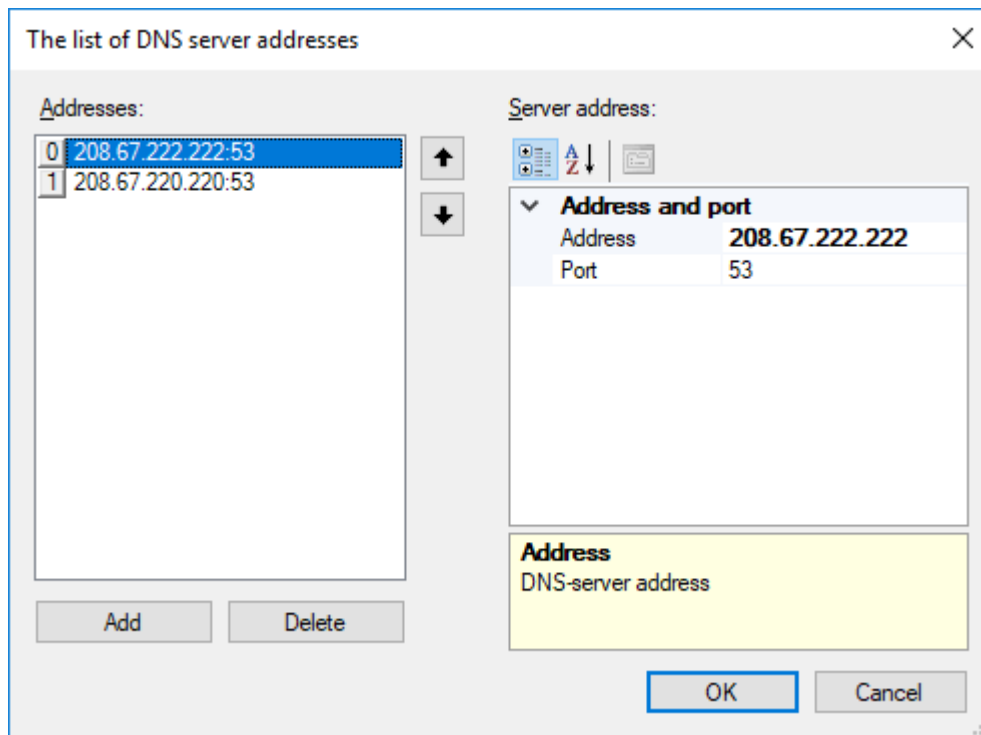


Figure 46. Detailed DNS configuration.

DNSBL

You may also need to categorize messages during filtration. For this purpose, **DeviceLock EtherSensor** supports DNSBL servers capable of detecting spam messages in the SMTP stream. This feature is configured identically to DNS server.

Anonymizers

The current version of **DeviceLock EtherSensor** (5.1.0.13519) supports monitoring the use of anonymizer service domains. You can create the list of such domains by adding each domain in a separate line or specifying a comma-separated list of domains in one line.

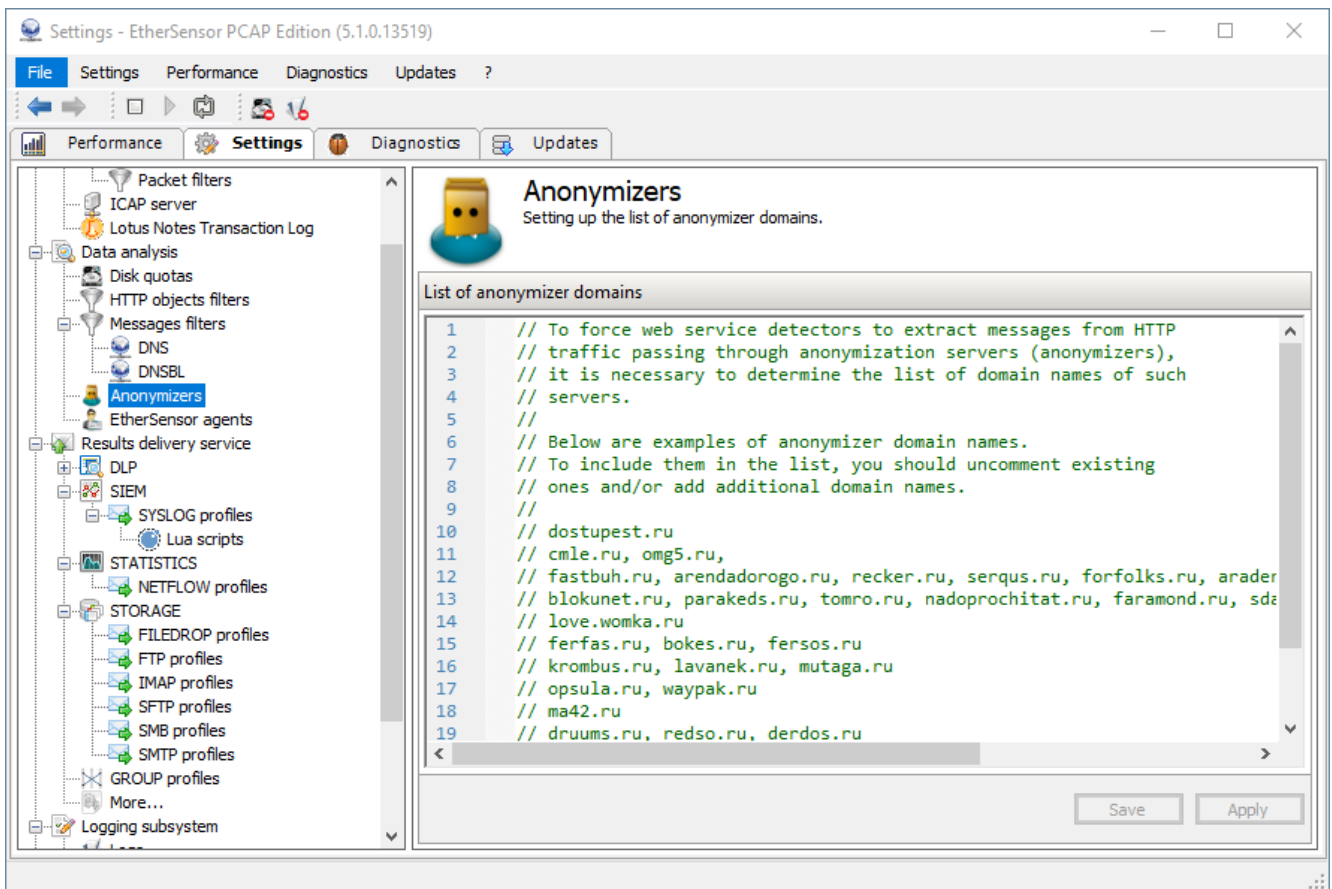


Figure 47. Setting up the list of anonymizer domains.

DeviceLock EtherSensor and EtherSensor Agent

The **EtherSensor Agent** service is used to bind TCP connections created by local processes on user workstations to their current names and IP addresses, including terminal servers.

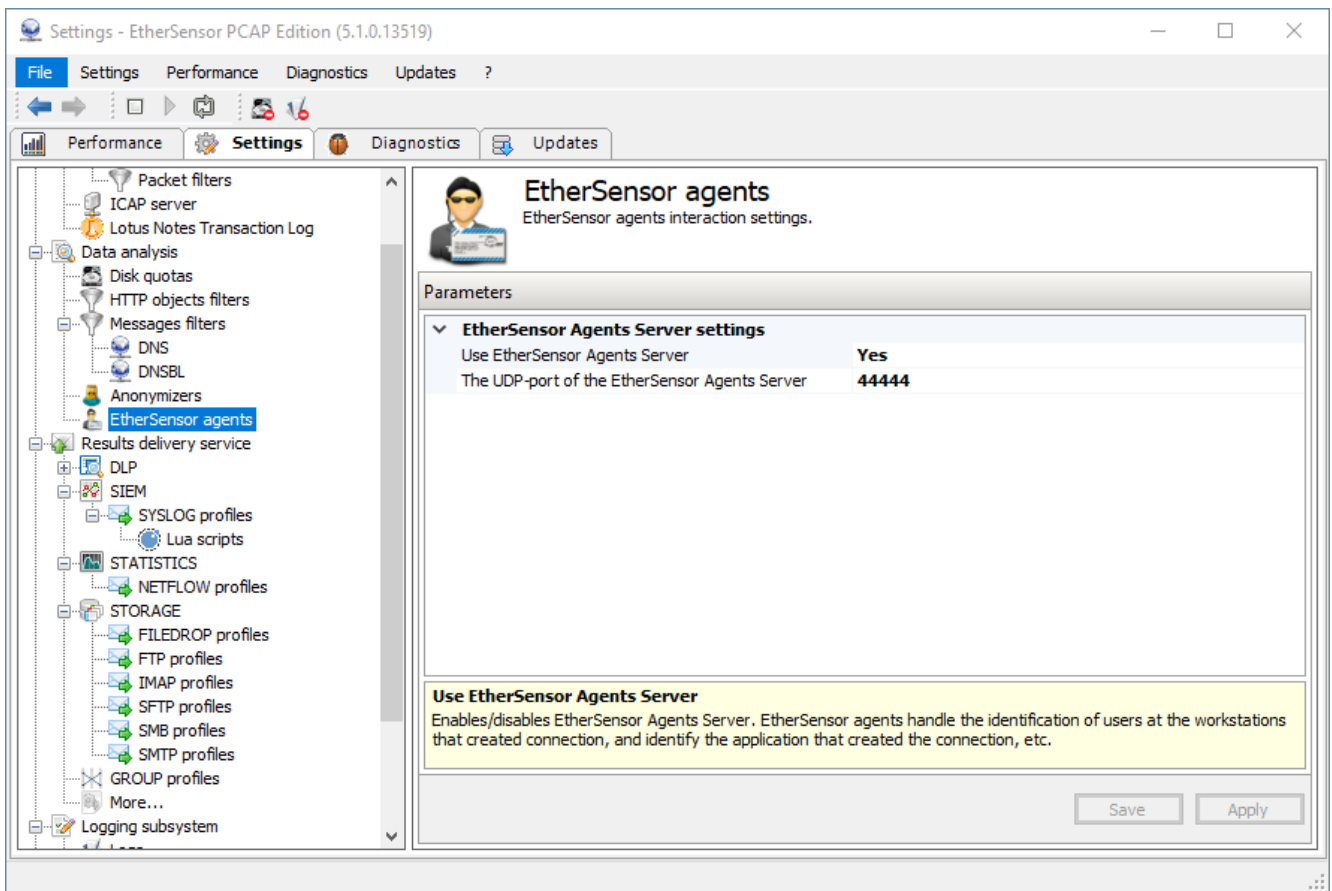


Figure 48. Setting up the interaction between DeviceLock EtherSensor and EtherSensor Agent.

4.2. Manual Setup (Config File)

The **EtherSensor Analyser** service configuration is stored in the **analyser.xml** and **quotas.xml** files in the common configuration directory - **DeviceLock EtherSensor [INSTALLDIR]\config**.

An example of the **analyser.xml** configuration file:


```
<?xml version="1.0" encoding="utf-8"?>
<AnalyserConfig version="4.5">
  <MaxFileSizeInMemory>10485760</MaxFileSizeInMemory>
  <AgentsServer enabled="false">
    <UdpPort>44444</UdpPort>
    <SessionTimeOut>10</SessionTimeOut>
  </AgentsServer>
  <RawFilter enabled="false" filename="" />

  <MessageFilter enabled="false" filename="">
    <Dns>
      <AttemptsCount>3</AttemptsCount>
      <TtlForUnknown>3600</TtlForUnknown>
      <MinTtl>300</MinTtl>
      <MaxTtl>604800</MaxTtl>
      <Server ipaddress="208.67.220.220" port="53" />
    </Dns>

    <DnsBl>
      <AttemptsCount>3</AttemptsCount>
      <TtlForUnknown>3600</TtlForUnknown>
      <MinTtl>300</MinTtl>
      <MaxTtl>604800</MaxTtl>
      <Server ipaddress="208.67.222.222" port="53" />
    </DnsBl>
  </MessageFilter>

  <Anonymous>
    <Host />
  </Anonymous>
</AnalyserConfig>
```

AnalyserConfig tag

The root tag of the service configuration. Its **version** attribute specifies the configuration version.

MaxFileSizeInMemory tag

The **MaxFileSizeInMemory** tag is nested within the **AnalyserConfig** tag and specifies the size in bytes of the cache stored in OS memory. Cache objects are OSI model application-level protocol objects received from the **EtherSensor EtherCAP** and **EtherSensor ICAP** services. The service analyzes cache object to detect messages exchanged by network participants. Use this parameter to minimize the load on the file system. All the objects in the cache with the size less than specified by this setting will be processed without the file system.

AgentsServer

The **AgentsServer** tag is nested within the **AnalyserConfig** tag and specifies the settings of the **DeviceLock EtherSensor** server communicating with the **EtherSensor Agent** instances. **EtherSensor Agent** delivers the information about connections created by network users. The **DeviceLock EtherSensor** server uses this information to identify the user when the intercepted message is being processed.

UdpPort tag

The **UdpPort** tag is nested within the **AgentsServer** tag and specifies the number of the UDP port listened by the **DeviceLock EtherSensor** server.

SessionTimeout tag

The **SessionTimeout** tag is nested within the **AgentsServer** tag and specifies the idle time of the connection being tracked. The connection will be removed from the cache of tracked connections after this period. The value of this parameter is specified in minutes and may be in the range between 1 and 59.

RawFilter tag

The **RawFilter** tag is nested within the **AnalyserConfig** tag. It specifies a description of the HTTP request filter settings. The **enabled** attribute specifies the HTTP request filter usage status. If set to **false**, the HTTP request filter is not used in the message processing. The **filename** attribute specifies the name of the filter settings file to use. The filter configuration is stored in the **.xml** file located in the **DeviceLock EtherSensor [INSTALLDIR]\config\filter\http** configuration directory.

MessageFilter tag

The **MessageFilter** tag is nested within the **AnalyserConfig** tag. It specifies a description of the messages filter settings. The **enabled** attribute specifies the messages filter usage status. If set to **false**, the messages filter is not used in the message processing. The **filename** attribute specifies the name of the filter settings file to use. The filter configuration is stored in the **.xml** file located in the **DeviceLock EtherSensor [INSTALLDIR]\config\filter** configuration directory.

Dns tag

The **Dns** tag is nested within the **Filter** tag and is used to set up the DNS module used by the filtering mechanism to resolve DNS names of network communication participants.

Example of the DNS module configuration:

```
<Dns>
  <AttemptsCount>3</AttemptsCount>
  <TtlForUnknown>3600</TtlForUnknown>
  <MinTtl>300</MinTtl>
  <MaxTtl>604800</MaxTtl>
  <Server ipaddress="208.67.220.220" port="53"/>
  <Server ipaddress="208.67.222.222" port="53"/>
</Dns>
```

DNS module properties

AttemptsCount

Specifies the number of attempts to query the DNS name of the host. Example: **3**.

TtlForUnknown

Specifies the time (in seconds) of storage of an unresolved DNS name in the local DNS cache if the host DNS name was not resolved within the specified number of attempts. Example: **3600**.

MinTtl

Specifies the minimum time (in seconds) of storage of a DNS name record in the local DNS cache. Example: **300**.

MaxTtl

Specifies the maximum time (in seconds) of storage of a DNS name record in the local DNS cache. Example: **604800**.

Server

Specifies the DNS server address and port. Example: **ipaddress="208.67.222.222" port="53"**.

DnsBl tag

The **DnsBl** tag is nested within the **Filter** tag and is used to set up the **DnsBl** module, which in turn is used by the filtering mechanism to get the information about network communication participants from the DNSBL (DNS blacklist) service.

Example of the DNSBL module configuration:

```
<DnsBl>
  <AttemptsCount>3</AttemptsCount>
  <TtlForUnknown>3600</TtlForUnknown>
  <MinTtl>300</MinTtl>
  <MaxTtl>604800</MaxTtl>
  <Server ipaddress="192.68.0.20" port="53"/>
  <Server ipaddress="192.68.0.21" port="53"/>
</DnsBl>
```

DNSBL module properties

AttemptsCount

Specifies the number of attempts to query the DNSBL service. Example: **3**.

TtlForUnknown

Specifies the time (in seconds) of storage of an "undefined" response from the DNSBL service in the local DNS cache if the DNSQL service failed to respond to the query within the specified number of attempts. Example: **3600**.

MinTtl

Specifies the minimum time (in seconds) of storage of the DNSBL service response in the local DNSBL cache. Example: **300**.

MaxTtl

Specifies the maximum time (in seconds) of storage of the DNSBL service response in the local DNSBL cache. Example: **604800**.

Server

Specifies the address and port of the DNSBL service. Example: **"192.68.0.20" port="53"**.

The Anonymous

The **Anonymous** tag is nested within the **AnalyserConfig** tag and specifies the settings of the list of anonymizer domains.

Host tag

The **Host** tag is nested within the **Anonymous** tag and contains the comma-separated list of names of tracked anonymizer server domains.

Example of the **Anonymous** module configuration:

```
<Anonymous>
  <Host>dostupest.ru</Host>
  <Host>fastbuh.ru, arendadorogo.ru, recker.ru</Host>
</Anonymous>
```

An example of the **quotas.xml** configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<QuotasConfig version="1.0" enabled="true">
  <NotProcessed>0</NotProcessed>
  <PreParseError>10</PreParseError>
  <RawFiltered>0</RawFiltered>
  <RawFilterError>10</RawFilterError>
  <ParseError>10</ParseError>
  <UnknownObject>10</UnknownObject>
  <DetectOk>-1</DetectOk>
  <DetectUnknown>10</DetectUnknown>
  <DetectError>10</DetectError>
  <DetectNoObjects>0</DetectNoObjects>
  <DetectFiltered>0</DetectFiltered>
  <MessageFilterError>10</MessageFilterError>
  <SendError>1000</SendError>
</QuotasConfig>
```

QuotasConfig tag

The root tag of the disk quota configuration. Its **version** attribute specifies the configuration version. The **enabled** attribute specifies the active status of the disk quota mechanism. If this attribute is **false**, disk quotas are not taken into account during message processing.

PreParseError tag

The **PreParseError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of processing results with errors related to the parsing of corresponding Internet protocol data. Data are prefiltered before the RAW filter is applied.

The **RawFiltered** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of results rejected by the RAW filter. This parameter is primarily used to debug (test) the logic of the RAW filter.

The **RawFilterError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects for which an error occurred during filtering.

ParseError tag

The **ParseError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of processing results with errors related to the parsing of corresponding Internet protocol data. TCP session data of such result were intercepted successfully, but an error occurred during parsing, usually caused by protocol violations. For example, a browser add-on may set the **00** end-of-line byte in the POST request instead of **0D0A**.

UnknownObject tag

The **UnknownObject** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects with unrecognized format. TCP session data have been reconstructed successfully, but there is no detector for this message type in the current version of **DeviceLock EtherSensor** (5.1.0.13519).

DetectOk tag

The **DetectOk** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of successfully recognized objects. TCP session data have been intercepted successfully, the message is recognized and passed to the delivery service.

DetectUnknown tag

The **DetectUnknown** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of recognized objects with unknown data. TCP session data have been intercepted successfully, the corresponding detector for this message type has been triggered, but it is unable to extract all the data - most likely the message format has changed.

DetectError tag

The **DetectError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects with data which trigger message detection errors.

DetectNoObjects tag

The **DetectNoObjects** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects which were recognized successfully but do not contain the required data for further processing.

DetectFiltered tag

The **DetectFiltered** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects which initially carry no useful information, since these are service data transmitted during the interaction of the user with the Internet service.

MessageFilterError tag

The **MessageFilterError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of objects for which an error occurred during message filtering.

SendError tag

The **SendError** tag is nested within the **Quotas** tag and specifies the disk quota for the storage of intercepted messages for which an unexpected error occurred when an attempt was made to send them to an external consumer.

4.3. Messages Created

DeviceLock EtherSensor generates messages from application-level objects (messages) extracted from the traffic. **DeviceLock EtherSensor** delivers such messages for further processing to consumer systems via the results delivery service.

The following ways and methods are used to deliver reconstructed communications data:

1. The captured message is converted into an XML passport, which contains all known details of captured data, senders, recipients, etc. The collected data are added to the passport during the processing.
2. After the object processing is completed by **DeviceLock EtherSensor** the object is passed to the results delivery service in the form of passport and other data files for delivery.
3. Depending on the required transport type and its settings, the results delivery service uses the passport to generate the object for delivery.

For example, in case of SMTP delivery, an email message is generated where the sender and the recipient addresses (FROM, TO, CC, BCC) are populated with captured data for the source and destination of the message (if such data is available). The text of the message is used as the text of the email message, and transmitted files are represented as attachments to the message. Other data collected by **DeviceLock EtherSensor** during the message processing are saved as MIME headers of the message.

Message format for the delivery of results

Messages transmitted by the results delivery service should meet the RFC requirements to SMTP messages. Long headers may be split and encoded; UTF-8 is used as the primary encoding. Object types are determined based on processed protocol headers.

Please note:

1. For certain web services, multiple messages may be captured for a single user operation in the user interface: this is how **DeviceLock EtherSensor** processes drafts and attachments sent to the service.
2. Sometimes the text and the attachment may be delivered as separate messages.
3. Sometimes a single container with the contents of several messages may be delivered (this mainly happens in case of IM protocols to minimize traffic and related overheads).

Service MIME headers of DeviceLock EtherSensor with sample values

X-Sensor-Version: 5.1.0.13519

Text string. Identifies the current version of **DeviceLock EtherSensor**.

X-Sensor-Id: sensor-01

Text string. You can use sensor ID to distinguish messages from different sensors or group them by sensors. It can be any ASCII ID, and is assigned UHID by default.

X-Sensor-Session-Id: 6612456

Integer. Internal ID of connections processed by **DeviceLock EtherSensor**. It is assigned by the message source - **EtherSensor ICAP** or **EtherSensor EtherCAP** service. It is then recorded to the **capture.log** file.

X-Sensor-Net-Interface-Id: 00-21-28-10-58-80

Text string. If the message is intercepted by the **EtherSensor EtherCAP** service, it is assigned the MAC address of the interface or **capdrop** - the virtual ID of the PCAP file parsing driver.

For the ICAP server, the ID is taken from the ICAP server configuration. You can use this header to trace the source of the message: the interface or service from which the data came for processing.

X-Sensor-Session-Level: 0

Integer. Specifies the number of protocols processed in order to reach the message. The list of protocols may include HTTP connection, HTTP proxy connection over it, and GRE encapsulation.

X-Sensor-Src-Address: 10.31.90.22:47016**X-Sensor-Dst-Address: 193.203.100.139:8080**

Connection IP addresses and ports (outgoing and destination). They may be not defined when ICAP traffic without **X-Client-IP** and **X-Server-IP** headers is processed. An example of ICAP headers: **X-Client-IP: 192.168.3.67**, **X-Server-IP: 123.45.67.89**.

X-Sensor-Src-Host: pc-test.msk.su**X-Sensor-Dst-Host: nns-team.ru**

Text strings. Host names corresponding to the **X-Sensor-Src-Address** and **X-Sensor-Dst-Address** headers. Many networks use DHCP to assign internal addresses, and that is why the host name should only be determined when the message is captured. To do so, **DeviceLock EtherSensor** sends a return DNS query to the DNS server specified in the **EtherSensor Analyser** service configuration. If the value cannot be recognized, it is set to .

X-Sensor-Protocol: HTTP

Text string. The name of the detector used to parse data. Possible options are SMTP, ICQ, MRA, etc.

X-Sensor-Detector: phpBB

Text string. The name of the **DeviceLock EtherSensor** detector which detected the presence of data in the connection.

X-Sensor-Attachments-Count: 0

Integer. The number of files in the intercepted message.

X-Sensor-Object-Date: Fri, 17 Sep 2010 17:30:24 +0400

Text string. The time when the message was intercepted (object creation date in **DeviceLock EtherSensor**). The time zone is based on the sensor OS settings.

X-Sensor-Object-Size: 2735

Integer. The size of the intercepted object in bytes before processing.

X-Sensor-Object-MD5Hash: c326230de58279229862b18e818a3912

Text string, the **md5** hash of the captured object. If messages contain identical text but have different size, hash, object date or source and destination addresses and ports, they are not duplicates but rather two very similar objects. Such objects normally do not have identical **md5** hash. If they do, then **DeviceLock EtherSensor** processes the same connection multiple times (a loop).

X-Sensor-Via: 1.1 off:1080 (squid/2.6.STABLE18)

X-Sensor-Forwarded-For: 10.255.241.31

Text strings. Headers from an HTTP connection; headers values are populated (if available). You can use these values to determine if the connection client is behind a proxy server, or sometimes find out the client's address at the moment of message registration.

X-Sensor-Icap-Client-Username: user1

X-Sensor-Icap-Subscriber-Id: mike.smith@mycompany.com

X-Sensor-Icap-Authenticated-User:

TERBUDovLzE5Mi4xNjguMTluMTAwL289bXljb21wYW55LCBvdT1lbmdpbmVlcmluZywgY249bWlrZS5zbWI0aA==

X-Sensor-Icap-Authenticated-Group:

TERBUDovLzE5Mi4xNjguMTluMTAwL289bXljb21wYW55LCBvdT1lbmdpbmVlcmluZw==

The values of these headers are generated when ICAP traffic is processed. Corresponding ICAP protocol headers are used for that (**X-Client-Username**, **X-Subscriber-ID**, **X-Authenticated-User**, **X-Authenticated-Groups**).

X-Sensor-Filter-Name: TEST

X-Sensor-Tags: Filtered=1

X-Sensor-Labels: filter-begin-time="2010-09-17T17:30:24.6318125+04:00",

dns-begin="2010-09-17T17:30:24.6318125+04:00",

dns-end="2010-09-17T17:30:24.6318125+04:00",

Filtered="true",

filter-end-time="2010-09-17T17:30:24.6318125+04:00"

Service headers of the **EtherSensor Analyser** service. You can use them to determine the filter that was triggered, the nature of the message content, tags and labels set for it, and track how the message is processed by the filter. Resulting headers may vary significantly depending on the filter policy.

Date: Fri, 17 Sep 2010 17:30:24 +0400

The **X-Sensor-Object-Date** value is copied to this header.

From: anonymous@nns-team.ru

To: forum@nns-team.ru

CC, BCC and other headers

Subject: Re: World of Tanks

Sender and receiver headers are populated with user addresses or IDs extracted from messages, query headers, etc. when possible. These may be not defined, and they depend on the detector: e.g. if a protocol does not set the message subject, it may be populated with the information from the sensor.

X-Sensor-RawSource-Type: LotusMail

All the messages are labeled with the "**X-Sensor-RawSource-Type**" header to specify the data source of the final message.

This header in the current version of **DeviceLock EtherSensor** (5.1.0.13519) may have the following values:

HttpGetRequest

Specifies a GET HTTP query as the primary data source.

HttpPostRequest

Specifies a POST HTTP query as the primary data source.

HttpPutRequest

Specifies a PUT HTTP query as the primary data source.

FtpFile

Specifies an FTP file as the primary data source.

SmtpeMl

Specifies an SMTP message in EML format as the primary data source.

Pop3Eml

Specifies a POP3 message in EML format as the primary data source.

IcqContactList

Specifies an ICQ contact list as the primary data source.

IcqMessageList

Specifies an ICQ message list as the primary data source.

IcqFile

Specifies a file transmitted between ICQ clients as the primary data source.

IcqLoginInfo

Specifies ICQ user details as the primary data source.

MraUserInfo

Specifies MRA user details as the primary data source.

MraContactList

Specifies an MRA contact list as the primary data source.

MraMessageList

Specifies an MRA message list as the primary data source.

MraFile

Specifies a file transmitted between MRA clients as the primary data source.

MsnContactList

Specifies an MSN contact list as the primary data source.

MsnMessageList

Specifies an MSN message list as the primary data source.

MsnFile

Specifies a file transmitted between MSN clients as the primary data source.

XmppContactList

Specifies an XMPP contact list as the primary data source.

XmppMessageList

Specifies an XMPP message list as the primary data source.

XmppFile

Specifies a file transmitted between XMPP clients as the primary data source.

IrcMessageList

Specifies an IRC message list as the primary data source.

IrcFile

Specifies a file transmitted between IRC clients as the primary data source.

SkypeVersionRequest

Specifies a latest version request to **ui.skype.com** as the primary data source.

SslSessionsList

Specifies an SSL session list as the primary data source.

LotusMail

Specifies a LOTUS protocol message list as the primary data source.

LotusAttachment

Specifies an attachment file of a LOTUS message as the primary data source.

X-Sensor-LicOption: Lotus

All the messages are labeled with the "**X-Sensor-LicOption**" header to specify the module used to process this message. This header in the current version of **DeviceLock EtherSensor** (5.1.0.13519) may have the following values:

WebMail

Indicates that the message was processed by the module with the "Web mail" licensed option.

WebSocial

Indicates that the message was processed by the module with the "Social networks" licensed option.

Email

Indicates that the message was processed by the module with the "E-mail" licensed option.

IM

Indicates that the message was processed by the module with the "Instant messages" licensed option.

FT

Indicates that the message was processed by the module with the "File transfer" licensed option.

WebMailRead

Indicates that the message was processed by the module with the "Reading incoming web mail" licensed option.

Lotus

Indicates that the message was processed by the module with the "Interception of **Lotus Notes** messages" licensed option.

LotusTxn

Indicates that the message was processed by the module with the "Extraction of messages from **Lotus Notes Transaction Log**" licensed option.

X-Sensor-UID: 0e515c8c-61eb-11e1-a529-000c29ff0707

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed on network users workstations. The header value uniquely identifies the company user on a specific computer within the company.

X-Sensor-UID-UserName: CN=Administrator,CN=Users,DC=bigbrother,DC=foo

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed at workstations of network users. The header value uniquely identifies the company user within the company.

X-Sensor-UID-UserSID: S-1-5-21-86032015-1269853868-1024056280-1001

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed at workstations of network users. The header value uniquely identifies the user within the company and indicates the security ID of the current user.

X-Sensor-UID-ComputerName: WS325-LOCK.bigbrother.foo

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed at workstations of network users. The header value uniquely identifies the computer within the company.

X-Sensor-UID-AdapterType: if_type_ethernet_csmacd

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed at workstations of network users. The header value contains the type of the network adapter used to send the message. See the complete list of possible values of this field on Microsoft website.

X-Sensor-UID-MacAddress: 00-1F-C6-2D-EA-40

This header is generated when the **DeviceLock EtherSensor** server communicates with **EtherSensor Agent** instances installed at workstations of network users. The header value contains the MAC address of the network adapter used to send the message.

X-Sensor-UHID: UO2D-RNVO-JRN7-R1EN-91C0-61TA-1HP7-YRVF

Contains a unique ID for each **DeviceLock EtherSensor** instance and equipment set (Unique Hardware Identifier).

X-Sensor-Lotus-Messageld: <OF4D026078.B21F0C4F-ON44257C15.002E1625-44257C15.002E2225@LocalDomain>

Contains the unique ID of the message transmitted over the Lotus Notes protocol.

X-Sensor-Lotus-Form: Reply

Contains the name of the form of the message transmitted over the Lotus protocol.

X-Sensor-Lotus-Mailer: Lotus Notes Release 8.5.2FP2 SHF236 October 24, 2011

Contains a string which identifies the type of the **Lotus Notes** user.

X-Sensor-Lotus-INetPrincipal: UserName/OU/O@ServerName.Domain.com

Contains extended details of the message sender.

X-Sensor-Lotus-RouteServers: CN=lotus1/O=Company

Contains the list of Lotus servers which participated in message transmission.

X-Sensor-Lotus-References: <OF02B9DAC2.33CDF581-ON44257C15.002DF8CD@LocalDomain>

Contains the list of message IDs referred by the current message.

4.4. Capture Results Filtering

Starting from version **3.0** , recognized message filter has been added to **DeviceLock EtherSensor**.

The message processing concept is based on chains of rules created and then combined into tables. The message is checked against rules which may modify its contents, metadata or processing depending on whether the message is affected by the rule.

This concept is very similar to filter rules used in the **iptables** utility.

Using filters allows you to manage message processing, send messages to various tools for further analysis and delete obviously unnecessary messages to lower the load on **DeviceLock EtherSensor** and the runtime environment.

4.4.1. Filtration Basics

The key concepts of filtration are:

Match

A logical expression which consists of one or more **conditions** connected with boolean operators: **OR, AND, XOR** and **NOT**, which analyses message contents or metadata and determines whether this specific message falls under the current rule.

Condition

Atomic check condition for the message and/or its metadata. It test one or more fields of the message or its metadata consistently against a certain condition (equation, inequation, template match, etc.).

Action

Description of the action to be performed with the message if it falls under the rule. Possible message actions are detailed in the Actions section.

Rule

Consists of conditions and actions. If conditions are fulfilled for the message, actions are applied to it. There may be no conditions at all, which implies the "all messages" or "any message" match type.

Table

An ordered sequence of rules. A table must have a unique name. There is one system table named "**main**". It is the entry point for message filtering. Each filter must contain the "**main**" table. Tables are detailed in the Tables section.

Principles of filtering

All the messages pass through filter tables starting with the "**main**" table. When a message passes through a table, all the rules from this table are applied to the message in the specified order.

Applying a rule includes:

- Checking the message (including its metadata) against conditions.
- Applying the action to the message if the condition is fulfilled.

The action may be a basic operation (a build-it action, such as **ACCEPT**, **DROP**, **JUMP** or **RETURN**) or a custom operation (setting a label or a tag, modifying the message or its metadata, etc.). Basic operations may be either terminating, i.e. they stop any further message filtering (these are **ACCEPT**, **DROP**), or non-terminating, i.e. they do not stop message processing (these are **JUMP** and **RETURN**).

The "**main**" table must always end with a rule that contains the "**all messages**" match and one of the terminating actions (**ACCEPT** or **DROP**). Other tables must always end with a rule that contains the "**all messages**" match and a terminating action (**ACCEPT** or **DROP**) or the **RETURN** action.

The **JUMP** action is used to pass the message from the current table to another one. If the **RETURN** action is applied to the message in that table, the message returns to the original table and goes on through it, starting with the next rule. Table other than the "**main**" table, and jumping to them, may be required to minimize the number of rules the message has to pass through for the **ACCEPT** or **DROP** decision to be made.

It can also be used to combine rules into groups to process messages with some "global" properties in common which cannot be described with matches within a single rule.

Please note:

This means that the **main** table may not end with the **RETURN** action rule, as there is nowhere to return to from it. You cannot **JUMP** to the **main** table either.

It should be particularly noted that any explicitly or implicitly "cyclical" loops are **not allowed**. This is verified during the rule compilation. For example, the following scenario will be blocked during the filter compilation: "**main**" table -> **jump** -> "**boss-messages**" table -> "**shopping**" table -> "**spam**" table -> "**boss-messages**" table.

4.4.1.1. Filter Configuration

Filter settings are stored as an XML file with a certain structure.

The configuration file begins with the standard XML document starting tag which specifies XML version 1.0 and document encoding.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
```

The **filter** root tag follows with the filter settings. The root tag has the following attributes: **name** - short name of the filter, and **version** - filter version. The only version currently supported is **"1.0"**.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="main filter" version="1.0">
  ...
</filter>
```

You can add an optional comment for the filter (it may be a text description of the filter's purpose). Comments may contain any text and they are ignored when the filter is applied. A comment is added as a separate **<comment>** tag.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="main filter" version="1.0">
  <comment>This is a comment.</comment>
  ...
</filter>
```

Similarly, you can specify comments for tables and rules.

4.4.1.2. Tables

A filter always contains the **"main"** table and, possibly, other tables. The filter always starts message processing with the **"main"** table. A table is defined in a filter with the **<table>** tag. Each table must have a unique name defined in the **name** attribute of the **<table>** XML tag. A table may have an optional comment in the **<comment>** XML tag.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="main filter" version="1.0">
  <comment>This is a comment.</comment>
  <table name="main">
    <comment>This is a comment for the table "main".</comment>
    ...
  </table>

  <table name="spam">
    ...
  </table>
</filter>
```

This filter has two tables: the mandatory **main** table and an optional **spam** table.

4.4.1.3. Rules

Rules consist of criteria (a match) with one or more conditions, and one or more actions which are performed if the criteria are matched.

Rules are defined within tables with the **<rule>** XML tag.

A rule can have an optional name specified in the **name** attribute of the **<rule>** XML tag.

A rule can have an optional comment specified with the **<comment>** XML tag.

A rule can be enabled (it will be applied in the current configuration) or disabled (it will be ignored when the current configuration is applied). The active status of the rule is defined by the mandatory **enabled** attribute of the **<rule>** XML tag. The **enabled** attribute can have the following values:

1 or **true**:

The rule is enabled and participates in messages filtering.

0 or **false**:

The rule is disabled and does not participate in messages filtering (is ignored).

Rule criteria are defined by the **<match>** XML tag inside the rule. Rule actions are defined by the sequence of **<action>** XML tags inside the rule.

For example:


```
<?xml version="1.0" encoding="utf-8"?>
<filter name="main filter" version="1.0">
  <comment>This is a comment.</comment>

  <table name="main">
    <comment>This is a comment for the table "main".</comment>
    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule name="spam" enabled="1">
      <comment>The rule for the messages of the SPAM category.</comment>
      <match ...> ... </match>
      <action ...> ... </action>
      <action ...> ... </action>
      <action ...> ... </action>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>

  </table>
</filter>
```

In this example, the **"main"** table contains three rules, all of them enabled. The second rule (unlike the first one) has a name, a comment and several actions. The third rule is the mandatory terminating rule which rejects all the messages not accepted by the rules above it.

If the rule contains no **<match>** criteria or the match is empty (**<match />**), the **"all messages"** match is implied: **<c name="all"/>**.

4.4.1.3.1. Criteria and Conditions

Criteria (**match**) of a rule are set with the **<match>** XML tag and define whether the rule's action is to be performed for this message.

The match of the rule consists of one or more **conditions** interconnected with boolean operations **AND**, **OR**, **XOR** or **NOT**. This allows you to create criteria based on logical expressions which consist of conditions.

Example:

```
<rule>
  <match>
    <c name="all" />
  </match>
  <action name="drop" />
</rule>
```

This match consists of a single **"all messages"** condition.

Conditions in criteria

A **condition** is an atomic check of a message or its metadata. A condition is defined with the **<condition>** or **<c>** XML tag. A condition is either fulfilled for the message and is considered **TRUE** or is not fulfilled and thus considered **FALSE**.

A **condition** has its name specified in the **name** attribute which defines what exactly is to be checked for the message. Other tag attributes specify action parameters. Attribute names for additional parameters depend on the condition.

The general structure of the XML tag of a condition:

```
<condition name="The name of the condition." [additional parameters] />
```

or

```
<condition name="The name of the condition." [additional parameters] >  
</condition>
```

or

```
<c name="The name of the condition." [additional parameters] />
```

or

```
<c name="The name of the condition." [additional parameters] ></c>
```

Most **conditions** use the **value="..."** attribute which specifies the values to check in the condition, or the **data="..."**, attribute which may specify an external file for the check to load data from (word sets, domain name lists or other parameters). Specific attributes and how they are checked are described in sections for corresponding **conditions**.

Logical expressions in matches

When you need to create a rule with a complex match containing multiple conditions, these conditions can be combined into logical expressions using the **AND**, **OR**, **XOR**, **NOT** boolean operations. Logical operations in XML tags are specified as follows:

<and> ... </and>:

Corresponds to (... & ... & ... & ...) - all the conditions inside the **<and>** tag are connected with the **AND** boolean operation.

<or> ... </or>:

Corresponds to (... | ... | ... | ...) - all the conditions inside the **<or>** tag are connected with the **OR** boolean operation.

<xor> ... </xor>:

Corresponds to (... ^ ... ^ ... ^ ...) - all the conditions inside the <xor> tag are connected with the **XOR** boolean operation.

<not> ... </not>:

Corresponds to **!(...)** - the negation operation is applied to the condition.

For example:

```
<and>
  <c name="ccc1"/>
  <c name="ccc2"/>
</and>
```

means **(ccc1 & ccc2)**,

and the following match:

```
<not><c name="ccc1"/></not>
```

means **not (ccc1)**.

Any logical operation tag can be nested.

For example:

```
<and>
  <c name="ccc1"/>
  <or>
    <c name="ccc2">
    <c name="ccc3">
  </or>
  <or>
    <c name="ccc4">
    <c name="ccc5">
    <not><c name="ccc6"></not>
  </or>
</and>
```

means **(ccc1 & (ccc2 | ccc3) & (ccc4 | ccc5 | !ccc6))**. That is, the criteria from this example are met for the message if: is true for **ccc1**, and is true (for **ccc2** or **ccc3**), and (is true (for **ccc4** or **ccc5**) or is not true for **ccc6**).

For clarity, we can split this match into condition blocks, which should return **TRUE** for the following checks:

1) The **ccc1** condition must be true for the message

AND

2) One of the (**ccc2** or **ccc3**) conditions must be true for the message

AND

3) One of the (**ccc4** or **ccc5**) conditions must be true for the message OR the **ccc6** condition must be false for the message.

4.4.1.3.1.1. ALL, * Condition

A special condition which is true for all the filtered objects.

Description

The condition is true for any object. It is implied if the `<match>...</match>` match is empty or is missing from the rule (the rule only contains `<actions>`).

Format

```
<c name="all" />
<c name="*" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="all"** or **name="*"**.

Example

The rule accepts all messages for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>

  <table name="main">

    <rule enabled="1">
      <comment>This rule accepts all messages for further processing.</comment>
      <match>
        <c name="all" />
      </match>
      <action name="accept" />
    </rule>
  </table>

</filter>
```

Example (implicit "all" condition)

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter comment.</comment>
  <table name="main">
    <rule enabled="1">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.2. DETECTOR Condition

Sets the name of the detector which identified the message.

Description

When a detector is triggered, the system stores its name in the metadata of the message. The DETECTOR condition can be used during message processing to determine the detector triggered by the message (there can be only one).

Format

```
<c name="detector" value="[detector-name(s)]" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="detector"**.

The "value" attribute:

The **value="..."** attribute specifies the name to compare to the name of the detector triggered by the message. There may be several names separated with a comma ','. You can place spaces after commas for readability purposes.

Example:

Messages from mail.ru, yandex.ru detectors.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Messages filter.</comment>
  <table name="main">
    <rule enabled="1">
      <comment>
        Messages from detectors mail.ru,
        yandex.ru.
      </comment>
      <match>
        <c name="detector" value="mail.ru, yandex.ru" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.3. PROTOCOL Condition

Checks which protocol over which the message was received.

Description

This condition checks which protocol over which the message was received.

Format

```
<c name="protocol" value="[protocol-name(s)]" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="protocol"**.

The "value" attribute:

The **value="..."** attribute specifies the name to compare to the protocol name over which the message was received.

There may be multiple protocols to be checked, separated with a comma ','. You can place spaces after commas for readability purposes.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Messages filter.</comment>
  <table name="main">
    <rule enabled="1">
      <comment>
        Drop messages received by SMTP or IMAP.
      </comment>
      <match>
        <c name="protocol" value="smtp, imap" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.4. MSG-SIZE, TOTAL-SIZE Condition

Checks the size of the message.

Description

This condition checks the size of the message.

The **msg-size** condition sums up the size of the extracted text and attachments.

The **total-size** condition sums up the total size of the extracted text, attachments and source data from which they were extracted.

Format

```
<c name="msg-size" op="<operation>" value="<compare pattern>" />  
<c name="total-size" op="<operation>" value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute contains the condition name: **name="msg-size"** or **name="total-size"**.

The "value" attribute:

The **value="..."** attribute specifies the number to which the message size is compared:

<number> or <number>B

Specifies the size in bytes.

<number>K

Specifies the size in kilobytes.

<number>M

Specifies the size in megabytes.

<number>G

Specifies the size in gigabytes.

The "op" attribute:

The **op="..."** attribute specifies the comparison operation and may have the following values:

"eq" or = or ==

The condition is considered true if the size **IS EQUAL TO** the specified number.

"ne" or != or <>

The condition is considered true if the size **IS NOT EQUAL TO** the specified number.

"lt" or <

The condition is considered true if the size **IS LESS THAN** the specified number.

"gt" or >

The condition is considered true if the size **IS GREATER THAN** the specified number.

"le" or <=

The condition is considered true if the size **IS LESS THAN OR IS EQUAL TO** the specified number.

"ge" or >=

The condition is considered true if the size **IS GREATER THAN OR EQUAL TO** the specified number.

Example:

The rule stops processing messages larger than 100 kilobytes (net of source data) or larger than 1 megabyte (including source data).

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Messages filter.</comment>

  <table name="main">

    <rule enabled="1">
      <match>
        <or>
          <c name="msg-size" op=">" value="100K"/>
          <c name="total-size" op="gt" value="1M"/>
        </or>
      </match>
      <action name="drop" />
    </rule>

  </table>
</filter>
```

4.4.1.3.1.5. CHECK-MD5 Condition

Checks if a message MD5 hash appears twice within a defined period of time (tracking duplicate messages).

Description

This condition checks if a message with the same MD5 hash has already occurred (during the specified period of time).

Format

```
<c name="check-md5" time="<timeout>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="check-md5"**.

The "time" attribute:

The **time="..."** attribute specifies the timeout in milliseconds.

This timeout should be not less than **1** millisecond and not more than **5** minutes, which is $5*60*1,000=300,000$ milliseconds.

Example:

Delete duplicate messages with the same MD5 if they arrive within 2 seconds.


```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="check-md5" time="2000" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.6. CHECK-MESSAGE-ID Condition

Checks the **Message-ID** (for LOTUS protocol, the **X-Sensor-Lotus-MessageId**) message header for repeated occurrence within the defined period of time (tracking duplicate messages).

Description

This condition checks if a message with the same **Message-ID** (for Lotus Notes, the **X-Sensor-Lotus-MessageId** header) has already occurred within the specified period of time.

Format

```
<c name="check-message-id" time="<timeout>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="check-message-id"**.

The "time" attribute:

The **time="..."** attribute specifies the timeout in milliseconds.

This timeout should be not less than **1** millisecond and not more than **5** minutes, which is $5*60*1,000=300,000$ milliseconds.

Example:

Delete duplicate messages with the same Message-ID header if they arrive within 2 seconds.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="check-message-id" time="2000" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.7. HOSTNAME Condition

Checks the message source and destination host names.

Description

This condition checks if the source or destination host names match a string, a pattern or a regular expression. For this condition to work correctly, you should first perform host name resolution (see "DNS Action").

Tip:

If you need to check only the destination host only for HTTP, there is no need to resolve names using the DNS action, because the destination host name is available from the HTTP request "Host" header.

Format

```
<c name="hostname"
  address="<address type>"
  op="<operation>"
  value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="hostname"**.

The "address" attribute:

The **address="..."** attribute specifies the type of the address to check. Possible values:

"src" or "client"

Only check the source address

"dst" or "server"

Only check the destination address

"both", "all" or *

Check both source and destination addresses

If this attribute is omitted, **"both"** is used by default. i.e. both source and destination addresses are checked.

The "op" attribute:

The **op="..."** attribute specifies the type of the comparison operation and may have the following values:

"eq", = or ==

The condition is considered true if the value being checked **CONTAINS** the specified value

"ne", != or <>

The condition is considered true if the value being checked **DOES NOT CONTAIN** the specified value

"wc" or "wildcard"

The condition is considered true if the value being checked **matches** the specified **wildcard** pattern

"re", "regex" or "regexp"

The condition is considered true if the value being checked **matches** the specified **regular expression**

The "value" attribute:

The **value="..."** attribute specifies a string or a pattern to match the value.

Example:

Drop messages sent to *.yandex.ru.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="hostname" address="server" op="wc" value="*.yandex.ru" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.8. IP Condition

Checks if client or server IP addresses belong to a range or a subnet.

Description

This condition checks if client or server IP addresses belong to a range or a subnet.

Please note:

1. Unwanted traffic should be isolated as early as possible. This affects the performance of **DeviceLock EtherSensor** and the runtime environment.
2. It is recommended to isolate all traffic from an IP address in the ethercap service IP filter.
3. It is recommended to isolate certain HTTP traffic from an IP address (if it is possible to specify such criteria) in the HTTP filter.
4. Certain messages from an IP address should be processed in the message filter.

Format

```
<c name="ip" address="<address type>" value="<ip-range>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="ip"**.

The "address" attribute:

The **address="..."** attribute specifies the type of the address to check. Possible values:

"src" or "client"

Check the source address

"dst" or "server"

Check the destination address

"any" or *

Matches any address.

If the attribute is omitted, "*" is used by default.

The "value" attribute:

The **value="..."** attribute specifies the value for the comparison. Possible values:

ipaddress

Checks the IP address for equality. For example, **value="192.168.0.10"**

ip1-ip2

Checks if the IP address belongs to a certain range. For example, **value="192.168.0.1-192.168.0.10"**

ip/netmask

Checks if the IP address belongs to the specified subnet. For example,
value="192.168.0.1/255.255.255.0"

ip/netmaskbits

Checks if the IP address belongs to the specified subnet. For example,
value="192.168.0.1/24"

Example:

Drop messages from 192.168.0.15.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">

    <rule enabled="1">
      <match>
        <c name="ip" address="client" value="192.168.0.15" />
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.1.9. HEADER Condition

Checks the value of one of the message headers.

Description

This condition checks if the value of the field contains a substring or matches a pattern or a regular expression. Message headers are any metadata headers generated by detectors; they look like "**X-Sensor-...**".

These are also mail message headers except for the following:

From

To

Cc

Bcc

Subject

Date

Content-Type

Content-Transfer-Encoding

Please note that for more efficient filtering you should avoid filtering the "**X-Sensor-...**" metadata headers if any special conditions are pre-defined for these headers. For example, for the "**X-Sensor-Detector**" header instead of checking the header value using **header** condition, it is more efficient to use the special condition.

Format

```
<c name="header"  
  headername="<header name>"  
  op="<operation>"  
  value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="header"**.

The "headername" attribute:

The **headername="..."** attribute specifies the name of the header to check.

The "op" attribute:

The **op="..."** attribute specifies the type of the comparison operation and may have the following values:

"eq", = or ==

The condition is considered true if the value being checked **CONTAINS** the specified value

"ne", != or <>

The condition is considered true if the value being checked **DOES NOT CONTAIN** the specified value

"wc" or "wildcard"

The condition is considered true if the value being checked **matches** the specified **wildcard** pattern

"re", "regex" or "regexp"

The condition is considered true if the value being checked **matches** the specified **regular expression**

The "value" attribute:

The **value="..."** attribute specifies a string or a pattern to match the value.

Examples:

```
<c name="header" headername="X-Priority" op="eq" value="3" />
```

The condition is considered true if a message has the **X-Priority** header, and its value contains **"3"**.

```
<c name="header" headername="X-Mailer" op!=" value="Outlook" />
```

The condition is considered true if a message has the **X-Mailer** header, and its value does not contain **"Outlook"**.

```
<c name="header"
  headername="X-Sensor-Net-Interface-Id"
  op="eq"
  value="01-icap" />
```

The condition is considered true if a message has the **X-Sensor-Net-Interface-Id** header, and its value contains **"01-icap"**.

```
<c name="header"
  headername="X-Sensor-Net-Interface-Id"
  op="wc"
  value="*-icap" />
```

or

```
<c name="header"
  headername="X-Sensor-Net-Interface-Id"
  op="wildcard"
  value="*-icap" />
```

The condition is considered true if a message has the **X-Sensor-Net-Interface-Id** header, and its value matches **"*-icap"** pattern.

Example:

Drop messages sent via Outlook.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="header"
          headername="X-Mailer"
          op=="="
          value="Outlook" />
      </match>
      <action name="drop" />
    </rule>

  </table>
</filter>
```

4.4.1.3.1.10. ATTACH-NAME Condition

Checks the message attachment names.

Description

This condition checks if any attachment name matches a pattern or a regular expression.

Format

```
<c name="attach-name" op="<operation>" value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="attach-name"**.

The "op" attribute:

The **op="..."** attribute specifies the type of the comparison operation and may have the following values:

"eq", = or ==

The condition is considered true if the value being checked **CONTAINS** the specified value

"ne", != or <>

The condition is considered true if the value being checked **DOES NOT CONTAIN** the specified value

"wc" or "wildcard"

The condition is considered true if the value being checked **matches** the specified **wildcard** pattern

"re", "regex" or "regexp"

The condition is considered true if the value being checked **matches** the specified **regular expression**

The "value" attribute:

The **value="..."** attribute specifies a string or a pattern to match the value.

Example:

```
<c name="attach-name" op="eq" value="instruction.doc" />
```

The condition is considered true if any attachment name contains **"instruction.doc"**.

```
<c name="attach-name" op="eq" value="instruction.doc" />
```


The condition is considered true if any attachment name does not contain "instruction.doc".

```
<c name="attach-name" op="wc" value="*.doc" />
```

or

```
<c name="attach-name" op="wildcard" value="*.doc" />
```

The condition is considered true if any attachment name matches the "*.doc" pattern.

```
<c name="attach-name" op="re" value=".+\.doc" />
```

or

```
<c name="attach-name" op="regexp" value=".+\.doc" />
```

The condition is considered true if any attachment name matches the ".+\.doc" regular expression.

```
<c name="attach-name" op="re" value=".+((\.doc)|(\.exe)|(\.zip))" />
```

The condition is considered true if any attachment name matches the ".+((\.doc)|(\.exe)|(\.zip))" regular expression.

Example:

Drop messages that have *.exe attachments.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="attach-name" op="re" value=".+\.exe" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.11. ATTACH-EXIST Condition

Checks if the message has an attachment.

Description

The condition is considered true if a message has any attachments.

Format

```
<c name="attach-exist" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="attach-exist"**.

Example:

Drop messages with attachments.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="attach-exist" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.12. TAG Condition

Checks if a tag is set and its counter value (see "TAG Action").

Description

This condition checks if a tag exists and the value if the tag's counter.

If the tag is missing, the condition is not met.

Format

```
<c name="tag" op="<operation>" value="<compare value>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="tag"**.

The "op" attribute:

The **op="..."** attribute specifies the check criteria:

"eq", = or ==

The condition is considered true if the counter value **IS EQUAL TO** the specified number

"ne", != or <>

The condition is considered true if the counter value **IS NOT EQUAL TO** the specified number

"lt" or <

The condition is considered true if the counter value **IS LESS THAN** the specified number

"gt" or >

The condition is considered true if the counter value **IS GREATER THAN** the specified number

"le" or <=

The condition is considered true if the counter value **IS LESS THAN OR EQUAL TO** the specified number

"ge" or >=

The condition is considered true if the counter value **IS GREATER THAN OR EQUAL TO** the specified number

exist

The condition is considered true if the tag **exists** (has been already set for this object)

If the **op** attribute is omitted, **"exist"** is used by default. For the **"exist"** operation, the **value** attribute is optional.

The "value" attribute:

The **value="..."** attribute specifies the number to compare to the tag counter value.

Example:

```
<c name="tag" tag="SPAM" op="exist" />
```

or

```
<c name="tag" tag="SPAM" />
```

The condition is considered true if the tag **"SPAM"** is set for the object.

```
<c name="tag" tag="SPAM" op="eq" value="1" />
```

The condition is considered true if the tag **"SPAM"** is set for the object, and its counter value is equal to **1**.

```
<c name="tag" tag="SPAM" op=">" value="1" />
```

The condition is considered true if the tag "**SPAM**" is set for the object, and its counter value is greater than **1**.

```
<c name="tag" tag="SPAM" op=">=" value="3" />
```

The condition is considered true if the tag "**SPAM**" is set for the object, and its counter value is greater than or equal to **3**.

Example:

Drop messages marked with the **SPAM** tag with the value greater than **1**.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match>
        <c name="tag" tag="SPAM" op=">=" value="1" />
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.1.13. FROM, TO, CC, BCC, ADDRESS, SUBJECT Condition

Checks the value of one of the following fields: **from**, **to**, **cc**, **bcc**, **subject**, **address**.

Description

This condition checks if the value of the field contains a substring or matches a pattern or a regular expression.

from

Checks the FROM (sender address) field

to

Checks the TO (recipient address) field

cc

Checks the CC field

bcc

Checks the BCC field

address

Checks all address fields (**from, to, cc, bcc**) If a match is found in any field, the condition is considered true.

subject

Checks the SUBJECT field

Format

```
<c name="from" op="<operation>" value="<compare pattern>" />
<c name="to" op="<operation>" value="<compare pattern>" />
<c name="cc" op="<operation>" value="<compare pattern>" />
<c name="bcc" op="<operation>" value="<compare pattern>" />
<c name="subject" op="<operation>" value="<compare pattern>" />
<c name="address" op="<operation>" value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="from"**, **name="to"**, **name="cc"**, **name="bcc"**, **name="subject"** or **name="address"**

The "op" attribute:

The **op="..."** attribute specifies the type of the comparison operation and may have the following values:

"eq", = or ==

The condition is considered true if the field value **CONTAINS** the specified value

"ne", != or <>

The condition is considered true if the field value **DOES NOT CONTAIN** the specified value

"wc" or "wildcard"

The condition is considered true if the field value **matches** the specified **wildcard** pattern

"re", "regex" or "regexp"

The condition is considered true if the field value **matches** the specified **regular expression**

The "value" attribute:

The **value="..."** attribute specifies a string or a pattern to match the value.

Example:

```
<c name="from" op="eq" value="xxx@mail.ru" />
```

The condition is considered true if the message **FROM** field contains "**xxx@mail.ru**".

```
<c name="to" op="!=" value="xxx@mail.ru" />
```

The condition is considered true if the message **TO** field contains "**xxx@mail.ru**".

```
<c name="cc" op="wc" value="*@mail.ru" />
```

or

```
<c name="cc" op="wildcard" value="*@mail.ru" />
```

The condition is considered true if the message **CC** field matches "***@mail.ru**" pattern.

```
<c name="address" op="re" value=".*@mail.ru" />
```

or

```
<c name="address" op="regex" value=".*@mail.ru" />
```

The condition is considered true if any address field (**FROM**, **TO**, **CC** or **BCC**) of the message matches the "**+@mail.ru**" regular expression.

```
<c name="subject" op="re" value=".*((badword1)|(badword2)|(badword3)).*" />
```

The condition is considered true if the message subject matches the "***((badword1)|(badword2)|(badword3)).***" regular expression.

In other words, the condition is considered true if the message subject contains any of **badword1**, **badword2** or **badword3**.

```
<c name="subject" op="re"
value="\+?\d([\ \-\()]\d{3}([\ \-\)]?)?([\ -])?((\d{7})|
(\d{3}([\ \-])?\d{2}([\ \-])?\d{2})|
(\d{2}([\ \-])?\d{3}([\ \-])?\d{2}))" />
```

The condition is considered true if the message subject contains a phone number(a string that matches the regular expression).

A phone number in this case may be in one of the following formats:

1234567

123 45 67

12 345 67

89031234567

8(903)1234567

8-903-123-45-67
8 903 123 45 67
+79031234567
+7(903)1234567
+7-903-123-45-67
+7 903 123 45 67

Example:

Continue processing messages from ***@mail.ru** address and drop all the rest.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">

    <rule enabled="1">
      <match>
        <c name="address" op="wildcard" value="*@mail.ru" />
      </match>
      <action name="accept" />
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>

  </table>
</filter>
```

4.4.1.3.1.14. TEXT Condition

Checks if the message text or subject contains keywords.

Description

This condition checks if the message text or subject contains certain keywords.

It looks for all occurrences of a keyword, not only whole words.

So to say, for each keyword, it tries to match the **"*keyword*"** pattern.

For example, for **"secret"** keyword, the condition will match **"secret"**, **"sECrEt"**, **"secretary"**, **"insecretory"** etc.

The check is case-insensitive.

Format

```
<c name="text" op="<operation>" value="<compare pattern>" />
<c name="text" op="<operation>" data="<data source>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="text"**.

The "op" attribute:

The **op="..."** attribute specifies the check criteria:

all

Looks for messages that have all specified tags.

one

Looks for messages that has at least one of the specified tags.

If the **op** attribute is omitted, **"one"** is used by default.

The "value" attribute:

The **value="..."** attribute lists the keywords. Use comma (",") to separate multiple keywords. Keywords may also be specified in the **key-word-list** tag value

The "data" attribute:

The **data="..."** attribute may specify another source of keywords. You can use it to specify long keyword lists instead of specifying them in the **value="..."** attribute.

Possible values:

data="<extern data name>"

Load the keyword list from an external block in the filter (the ...) tag.

data="extern://<extern data name>"

Load the keyword list from an external block in the filter (the ...) tag. Use commas to separate keywords.

data="file://<full-file-path>"

Load the keyword list from a file. Provide each keyword in a new line (no commas are required).

Example:

```
<c name="text" op="one" value="secret1, secret2, secret3, secret4" />
```

or

```
<c name="text" op="one">secret1, secret2, secret3, secret4</c>
```


The condition is considered true if the message subject or text contains at least one of the specified keywords: either separately or as parts of other words.

Text examples:

1. Message subject: "**RE: secret 1 secret2**" — the condition is TRUE.
2. Message subject: "**RE: secret3 secret4**" — the condition is TRUE.
3. Message text: "**I'm sending our secret 1 secret2**" — the condition is TRUE.
4. Message subject: "**The quick brown fox jumps**" — the condition is FALSE.

```
<c name="text" op="all" value="secret1, secret2" />
```

or

```
<c name="text" op="all">secret1, secret2</c>
```

The condition is considered true if the message subject or text contains both specified keywords either separately or as parts of other words.

Text examples:

1. Message subject: "**RE: secret1 secret2**" — the condition is TRUE.
2. Message subject: "**RE: secret1 from the accounting**" — the condition is FALSE.
3. Message text: "**I'm sending our secret2**" — the condition is FALSE.
4. Message text: "**I'm sending our secret1 from secret2**" — the condition is TRUE.

```
<c name="text" op="one" data="dictionary.txt" />
```

Load the keyword list from the "**dictionary.txt**" file.

Example:

Accept messages with keywords from the **keywords** list for further processing. Drop all other messages.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">

    <rule enabled="1">
      <comment></comment>
      <match>
        <c name="text" data="extern://keywords" />
      </match>
      <action name="accept" />
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>

  </table>

  <data name="keywords">
    secret1,
    secret2,
    secret3,
    secret4
  </data>
</filter>
```

4.4.1.3.2. Actions

All the **actions** are described in XML tags **<action>**. For each action, the **name** attribute specifies its name which determines the operation to be performed with the message. Other tag attributes specify action parameters. Attribute names for additional parameters depend on the action.

The general structure of the XML action tag is the following:

```
<action name="action name" [parameters] />
```

or

```
<action name="action name" [parameters]></action>
```

Most **actions** use the **value="..."** attribute which specifies the values required to perform the action, or the **data="..."** attribute which may specify an external file for the action to load data from (word sets, domain name lists or other parameters). Attributes and their effects are detailed in descriptions of corresponding **actions**.

Basic actions

Basic actions affect how the message passes through the filter. Basic actions include **ACCEPT**, **DROP**, **JUMP** and **RETURN**.

User actions

User actions modify the message or its metadata, or pass the message through external services (antivirus software, URL classifiers, DNS blacklists, etc.).

4.4.1.3.2.1. ACCEPT Action

Pass the message for further processing.

Description

The **ACCEPT** stops filtering for the current message and immediately passes it for further processing (i.e. delivery based on a predefined or the default delivery profile).

Format

```
<action name="accept" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="accept"**.

Example:

All messages that reach this rule are accepted for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule enabled="1">
      <comment>
        All the messages reaching this rule are accepted for
        further processing.
      </comment>
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.1.3.2.2. DROP Action

Discard the message without further processing.

Description

This action stops processing of the message and instructs **DeviceLock EtherSensor** to destroy ("drop") all data stored about it.

Format

```
<action name="drop" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="drop"**.

Example:

All data and metadata for messages that reach this rule will be destroyed ("dropped").

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Details/metadata of any message reaching this point
        are destroyed (discarded).
      </comment>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.1.3.2.3. JUMP Action

Continue message processing in another table.

Description

This actions continues processing of the message in another table.

Format

```
<action name="jump" value="<table name=" />
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="jump"**.

The "value" attribute:

The **value="..."** attribute specifies the name of the table to jump to for further processing of the message.

Please note:

Remember that jumps to the **main** table are prohibited.

Jumps which can result in an explicit or implicit recursion are also prohibited.

Example:

From the **main** table, jump to the **yandex** table to process messages from mail.ru and yandex.ru. The **yandex** table: the only rule destroys all messages larger than 100 KB. After that, jump back to the **main** table.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Processing of messages from the mail.ru,
        yandex.ru detectors in the yandex table.
      </comment>
      <match>
        <c name="detector" value="mail.ru, yandex.ru" />
      </match>
      <action name="jump" value="yandex"/>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>
  </table>

  <table name="yandex">
    <comment>
      The table processes messages from the mail.ru,
      yandex.ru detectors.
    </comment>

    <rule enabled="1">
      <comment>
        Discard messages larger than 100 Kb.
      </comment>
      <match>
        <c name="size" op=">" value="100K"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <comment>
        Returning to the main table for further processing.
      </comment>
      <action name="return" />
    </rule>
  </table>
</filter>
```

4.4.1.3.2.4. RETURN Action

Returns to the previous (caller) table and continues processing with the next rule.

Description

This action returns message processing to the previous (caller) table and continues processing with the next rule.

Format

```
<action name="return" />
```

The "name" attribute:

The name attribute contains the action name: **name="return"**.

Please note:

You cannot use this action in the **main** table.

Examples:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Processing of messages from the mail.ru,
        yandex.ru detectors in the yandex table.
      </comment>
      <match>
        <c name="detector" value="mail.ru, yandex.ru" />
      </match>
      <action name="jump" value="yandex"/>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>
  </table>

  <table name="yandex">
    <comment>
      The table processes messages from the mail.ru,
      yandex.ru detectors.
    </comment>

    <rule enabled="1">
      <comment>
        Discard messages larger than 100 Kb.
      </comment>
      <match>
        <c name="size" op=">" value="100K"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <comment>
        Returning to the main table for further processing.
      </comment>
      <action name="return" />
    </rule>
  </table>
</filter>
```

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="main filter" version="1.0">

  <table name="main">

    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule name="spam" enabled="1">
      <match ...> ... </match>
      <action name="jump" value="spam"/>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>

  </table>

  <table name="spam">
    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule name="return-to-main" enabled="1">
      <action name="return" />
    </rule>

  </table>
</filter>
```

In this example, the **main** table processes some message. The second processing rule is named **spam**: if the message matches this rule's criteria, it is passed over for further processing to the second **spam** table.

If the message processing in the **spam** table is not aborted by foregoing rules, the **return-to-main** rule returns it for further processing to the **main** table starting with the rule the follows the **spam** rule.

4.4.1.3.2.5. LABEL Action

Adds a string label to the message metadata.

Description

This action adds a string label to the metadata of the currently processed message.

If this label has already been set for the message (or for the HTTP object from which the message was extracted), then its value is replaced with a new one.

Is used to add descriptions to messages.

Format

```
<action name="label" label="<label name>" value="<label value>" />
```

or

```
<action name="label" label="<label name>" > label value </action>
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="label"**.

The "label" attribute:

The **label="..."** attribute specifies the name of the string tag to set.

The "value" attribute:

The **value="..."** attribute specifies the value (string) for the tag.

The value can also be enumerated in the **<action>** tag.

Example

```
<action name="label"  
  label="VIRUS-DESCR"  
  value="Win.32.BlackHorse.trojan.virus - mail worm, very dangerous!!!" />
```

or

```
<action name="label"  
  label="VIRUS-DESCR">  
  Win.32.BlackHorse.trojan.virus -  
  mail worm, very dangerous!!!  
</action>
```

Sets the string tag named "VIRUS-DESCR" for the message and adds the following string to it:
"Win.32.BlackHorse.trojan.virus - mail worm, very dangerous!!!".

Example

Mark messages processed by the mail.ru, yandex.ru detectors with the CONTENT-DESCR tag.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Mark messages detected by the mail.ru,
        yandex.ru detectors with the CONTENT-DESCR label.
      </comment>
      <match>
        <c name="detector" value="mail.ru, yandex.ru" />
      </match>
      <action name="label"
        label="CONTENT-DESCR"
        value="Russian mail services"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.6. TAG Action

Adds a tag (number label) to the message metadata.

Description

This action adds a numeric tag to the message metadata. Objects may have multiple tags separated by a comma (,) or a semicolon (;). If the same tag is set for an object more than once, its "level" (the numeric value) increases. In other words, each tag has an internal counter, which shows how many times the tag was set for this object.

You can use filter conditions to verify whether the tag is set and check its counter (how many times it was set for this object) in order to make decisions.

The counter is increased by **1** by default when the tag is set. If you want to increase it by more than **1**, you can specify the increment step after the tag name in parentheses: "**TAG(...)**". For example, **SPAM(3)** increases the counter for the **SPAM** tag by **3**, while **SPAM(1)** is equal to **SPAM**. This may be useful for conditions having different significance, importance or priority but setting the same tag.

Counter change values may be negative. **SPAM(-3)** decreases the counter for the **SPAM** tag by **3**.

Format

```
<action name="tag" value="<tag list>" />
```

or

```
<action name="tag" > tag list </action>
```

The "name" attribute:

The **name** attribute contains the action name: **name="tag"**.

The "value" attribute:

The **value="..."** attribute enlists names of the tags.

Tag names can also be enlisted directly in the **<action>** tag.

Example

```
<action name="tag" value="SPAM" />
```

Sets the tag named "SPAM".

```
<action name="tag" value="SPAM(3)" />
```

Sets the tag named "SPAM" and increases its counter by 3.

```
<action name="tag" value="SPAM, shopping" />
```

Sets tags named "SPAM" and "shopping".

```
<action name="tag" value="SPAM(3), shopping(2)" />
```

Sets tags named "SPAM" and "shopping" and increases their counters by 3 and 2, respectively.

```
<action name="tag" value="SPAM, shopping">VIP-OFFICE</action>
```

Also sets tags names "SPAM" and "shopping".

```
<action name="tag" value="SPAM, shopping">VIP-OFFICE</action>
```

Sets tags named "SPAM", "shopping", "VIP-OFFICE".

Example

Mark requests to popular Russian mail services with the `RUS_MAIL` tag.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Mark requests to popular Russian mail services
        with the RUS_MAIL tag.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
        <c name="req-header"
          headername="Host"
          op="eq"
          value="mail.yandex.ru" />
        <c name="req-header"
          headername="Host" op="eq"
          value="mail.rambler.ru" />
      </match>
      <action name="tag" value="RUS_MAIL"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.7. DATETIME Action

Sets a string label with the current date and time for the message.

Description

This action sets a string label (**label**) or the message, with its value being the current date and time string. If this string label has already been set for the message, its value is replaced with a new one. Is used to track the time taken by the message to passes through the filter.

Format

```
<action name="datetime" value="label-name" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="datetime"**.

The "value" attribute:

The **value="..."** attribute specifies the name of the string label to be set.

Example:

Set the FILTER-BEGIN processing start and the FILTER-END processing end tags for all messages.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Set the FILTER-BEGIN processing start label
        for all the messages.
      </comment>
      <action name="datetime" value="FILTER-BEGIN" />
    </rule>

    <rule enabled="1">
      <comment>
        Set the FILTER-END processing end label
        for all the messages.
      </comment>
      <action name="datetime" value="FILTER-END" />
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.1.3.2.8. DNS Action

Resolves DNS names to IP addresses and IP addresses to names for unknown host addresses.

Description

This action resolves DNS names of hosts and IP addresses of the session.

Each session has a source (client) and a receiver (server) with corresponding names or IP addresses. Sometimes the client or server host name may be known and the IP address may be unknown for the message, or vice versa. The **DNS** action resolves the missing information on the client or server address.

For a known IP address and unknown host name the host name for this IP is resolved (if possible). The same is done for a known host name with an unknown IP address. If the message contains both the host name and the IP address then no action is taken.

If names are retrieved successfully then the **X-Sensor-Src-Host** and **X-Sensor-Dst-Host** headers are added to the metadata and can be used in the "**header**" condition (check the header value).

You should also remember that applying the "**hostname**" condition (check the host name) only makes sense after the **DNS** action is completed.

Format

```
<action name="dns" address="<address type for resolving>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="dns"**.

The "address" attribute:

The **address="..."** attribute specifies the type of address for name resolving. Possible values:

"src" or "client"

Only resolve names for the source address

"dst" or "server"

Only resolve names for the destination address

"both" or "all" or *

Only resolve names for both addresses (source and destination).

If this attribute is not specified, the **"both"** action is assumed (resolve names for both addresses).

Example

Resolve names for a message.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Resolve names for the message.
      </comment>
      <action name="dns" address="server"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.9. DNSBL-LH, DNSBL-RH Action

Checks if the message addresses belong to the DNSBL lists and sets this tag if they do.

Description

dnsbl-rh - check the IP address in DNSBL-RHSBL.

dnsbl-lh - check the host name in DNSBL-LHSBL.

If one of the addresses is on one of the DNSBL lists, the action increments the value of the specified tag by **1**.

The list is always checked completely.

Each address (or host name) included in each DNSBL list increases the tag by **1**. I.e. if both the source address and the destination address are on the same DNSBL list, the tag is increased for each hit.

The tag value may be further analysed in filter conditions, and decisions may be taken based on the inclusion of message addresses in DNSBL lists.

Format

```
<action name="dnsbl-rh"
  address="<address-type>"
  tag="<tag-name>"
  value="<dns-bl domains list>" />
<action name="dnsbl-rh"
  address="<address-type>"
  tag="<tag-name>"
  data="<dns-bl domains list source>" />
<action name="dnsbl-lh"
  address="<address-type>"
  tag="<tag-name>"
  value="<dns-bl domains list>" />
<action name="dnsbl-lh"
  address="<address-type>"
  tag="<tag-name>"
  data="<dns-bl domains list source>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="dns"**.

The "address" attribute:

The **address="..."** attribute specifies the address type for name resolving. Possible values:

"src" or "client"

Check only source address.

"dst" or "server"

Check only destination address.

"both" or "all" or *

Check both addresses (source and destination).

If this attribute is not specified, the **"both"** action is assumed (check both addresses).

The "tag" attribute:

The **tag="..."** attribute specifies the name of the tag to be increased.

The "value" attribute:

The **value="..."** attribute lists DNSBL domains.

Multiple domains are separated with a comma ','.

Domains may also be specified in the value of the tag **<action>dns-bl-domains-list</action>**

The "data" attribute:

The **data="..."** attribute may contain another source for the DNSBL domains list. You can use it to specify long lists instead of specifying them in the **value="..."** attribute.

Possible values:

data="<extern data name>"

Load the list from the external block in the filter (the **<data name="extern-data-name">...</data>** tag).

data="extern://<extern data name>"

Load the list from the external block in the filter (the **<data name="extern-data-name">...</data>** tag). DNSBL domains are separated with commas.

data="file://<full-file-path>"

Load the list from a file. Each DNSBL domain is specified in a new line (no commas required).

Example

```
<action name="dnsbl-rh" tag="SPAM">
  bl.spamcop.net, vote.drbl.sandy.ru,
  sbl.spamhaus.org, cblplus.anti-spam.org.cn
</action>
```

Increases the **"SPAM"** tag if message addresses belong to one of the DNSBL lists. If an address belongs to one of the lists, the **"SPAM"** value is equal to **1**. If the address is on two lists, the **"SPAM"** value is equal to **2**, etc.

In a subsequent check, the value (**"SPAM" > 3**) would indicate apparent spam, while a lesser value would just arouse suspicion. If both addresses are on the same DNSBL list, the tag is increased by **2**.

```
<action name="dnsbl-rh" address="src" tag="SPAM">
  bl.spamcop.net, vote.drbl.sandy.ru,
  sbl.spamhaus.org, cblplus.anti-spam.org.cn
</action>
```

Checks only source IP address.

```
<action name="dnsbl-lh" address="dst" tag="SPAM">
  bl.spamcop.net, vote.drbl.sandy.ru,
  sbl.spamhaus.org, cblplus.anti-spam.org.cn
</action>
```


Checks only the **hostname** of the destination.

Example

Checks all message addresses against the DNS-BL list. Mark hits with the SPAM tag. Delete messages with the SPAM tag and accept all other messages.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Checks all message addresses against the DNS-BL list. Hits
        are to be marked with the SPAM tag.
      </comment>
      <action name="dnsbl-rh"
        address="both" tag="SPAM"
        data="extern://dns-bl-list" />
    </rule>

    <rule enabled="1">
      <comment>Delete spam.</comment>
      <match>
        <c name="tag-exist" value="SPAM" />
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <comment>Accept the rest.</comment>
      <action name="accept" />
    </rule>
  </table>

  <data name="dns-bl-list">
    bl.spamcop.net,
    vote.drbl.sandy.ru,
    sbl.spamhaus.org,
    cblplus.anti-spam.org.cn
  </data>
</filter>
```

4.4.1.3.2.10. SAVE RAW DATA Action

Enables or disables saving of the source (raw) data from which the message was received.

Description

The action enables or disables saving of the source (raw) data from which the message was received. If the message was intercepted in HTTP traffic, its source data are the two files containing the HTTP query and the HTTP response. If the message was intercepted in SMTP- or POP3 traffic, its source data is the file with the original email in the EML format.

Saving source data for each message is **disabled** by default. If saving source data is enabled for messages, the files with its source data will be included in the message as attachments.

Please remember that when source data saving is enabled the stored data amount is increased more than twice, so you may want to use this option only for debugging or when it is imperative to have source data.

Format

```
<action name="save-raw-data" value="<true/false/1/0>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="save-raw-data"**.

The "value" attribute:

The **value="..."** specifies the action active status.

true or 1

Enable saving source data for messages.

false or 0

Disable saving source data for messages (if it was previously enabled in the filter).

Example:

Enable saving source data for messages received via HTTP.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">
    <rule enabled="1">
      <comment>
        Enable saving source data for messages
        intercepted over the HTTP protocol.
      </comment>
      <match>
        <c name="protocol" value="http" />
      </match>
      <action name="save-raw-data" value="true" />
    </rule>
  </table>
</filter>
```

4.4.1.3.2.11. TRANSPORT Action

Sets the transport profile for the message to deliver it with after it is successfully processed by the filter with the **ACCEPT** action.

Description

If this transport profile has already been set for the message, the action does nothing.

You can set multiple transport profiles for the message by specifying multiple **transport** actions.

If no transport profile was applied to the message by the filters, the message (if it is accepted) will be delivered by the **default profile**.

Format

```
<action name="transport" value="<transport-profile-name>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="transport"**.

The "value" attribute:

The **value="..."** attribute specifies the name of the transport profile.

Example:

Set the "smtp-archive" and "smtp-archive-rezerv" transport profiles for all messages. If a message is accepted by the filter, it will be delivered by BOTH profiles.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">
  <comment>Message filter.</comment>

  <table name="main">

    <rule enabled="1">
      <comment>
        Set the "smtp-archive" and "smtp-archive-rezerv"
        transport profiles for all messages. If the message
        is accepted by the filter, it will be delivered by
        BOTH profiles.
      </comment>
      <action name="transport" value="smtp-archive" />
      <action name="transport" value="smtp-archive-rezerv" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.12. HEADER Action

Adds a custom **X-Sensor-...** header to the object's metadata or modifies the value of the existing **X-Sensor-...** header.

Description

The action adds a custom header to the message: **X-Sensor-...**

If a header with such name already exists in the message, its value is replaced with a new one.

Format

```
<action name="header" headername="<UserHeader>" value="<user header value>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="header"**.

The "headername" attribute:

The **headername="..."** attribute specifies the name of the string label to be set.

The "value" attribute:

The **value="..."** attribute specifies the header value.

Example

Adds the following header to the message: **X-Sensor-UserHeader: user header value.**

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Adds the following header to the message:
        X-Sensor-UserHeader: user header value.
      </comment>
      <action name="header"
        headername="UserHeader"
        value="user header value" />
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.13. HEADER_EX Action

Adds a custom header to the object's metadata or modifies the value of the existing header, except of the **From, To, Cc, Bcc, Subject, Date** headers.

Description

The action adds a custom header to the message:

If the header with the same name (case-insensitive) already exists in the message, its value is replaced with a new one.

The header name can not be:

From

To

Cc

Bcc

Subject

Date

Format

```
<action name="header_ex"  
  headername="<UserHeader>"  
  value="<user header value>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="header_ex"**.

The "headername" attribute:

The **headername="..."** attribute specifies the name of the string label to be set.

The "value" attribute:

The **value="..."** attribute specifies the header value.

Example

Adds the following header to the message: **X-SomethingElse-UserHeaderEx: user header value.**

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Adds the following header to the message:
        X-SomethingElse-UserHeaderEx: user header value.
      </comment>
      <action name="header_ex"
        headername="X-SomethingElse-UserHeaderEx"
        value="user header value" />
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.3.2.14. LOG Action

Adds an entry to the log for a specified channel and receiver; can be used for filter debugging.

Description

This action sends a text string, values of message tags and labels, or message metadata to the specified receiver (a file, **syslog** server).

The data are recorded in the following format:

```
[<timestamp>] <value>
```

The following can be recorded to the log:

- The string specified in the **value="..."** attribute.
- The list of labels and their values specified in the **field="#labels"** attribute.
- The list of tags and their values specified in the **field="#tags"** attribute.
- The value of the message metadata header. The **field="X-Sensor-..."** attribute specifies the name of the header.

The **LOG** action is useful for the debugging of filters. Add it to certain filter rules to get a detailed report of message processing and modification of metadata, tags and labels.

Please note:

This action is used to debug filters and their behavior during message processing. Applying it too often can result in degraded general performance **DeviceLock EtherSensor** and execution environment performance in case of heavy disk writing.

Format

```
<action name="log" dst="<log-destination>" value="<user string value>" />
<action name="log" dst="<log-destination>" field="<field type>" />
```

The "name" attribute:

The **name** attribute contains the action name: **name="log"**.

The "dst" attribute:

The **dst="..."** attribute specifies the log entry receiver. This can be:

dst="syslog://<syslog-server-ip:port>"

Sends the message to the **syslog** server over the UDP protocol (RFC-3164).

dst="channel://<channel-name>"

Sends the message to the channel pre-configured in the **Watcher** service.

dst="file://<full-file-path>"

Saves the message to a file.

The "field" attribute:

The **field="..."** attribute specifies the type of the message field to be sent to the log. Possible values:

#labels

Output the list of message labels and their values.

#tags

Output the list of message tags and their values.

#from

Output the list of FROM addresses.

#to

Output the list of TO addresses.

#subject

Output the message subject.

X-Sensor-...

Output the value of the message metadata header.

The "value" attribute:

The **value="..."** attribute specifies the message string to be sent to the log.

Example

```
<action name="log" dst="file://d:\file\path.log.txt"
  value="user string value" />
```

Saves the following string to the **d:\file\path.log.txt** log file:

```
"[<timestamp>] user string value".
```

Action:

```
<action name="log" dst="file://d:\file\path.log.txt" field="#tags" />
```

Saves the following string to the **d:\file\path.log.txt** log file:

```
"[<timestamp>] Tags:"
"          <TAG = value>"
"          <TAG = value>"
"          <TAG = value>"
```

Each tag is written in a separate line.

Action:

```
<action name="log" dst="channel://debug.log.txt" field="#labels" />
```

Sends the following string to the **debug.log.txt** channel:

```
"[<timestamp>] Labels:"
"          <LABEL = "value">"
"          <LABEL = "value">"
"          <LABEL = "value">"
```

Each label is written in a separate line.

Action:

```
<action name="log"
  dst="syslog://192.168.0.1:514"
  field="X-Sensor-Src-Address" />
```

Sends the following string to the **syslog** server with the **192.168.0.1:514** address:

```
"[<timestamp>] X-Sensor-Src-Address: <metadata header value>"
```

Example

Send message details to the syslog server.


```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is the comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Sending message details to syslog.
      </comment>
      <action name="log"
        dst="syslog://192.168.0.1:514"
        value="Message info dump:" />
      <action name="log"
        dst="syslog://192.168.0.1:514"
        field="X-Sensor-Src-Address" />
      <action name="log"
        dst="syslog://192.168.0.1:514"
        field="X-Sensor-Dst-Address" />
      <action name="log" dst="syslog://192.168.0.1:514"
        field="#labels" />
      <action name="log"
        dst="syslog://192.168.0.1:514"
        field="#tags" />
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.1.4. Short rules for developing filters

1. A filter must always contain the **main** table, which is the starting point to this filter.
2. A table may not be empty. Each table must end with a rule true for each message, with one of the following actions: **ACCEPT**, **DROP** or **RETURN**. This rule must be enabled.

For example:

```
<rule name="end" enabled="1">
  <match>
    <c name="all"/>
  </match>
  <action name="drop" />
</rule>
```

or

```
<rule enabled="1">
  <action name="accept" />
</rule>
```

3. **RETURN** can be used in any tables except for **main**.
4. Jumps (**JUMP**) to the **main** table are not allowed (to avoid circular loops).
5. **JUMP** actions forming circular loops (either explicit or implicit) are not allowed.

4.4.1.5. Tips

This section describes techniques you can use to debug and test filters.

Timestamps

Timestamps (`<action name="datetime" value="label-name">`) can be used not only at the start and at the end of message processing, but in each rule to specify when it was processed relative to the start of the entire message filtering process. For this purpose, you can specify the **datetime** action directly before the terminating action.

For example:

```
<rule name="rule2" enabled="1">
  <match>
    <c name="attach-exist"/>
  </match>
  <action name="datetime" value="rule2-end" />
  <action name="accept" />
</rule>
```

In this example, the **"rule2-end"** label will be added to the message after it is processed by the rule in order to specify the time when the message was accepted (**ACCEPT**) by this rule.

Certain actions can take a significant time to perform (such as resolution of DNS names, message inspection by external antivirus software, etc.). All these actions are performed successively, so you can measure the time required to perform a long action by enclosing it with **datetime** actions.

For example:

```
<rule name="rule2" enabled="1">
  <match>
    <c name="attach-exist"/>
  </match>
  <action name="datetime" value="rule2-dns-start" />
  <action name="dns" />
  <action name="datetime" value="rule2-dns-end" />
</rule>
```

In this example, you can compare the values of the **"rule2-dns-start"** and **"rule2-dns-end"** labels after the rule finishes processing to find out the time required by the **dns** operation.

Rule tags

You can use the tag setting **action** in any rule with the tag name identical to the action name. This allows you to record the "history" of rules applied to the message while it passes through the filter. This can be useful to test filters and track filtering "routes" when the filter is complex and contains multiple rules and tables.

For example:

```
<rule name="rule-attach" enabled="1">
  <match>
    <c name="attach-exist"/>
  </match>
  <action name="tag" value="rule-attach" />
</rule>

<rule name="rule-bad-words" enabled="1">
  <match>
    <c name="text" op="all" value="tel, respect" />
  </match>
  <action name="tag" value=" rule-bad-words" />
</rule>

<rule name="rule-only-for-dns" enabled="1">
  <action name="dns" />
  <action name="tag" value=" rule-only-for-dns" />
</rule>

<rule name="accept-all" enabled="1">
  <action name="tag" value="accept-all" />
  <action name="accept" />
</rule>
```

You can do the same with timestamps to record not only the rules applied but also the time of application.

"From simple to complex"

Certain actions can take a significant time to perform (such as resolution of DNS names, message inspection by external antiviruses, etc.). Place such actions as close as possible to the end of the processing route in order to minimize situations when a long operation is performed, and then the **DROP** action is applied due to failed verification of the **FROM** field. You may want to check **FROM** first and drop some messages, and then perform **dns** or other long operations like antivirus inspection.

4.4.2. Prefiltering HTTP Requests

Starting from version **4.0**, a mechanism to prefilter HTTP requests has been added to **DeviceLock EtherSensor** with the following tasks in mind:

1. To filter out unnecessary HTTP traffic to reduce the workload to **DeviceLock EtherSensor** and the runtime environment when analyzing messages (ACCEPT and DROP actions).
2. To accumulate information on possible new trends in HTTP traffic, which can be useful for **DeviceLock EtherSensor** support service and/or developers (COPY action).
3. To log the information on HTTP requests in SQUID-ACCESS-LOG format (ACCESS-LOG action).
4. To manipulate tags and labels at preprocessing stage to use the information they accumulated when analyzing the messages (TAG and LABEL actions).

HTTP filter configuration is defined in the XML file stored in **[INSTALLDIR]\config\filter\http subdirectory**.

To edit the filter, use any external text/XML editor or **DeviceLock EtherSensor** editor built into the configuration utility (**ethersensor_console.exe** from the bundle), which is designed specifically to edit filters:

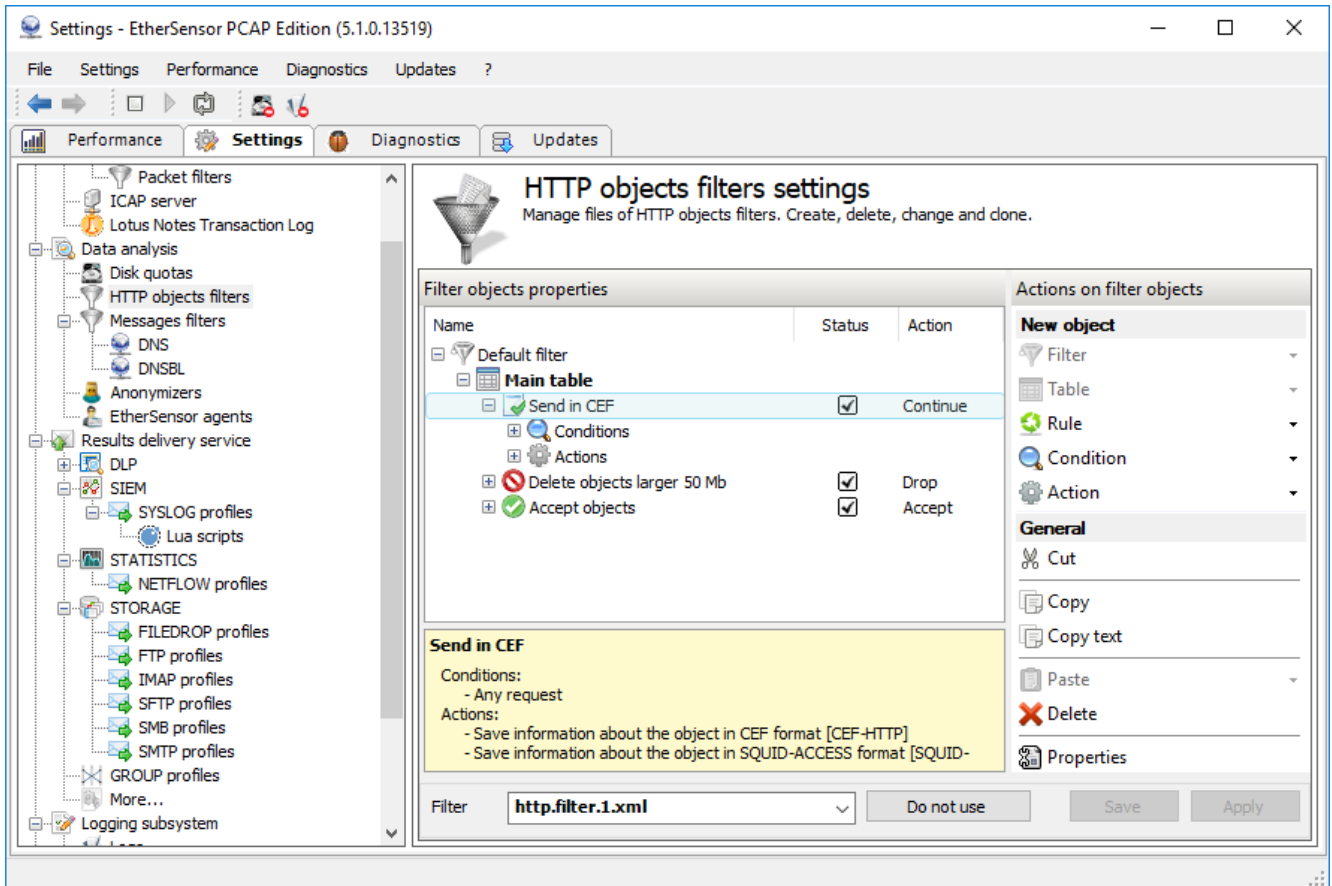


Fig. 49. Editing an HTTP filter.

4.4.2.1. Conditions

Conditions used in HTTP request prefilter rules are listed below.

4.4.2.1.1. ALL, * Condition

A special condition, which is true for all filtered objects.

Description

The condition is true for any object. This condition is implicated if `<match>...</match>` criterion is empty or missing from a rule (the rule only contains `<actions>`).

Format

```
<c name="all" />
<c name="*" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="all"** or **name="*"**.

Example

The rule accepts all messages for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>

  <table name="main">

    <rule enabled="1">
      <comment>
        This rule accepts all messages for further processing.
      </comment>
      <match>
        <c name="all" />
      </match>
      <action name="accept" />
    </rule>

  </table>

</filter>
```

Example (implicated "all" condition)

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <action name="accept" />
    </rule>

  </table>

</filter>
```

4.4.2.1.2. METHOD Condition

The condition checks the HTTP request method.

The **condition** checks the name of the HTTP request method. The result is true if the method name matches the specified value.

Format

```
<c name="method" value="<HTTP method>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="method"**.

The "value" attribute:

The **value="..."** attribute specifies the method name to use for the comparison.

Example

The rule stops processing all GET requests.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>

  <table name="main">
    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule enabled="1">
      <comment>
        The rule stops further processing of GET requests.
      </comment>
      <match>
        <c name="method" value="GET"/>
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.2.1.3. IP Condition

Checks if the client or server IP addresses belong to a range or a subnet.

Description

This condition checks if the client or server IP addresses belong to a range or a subnet.

Please note:

Unwanted traffic should be isolated as early as possible. This affects the performance of **DeviceLock EtherSensor** and the runtime environment.

1. It is recommended to isolate all traffic from a certain IP address or a range of addresses in the EtherSensor EtherCAP service IP filter.
2. It is recommended to isolate certain HTTP traffic from a specific IP address or a range of addresses (if it is possible to specify such criteria) in an HTTP prefilter, but not at the message analysis stage.
3. It is recommended to filter certain messages from a specific IP address in the message filter.

Format

```
<c name="ip" address="<address type>" value="<ip-range>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition - **name="ip"**.

The "address" attribute:

The **address="..."** attribute specifies the address type to check. Possible values:

src or client

Check the source address

dst or server

Check the destination address

The "value" attribute:

The **value="..."** provides a value for the comparison. Possible values:

ipaddress

Checks if the IP address is equal to this value. For example: **value="192.168.0.10"**

ip1-ip2

Checks if the IP address is within this range. For example: **value="192.168.0.1-192.168.0.10"**

ip/netmask

Checks if the IP address belongs to the specified subnet. For example: **value="192.168.0.1/255.255.255.0"**

ip/netmaskbits

Checks if the IP address belongs to the specified subnet. For example: **value="192.168.0.1/24"**

Example

Drop all messages from 192.168.0.15.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>

  <table name="main">

    <rule enabled="1">
      <comment>
        Discard messages from 192.168.0.15.
      </comment>
      <match>
        <c name="ip" address="client" value="192.168.0.15" />
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.2.1.4. REQ-SIZE, RESP-SIZE, SIZE Condition

The condition checks the full size (including headers) of an HTTP object.

Description

Conditions of the **size** group check the full (including headers) size of the HTTP request/response.

req-size

True if the size of the HTTP request matches the condition.

resp-size

True if the size of the HTTP response matches the condition.

size

True if the size of the HTTP request **or** one of the HTTP response matches the condition.

Format

```
<c name="req-size" op="<operation>" value="<compare pattern>" />
<c name="resp-size" op="<operation>" value="<compare pattern>" />
<c name="size" op="<operation>" value="<compare pattern>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition: **name="req-size"**, **name="resp-size"** or **name="size"**.

The "value" attribute:

The **value="..."** attribute specifies the number to which the size of the HTTP object is compared.

<number> or <number>B

Specifies the size in bytes

<number>K

Specifies the size in Kbytes

<number>M

Specifies the size in Mbytes

<number>G

Specifies the size in Gbytes

The "op" attribute:

The **op="..."** attribute specifies the type of comparison. Possible values:

eq or = or ==

True if the size is **EQUAL TO** the specified value

ne or != or <>

True if the size is **NOT EQUAL TO** the specified value

lt or <

True if the size is **LESS THAN** the specified value

gt or >

True if the size is **GREATER THAN** the specified value

le or <=

True if the size is **LESS THAN OR EQUAL TO** the specified value

ge or >=

True if the size is **GREATER THAN OR EQUAL TO** the specified value

Example

The rule stops processing any HTTP objects with the request size over 100KB or the response size over 1MB.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <comment>
        The rule stops processing HTTP objects with the request size
        over 100KB or the response size over 1MB.
      </comment>
      <match>
        <or>
          <c name="req-size" op=">" value="100K"/>
          <c name="resp-size" op="gt" value="1M"/>
        </or>
      </match>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.2.1.5. REQ-HEADER, RESP-HEADER Condition

Checks the value of one of HTTP request or response headers.

Description

This condition checks if the HTTP request header value contains a substring or matches a wildcard pattern/regular expression.

Format

```
<c name="req-header" headername="..." op="..." value="..." />
```

or

```
<c name="resp-header" headername="..." op="..." value="..." />
```

The "name" attribute:

The **name** attribute specifies the name of the condition - **name="req-header"** or **name="resp-header"**.

req-header

Checks HTTP request headers

resp-header

Checks HTTP response headers

The "headername" attribute:

The **headername="..."** attribute specifies the name of the header to check.

The "headername" attribute:

Specifies the string the value is compared to or the pattern to check **value="..."**.

The "op" attribute:

The **op="..."** attribute specifies the type of comparison. Possible values:

eq or = or ==

True if the header value **CONTAINS** the specified value

ne or != or <>

True if the header value **DOES NOT CONTAIN** the specified value

wc or wildcard

True if the header value **matches** the specified **wildcard** pattern

re or regex or regexp

True if the header value **matches** the specified **regular** expression

Operations available for **Content-Length** header are listed below:

eq or = or ==

True if the header value **CONTAINS** the specified value

ne or != or <>

True if the header value **DOES NOT CONTAIN** the specified value

lt or <

True if the size is **LESS THAN** the specified value

gt or >

True if the size is **GREATER THAN** the specified value

le or <=

True if the size is **LESS THAN OR EQUAL TO** the specified value

ge or >=

True if the size is **GREATER THAN OR EQUAL TO** the specified value

These operations treat the header value as a NUMBER, not a string.

The "value" attribute:

The **value="..."** attribute specifies the value to search for (a string, a wildcard or a regular expression).

Warning!

For **Content-Length** header, provide a numeric value for the comparison.

The number should be specified as:

<number> or <number>B

Specifies the size in bytes.

<number>K

Specifies the size in Kbytes.

<number>M

Specifies the size in Mbytes.

<number>G

Specifies the size in Gbytes.

Example

Ignore requests if the Content-Length is more than 100K. Accept requests to win.mail.ru and *.yandex.ru.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <comment>
        Ignore requests where Content-Length is more than 100K.
      </comment>
      <match>
        <c name="req-header"
          headername="Content-Length"
          op=">" value="100K" />
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <comment>
        Accept requests to win.mail.ru and *.yandex.ru.
      </comment>
      <match>
        <or>
          <c name="req-header"
            headername="Host" op="eq"
            value="win.mail.ru" />
          <c name="req-header"
            headername="Host"
            op="wc"
            value="*.yandex.ru" />
        </or>
      </match>
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.2.1.6. URL Condition

Checks the URL value of a HTTP request.

Description

This condition checks if the URL value of the HTTP request contains a substring or matches a wildcard pattern/regular expression.

Please keep in mind, that the entire URL of the request is checked, i.e. `http://www.server.com/script-or-page-path.htm`.

Warning!

If you need to check only the host name, use the **"HEADER"** condition. This will speed up the check and save the resources.

Format

```
<c name="url" op="..." value="..." />
```

The "name" attribute:

The **name** attribute specifies the name of the condition - **name="url"**.

The "op" attribute:

The **op="..."** attribute specifies the type of comparison. Possible values:

eq or = or ==

True if the header value **CONTAINS** the specified value

ne or != or <>

True if the header value **DOES NOT CONTAIN** the specified value

wc or wildcard

True if the header value **matches** the specified **wildcard** pattern

re or regex or regexp

True if the header value **matches** the specified **regular** expression

The value attribute:

The **value="..."** attribute specifies the value to search for (a string, a wildcard or a regular expression).

Example

```
<c name="url" op="eq" value="game" />
```

True if the URL contains the substring "game".

```
<c name="url" op="wc" value="http://*.mail.ru/*send*" />
```

True if the URL matches the wildcard pattern "http://*.mail.ru/*send*".

```
<c name="url" op="re" value="satan|shopping|dating|movies|hexogen" />
```

True if the URL matches the following regular expression: "satan|shopping|dating|movies|hexogen".

Example

Detect requests with bad words (satan|shopping|dating|movies|hexogen), mark them with the **shopping** tag. Ignore requests marked as **shopping**.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <comment>
        Detect requests possibly related to bad content.
      </comment>
      <match>
        <c name="url"
          op="re"
          value="satan|shopping|dating|movies|hexogen" />
      </match>
      <action name="tag" value="shopping"/>
    </rule>

    <rule enabled="1">
      <comment>
        Ignore requests tagged as "shopping".
      </comment>
      <match>
        <c name="tag" tag="shopping"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.2.1.7. TAG Condition

Checks if a tag is set and gets the value of the tag's counter (refer to "TAG Action" section).

Description

This condition checks the specified tag's activity for an object and the level of this tag.

Format

```
<c name="tag" tag="<tag name>" op="<operation>" value="<compare value>" />
```

The "name" attribute:

The **name** attribute specifies the name of the condition - **name="tag"**.

The "tag" attribute:

The **tag="..."** attribute specifies the name of the tag to check.

The "op" attribute:

The **op="..."** attribute specifies the type of comparison. Possible values:

eq or = or ==

True if the value is **EQUAL TO** the specified value

ne or != or <>

True if the value is **NOT EQUAL TO** the specified value

lt or <

True if the value is **LESS THAN** the specified value

gt or >

True if the value is **GREATER THAN** the specified value

le or <=

True if the value is **LESS THAN OR EQUAL TO** the specified value

ge or >=

True if the value is **GREATER THAN OR EQUAL TO** the specified value

exist

True if the tag **exists** (has already been set for this HTTP object)

By default (if the **op** attribute is missing), **"exist"** is used. For the **"exist"** operation, there's no need to specify the **"value"** attribute.

The "value" attribute:

The **value="..."** attribute specifies the value to check in case the value is a tag counter.

Example

```
<c name="tag" tag="SPAM" op="exist" />
```

or

```
<c name="tag" tag="SPAM" />
```

True if the object is tagged as **SPAM**.

```
<c name="tag" tag="SPAM" op=">=" value="3" />
```

True if the object is tagged as **SPAM** and its counter value is greater than or equal to **3**.

Example

Detect requests with bad words (satan|shopping|dating|movies|hexogen), mark them with the **shopping** tag. Ignore requests marked as **shopping**.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Detect requests possibly related to bad things.
      </comment>
      <match>
        <c name="url"
          op="re"
          value="satan|shopping|dating|movies|hexogen" />
      </match>
      <action name="tag" value="shopping"/>
    </rule>

    <rule enabled="true">
      <comment>
        Ignore a request tagged as "shopping".
      </comment>
      <match>
        <c name="tag" tag="shopping"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="1">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.2.2. Actions

Actions performed by HTTP prefiltering rules are listed below.

4.4.2.2.1. ACCEPT Action

Accepts an HTTP object for further processing.

Description

The **ACCEPT** action stops filters for the current HTTP object and accepts it for further processing by the detectors.

Format

```
<action name="accept" />
```

The name attribute:

The **name** attribute specifies the name of the action - **name="accept"**.

Example:

All messages that reach this rule are accepted for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <match ...> ... </match>
      <action ...> ... </action>
    </rule>

    <rule enabled="1">
      <comment>
        All messages that reach this rule are accepted for
        further processing.
      </comment>
      <action name="accept" />
    </rule>
  </table>
</filter>
```

4.4.2.2.2. DROP Action

Ignore an HTTP object without further processing.

Description

The action stops processing the current HTTP object in **DeviceLock EtherSensor** and instructs the system to ignore ("drop") it and destroy all data stored about it.

Format

```
<action name="drop" />
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="drop"**.

Example:

text styleclass="Normal" translate="true">Drop all messages that reach this rule.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <comment>
        All messages that reach this rule will be discarded.
      </comment>
      <action name="drop" />
    </rule>
  </table>
</filter>
```

4.4.2.2.3. JUMP Action

Continue processing an HTTP object in another table.

Description

This action continues processing an HTTP object in another table.

Format

```
<action name="jump" value="<table name>" />
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="jump"**.

The value attribute:

The **value="..."** attribute specifies the name of the table to jump for the further processing of the object HTTP object.

Please note:

Jumping to the **main** table is prohibited to avoid loops.

Jumps, which can lead to an explicit or implicit recursion, are prohibited as well.

Example:

GET requests are passed for processing to the **get-process** table. GET requests are processed in the **get-process** table. ACCEPT for requests to win.mail.ru. Return to the **main** table for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">
    <rule enabled="1">
      <comment>
        Processing of GET requests is passed to the "get-process" table.
      </comment>
      <match>
        <c name="method" value="GET"/>
      </match>
      <action name="jump" value="get-process"/>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>
  </table>

  <table name="get-process">
    <comment>
      GET requests are processed in the table get-process.
    </comment>

    <rule enabled="1">
      <comment>
        ACCEPT for requests for win.mail.ru.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
      </match>
      <action name="accept" />
    </rule>

    <rule enabled="1">
      <comment>
        Return to "main" table for further processing.
      </comment>
      <action name="return" />
    </rule>
  </table>
</filter>
```

4.4.2.2.4. RETURN Action

Returns to the previous table (that called the current one) and continues execution there starting with the next rule.

Description

This action returns the current HTTP object processing to the previous table (that called the current one) and continues execution there starting with the next rule.

Format

```
<action name="return" />
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="return"**.

Warning!

This action is not applicable to the **main** table.

Example:

GET requests are passed for processing to the **get-process** table. GET requests are processed in the **get-process** table. ACCEPT for requests to win.mail.ru. Return to the **main** table for further processing.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <comment>HTTP filter.</comment>
  <table name="main">

    <rule enabled="1">
      <comment>
        Processing of GET requests is passed to the "get-process" table.
      </comment>
      <action name="jump" value="get-process"/>
    </rule>

    <rule enabled="1">
      <action name="drop" />
    </rule>
  </table>

  <table name="get-process">
    <comment>
      GET requests are processed in the table.
    </comment>

    <rule enabled="1">
      <comment>
        ACCEPT for requests for win.mail.ru.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
      </match>
      <action name="accept" />
    </rule>

    <rule enabled="1">
      <comment>
        Return to "main" table for further processing.
      </comment>
      <action name="return" />
    </rule>
  </table>
</filter>
```

4.4.2.2.5. COPY Action

Copies the HTTP object currently being processed (the request and the response, if any) and its passport to the specified folder.

Description

The action copies the HTTP object currently being processed (the request and the response, if any) and its passport to the specified folder.

Format

```
<action name="copy" value="<folder path>"/>
```

The "name" attribute:

The **name** attribute specifies the name of the action: **name="copy"**.

The value attribute:

The **value="..."** attribute specifies the path to the folder to which the data should be copied.

If the folder doesn't exist, it will be created.

Example:

Copy requests to win.mail.ru to d:\data_from_mail.ru.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is a comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Copy requests for win.mail.ru to d:\data_from_mail.ru.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
      </match>
      <action name="copy" value="d:\data_from_mail.ru"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.2.2.6. ACCESS-LOG Action

Saves the information about the current HTTP object to the log channel in the SQUID-ACCESS-LOG format.

Description

The **access-log** action saves the information about the current HTTP object to the log channel in the SQUID-ACCESS-LOG format.

Format

```
<action name="access-log" value="<log-channel-name>" />
```

The name attribute:

The **name** attribute specifies the name of the action - **name="access-log"**.

The value attribute:

The **value="..."** attribute specifies the log channel name. This channel should be pre-configured in the **EtherSensor Watcher** service.

Example

Log all requests to **all-log-channel**, and additionally log requests to win.mail.ru to **mail-ru-log-channel**.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is a comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Log requests for win.mail.ru to mail-ru-log-channel.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
      </match>
      <action name="access-log" value="mail-ru-log-channel"/>
    </rule>

    <rule enabled="true">
      <comment>
        Log all requests to all-log-channel.
      </comment>
      <action name="access-log" value="all-log-channel"/>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.2.2.7. TAG Action

Adds a tag (a numeric label) to the object metadata.

Description

This action sets a tag (a numeric label) in the object metadata. There may be more than one tag. In this case, they must be separated by a comma ',' or a semicolon ';'. If the same tag is set for an object more than once, the "level" of the tag (its numeric value) is increased. Each tag has an internal counter that shows how many times this tag has been set for this object.

You can then use a filter condition to check if a tag exists and what is its counter value (how many times the tag has been set for the current object) to make decisions.

The default increment of the value is **1**. If you need to increase the value by more than **1**, specify the increment value after the tag name in parentheses: "**TAG(...)**". For example, **SPAM(3)** increases the counter value for the **SPAM** tag by **3**, and **SPAM(1)** is equal to **SPAM**. This may be helpful if conditions that set the same tag have different relevance/importance/priority.

Counter change values may be negative. **SPAM(-3)** decreases the counter for the **SPAM** tag by **3**.

Please note:

The set tags will be available in the message filter conditions, if a message was extracted from this HTTP object.

Format

```
<action name="tag" value="<tag list>" />
```

or

```
<action name="tag" > tag list </action>
```

The "name" attribute:

The **name** attribute specifies the name of the action - **name="tag"**.

The "value" attribute:

The **value="..."** attribute lists tag names.

You can also provide names in the **<action>** tag.

Example 1:

```
<action name="tag" value="SPAM" />
```

Sets a tag named "SPAM".

```
<action name="tag" value="SPAM(3)" />
```

Sets the tag named "SPAM" and increases its value by 3.

```
<action name="tag" value="SPAM, shopping" />
```

Sets tags named "SPAM" and "shopping".

```
<action name="tag" value="SPAM(3), shopping(2)" />
```

Sets tags named "SPAM" and "shopping" and increases their counters by 3 and 2.


```
<action name="tag">SPAM, shopping</action>
```

Also sets tags names "SPAM" and "shopping".

```
<action name="tag" value="SPAM, shopping">VIP-OFFICE</action>
```

Sets tags named "SPAM", "shopping" and "VIP- OFFICE".

Example 2:

Mark requests to popular Russian mail services with the RUS_MAIL tag.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>Filter comment.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Mark requests to popular Russian mail services
        with RUS_MAIL tag.
      </comment>
      <match>
        <c name="req-header"
          headername="Host"
          op="eq"
          value="win.mail.ru" />
        <c name="req-header"
          headername="Host"
          op="eq"
          value="mail.yandex.ru" />
        <c name="req-header"
          headername="Host"
          op="eq"
          value="mail.rambler.ru" />
      </match>
      <action name="tag" value="RUS_MAIL"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.2.2.8. LABEL Action

Adds a string label to the object metadata.

Description

This action sets a string label in the metadata of the HTTP object currently being processed. If such label has already been set for the HTTP object, its value will be replaced with the new one. It is used to add descriptions to HTTP objects.

Please note:

The set tags will be available in the message filter (if a message was extracted from this HTTP object).

Format

```
<action name="label" label="<label name>" value="<label value>" />
```

or:

```
<action name="label" label="<label name>" > label value </action>
```

The "name" attribute:

The **name** attribute specifies the name of the action - **name="label"**.

The "label" attribute:

The **label="..."** attribute specifies the name of the string label to set.

The "value" attribute:

The **value="..."** attribute defines the string value of the label.

You can also provide the value in the **<action>** tag.

Example

```
<action name="label"
  label="VIRUS-DESCR"
  value="Win.32.BlackHorse.trojan.virus -
        mail worm, extremely dangerous!!!" />
```

or:

```
<action name="label"
  label="VIRUS-DESCR">Win.32.BlackHorse.trojan.virus -
                        mail worm, extremely dangerous!!!
</action>
```

Sets the string tag named "VIRUS-DESCR" for the message and adds the following string to it: "Win.32.BlackHorse.trojan.virus - mail worm, extremely dangerous!!!".

Example

Mark requests to popular Russian mail services with the CONTENT-DESCR tag.

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="TEST" version="1.0">
  <comment>This is a comment for the filter.</comment>
  <table name="main">

    <rule enabled="true">
      <comment>
        Mark requests for popular Russian mail services
        with CONTENT-DESCR label.
      </comment>
      <match>
        <or>
          <c name="req-header"
            headername="Host"
            op="eq"
            value="win.mail.ru" />
          <c name="req-header"
            headername="Host"
            op="eq"
            value="mail.yandex.ru" />
          <c name="req-header"
            headername="Host"
            op="eq"
            value="mail.rambler.ru" />
        </or>
      </match>
      <action name="label"
        label="CONTENT-DESCR"
        value="Russian mail services"/>
    </rule>

    <rule enabled="true">
      <match><c name="all"/></match>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

4.4.3. Filter Use Cases

Some examples of using the filters to process the interception results are provided below.

4.4.3.1. Adding a Host Name

Problem

Reconstructed messages need to contain domain names of the sender and the recipient in addition to their IP addresses.

Solution Logic

To achieve this, **DeviceLock EtherSensor** has to resolve domain names via DNS.

Perform the following steps:

1. In the **EtherSensor Analyser** service configuration, specify DNS servers to use for the name resolution.

2. Create a message filter with a rule action to resolve host names via DNS for the message currently being processed.

Solution

1. In the EtherSensor Analyser service configuration, specify DNS server settings for the name resolution.

You can do this in the configuration tool or the configuration file.

In the **EtherSensor Analyser** service configuration file:

<AnalyserConfig> root tag, then **<Filter>** nested tag for filter settings (turning on the message filter), then the **<DNS>** nested tag with DNS server settings to resolve the names in the message filter.

In the **EtherSensor Analyser** service configuration file:

The **<AnalyserConfig>** root tag, then the **<Filter>** nested tag for filter settings (turning on the message filter), then the **<DNS>** nested tag with DNS server settings to resolve the names in the message filter.

```
<?xml version="1.0" encoding="utf-8"?>
<AnalyserConfig version="3.2">

<!-- specify other settings for the service here -->

  <Filter enabled="true" filename="msg_filter.xml">

<!-- specify other filter settings here -->

    <Dns>
      <AttemptsCount>3</AttemptsCount>
      <TtlForUnknown>3600</TtlForUnknown>
      <MinTtl>300</MinTtl>
      <MaxTtl>604800</MaxTtl>
      <Server ipaddress="127.0.0.1" port="53" />
    </Dns>

  </Filter>
</AnalyserConfig>
```

Here we assume that the DNS server's IP address is **127.0.0.1:53**.

2. Setting up the message filter

For example, **msg_filter.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="DNS resolve" version="1.0">

  <table name="main">

    <rule enabled="true">
      <comment>
        Resolve the sender and the recipient host names
        for the message.
      </comment>
      <action name="dns" address="both"/>
    </rule>

    <rule enabled="true">
      <comment>
        Accept all message that reached this point.
      </comment>
      <action name="accept" />
    </rule>

  </table>
</filter>
```

Comments and General Recommendations

1. When the **DNS** action is completed, **X-Sensor-Src-Host** and **X-Sensor-Dst-Host** headers are added to the message metadata. They contain domain names or "**<not resolved>**", if the system failed to obtain a domain name.
2. For faster DNS name resolution, it is recommended to specify the fastest DNS servers in the **EtherSensor Analyser** service configuration. These may be your ISP servers or your own DNS servers.
3. The **DNS** action (DNS name resolution) is a lengthy operation. For this reason, make sure you run it in the filter only for the messages that really require this.
4. Try to run the **DNS** action at the end of the filter exactly where its results are required.

For example, if you need the host addresses of the sender and the recipient in the output message to contain domain names, run the **DNS** action right before the end of message filtering and the **ACCEPT** action. There is no need to do it at the beginning of the filter because the **DROP** action may reject some messages. The **DNS** action for such dropped messages (if it is located at the beginning of the filter) would create an unnecessary load on **DeviceLock EtherSensor** and the runtime environment.

If you want to use DNS names of the sender and the recipient further in the filter criteria to drop the messages based on their host names, then you need to resolve DNS names in the rule located directly before one that uses these DNS names.

Troubleshooting

If DNS names for IP addresses of the sender or the recipient do not resolve:

1. Make sure the DNS server is available directly from the sensor machine using **ping** (for example, **ping <dns-server-ip-address>**) and **telnet** (for example, **telnet <dns-server-ip-address>**).
2. Make sure the DNS server you use can resolve names for the sensor machine. To do that, you can use **nslookup**.
 - Run **nslookup** from the command prompt.
 - In the utility, run **server <dns-server-ip-address>** command to define the DNS server used to resolve names.
 - Enter the IP address for which you cannot resolve the name.

If **nslookup** resolves the host name then check your filter:

1. Make sure the rule that contains **DNS** action is enabled and its conditions match the messages.
2. Make sure the rule that contains **DNS** action actually runs, i.e. there is no rule before it that accepts the messages, so filtering doesn't reach this rule (in particular, make sure there are no rules with **ACCEPT** or **DROP** action before this rule).
3. Make sure that when **DeviceLock EtherSensor** is started, the **EtherSensor Analyser** service logs don't contain messages about incorrect filter loading or error messages.

If the problem persists, send the following items to the **DeviceLock EtherSensor** manufacturer:

- Several sample messages for which name resolution fails.
- The message filter you use.
- The diagnostic report on **DeviceLock EtherSensor** work generated by **ethersensor_console.exe** tool from the **DeviceLock EtherSensor** software bundle.
- Your comments and thoughts regarding the steps listed above.

4.4.3.2. Host Filtering

Problem

In case of requests to **<*baender*.com, benderlog.ru, benderlog.biz>** sites, we need to stop processing such messages and destroy all data stored about it.

Solution Logic

There are two options to solve the problem:

1. Ignore all traffic to hosts with the specified names in HTTP filter. To do this, we need to use a filtering condition for **"Host"** HTTP request header.
2. Delete messages sent to the specified hosts in the message filter. To do this, we need to resolve DNS names and use a **"hostname"** filtering condition.

Option 1 is recommended because it filters unnecessary data out at an earlier stage thus decreasing the workload on **DeviceLock EtherSensor** and increasing its overall performance.

Solution

1. Option 1 - ignore the traffic at an earlier stage using HTTP filter.

The HTTP filter file may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <table name="main">

    <rule enabled="true">
      <match>
        <or>
          <c name="req-header"
            headername="Host"
            op="wc"
            value="*baender*.com*" />
          <c name="req-header"
            headername="Host"
            op="eq"
            value="benderlog.ru" />
          <c name="req-header"
            headername="Host"
            op="eq"
            value="banderlog.biz" />
        </or>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="true">
      <action name="accept" />
    </rule>

  </table>
</filter>
```

For a detailed description of the "**req-header**" filtering condition, refer to "REQ-HEADER, RESP-HEADER Condition" section.

2. Option 2 - deleting messages in the message filter.

Make sure DNS name resolution is enabled. The message filter file may look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">

  <table name="main">

    <rule enabled="1">
      <match>
        <or>
          <c name="hostname"
            address="server"
            op="wc"
            value="*baender*.com*" />
          <c name="hostname"
            address="server"
            op="eq"
            value="benderlog.ru" />
          <c name="hostname"
            address="server"
            op="eq"
            value="benderlog.biz" />
        </or>
      </match>
      <action name="drop"/>
    </rule>

    <rule enabled="1">
      <action name="accept" />
    </rule>

  </table>
</filter>
```

For a detailed description of the "**hostname**" filtering condition, refer to "HOSTNAME Condition" section.

Comments and General Recommendations

1. In option **1**, note that the check for the host name from the **Host** header against the "***baender*.com*** wildcard mask contains "*" at the beginning and at the end of the mask. At the beginning of the mask, it is required because the host name may start with upper level domain names (for example, **www.baender.com** or **123.baender.com**). At the end of the mask, it is required because the host name in the **Host** header may end with a destination port (for example, **baender.com:80**). Equality check ("eq") would only search for a substring in a string. Therefore, "eq" with the value of "benderlog.ru" will work for **www.benderlog.ru** and **benderlog.ru:80** as well.

2. For option **2**, generally, for the **hostname** condition to work, you need first to resolve DNS names. However, in this case (we check only the destination host and only for HTTP) it is not necessary, because the host value will be available from the **Host** HTTP header.

4.4.3.3. URL Filtering

Problem

If a URL contains words **<forum, phorum, post, submit>** for **get** method, we need to label this message as "**user reads forums**".

Solution Logic

Check URL and HTTP request method using the HTTP filter. To check the URL, use the "**url**" condition. To check the HTTP method, use the "**method**" condition.

Solution

The HTTP filter file may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">

  <table name="main">

    <rule enabled="true">
      <comment>
        Detect requests possibly related to forums.
      </comment>
      <match>
        <and>
          <c name="method"
            value="GET"/>
          <c name="url"
            op="re"
            value="http://.*/*(forum|phorum|post|submit).*" />
        </and>
      </match>
      <action name="tag" value="USER_READS_FORUMS"/>
    </rule>

    <rule enabled="true">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

For a detailed description of the "**url**" and "**method**" filtering conditions, refer to "**URL Condition**" and "**METHOD Condition**" sections.

Comments and General Recommendations

1. Remember that at the stage of the HTTP filtering, messages don't exist; the traffic will be checked for these messages later. However, labels and tags set for requests at this stage will be saved to the message (if the message is extracted from these requests). These labels and tags will be available for the check in the message filter and later will be available as **X-Sensor-Tags** and **X-Sensor-Labels** headers.

2. Remember that the "**url**" condition gets the entire URL of the request, including the host name (i.e. **http://www.mail.ru/mail/read.php?some=parameter¶m2**). Account for this in the check condition.

4.4.3.4. Filtering by HTTP+DNSBL

Problem

If the message is received or sent using the HTTP GET method, and the server address is in the **dnsbl** list (for example, **dnsbl.sorbs.net**), the message should be labeled as "**possible malware or proxy**".

Solution Logic

Check HTTP request method using the HTTP filter. To make sure the server address is in a DNSBL list, we need to use the message filter. Therefore, two filters will form the solution of the entire problem: the HTTP filter and the message filter. The HTTP filter will check the GET method and label all GET requests (for example, as "HTTP_GET").

The label will be kept in messages extracted from these requests and will become available in the message filter. For all messages that have "HTTP_GET" label, the message filter will check if the server address is in a DNSBL list. To do this, it completes "dnsbl" action. If the server address is in a DNSBL list, the message will get a second label (for example, "DNSBL_EXIST"). Then the message filter will have to check two labels for each message ("HTTP_GET" and "DNSBL_EXIST") and set "possible malware or proxy" label for messages that have both.

For the "dnsbl" action to work correctly, we need to set up DNS servers to check DNSBL in the **EtherSensor Analyser** service configuration.

Solution

1. Set up DNS servers in the EtherSensor Analyser service configuration to check DNSBL.

You can do this in the configuration tool or the configuration file.

In the **EtherSensor Analyser** service configuration file:

<AnalyserConfig> root tag, then <RawHttpFilter> nested tag - enable HTTP filter. The <Filter> filter settings tag (enable message filter). Then the nested <DnsBl> tag - DNS server settings to check addresses in DNSBL lists in the message filter.

```
<?xml version="1.0" encoding="utf-8"?>
<AnalyserConfig version="3.2">

<!-- other service settings -->
  <RawHttpFilter enabled="true" filename="http-filter.xml" />
  <Filter enabled="true" filename="msg_filter.xml">

<!-- other filter settings -->

    <DnsBl>
      <AttemptsCount>3</AttemptsCount>
      <TtlForUnknown>3600</TtlForUnknown>
      <MinTtl>300</MinTtl>
      <MaxTtl>604800</MaxTtl>
      <Server ipaddress="127.0.0.1" port="53" />
    </DnsBl>

  </Filter>
</AnalyserConfig>
```

Here we assume that the DNS server's IP address is **127.0.0.1:53**.

2. The HTTP filter file may look like this (http-filter.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">

  <table name="main">

    <rule enabled="true">
      <match>
        <c name="method" value="GET"/>
      </match>
      <action name="tag" value="HTTP_GET"/>
    </rule>

    <rule enabled="true">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

For a detailed description of the **"method"** filtering condition and the **"tag"** action, refer to **"METHOD Condition"** and **"TAG Action"** sections.

3. The message filter file may look like this (msg_filter.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="Message filter" version="1.0">

  <table name="main">

    <rule enabled="true">
      <match>
        <c name="tag" tag="HTTP_GET"/>
      </match>
      <action name="dnsbl-rh"
              address="both"
              tag="DNSBL_EXIST"
              value="dnsbl.sorbs.net" />
    </rule>

    <rule enabled="true">
      <match>
        <and>
          <c name="tag" tag="HTTP_GET"/>
          <c name="tag" tag="DNSBL_EXIST"/>
        </and>
      </match>
      <action name="tag" value="MALWARE_OR_PROXY"/>
    </rule>
  </table>
</filter>
```

For a detailed description of the **"tag"** condition, **"tag"** action and the **"dnsbl"** action, refer to **"TAG Condition"**, **"TAG Action"** and **"DNSBL-LH, DNSBL-RH Action"** sections.

Comments and General Recommendations

1. For faster DNS name resolution, it is recommended to specify the fastest DNS servers in the **EtherSensor Analyser** service configuration. These may be your ISP servers or your own DNS servers.

2. "**dnsbl**" action (DNS name resolution for DNSBL) is a lengthy operation (especially if you specify several DNSBL services to use); for this reason, make sure you run it in the filter only for the message that really require this.

3. Try to run the "**dnsbl**" action at the end of the filter where its results are required.

4. Remember that at the stage of the HTTP filtering, messages don't exist; the traffic will be checked for these messages later. However, at this stage labels and tags set for requests will be saved to the message (if the message is extracted from these requests).

4.4.3.5. Filtering Large HTTP Objects

Problem

Sometimes very large objects are transferred via HTTP (file download, watching movies online). When too many such objects are transferred at the same time, **DeviceLock EtherSensor** may use a lot of RAM. As a result, the overall **DeviceLock EtherSensor** and the environment performance may degrade.

We need to use filters to block large HTTP objects and delete them before the analysis.

Solution Logic

To delete large HTTP objects before full analysis (and before loading them into the memory for the analysis), we need to use an HTTP filter. To check the size of a HTTP request or response, we need to use "**size**" condition, which checks the size of the request and the response.

Solution

The HTTP filter file may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">

  <table name="main">

    <rule enabled="1">
      <comment>
        The rule stops processing HTTP objects for which
        the request or response size is greater than 1MB.
      </comment>
      <match>
        <c name="size" op="gt" value="1M"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="true">
      <action name="accept" />
    </rule>
  </table>
</filter>
```

For a detailed description of the "**size**" HTTP filtering condition, refer to "**REQ-SIZE, RESP-SIZE, SIZE Condition**" section.

Comments and General Recommendations

1. Instead of the "size" rule, which checks both the request and response size, you can use "req-size" (to check only the size of the HTTP request) or "resp-size" (to check only the size of the HTTP response) conditions.

5. EtherSensor Updater Service

The **EtherSensor Updater** service (**ethersensor_updater.exe** process) is used to automatically download and install updates and patches. **EtherSensor Updater** is a part of the manufacturer's service to maintain normal work of **DeviceLock EtherSensor** and update its functions.

EtherSensor Updater functions:

- Check for updates and patches for **DeviceLock EtherSensor**, download and install them.
- Promptly update the product license file in case of change.
- Create a backup copy of the product previous version and make sure all necessary files are included.
- Restore a product previous version in case of update installation error.

You can specify the interval to check for updates or use default settings. **EtherSensor Updater** allows flexible network connection and supports use of proxy servers.

How it works:

The update service checks for new software versions as well as for updated license versions. To do this, it connects to the update server with the specified intervals and sends the information on **DeviceLock EtherSensor** version and its license.

If newer versions of the installed software exist, the server responds with a manifest file that specifies files to be downloaded and installed. The service downloads these files from the update server and adds them to the installation queue. The service starts these files at the time specified in the configuration, and the software is updated.

EtherSensor Updater installation:

To install **EtherSensor Updater** for the first time, run **ethersensor_updater_X.X.X.XXXX_x64.msi** from the distribution package. After that, the service will be updated automatically as soon as new versions are released.

To install to a directory other than the default one, use Windows **msiexec.exe** utility.

For example:

```
msiexec.exe /i ethersensor_updater_4.6.3.12232_x64.msi INSTALLDIR="C:\Program Files\Microoolap Ether
```

Technical requirements:

- The **EtherSensor Updater** service should have Internet access via port 80 or 443 to the following servers: **license.microoolap.com** and **kpps-downloads.microoolap.com**.
- If using a proxy server, it should work correctly with SOAP.
- Windows Server 2008, Windows Server 2012 R2 or Windows Server 2016.

EtherSensor Updater consists of the following:

The EtherSensor Updater update service (**ethersensor_updater.exe** process)

Starts in the background mode and performs all actions necessary to check for updates and install them. It also creates the **status.xml** status file that contains the information on the current status of the service.

The "Updates" section of the **DeviceLock EtherSensor** configurator (**ethersensor_console.exe** process)

The "Updates" section of the **DeviceLock EtherSensor** configurator (**ethersensor_console.exe** process)

Used to configure the service and display its status. The main update log file is displayed on a separate tab.

Notes:

1. The **DeviceLock EtherSensor** configurator connects to **ethersensor_updater.exe** using TCP/IP, port 52076.
2. If the **DeviceLock EtherSensor** configurator detects that **ethersensor_updater.exe** is not running, it doesn't attempt to make a connection. If you need to connect to a service working in the console mode, use the following command line parameter: **/ignore-service-status**
3. You can also use the **/no-connect** command line parameter, which forbids all attempts to contact the service. In this case, the **DeviceLock EtherSensor** configurator only receives the information on the server status via the **status.xml** file.

[INSTALLDIR]\config directory

Contains configuration files for the service (service configuration).

[INSTALLDIR]\downloads directory

When using default setup values, the service uses this directory to download all files necessary to update the software.

[INSTALLDIR]\log directory

Contains all log files and the status file:

status.xml

The status file that contains the information on the current status of the service.

log.txt

The service main operation log.

uupdater_log.txt

Logs of **ethersensor_uupdater.exe**, a special application that is called when installing a new version of **EtherSensor Updater**.

5.1. Setting up the Configurator

EtherSensor Updater configuration is stored in the **system.xml** file in the **[INSTALLDIR]\config** directory.

When started, the service reads the config file. If you change the configuration file, restart the service to apply the changes.

Current status

The **Current status** section displays the information from the **EtherSensor Updater** status file related to the update service operation.

It also lists the software on the sensor, for which it can check the updates, the files currently being downloaded (if any) and the updates ready for installation.

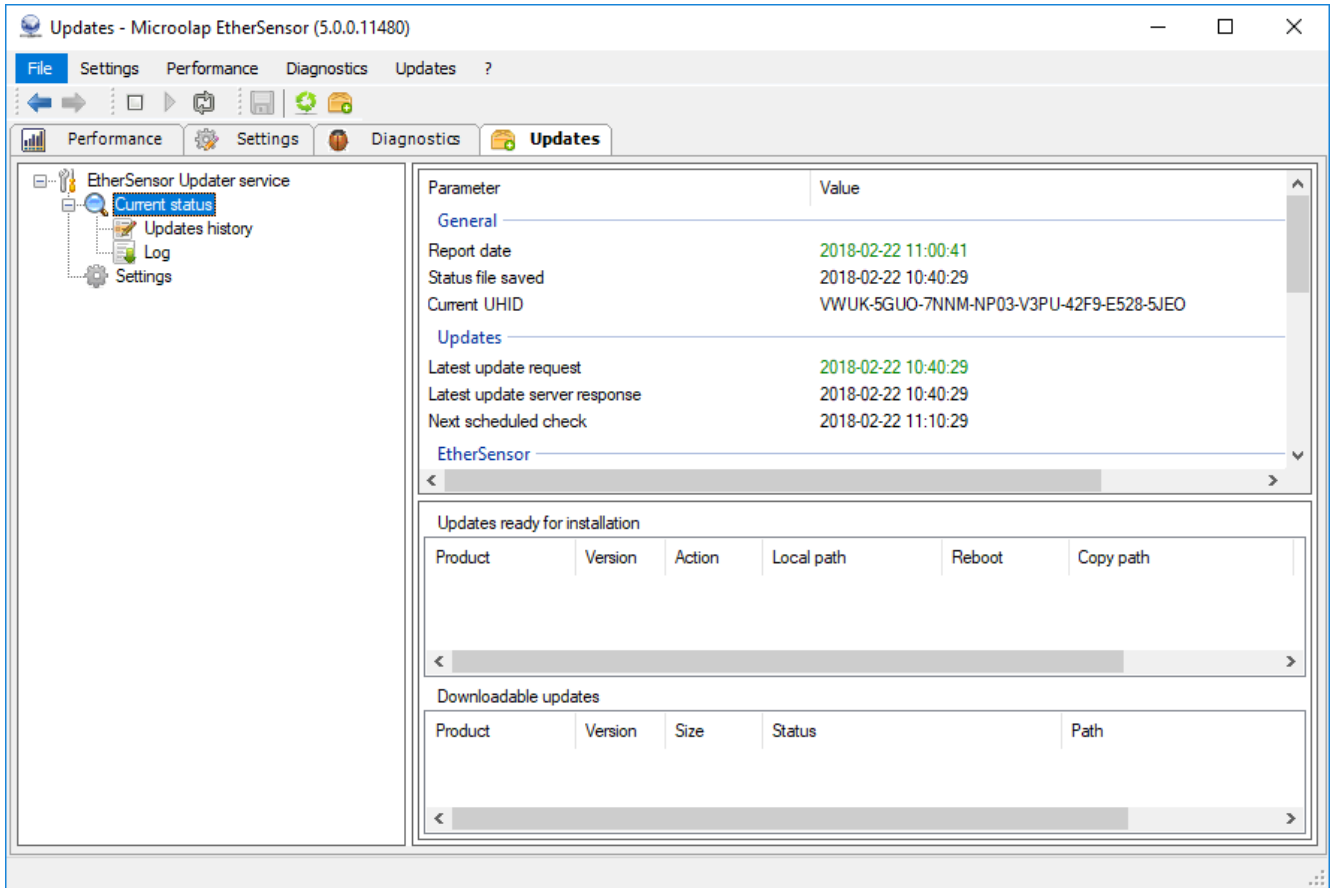


Fig. 50. The "Current status" section

Settings, Download updates tab

The **Download updates** section configures **EtherSensor Updater** operation modes and the interval for the update checks. You can also enable the test mode here. In this mode, the log files contain more detailed messages about the service operation.

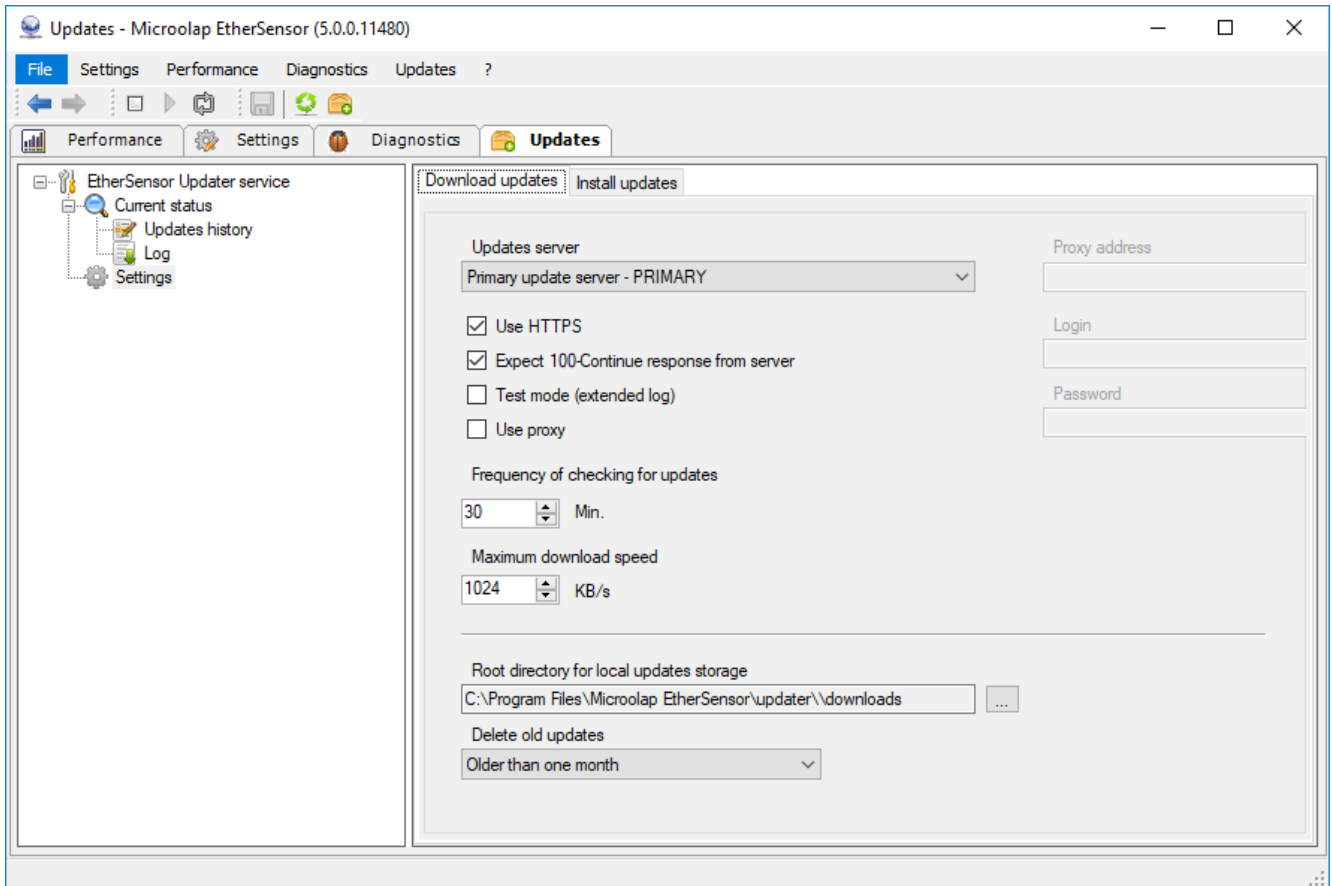


Fig. 51. The "Download updates" section

For more details on **EtherSensor Updater** operation mode parameters, refer to system.xml configuration file section

Settings, Install updates tab

The **Install updates** section configures the time to automatically install the updates. After updates are installed, the operation system reboots (if it is required).

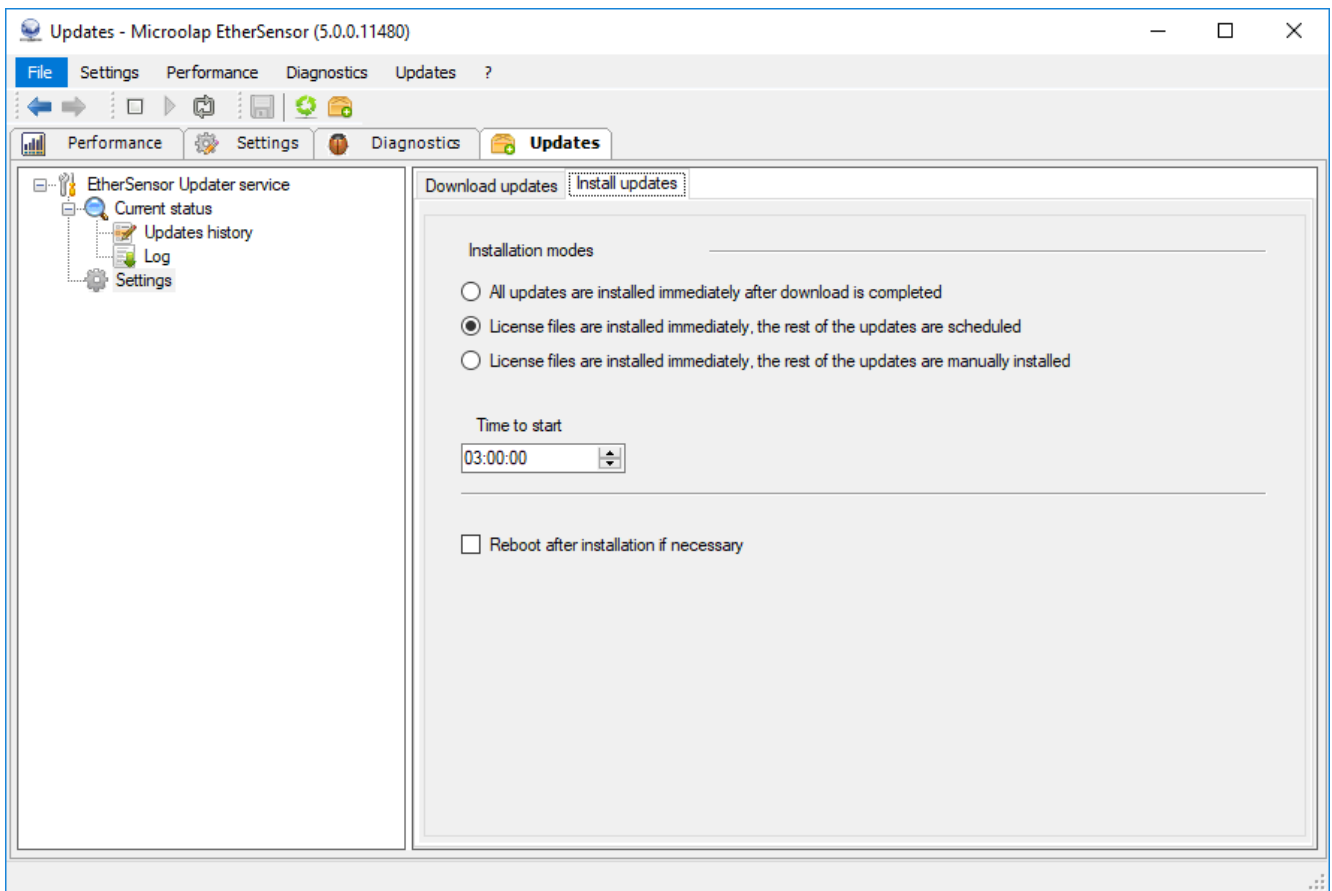


Fig. 52. The "Install updates" section

Log

The **Log** section displays the last records from the **EtherSensor Updater** log files.

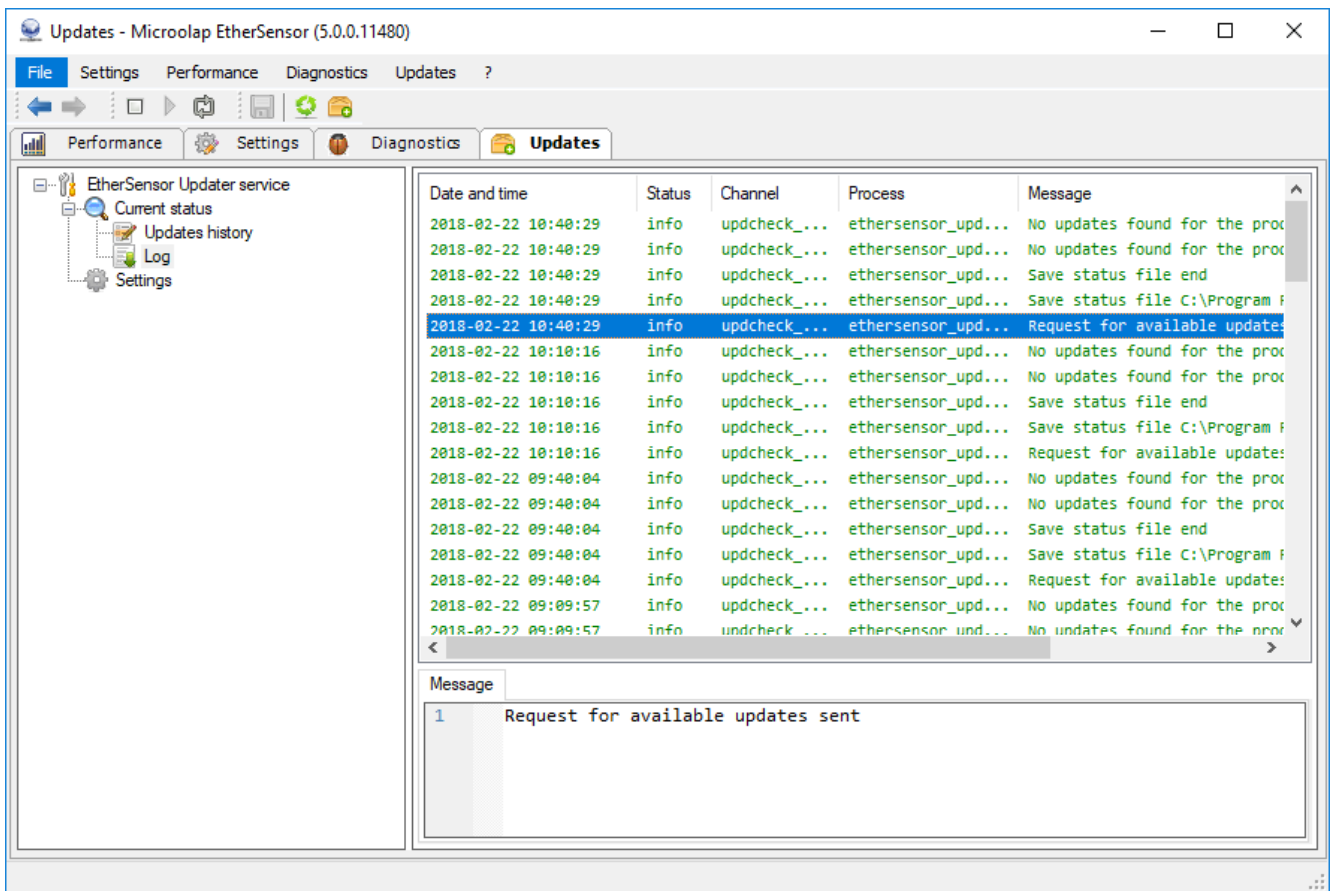


Fig. 53. The "Log" section

Update history

The **Update history** section lists all previously installed updates.

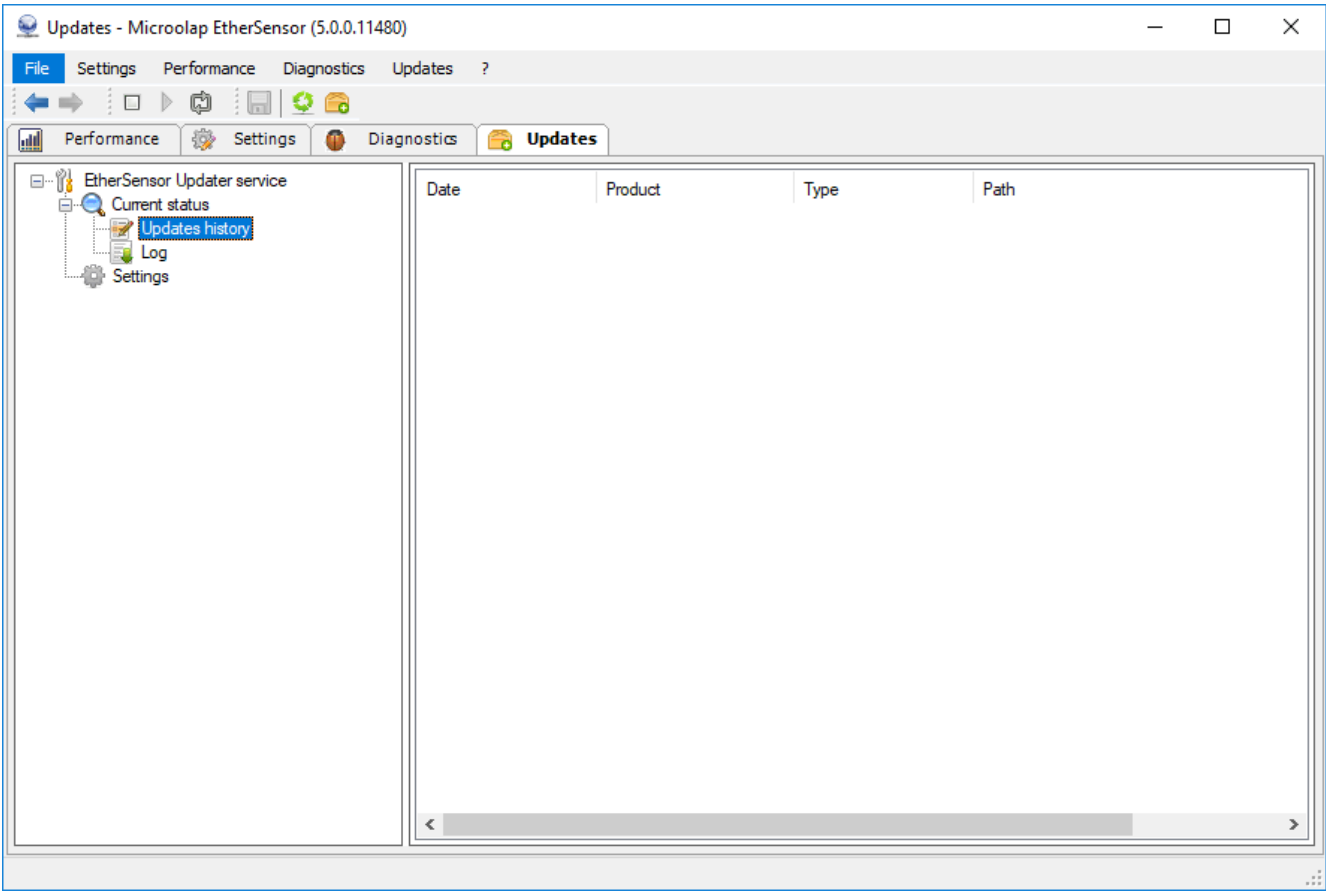


Fig. 54. The "Update history" section

5.2. Manual Setup (Config File)

The **EtherSensor Updater** configuration is stored in the **system.xml** file in the **[INSTALLDIR]\config** directory.

A sample **system.xml** configuration file:

```
<?xml version="1.0"?>
<UpdaterConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Store>
    <Location>c:\Program Files\Microolap DLP SUS\Downloads\</Location>
    <Rate>SixMonth</Rate>
  </Store>
  <Version>1.0</Version>
  <UpdatesServer>PRIMARY</UpdatesServer>
  <UseHTTPS>>false</UseHTTPS>
  <UseWebProxy>>false</UseWebProxy>
  <WP_Address>192.168.10.2</WP_Address>
  <WP_Port>3128</WP_Port>
  <WP_Username>test</WP_Username>
  <WP_Password>TscOFRJHHBwXOM5QgazQ8w==</WP_Password>
  <CheckRate>30</CheckRate>
  <mode>wm_standard</mode>
  <InstallationTime>3:00</InstallationTime>
  <AutoReboot>>false</AutoReboot>
  <MaxDownloadRate>128</MaxDownloadRate>
</UpdaterConfig>
```

The XML tags used in the configuration file are explained below:

CheckRate

Update check period in minutes. Sets the interval in minutes between periodic update checks. The first check occurs in **5** minutes after the service has been started. All further checks occur periodically with the specified interval. To immediately check for updates, use the **Check for updates** action in the **DeviceLock EtherSensor** configurator.

InstallationTime

Time of the day when downloaded updates will be installed. Default value is **3:00** (3:00AM). To immediately install updates, use **Install available updates** action in the **DeviceLock EtherSensor** configurator.

AutoReboot

Reboot the OS after the installation, if necessary. If an installed update requires a reboot, **DeviceLock EtherSensor** configurator will display the respective message. If this option is enabled, the operating system will reboot automatically if at least one update requires this.

mode

The operating mode. Only one operating mode is supported at the moment: **wm_standard**.

UseHTTPS

Use HTTPS to connect to the update server. **true** allows to use HTTPS to connect to the update server. **false** means HTTP will be used for all connections.

UseWebProxy

Use a proxy server to connect. This option instructs the service to connect via a proxy server when working in the standard mode.

WP_Address

The address of the proxy server.

WP_Port

The port number of the proxy server.

WP_Username

The user name to access the proxy server. Parameters to connect via a proxy server. If the proxy server doesn't require authorization, leave **Login** and **Password** blank.

WP_Password

The encrypted password to access the proxy server. To change the password, use the **DeviceLock EtherSensor** configurator.

<Store><Location>

The storage root directory. The path to the directory, to which files will be downloaded when working in the standard mode and where the files from the update directory will be stored when working in the local mode. The names of subdirectories in this directory will contain the product ID, for which the updates are intended as well as the date and time when the update has been received. By default, the **downloads** subdirectory in the update service installation directory is used.

<Store><Rate>

The storage clean period. Files are deleted from the storage after the specified period of time. Possible values:

None

Don't delete files

OneMonth

Delete files older than one month

TwoMonth

Delete files older than two months

ThreeMonth

Delete files older than three months

SixMonth

Delete files older than six months

Year

Delete files older than one year

<MaxDownloadRate>

The maximum file download speed. This is the maximum speed (in KB/sec) to download update files from the server. It is recommended to limit this speed to prevent the Internet channel from being fully used by the update service.

6. Sensor Routine Maintenance

To ensure the best performance of the sensor and prevent problems, it is recommended to regularly perform the following actions.

1. Check sensor logs for any service warnings or errors.
2. Perform periodic backups. Make sure you backup the sensor's configuration file to an external drive on a regular basis.
3. Check performance indicators of the server where is running **DeviceLock EtherSensor**, such as drive free space, CPU utilization and RAM usage. To do this, you can use Windows Performance Monitor or **ethersensor_console.exe** in the installation folder **DeviceLock EtherSensor**. Use of notification allows to notify the **DeviceLock EtherSensor** service team on unexpected changes or problems with the server performance.
4. Check log files in the network device. Analyze log files to make sure the delivery service is working correctly and evaluate the amount of tapped traffic to avoid overloading its receivers.
5. Check if intercepted message receivers are available to the sensor. Make sure messages generated by sensor services are delivered to the servers of message receivers. If messages are not delivered to the receivers, they may overflow the sensor's disk space (it depends on the disk quotas set up by the administrator).

6.1. Questions on the Sensor Maintenance

How to manually delete DeviceLock EtherSensor services?

1. Run the following commands from the command prompt:

```
sc delete "EtherSensorEtherCAP"  
sc delete "EtherSensorICAP"  
sc delete "EtherSensorLotusTXN"  
sc delete "EtherSensorAnalyser"  
sc delete "EtherSensorTransfer"  
sc delete "EtherSensorWatcher"
```

2. Run **regedit** and delete the following registry branches:

```
HKLM/System/CurrentControlSet/EtherSensorAnalyser  
HKLM/System/CurrentControlSet/EtherSensorEtherCAP  
HKLM/System/CurrentControlSet/EtherSensorICAP  
HKLM/System/CurrentControlSet/EtherSensorLotusTXN  
HKLM/System/CurrentControlSet/EtherSensorTransfer  
HKLM/System/CurrentControlSet/EtherSensorWatcher
```

3. Delete the old version files.
4. Restart the server.

Why there are many duplicated messages?

1. Web services may send forms more than once, e.g. when saving drafts or adding attachments.
2. When network conditions are complicated (for example, when using a chain of proxy servers or load balancers), the sensor may see the same connection through several interfaces. In this case, use **check-md5** filter. This will help decide whether it is necessary to process this message again or not.

I see traffic from the <another sniffer> mirror port, but the sensor doesn't intercept anything.

Traffic counters are increasing, but the sensor doesn't intercept anything.

In the interception record there are no back packets from HTTP services, i.e. the packets coming from the client IP address to the remote server/port are visible, but the remote server/port responses are not. ACK, FIN/ACK packets from the client a coming meaning that these are real traffic working sessions, and the traffic reaches the client.

Perform the following steps:

1. Make sure the services are up and running.
2. Check IP filtering rules. To apply rules, restart the services.
3. Check traffic counters. The value of **Received** should not be equal to one of **Rejected**.
4. Check if the intercepted data are in the **data** subfolder of the **DeviceLock EtherSensor** installation folder. When using filters, also check **[INSTALLDIR]\data\filter**.
5. Check the **[INSTALLDIR]\data\result** folder for any intercepted messages that have not been send.
6. Check logs and counters for any errors. If there are no errors, the services are working correctly.
7. Check settings of the mirror port. For example, Cisco hardware by default mirrors either **RX** or **TX** packets. Change the setting to use **both** keyword:

```
monitor session 1 source interface <interface-id> both
```

8. Check if a profile is set up in the delivery service settings and if a correct profile is set up as the default profile.
9. If the problem persists, please contact support.

7. Licensing EtherSensor

The **DeviceLock EtherSensor** licensing is described below.

7.1. The License File

All information about **DeviceLock EtherSensor** licenses is stored in the **license.licx** file, which should be located in the root folder of **DeviceLock EtherSensor**.

Warning!

DeviceLock EtherSensor versions from **4.0.15** don't support licenses issued before March 2012.

Please make sure you have updated **license.licx** before updating **DeviceLock EtherSensor** to versions **4.0.15** and higher.

(If you open **license.licx** in a text editor, it should contain the following XML tag:

```
<UPDSUBEND></UPDSUBEND>
```

The license contains information on modules licensed for this particular installation, license terms for each module, update subscription expiration date and other data. You can easily view the license file contents; however, it is protected using CRC and digital signature. The license is issued to install **DeviceLock EtherSensor** on a particular hardware, and there is a dedicated check to verify this.

You can check the license status using the configuration tool (**Licensing** section).

Based on the presence of **license.licx** in the root folder and on the **DeviceLock EtherSensor** license types, all modules work in one of the following modes:

Demo Mode

In this mode, all messages are detected, but only 1 of each 5 messages is passed to the delivery service in full. **DeviceLock EtherSensor** can work in demo mode for an unlimited amount of time. To enable full mode for **DeviceLock EtherSensor**, move the valid **license.licx** file to the root folder (there is no need to restart services after that).

Full Mode

In full mode, the sensor software works for an unlimited amount of time and ensures all modules available according to the license are working correctly. After the end date of the subscription for updates, which is specified in **license.licx**, the software updates stop automatically. If you manually install a version issued after this date, it will be working in demo mode until it gets a new **license.licx** file with the new license data. When a software version issued after the end date of the subscription for updates is started in demo mode, a message on the latest version supported by the current license is added to the log file.

In some cases (for example, for testing) the license may include modules with their expiration dates. After these expiration dates, the messages corresponding to the respective modules will be intercepted in demo mode.

DeviceLock EtherSensor adds a message on the license expiration to **DeviceLock EtherSensor** log files.

The update system may automatically pass the license file to the working sensor.

The **DeviceLock EtherSensor** manufacturer generates the **license.licx** file based on the **request.licx**, a license request file created on the particular sensor hardware. If **DeviceLock EtherSensor** is moved to another server or the current server's configuration is changed, **DeviceLock EtherSensor** may not recognize the license as a valid one. In this case, send a new license request and receive a new **license.licx** file.

Warning!

Licenses are linked to the runtime environment hardware for which they have been issued.

Therefore, to avoid problems with the change of the UHID during operation, before creating the **request.licx** file, make sure all runtime environment hardware is in the state you're planning it to operate, i.e. disconnect all temporary devices (such as flash drives, external HDDs, etc.), temporary, virtual or unused network adapters, set up MAC addresses, etc.

For more details on the UHID, please refer to "Runtime Environment UHID (HardwareID)" section.

To create the **request.licx** file, open the **Request for license** section in the configuration tool. A window will open. Fill out the fields and click **Save** button:

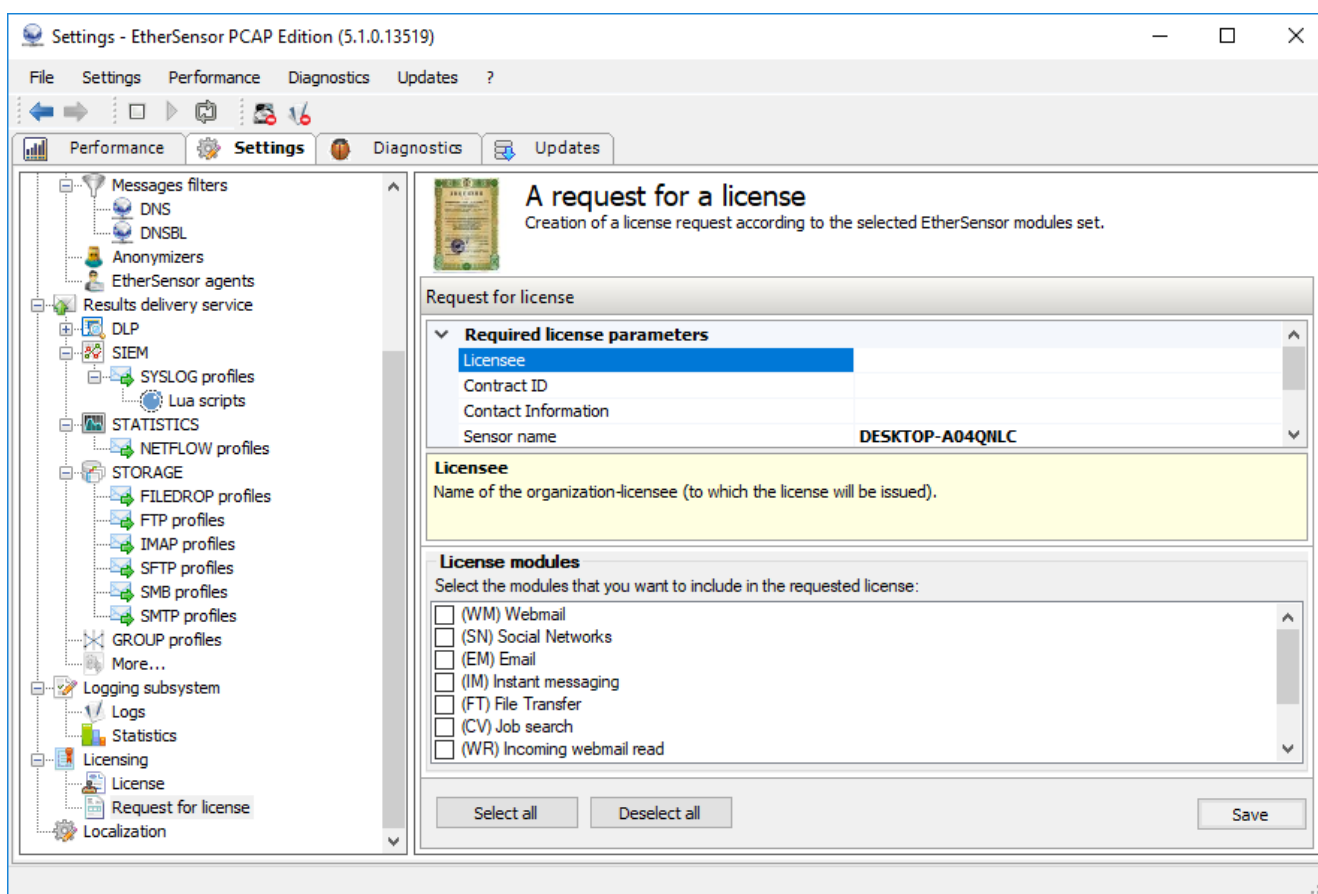


Fig. 55. License request form.

License request file are to be saved in the **DeviceLock EtherSensor** installation folder. The filename will be **request.licx**.

Licensee:

Full name of the licensee organization.

Contract ID:

License agreement info; the contractor name is required.

Contact Information:

Full contact data of the person responsible for **DeviceLock EtherSensor** operation (name, phone number, email, etc.).

Sensor name:

The name of the sensor (server name). Used to distinguish between different sensors within the same organization. Here you can enter the host name or any conventional name for the sensor, which uniquely identifies it within the licensee organization.

License modules:

DeviceLock EtherSensor modules which are subject to licensing.

When you click the **Save** button, **DeviceLock EtherSensor** the **request.licx** file will appear in the root folder. Please send it to the **DeviceLock EtherSensor** supplier to create a new **license.licx** file.

7.2. Runtime Environment UHID (HardwareID)

Each product license (**license.licx** file) is linked to the **DeviceLock EtherSensor** runtime environment hardware for which it has been issued. This is done via **UHID**, which is a special code unique for each system and hardware set (**Unique Hardware IDentifier**). UHID is added to the license file and **DeviceLock EtherSensor** then the system checks if the system UHID matches the UHID in the license.

If the UHID in the license doesn't match the UHID of the system, **DeviceLock EtherSensor** automatically switches to the demo mode (for more information, refer to "The License File" section).

To calculate the current system UHID, the software uses the list of network adapters found in the system as well as the list of storage devices. This means that any changes in the list of network adapter or storage devices may also change the current UHID.

Warning!

To avoid problems arising from UHID change during operation, before creating **request.licx** file, make sure that all hardware of the **DeviceLock EtherSensor** runtime environment is in the state you're planning it to operate, i.e. disconnect all temporary devices (such as flash drives, external HDDs, etc.), temporary, virtual or unused network adapters, set up MAC addresses, etc.

After that, a link to the hardware will be created in **request.licx** and as long as this hardware is found in the **DeviceLock EtherSensor** runtime environment, the license will be working.

After you receive the license (**license.licx** file), **DeviceLock EtherSensor** checks if the UHID specified in the license matches the UHID of the runtime environment.

While you're using **DeviceLock EtherSensor**, you can temporarily connect storage devices (flash drives, external HDDs, etc.) to the server. The software will detect it and continue working correctly despite the temporary UHID change.

However, if you replace or add any hardware to the server, the UHID may change permanently. In this case you need to generate a new **request.licx** file (for more information, refer to "The License File" section) and send it to license@microolap.com along with the description of the hardware changes in your server. The **DeviceLock EtherSensor** manufacturer's operator will check the data and change the license so it matches the new UHID of the **DeviceLock EtherSensor** runtime environment.

To check the current runtime environment UHID and the UHID specified in the license and make sure they match, use the configuration tool (**ethersensor_console.exe**):

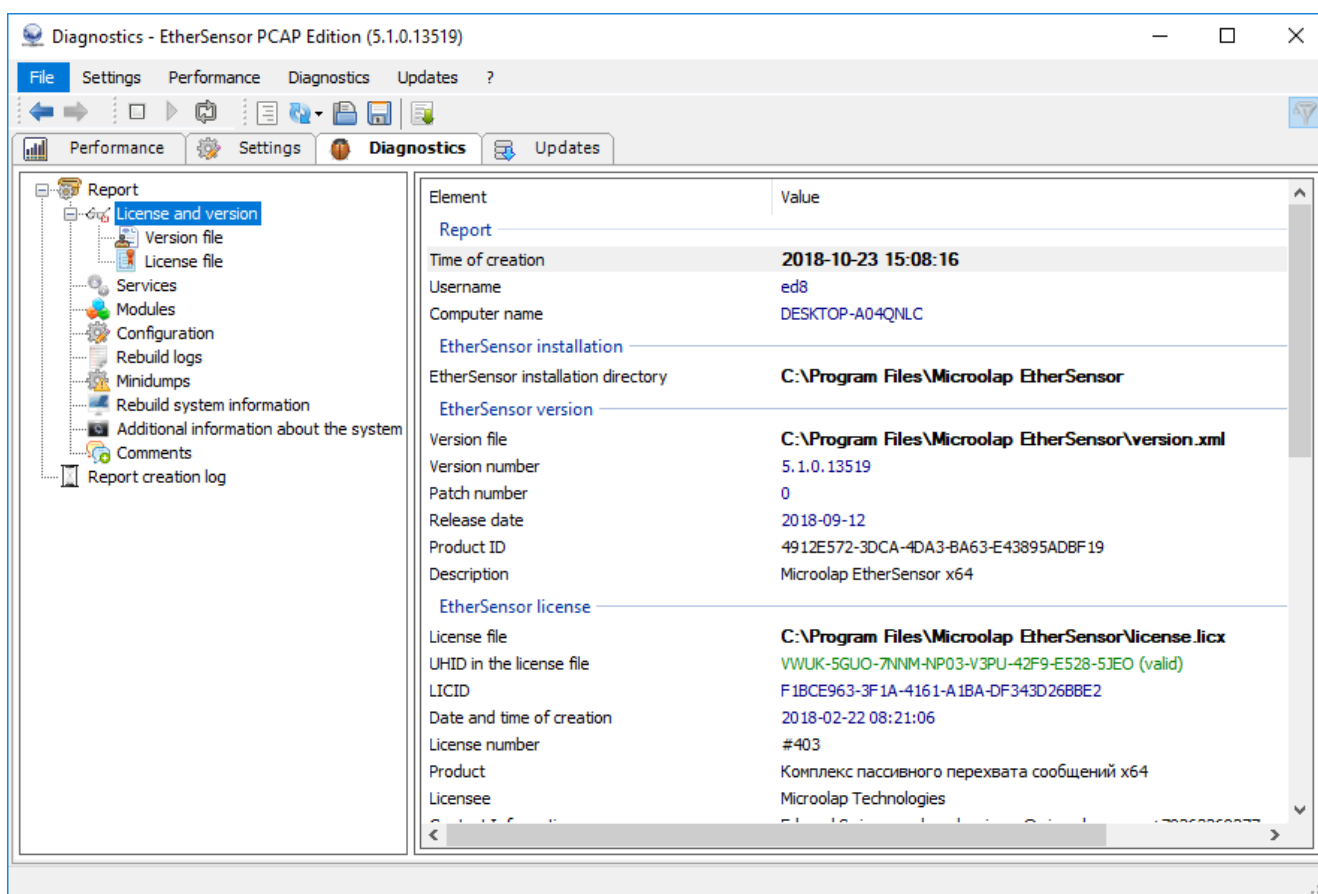


Fig. 56. Runtime environment and license UHID information in ethersensor_console.exe.

Here we can see that the runtime environment UHID is different from the license UHID (this is because an additional storage device has been attached). However, **DeviceLock EtherSensor** still considers the license UHID to be valid.

7.3. How the Licensing System Works

Until you install **DeviceLock EtherSensor** to a server, it is impossible to issue a license for this installation, because data on the runtime environment are required to issue a license. Licenses are linked to the runtime environment hardware and should not work on different platforms.

When **DeviceLock EtherSensor** is installed, use the configuration tool (**ethersensor_console.exe** file from the **DeviceLock EtherSensor** bundle) and create a license request, which will contain the data on the runtime environment. The request will be saved in **request.licx** file in **DeviceLock EtherSensor** installation root folder. For more information on generating **request.licx**, refer to "The License File" section.

Based on this request, the **DeviceLock EtherSensor** manufacturer issues a license of a certain type. You can send this file by email, save it to a removable media or use the web interface of the license system available to service engineers.

The URL of the interface is <https://license.microolap.com>

Created license should be approved by the **DeviceLock EtherSensor** manufacturer before an engineer can receive it in the web interface or by email. The approval takes less than 3 hours.

The access to the license system is granted to service engineers after a request sent to license@microolap.com.

Warning!

Please remember that the license status directly impacts the update system of **DeviceLock EtherSensor**. If the license is expired, you won't be able to receive any updates. Valid licenses define when and what updates **DeviceLock EtherSensor** will be receiving. Particularly, the update system allows to replace the license file for the installation, so you won't need to change it manually.

Express Licenses

If you need to start **DeviceLock EtherSensor** quickly, your engineer may receive an express license for a new installation. It is a full **DeviceLock EtherSensor** license; the difference is that it works only for one week after being issued and you can download it from the web interface immediately, without waiting for the **DeviceLock EtherSensor** manufacturer approval. For each installation, you can issue only one express license. Any further licenses should be full and have the **DeviceLock EtherSensor** manufacturer approval.

Warning!

It is recommended to use express licenses only when you really need to start **DeviceLock EtherSensor** urgently.

Use of express licensing system to issue temporary, test and similar licenses is prohibited. To get these licenses, please send the **request.licx** file to license@microolap.com explaining why you need a license and specifying the license expiration date.

The engineer who issued an express license for a site **has** to send a message to license@microolap.com and ask a replacement with a full license. The **DeviceLock EtherSensor** manufacturer's operator will check the service agreement data and issue the license for this particular installation via the update service.

Important: The update service automatically applies updates once per day. You can expect to automatically receive updates and replacement licenses one day after your application. Alternatively, you can manually install the updates received by the update service.

Warning!

For **DeviceLock EtherSensor** to work normally, the software installation server requires the access to the update service. If the update service is not accessible, we cannot guarantee the software will be operating correctly and won't provide support.

Beta Licenses

The licensing system allows to define beta licenses. Installations with beta licenses receive updates before they are released to all other installations. This allows to have test installations of the product to test new versions.

8. What to Do in Case of Emergency

Below is the procedure for emergency situations.

8.1. Hardware Failure

Below is the list of possible failures of equipment used to run **DeviceLock EtherSensor**, corrective and recovery actions for the sensor.

Unable to start DeviceLock EtherSensor services due to incorrect server reboot or stop.

This emergency is indicated by the (Error 1053) system message telling that the service cannot be started.

If any of the **DeviceLock EtherSensor** services cannot be started, do the following:

- Try to start the services again, taking into account that in order for the **DeviceLock EtherSensor** services to be started, the **EtherSensor Watcher** service must be started.
- If you failed to restart the services, contact support.

There is no free disk space for message interception spools.

The following are the indications of this emergency:

- OS alerts about the lack of free space on the disk.
- Records in the OS system log about the lack of free space on the disk.

The system administrator must find out why the system has run out of space. You can use the **Disk Management** snap-in. If the reason is the spool overflow (**[INSTALLDIR]\data** directory), delete spool data or move them to a free disk partition. Other possible reasons are:

- Server that supposed to receive messages sent from sensor is unavailable. If this is the case, troubleshoot communication errors.
- Too much traffic processed by the sensor: if this is the case, set up additional sensors and distribute the load among them.

DeviceLock EtherSensor server emergency power shutdown..

After the power is restored and the server on which the **DeviceLock EtherSensor** runs is back online, the system administrator must make sure that all the services have been started successfully.

If there are any deviations from the normal startup procedure, inspect service logs. If these logs contain error messages, contact support.

DeviceLock EtherSensor server network disconnect.

The following are the indications of this emergency:

- The network card of the server on which the sensor runs alerts about the mechanical loss of integrity of the communications channel.
- Packets sent over the ICMP protocol to servers for further message processing are not returned or returned partially.

The system administrator must verify and recover the communications channel of the server on which the **DeviceLock EtherSensor** runs. After that, check network settings in the operating system. After the network connection problems are resolved, make sure that all the **DeviceLock EtherSensor** services are operational

8.2. Unauthorized Access to the Software or OS

You can identify unauthorized interference with **DeviceLock EtherSensor** or the OS by the following criteria:

- The Windows security log contains information about attempts of unauthorized access to the system.
- Arbitrary files are deleted or created, or arbitrary services are started in the system.

If any of these indications are present, do the following:

1. Block the sensor's access to the network.
2. Stop **DeviceLock EtherSensor** by stopping all the services.
3. Remove the cause of unauthorized access to the system.
4. Recover **DeviceLock EtherSensor** operation.

9. GUI Localization

In the operation mode, the **Localization** node of the **DeviceLock EtherSensor** configurator is hidden. To make it visible, create a file named **!.xml** in the **lang** subdirectory of the **DeviceLock EtherSensor** installation directory (the file contents may be arbitrary).

After **ethersensor_console.exe** is restarted, the **Localization node will appear in the function tree on the left**. Click it to open GUI localization window:

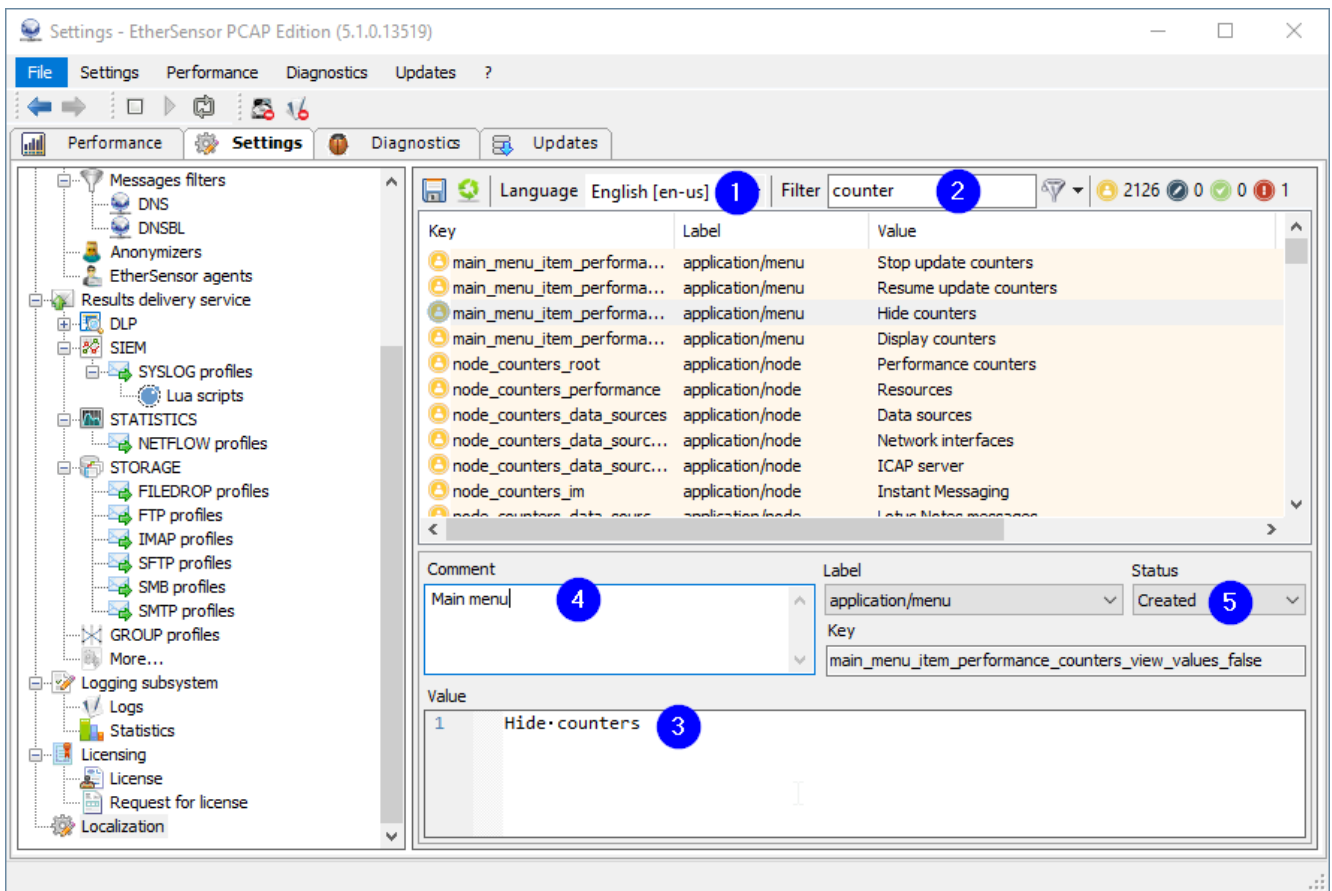


Fig. 57. Localization node window.

Text strings for GUI elements are stored in language XML files in the **lang** subdirectory of the **DeviceLock EtherSensor installation directory**. You can edit these files directly in the **Localization** node window.

To save the changes in a language XML file, press **Ctrl-S**. To make updates in GUI elements visible, press **Ctrl-R**. You can also perform these tasks with the buttons to the left of the **Language** drop-down menu (1) on the toolbar.

Main elements of the localization windows are described below:

1. Language:

Selects the XML language file to edit (to switch the GUI language, use **Settings -- Language**). The labels in this element are defined in the XML language files.

2. Filter:

Filters the string values. You can filter by **Key**, **Label**, **Value**, **Comment**, **Status** columns or by all of them together. You define the filter scope in the drop-down menu next to the funnel icon to the right of the **Filter**.

3. Value:

Here you can edit the **Value** field. The contents of this window is used when displaying the GUI element.

4. Comment:

Here you can edit the **Comment** field. You can enter any text here. It may be a reminder or an explanation like **What did they mean?**, **Need to google the term**, **Need to shorten this**, etc.

5. Status:

Toggles the status of the GUI element. Default status values are: **Created**, **Edited**, **Approved**, **Error**, but you can change statuses right here. For each status there is a defined background color of the interface element string. The counters of elements with different statuses are located on the top right of the localization window. They allow to evaluate the amount of the remaining work.

The **Key** and **Label** fields:

Key:

GUI text elements are stored in the XML file as "key-value" pairs. The **Key** field is a unique key, and the **Value** field contains the text to display in the GUI. All other fields are not required and are only used for convenience.

Label:

Although the developers tried to make the **Key** field values intuitive and hierarchical, they decided it was not enough, so they added the **Label** field to better define the group of elements to which each particular GUI element belongs. To restrict the list of displayed elements to the section defined by the label, enter the respective string in the filter.

You can also group text elements by their group using the **Label** field. The list of labels is hardcoded in **ethersensor_console.exe** and cannot be changed in the current version.


The **Links** field is required **only** for the developers to control unused GUI elements ("orphaned" or saved for the future use).

Tips:

1. To sort the rows in the table of GUI text elements, click the necessary column header. This will help to quickly locate elements by their status, elements that belong to a group or just to alphabetically sort them.
2. If you don't need a field for your work, you can hide the respective column by minimizing its width.

9.1. Language Files

Text strings for GUI elements are stored in language XML files in the **lang** subdirectory of the **DeviceLock EtherSensor** installation directory. The file naming convention is **-.xml**, e.g. **en-us.xml**, **pt-br.xml** etc.



```

1  <?xml version="1.0" encoding="utf-8"?>
2  <root lang="English">
3  <data name="application_name" label="application" status="edited">
4  <value>Microolap EtherSensor</value>
5  </data>
6  <data name="application_caption_performance" label="application/header" status="edited">
7  <value>Performance</value>
8  </data>
9  <data name="application_caption_settings" label="application/header" status="edited">
10 <value>Settings</value>
11 </data>
12 <data name="application_caption_bug_report" label="application/header" status="edited">
13 <value>Diagnostics</value>
14 </data>
15 <data name="application_caption_updater" label="application/header" status="edited">
16 <value>Updates</value>
17 </data>

```

Fig. 58. The contents of a GUI XML language file.

The value of the **lang** attribute of the **root** tag together with the file name are used to display the current language file in the **Language element**.

To start localizing the GUI to a new language, follow this steps:

1. Copy an existing language file, such as **en-us.xml** to a new one, e.g. **pt-br.xml**.
2. Open the newly created **pt-br.xml** in any text editor and change **English** to, for example, **Português brasileiro**.
3. Close and restart the **ethersensor_console.exe** configuration tool. The new language and new file will appear in the **Language** drop-down menu.

Now you can edit GUI text elements in the **pt-br.xml** file.

Tip:

Some GUI elements may contain HTML code. Please edit the XML files in a text editor only if you are absolutely sure in what you're doing. The safest method to edit GUI text elements is doing so in the **Localization** node window.

9.2. Editing GUI Elements

To edit the text of a GUI element, click the **Value** window, edit the text, press **Ctrl-S** (the new element value will be saved to the language file), then press **Ctrl-R** (the system will read the new element value from the language file and display it in the respective element). You can perform these tasks using the buttons to the left of the **Language** drop-down menu.

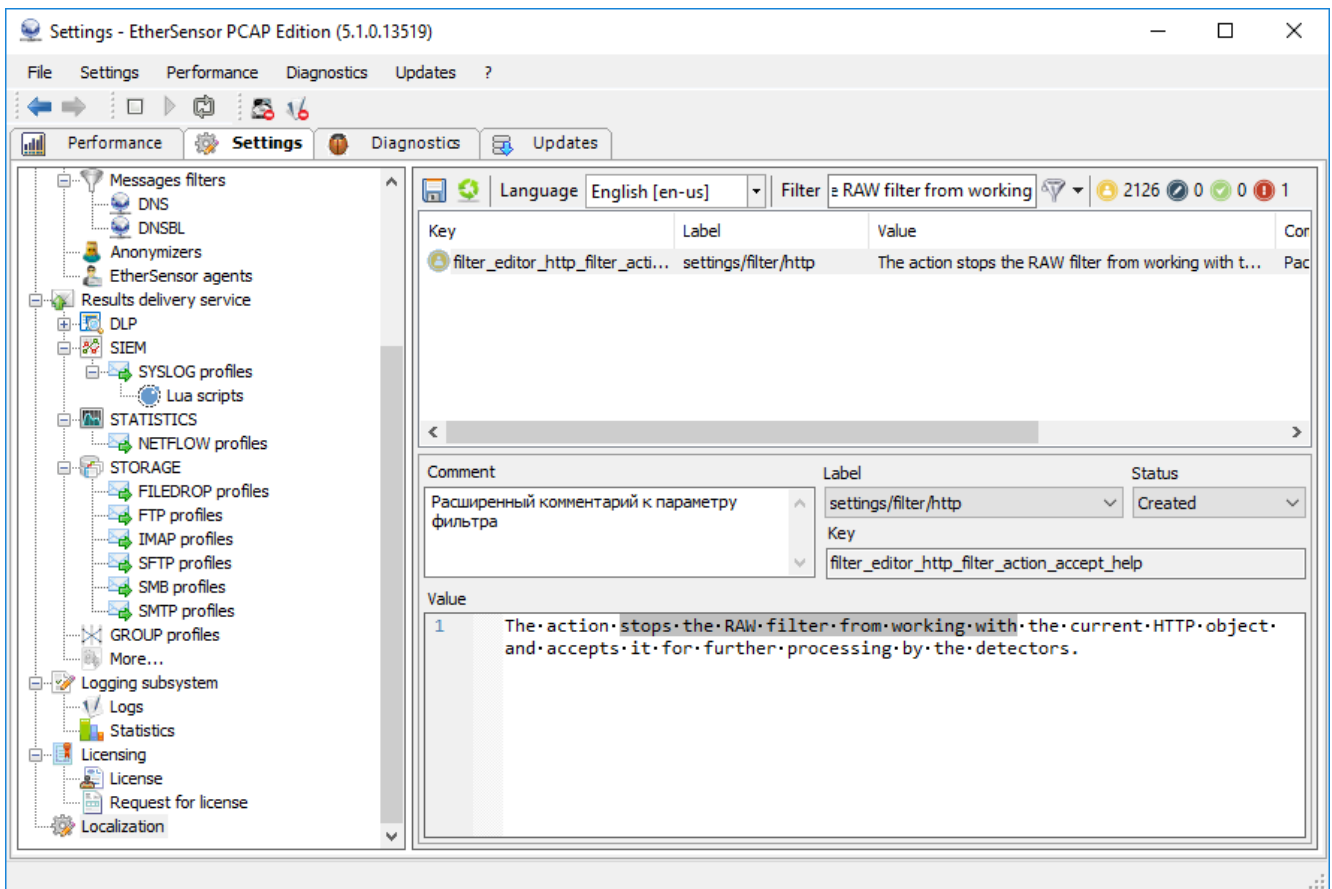


Fig. 59. Editing a GUI element text.

Do not forget to change the element status if necessary.

Tip:

You can simultaneously edit or delete comments from any number of elements as well as change their statuses. To do this, highlight the elements using **Shift-Down/Up** buttons and type a value you want in the **Comment** field or select the status you want.

Editing large elements

Some GUI elements may be as large as HTML pages, e.g. a description of the **Data sources** node:

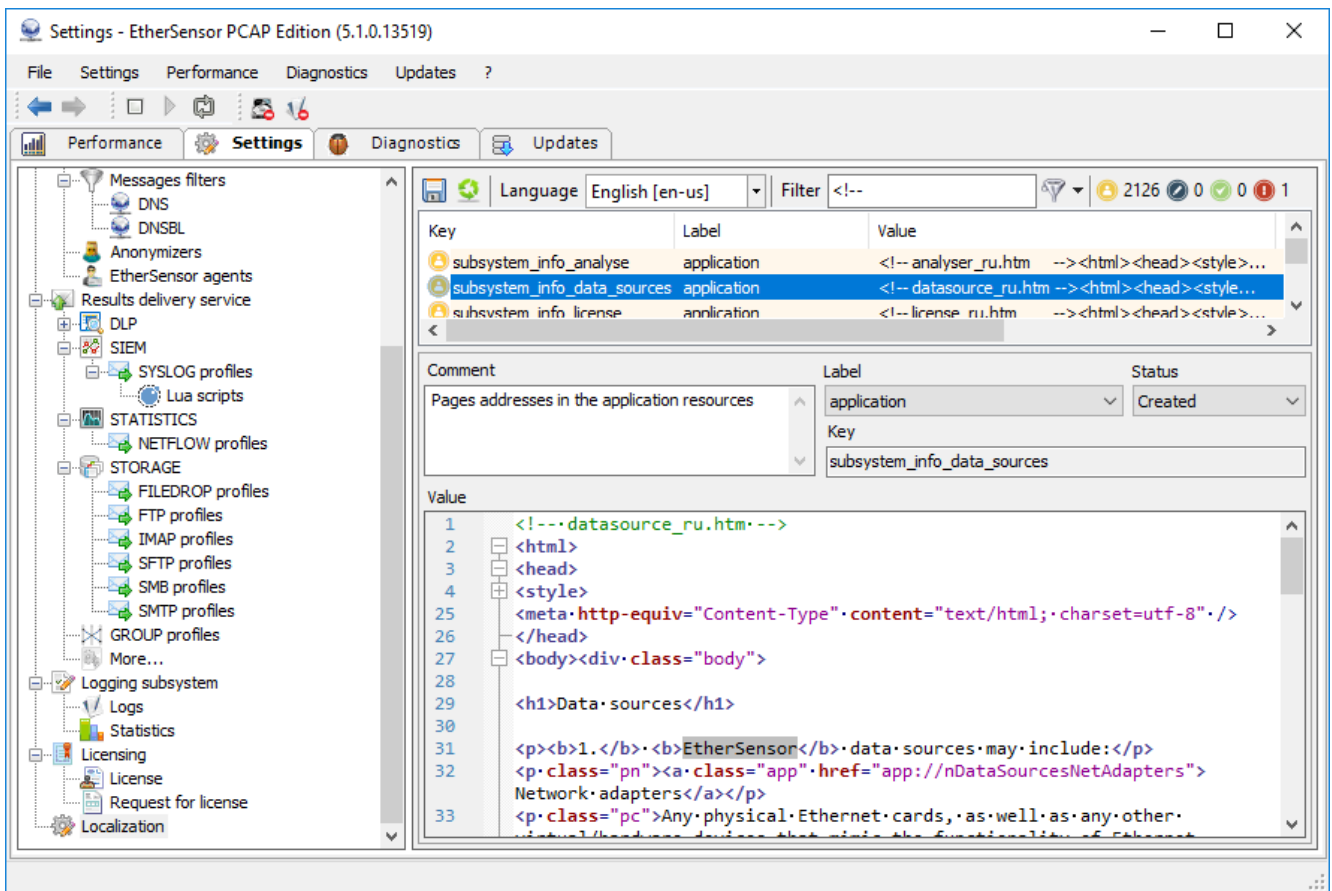


Fig. 60. An example of a GUI element that contains a lot of HTML code.

You can edit these elements in **Value** window, but it may be more convenient to copy the entire element text to an external editor and paste it back to **Value window after editing**.

Tip:

GUI elements that contain HTML, like the one shown above, may include links to configuration tool windows, Help topics, etc. Please be cautious when working with links.

Working with two GUI languages simultaneously

To avoid constant switching between two languages of the GUI language pair, we recommend using two configuration tool windows simultaneously. The procedure is as follows:

1. Create a work folder for the first language, for example **en-us.xml**. name it **C:\Ethersensor-en-us**.
2. Create a work folder for the second language, for example **pt-br.xml**. name it **C:\Ethersensor-pt-br**.
3. Copy the configuration tool application to both folders. In the first folder, in the **lang** subfolder, leave only **en-us.xml** file, and in the second folder, leave only **pt-br.xml**.

Now you can open two configuration tool windows simultaneously and edit the files safely.

Use of regular expressions when searching the element fields (Filter).

In the **Filter** element, add the following character sequence before the text to search: **re.** ("r", "e" and "."). Such strings are interpreted as regular expressions.

For example:

re.[a-zA-Z] will search for elements that contain at least one *Latin letter*.

re.[0-9]\$ will search for elements that end with *a digit*.

re.: \$ will search for elements that end with *colon followed by a space*.

re.^.{4,6}\$ will search for elements that consist of strings of **4 to 6** characters.

10. DeviceLock EtherSensor Changelog

2018-09-08 Version 5.1.0.13519

Runtime environment

Windows Server 2008, Windows Server 2012, Windows Server 2016.

Data sources and objects capture

EtherSensor EtherCAP service:

[+] Implemented support for IPv6.

[+] Improved processing of TCP/UDP traffic with various encapsulation depth levels, for example:

```
eth -> vlan -> ip4 -> ip6 -> ip4 -> gre -> ppp -> ip4 -> tcp/udp -> application
data.
```

[+] The metadata of the events being processed now contains VLAN information.

[+] Interception of FTP over IPv6 is implemented both for active and passive modes.

[+] Implemented the work of IPC channels TCP-SESSIONS and UDP-SESSIONS to deliver the results of the analysis by NETFLOW protocol.

[+] Reduced by 30% consumption of resources for traffic capture.

[-] The error of unpacking socks tunnels has been fixed.

Captured objects analysis:

[+] The internal format of the processed events is changed. On the basis of this, a strong typing of the analysis results is implemented.

Now all the detected events (messages) fit into the following scheme:

```
registration <- Event of registration in the system.
```

```
login <- Logon event.
```

```
profile <- Event of changing the profile in the system.
```

```
send_msg <- Event of sending a message.
```

```
recv_msg <- Event of receiving a message.
```

```
send_file <- Event of sending a file.
```

```
recv_file <- Event of receiving a file.
```

```
conversation <- Event of conversation interception.
```

```
contacts <- Event of contacts forwarding.
```

```
search_query <- Event of search request.
```

[+] Reduced by 15% consumption of resources for messages detecting.

Delivering analysis results to consumer system:

[+] Integration with DLP solution FALCONGAZE Secure Tower has been added.

- [+] Direct delivery of the analysis results from the IPC channel via the SYSLOG protocol was implemented, including the objects received from the RAW filtering in the analysis service. The analysis results maybe delivered either CEF-HTTP or SQUID ACCESS LOG format, the number of consumers is unlimited. Also, TCP over SSL delivery option for analysis results was implemented.
- [+] Direct delivery of analysis results from IPC channel via NETFLOW version 1, 5 and 9 is implemented. Now information about all TCP or UDP sessions processed by EtherSensor can be delivered to NETFLOW servers, the number of consumers is unlimited. TCP over SSL delivery option for analysis results was implemented. Consumers examples: Splunk (NetFlow Analytics for Splunk), McAfee Security Information and Event Management (SIEM), IBM QRadar Security Intelligence Platform, HP ArcSight.
- [+] The LUA integration module for sending results via SYSLOG protocol in CEF format has been updated. Consumers examples: Splunk, HP ArcSight, IBM QRadar, LogRhythm, EMC-RSA NetWitness, McAfee Enterprise Security Manager/NitroView, Symantec Security Information Manager (SSIM).
- [+] Updated results delivery module for JSON format.
- [+] Updated results delivery module for XML format.
- [+] Updated results delivery module for EML format.
- [+] Results delivery resource consumption was reduced.

Logging:

- [+] Added information about processed IPv6 connections.

Configuration console:

- [+] Added interface for managing FALCONGAZE delivery profiles.
- [+] Added interface for managing SYSLOG delivery profiles for direct work with IPC channel.
- [+] Added interface for managing NETFLOW delivery profiles for direct work with IPC channel.

2018-04-25 Version 5.0.3.12929

Runtime environment

Windows Server 2008, Windows Server 2012, Windows Server 2016.

Data sources and objects capture**EtherSensor EtherCAP service:**

- [+] Updated FTP interception in case the client and/or server are behind NAT.
- [+] Improved work with fragmented IP packets.
- [+] Integration with the updated IPC at the operating system kernel level.

Captured objects analysis:

- [+] Updated detectors: odnoklassniki.ru, !generic.
- [+] Added field <PRI> to the CEF log for HTTP requests.
- [-] Fixed the composing of the CEF|SquidAccess string. Excess slash in request following "=" was possible: request=/https://r3--.

Delivering analysis results to consumer system:

- [+] Updated libraries iwthrift.dll, libcrypto-1_1-x64.dll, libcurl.dll, libssh2.dll, libssl-1_1-x64.dll, libxml2.dll, libxmlsec-openssl.dll, libxmlsec.dll, libxslt.dll, zlib1.dll.
- [+] Added download/upload direction for SMB events in the InfoWatch Traffic Monitor.
- [+] Added download/upload direction for FTP events in the InfoWatch Traffic Monitor.
- [-] Fixed a rarely reproducible error when converting messages to EML: in some cases the line breaks in the headers were not handled correctly.
- [-] Fixed a minor error in the calculation of the amount of data sent to the consuming system.
- [+] Updated Lua script for SYSLOG transport, included in the distribution.

- [-] Fixed a bug in Lua engine for SYSLOG transport: incorrect retrieval of message body content in Lua script.

Logging:

- [+] A more accurate calculation and display of performance counters.

Configuration console:

- [+] The ability to delete quoted results is added.
- [+] Added the ability to empty EtherSensor logs.
- [+] The "Apply" button has been added to the service settings: it saves the changes and restarts the service (services).
- [+] The logic of controlling the start and stop of services when updating EtherSensor is improved.

Updater

- [+] Improved the performance of the update service through a proxy server.
- [-] Fixed handling of Russian characters in the update service configuration.

2018-03-20 Version 5.0.2.12765

Data sources and objects capture**EtherSensor EtherCAP service:**

- [+] Traffic capture engine was updated.
- [+] Support for RSS technology. Hardware acceleration support was added for processing traffic in multicore systems using standard equipment.
- [+] Integration with the updated IPC at the operating system kernel level was added.
- [+] Processing of traffic streams up to 20 Gbit was added.
- [+] Resource usage was decreased by a factor of 4.
- [+] Capture and processing of WebSocket protocol was added.
- [+] Capture and processing of SMB1 and SMB2 protocol was added.
- [+] ICQ and MRA protocols were updated.
- [+] WEB tunnels decapsulation (CONNECT method, WebSocket) was added.
- [+] SOCKS tunnels decapsulation, recognition and processing for protocols in SOCKS was added.
- [-] Errors were fixed in IMAP4 processing.

Captured objects analysis:

- [+] IPC (Inter Process Communications) engine was updated.
- [+] Speed of real time data processing was increased.
- [+] Resource usage was decreased by a factor of 2.
- [+] HTTP request processing was extended, support for ACL, AJAX, BAN, BASELINE-CONTROL, BCOPY, BDELETE, BIND, BITS_POST, BMOVE, BPROPFIND, BPROPPATCH, CCM_POST, CHECKIN, CHECKOUT, CONNECT, COPY, DELETE, GET, HEAD, HTML, INVOKE, JSON, LABEL, LINK, LOCK, LOG, MERGE, MKACTIVITY, MKCOL, MKREDIRECTREF, MKWORKSPACE, MOVE, M-SEARCH, NETHCMD, NOTIFY, OPTIONS, ORDERPATCH, PATCH, POLL, POST, PROPFIND, PROPPATCH, PURGE, PUT, REBIND, REPORT, REQMOD, RESPMOD, SEARCH, SCRIPT, SOURCE, SUBSCRIBE, TRACE, UNBIND, UNCHECKOUT, UNLINK, UNLOCK, UNSUBSCRIBE, UPDATE, UPDATEREDIRECTREF, VERSION-CONTROL, X-MS-ENUMATTS methods was added.
- [+] A possibility to generate an HTTP request log in CEF format to deliver to SIEM systems was added to HTTP filter.
- [+] Detection of WebSocket based chats (Skype, Mobile Applications, Web Chats) was added.
- [+] Web WhatsApp (contact lists, user identification) events processing was added.
- [+] Detection of Google Protobuf (Gmail) based messages was added.
- [+] Detection of Web Skype messages was added.
- [+] Detection of Web ICQ, MRA (Mail.ru Group) messages was added.
- [+] Detection of file transfer via SMB1 and SMB2 protocols was added.
- [+] Web detectors were updated: !generic,!fileupload, accounts, facebook.com, google.com, mail.ru, mamba.ru, odnoklassniki.ru, vkontakte.ru, yandex.ru.

Delivering analysis results to consumer system:

- [+] IPC integration was added.
- [+] Resource usage was decreased by a factor of 2.
- [+] Message delivery via SYSLOG protocol. Forming messages for SYSLOG protocol was customized via the integration with Lua script language. A possibility to form messages in custom formats was added.
- [+] Message delivery via SYSLOG protocol using TCP was added, SSL support was implemented.
- [+] A module of LUA integration was added to send via SYSLOG protocol in CEF format. Example consumers: Splunk, HP ArcSight, IBM QRadar, LogRhythm, EMC-RSA NetWitness, McAfee Enterprise Security Manager/NitroView, Symantec Security Information Manager (SSIM).

Logging:

- [+] EtherSensor log records were translated and are now being logged in English.
- [+] HTTP request log in CEF format was added.

Configuration console:

- [+] Update service was integrated into Ethersensor installation package.

2016-09-16 Version 4.5.6.10479

Data sources and objects capture**EtherSensor EtherCAP service:**

- [+] Protocol parser was added for **httpv2**.
- [+] Protocol parser was added for **skype**: interception and transfer of files, text messages, contact lists.
- [-] An error was fixed in http, ftp, and pop3 protocol processing.
- [-] An error was fixed in the processing of the old version configuration.

EtherSensor ICAP service:

- [+] The X-Sensor-Net-Interface-Id header value now includes the port being listened, e.g: **icap-000-1344**.
- [-] An error was fixed in ICAP protocol header processing which resulted in incorrect message generation.

Captured objects analysis:

- [+] The **X-Sensor-Httpv2: Microolap EtherSensor** header was added to distinguish HTTP query filter processing results during the processing of messages transmitted over the httpv2 protocol.
- [+] Processing of messages transmitted over the **skype** protocol.
- [+] The "Check network interface ID (X-Sensor-Net-Interface-Id)" rule was added to the HTTP filter.
- [+] The "Remove message attachment by name" action was added to the message filter.
- [+] The following detectors were updated: !generic,!fileupload,cv (careerist.ru, hh.ru, job.ru, job50.ru, job-mo.ru, jobsmarket.ru, rabotavia.ru, rabota.by, rabota.ru, rosrabota.ru, superjob.ru, zarplata.ru), facebook.com, google.com, gorod55, ipboard, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, mfd.ru, my.mail.ru, mybb, odnoklassniki.ru, phpBB, pochta.ru (qip.ru), rambler.ru, smsmms (skylink.ru, megafon.ru, mts.ru, tele2.ru), twitter.com, vbulletinboard, vkontakte.ru, wordpress.com, yandex.ru, xmpp
- [-] An error was fixed in search query processing.
- [-] An error was fixed in MIME part processing.

Delivering analysis results to consumer system:

- [+] DeviceLock Enterprise Server (DLES) transport was added. It can be used to deliver messages natively to the DeviceLock Enterprise Server.
- [-] An error was fixed in message translation to the EML.
- [-] An error was fixed in message attachment encoding into base64.
- [-] An error was fixed in sending chats to the INFOWATCH Traffic Monitor.

Logging:

- [+] Debug counter logging was added.

Configuration console:

- [+]The "Check network interface ID (X-Sensor-Net-Interface-Id)" rule was added to the HTTP filter.
- [+]The "Remove message attachment by name" action was added to the message filter.
- [+]Message transport profile was added to DeviceLock Enterprise Server.

2016-05-11 Version 4.5.5.10199**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]Code profiling; EtherSensor EtherCAP service performance increased by 20%.
- [-]The protocol parser plug-in system was removed.

Captured objects analysis:

- [*]Code profiling; EtherSensor Analyser service performance increased by 50%.
- [+]Intercepted messages can now be saved in JSON format.
- [+]Message encoding detection reliability was increased.
- [+]Message detection reliability was increased.
- [+]The following detectors were updated: !generic, accounts, chanboard, cv (careerist.ru, hh.ru, hotjob.ru, jobsmarket.ru, job.ru, rabotamedikam.ru, rabotavgorode.ru, rabota.mail.ru, rabota.ru, rosrabota.ru, superjob.ru, zarplata.ru), hotmail.com, facebook.com, google.com, gorod55, icq, ipboard, linkedin.com, livejournal.com, loveplanet.ru, lync, mail.ru, mamba.ru, my.mail.ru, mybb, myspace.com, newmail.ru, nextmail.ru, odnoklassniki.ru, phpBB, pochta.ru (qip.ru), rambler.ru, smsmms (beeline.ru, megafone.ru, mts.ru, tele2.ru), twitter.com, vbulletinboard, vkontakte.ru, ukr.net, wordpress.com, yandex.ru, yahoo.com.
- [-]Message detector plug-in system was removed.
- [-]A rarely reproduced error was fixed which resulted in memory leaks.
- [-]An error was fixed in temporary file processing which resulted in disk space leaks.
- [-]An error was fixed in processing Microsoft Skype for Business messages.

Delivering analysis results to consumer system:

- [+]The **INFOWATCH Traffic Monitor** transport profile was added.
- [-]An error was fixed in message translation to EML.
- [-]A rarely reproduced error was fixed which resulted in memory leaks during EML envelope generation.
- [-]A rarely reproduced error was fixed: sometimes the transport service crashed when configuration was being read.

Configuration console:

- [+]The INFOWATCH Traffic Monitor message transport profile was added.
- [+]The configuration file last updated time was added to **DeviceLock EtherSensor** diagnostic reports.
- [-]An error was fixed in rule attribute processing in message filters.

2016-02-09 Version 4.5.4.9893**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]ICQ protocol parser was updated.
- [*]ICAP protocol parser was updated for integration with ICAP clients via passive traffic listening.

EtherSensor ICAP service:

- [-]An error was fixed with writing to logs.

Captured objects analysis:

- [+]Attachment translation to EML is now faster.
- [+]A mechanism was added to bypass the disk subsystem when passing messages to the transport service.
- [+]Message detection reliability was increased.

[+]The following detectors were updated: !generic, accounts, chanboard, cv (careerist.ru, hh.ru, hotjob.ru, jobsmarket.ru, job.ru, rabotamedikam.ru, rabotavgorode.ru, rabota.mail.ru, rabota.ru, rosrabota.ru, superjob.ru, zarplata.ru), hotmail.com, facebook.com, google.com, gorod55, icq, ipboard, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, my.mail.ru, mybb, myspace.com, newmail.ru, nextmail.ru, odnoklassniki.ru, phpBB, pochta.ru (qip.ru, borda.ru, pochta.com), plaxo.com, rambler.ru, searchquery, smsmms (beeline.ru, megafone.ru, mts.ru, mysmsbox.ru, tele2.ru), twitter.com, vbulletinboard, vkontakte.ru, wordpress.com, yandex.ru, yahoo.com.

[-]A rarely reproduced error was fixed which resulted in memory leaks.

[-]A rarely reproduced error was fixed with message filtering.

Delivering analysis results to consumer system:

[*]DNS Round Robin is now used to distribute the load over consumers of the results of analysis of reconstructed objects.

[+]Error counters were added for the transport service cache.

[-]An error was fixed with EML envelope generation.

Logging:

[+]Counters of the results delivery service cache are now logged to counters.log.

Configuration console:

[*]Separate threads are now used for starting, stopping, pausing and restarting.

[*]User interface of the message filter editor was changed.

[+]Counter descriptions were clarified.

2015-12-16 Version 4.5.3.9707

Configuration console:

[+]Module names in the licensing system were updated.

2015-12-15 Version 4.5.2.9700

Data sources and objects capture

EtherSensor EtherCAP service:

[*]GRE protocol processing was improved.

[*]PCAP file processing was improved.

[+]Support was added for Windows 10 (Windows Server 2016 Preview).

[+]When the service is stopped threads are now forced to close after the 10-second timeout (to make service restart more responsive).

EtherSensor ICAP service:

[+]Reports can now be created when the service crashes.

[+]When the service is stopped threads are now forced to close after the 10-second timeout (to make service restart more responsive).

[-]An error was fixed in the Windows Server 2003 environment.

[-]An error was fixed in interoperation with Allow 204.

[-]An error with the incorrect month in file and folder names for the ICAP Raw dump was fixed.

Captured objects analysis:

[+]When the service is stopped threads are now forced to close after the 10-second timeout (to make service restart more responsive).

[+]The "This is a multipart message in MIME format." body is now forcibly moved to the end of the message body list.

[+]Reports can now be created when the service crashes.

[+]The error log for message detection context initialization was made more detailed.

[-]An error was fixed with the "Write to log" action of the message filter.

[-]An error was fixed in processing data from data\replay directory.

[-]An error was fixed in data processing from EtherSensor agents.

[-]An error was fixed in XML file processing.

[-]An error was fixed in RFC822 EML address parsing.

Delivering analysis results to consumer system:

[+]When the service is stopped threads are now forced to close after the 10-second timeout (to make service restart more responsive).

[-]An error was fixed with sending results from a group profile - a thread using a group profile could go on an infinite loop.

[-]An error was fixed with EML envelope generation.

Logging:

[+]When the service is stopped threads are now forced to close after the 10-second timeout (to make service restart more responsive).

Configuration console:

[+]Counter display was clarified.

[-]An error was fixed with losing focus when tree nodes in the "Performance" tab are switched.

2015-11-11 Version 4.5.1.9623

Data sources and objects capture

EtherSensor EtherCAP service:

[*]Memory loss during traffic capture was significantly reduced.

[*]TCP connection reconstruction performance is now higher in the Packet Sniffer SDK traffic capture library.

[+]VLAN 802.1Q packet processing was added to the Packet Sniffer SDK traffic capture library.

[+]The maximum number of packets held per a TCP thread can not be set in the Packet Sniffer SDK traffic capture library.

[+]Protocol parsers were added for SOCKS4 and SOCKS5. They can be used to monitor and intercept messages tunneled over SOCKS.

[+]ICAP parser was added. It can be used to passively monitor ICAP connections without any changes to the existing architecture.

EtherSensor ICAP service:

[+]SECURE ICAP operating mode was added. The ICAP server can now use SSL to create secure connections to ICAP clients.

[+]LYNC messages on Microsoft Lync (Microsoft Skype for Business) servers can now be intercepted. This feature is integrated with the Microoolap LYNC agent which operates over the ICAP protocol.

Captured objects analysis:

[*]Memory consumption during message detection and analysis is now significantly lower.

[+]Message filters can now send arbitrary (compound) messages to SYSLOG servers for integration with SIEM systems.

[+]Detected search queries for search engines (google.com, rambler.ru, yandex.ru, mail.ru, aport.ru, bing.com, yahoo.com, wikipedia.org) can now be sent over the SYSLOG protocol.

[+]The following message detectors were updated: blogger.com, cv (careerist.ru, hh.ru, job50.ru, job.ru, rabota.ru, superjob.ru, zarpalata.ru), facebook.com, hotmail.com, linkedin.com, livejournal.com, loveplanet.ru, mamba.ru, mail.ru, my.mail.ru, moikrug.ru, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (beeline.ru, megafon.ru, mts.ru, skylink.ru, tele2.ru, wsms.ru), ukr.net, vkontakte.ru (reading incoming messages was added), wordpress.com.

Delivering analysis results to consumer system:

[*]Memory consumption during message sending is now significantly lower.

[+]The new SFTP transport protocol was added to send messages over SSH.

Configuration console:

[+]SFTP transport profile was added.

2015-10-01 Version 4.5.0.9505

Data sources and objects capture

EtherSensor EtherCAP service:

[+]Protocol parsers were added for ICAP and SOCKS.

EtherSensor ICAP service:

[+]The ICAP server now supports Secure ICAP (SSL) secured connections.

[+]LYNC messages on Microsoft Lync servers can now be intercepted. This feature works in combination with the Microolap LYNC agent.

[+]Detected search queries can now be sent over the SYSLOG protocol.

Captured objects analysis:

[*]Memory consumption was significantly reduced.

[+]Protocol detectors were updated.

[+]Incoming vkontakte.ru messages can now be intercepted.

Delivering analysis results to consumer system:

[+]The new SFTP transport protocol was added.

Logging:

[+]Channel names can now be assigned to messages sent over the SYSLOG protocol.

Configuration console:

[+]Statistics can now be sent over the SYSLOG protocol.

2014-07-08 Version 4.4.1.8036

Data sources and objects capture**EtherSensor EtherCAP service:**

[*]The Packet Sniffer SDK traffic capture library was updated: TCP connection reconstruction performance is now higher.

[*]POP3 parser was improved:

Added support for AUTH extension with a multipage response

The situation when Login and Password are not provided is handled correctly.

[+]IMAP4 protocol parser was added.

[+]Protocol parsers were added for NMDC and ADC (DC++).

[+]TORRENT protocol detector for TCP connections was added.

[+]SSL protocol detector was improved (support was added for TLS 1.1 and TLS 1.2).

[-]An error was fixed in the SSL protocol detector which sometimes resulted in insignificant memory leaks.

[-]An error was fixed in calculation of sessions closed by timeout.

[-]An error was fixed in PCAP file processing: "hard" session reset after the end of PCAP file processing was removed; connections are now closed by timeout.

EtherSensor ICAP service:

[-]An error was fixed with X-Sensor-Icap-Authenticated-User and X-Sensor-Icap-Authenticated-Group header transcoding.

[-]An error was fixed in passing requests to the analysis service: in rare cases requests had not been passed for further processing.

Captured objects analysis:

[*]Recognition validity and processing efficiency in the !generic detector are now higher.

[*]Recognition validity for file uploading (downloading) in the !file-upload detector is now higher. URL-based file name generation (when the HTTP query contains no explicit file name) was made more logical.

[+]The following detectors were updated: blogger.com, cv (careerist.ru, hh.ru, job50.ru, job.ru, job-mo.ru, job.ws, jobsmarket.ru, rabotamedikam.ru, rabotavgorode.ru, rabota.mail.ru, rabota.ru, superjob.ru, zarpalata.ru), diary.ru, google.com, gorod55, facebook.com, hotmail.com, linkedin.com, livejournal.com, loveplanet.ru, mamba.ru, mail.ru, my.mail.ru, mfd.ru, moikrug.ru, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (beeline.ru, megafon.ru, mts.ru, skylink.ru, tele2.ru, wsms.ru), twitter.com, yandex.ru, yahoo.com, ukr.net, vkontakte.ru, wordpress.com.

- [+]The OWA (Outlook Web Access) detector was added. It detects messages sent, edited and viewed in the Outlook Web Access system.
- [-]An error with message filter condition processing was fixed which affected filtering message duplicates by the Message-ID field or the MD5 hash of the message (CHECK-MESSAGE-ID, CHECK-MD5). These actions sometimes resulted in errors.
- [-]Corrected the error in processing email messages with empty TO field. Sometimes such processing threw an error.
- [-]An error was fixed with file name processing in the FTP detector. Processing such messages resulted in failure sometimes.
- [-]An error was fixed with HTTP query decoding; sometimes HTTP parameters were left not decoded.
- [-]An error was fixed with IM detector: decoding BASE64 data sometimes resulted in errors.

Delivering analysis results to consumer system:

- [*]SMTP transport performance was improved.

Logging:

- [*]Resource consumption of the DeviceLock EtherSensor counter value collection and storage process was reduced.
- [+]Transport service log entry format was extended: number IDs of sending threads were added to logged messages, allowing to track the event sequence of a given sending thread.

Configuration console:

- [+]Functionality check was added for transport profiles (SMTP, FTP, FILEDROP, SMB). You can now bypass message interception and processing and check how the transport profile operates directly after changing its settings.
- [+]ipconfig /all output was added to the DeviceLock EtherSensor operation report.
- [+]HTTP query filter and message filter editors were added. You can now use the graphic environment to manage filters.
- [+]Editing of quotas for processed results is now done in the quotas.xml file.

2013-11-12 Version 4.4.0.7340

Data sources and objects capture

EtherSensor EtherCAP service:

- [*]The pssdk6.sys traffic interception driver was changed. The old version sometimes resulted in BSOD.
- [*]URI parsing logic was changed in the HTTP protocol parser.
- [*]FTP protocol parser was updated: sometimes it resulted in the crash of the ethcapvc.exe service.
- [*]SMTP protocol parser was updated: multiline server responses for the WELCOME and other commands had been processed incorrectly.
- [*]TCP connections without proper ending according to RFC are now logged with the "warning" status.

Captured objects analysis:

- [*]Recognition validity and processing efficiency for Lotus Notes messages are now higher.
- [*]The (CHECK-MESSAGE-ID) message filter condition for message duplicate filtering by the Message-ID field was updated.
- [+]The X-Sensor-Lotus-MessageId header is now checked for the LOTUS protocol.
- [*]The message filter condition for message filtering by IP addresses was updated. The address check type for "any" addresses was added.

```
<rule enabled="true">
  <match>
    <c name="ip"
      address="any"
      value="10.64.40.24" />
    </match>
    <action name="drop" />
  </rule>
```

[+] The SAVE RAW DATA action was added to the message filter. This message can be used to save source (original) data of the message.

[+] You can now attach source data to the message:

```
<rule enabled="1">
  <comment>For all HTTP objects save original data.</comment>
  <match>
    <c name="protocol" value="http" />
  </match>
  <action name="save-raw-data" value="true" />
</rule>
```

[+] The following detectors were updated: CV (hh.ru, job50.ru, job.ru, job.ws, jobsmarket.ru, rabotamedikam.ru, rabotavgorode.ru, rabota.mail.ru, rabota.ru, rabota.by, superjob.ru), diary.ru, google.com (Google Hangouts web message interception was added), gorod55, facebook.com, linkedin.com, livejournal.com, loveplanet.ru, mamba.ru, mail.ru, my.mail.ru, mfd.ru, moikrug.ru, pochta.ru, smsmms (mysmsbox.ru, megafon.ru, mts.ru, skylink.ru, tele2.ru, wsms.ru), yandex.ru, yahoo.com, ukr.net, vkontakte.ru, wordpress.com.

[+] Reconstruction of files downloaded in parts over the HTTP protocol was added.

[+] An error was fixed with accidentally switching to demo mode with an active license: module license expiration dates were checked incorrectly.

Delivering analysis results to consumer system:

[+] Message headers now include the current DeviceLock EtherSensor UHID: the X-Sensor-UHID header.

[-] An error was fixed in SMTP transport. Sometimes closed connections were used to send messages to archive.

Logging:

[*] Incorrectly closed TCP connections are now logged with the "warning" status. A special rule is created in the default configuration of the watcher service which logs such messages to a separate file.

```
<LogRule output="file://capstrange.log"
  maxsize="10Mb"
  encoding="utf-8"
  newline="CR,LF">
  <Channel name="CAPMAIN" loglevels="error, warning, criterr" />
</LogRule>
```

[-] An error was fixed with the calculation of module expiration time in the logging system messages.

[-] An error was fixed with getting the full path to the message log file.

[-] An error was fixed with saving statistics which sometimes resulted in memory leaks.

Configuration console:

[*] mconsole.exe, kppsreport.exe, perfmonitor.exe utilities were integrated into a single DeviceLock EtherSensor management application - mconsole.exe.

[+] The Ctrl+S hotkey was added to save a modified configuration.

[+] All DeviceLock EtherSensor services can now be stopped, started and restarted at once.

- [+]A DeviceLock EtherSensor diagnostic report can now be unpacked to a separate directory.
- [+]Message and HTTP query filters can now be re-formatted to improve filter readability.
- [-]An error was fixed with filter display (incorrect filter encoding).
- [-]An error was fixed with getting the full path to the message log file.
- [-]An error was fixed with performance counter update.
- [-]An error was fixed with license load and display. Sometimes the application crashed.
- [-]An error was fixed with saving filefrop profile settings.

2013-07-10 Version 4.3.9.7149**Data sources and objects capture****EtherSensor EtherCAP service:**

- [-]An error was fixed in the SSL protocol parser. Sometimes its incorrect operation resulted in the crash of the ethcapvc.exe service.
- [-]An error was fixed with counting recognized connections by protocol parsers.

Captured objects analysis:

- [*]The algorithm of directory spool index recovery was updated. The System.Data.SQLite.dll library (version 1.0.86.0) was updated.
- [+]Support for anonymizers was added: if the anonymizer list is not empty then service detectors process traffic for domains from this list. EtherSensor Analyser service configuration was updated.
- [+]Detector was added for the gorod55.ru service.

Delivering analysis results to consumer system:

- [+]SMB transport profile was added. You can use this profile to write messages to network shares.
- [+]GROUP transport profile was added. This profile is used to reserve transport profiles and balance the load on message consumers.
- [+]The keep-connection flag was added to the SMTP transport profile algorithm, which can be used to enable/disable connection keeping to an SMTP server.
- [+]The profile-fail-timeout option was added to all transport profiles, which can be used to set the time in seconds after which the profile will be locked if messages cannot be sent to that receiver.
- [+]The pgo-profile-reserve option was added to all profiles, which can be used to set the profile reserve flag. This setting is only applicable in a group profile.
- [+]The profile-fail-timeout option was added to all profiles, which can be used to set profile weight. This setting is only applicable in a group profile.

Logging:

- [-]An error was fixed in archiving the **DeviceLock EtherSensor** statistics.

Configuration console:

- [+]Anonymizer domain list management was added.
- [+]The profile-fail-timeout, pgo-profile-reserve, profile-fail-timeout options were added to all transport profiles.
- [+]SMB and GROUP transport profile management was added.

2013-05-30 Version 4.3.8.6986**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]The Packet Sniffer SDK (pssdk.dll) library was updated. The EtherSensor EtherCAP service sometimes crashed with the previous version of this library when intercepted traffic was processed.

EtherSensor ICAP service:

- [+]Support was added for the new "ICAP integration" licensing option.

Captured objects analysis:

[*]The following detectors were updated: CV (hh.ru, jobsmarket.ru, job.ru, rabota.ru, superjob.ru, zarpalata.ru), blogger.com, chanboard, facebook.com, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, my.mail.ru, pochta.ru, rambler.ru, twitter.com, vkontakte.ru, yahoo.com, yandex.ru, ukr.net, wordpress.com.

[+]A new condition for the Message-ID header was added to message filters.

[+]A new action was added to message filters which adds or modifies a certain header in the message (except for From, To, Cc, Bcc, Subject, Date headers).

Delivering analysis results to consumer system:

[*]Service configuration was updated: the save-xheaders option was replaced with the save-headers option in all transport profiles in order to save message headers to a separate attachment named microolap_msis_headers.txt.

[*]SMTP transport performance was improved: now multiple messages can be sent over an existing connection.

Logging:

[*]EtherSensor Watcher service configuration was updated: the gathering of DeviceLock EtherSensor statistics can now be controlled.

[*]The performance of DeviceLock EtherSensor log and statistics archiving was improved, required disk space was reduced.

Configuration console:

[*]The visual appearance of counters was modified.

[+]Added the option to save headers in a separate save-headers attachment to all delivery profile types

[+]The gathering of DeviceLock EtherSensor statistics can now be controlled.

2013-04-08 Version 4.3.7.6840

Captured objects analysis:

[*]E-mail address detection algorithm (From, To, Cc, Bcc) was updated.

[*]The code was profiled to reduce memory usage.

[+]The following detectors were updated: CV (careerist.ru, hh.ru, job-mo.ru, job50.ru, jobsmarket.ru, job.ru, rabotavgorode.ru, rabota.mail.ru, rabota.ru, superjob.ru, zarpalata.ru), blogger.com, facebook.com, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, my.mail.ru, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (beeline.ru, megafon.ru, mts.ru, smste.ru, tele2.ru), twitter.com, vkontakte.ru, yahoo.com, yandex.ru, wordpress.com.

Delivering analysis results to consumer system:

[+]The number of sending threads in the EtherSensor Transfer service can now be managed.

[+]The save-xheaders option was added to all transport profiles in the EtherSensor Transfer service in order to save X-Sensor, X-Sensor headers to a separate message body.

Configuration console:

[*]The visual appearance of counters was modified in the "Performance", "Network adapters", "Intercepted object cache", "Disk quotas" nodes.

[+]Sensor ID can now be edited in the EtherSensor Transfer service configuration.

[+]The number of sending threads of the EtherSensor Transfer service can now be managed depending on the sensor hardware (it can be in the range from 1 to CPU * 2).

[+]The save-xheaders option was added to all transport profiles in order to save X-Sensor headers to a separate message body.

2013-03-11 Version 4.3.6.6714

Data sources and objects capture

[-]An error was fixed in the crashreport.dll exception processing module.

Captured objects analysis:

- [*]E-mail address detection reliability (From, To, Cc, Bcc) was improved.
- [*]The code was profiled to reduce memory usage.

Configuration console:

- [*]Collection of data about PCAP files and minidumps was restricted: up to 50 in a list.

2013-02-21 Version 4.3.5.6608**Captured objects analysis:**

- [+]The following detectors were updated: blogger.com, hotmail.com, myspace.com, moikrug.ru, rambler.com, twitter.com, ukr.net, yahoo.com.
- [-]An error was fixed with analyzing HTTP queries from the DeviceLock EtherSensor ICAP server.

2013-02-14 Version 4.3.4.6584**Data sources and objects capture****EtherSensor EtherCAP service:**

- [+]The service is now forced to restart in case of exceptions. Previously it attempted to continue operation which resulted in uncontrolled memory consumption and other failures (such as a great number of service process dumps).
- [-]An error was fixed in the traffic interception driver, which sometimes resulted in the OS (Win2008/Win7/Win2012/Win8) restarting during uninstallation.

Captured objects analysis:

- [*]Memory and CPU resource consumption was reduced for reconstructed object processing.
- [+]The following detectors were updated: yahoo.com, yandex.com.
- [+]To enable support of DeviceLock EtherSensor agents, the DeviceLock EtherSensor license must now have the option for DeviceLock EtherSensor agents.
- [-]An error was fixed with the analysis of HTTP query form parameters.

Configuration console:

- [-]An error was fixed with diagnostic report generation: sometimes the management console could consume too much memory when the DeviceLock EtherSensor operation report was being generated.
- [-]An error was fixed with checking the version.xml file in the DeviceLock EtherSensor update system.

2013-02-01 Version 4.3.3.6536**Captured objects analysis:**

- [+]Support was added for the new DeviceLock EtherSensor agent message, which allows you to add X-Sensor-UID-AdapterType, X-Sensor-UID-MacAddress headers to interception results.
- [+]The following detectors were updated: !generic.

Configuration console:

- [*]The application now loads faster.
- [*]Corrections were made to bulk filter editing.

2012-12-26 Version 4.3.2.6488**Data sources and objects capture****EtherSensor EtherCAP service:**

- [+]Support was added for a new traffic interception driver. The new driver is now used for installations on Windows Vista and later (up to and including Windows 8/Windows 2012);
- [-]An error was fixed with TCP connection reconstruction. Now the ECE and CWR flags (RFC 3168) are supported.

2012-12-07 Version 4.3.1.6448**Data sources and objects capture****EtherSensor LotusTXN service:**

- [+]Message sent time can now be retrieved.
- [-]An error was fixed which occurred when a network share with Lotus Notes Transaction Log files was being closed.

Captured objects analysis:

- [+]The following detectors were updated: blogger.com, CV (hh.ru, job-mo.ru, job.ru, rabota.ru, rabota.mail.ru, superjob.ru), facebook.com, livejournal.com, linkedin.com, mail.ru, mamba.ru, my.mail.ru, myspace.com, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (megafon, mts), twitter.com, vkontakte.ru, yahoo.com, yandex.ru.
- [-]An error was fixed with email address detection.

Configuration console:

- [+]SMTP port setting was added to the SMTP profile.
- [-]Errors were fixed with adding logging rules.

2012-11-21 Version 4.3.0.6413**Data sources and objects capture**

- [+]Release of the **EtherSensor LotusTXN**, which extracts messages from **Lotus Notes Transaction Log**.

Captured objects analysis:

- [+]Analysis of objects provided by the **EtherSensor LotusTXN service**

2012-11-15 Version 4.2.3.6399**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]The Packet Sniffer SDK traffic capture library was updated. TCP connection reconstruction performance is now higher, memory consumption was reduced.
- [*]The visual appearance of traffic capture performance counters was modified.

EtherSensor LotusTXN service:

- [*]Pre-release of the **EtherSensor LotusTXN** service.

Captured objects analysis:

- [*]A separate queue was implemented for analysis of big raw interception data. Now all cached objects which are dumped to disk due to their size are analyzed in a separate queue in order to save memory.
- [+]The following detectors were updated: blogger.com, CV (hh.ru, job-mo.ru, job.ru, rabota.ru, rabota.mail.ru, superjob.ru), facebook.com (210 new urls were added), google.com, hotmail.com, livejournal.com, linkedin.com, mail.ru, mamba.ru, my.mail.ru, moikrug.ru, myspace.com, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (megafon, mts, skylink, tele2), taba.ru, twitter.com, vkontakte.ru, ukr.net, wordpress.com, yahoo.com, yandex.ru.

Configuration console:

- [-]Errors were fixed in DeviceLock EtherSensor service startup, stop and restart configuration.
- [-]An error was fixed in the display of TCP connection detection counters.
- [-]An error was fixed with display of logs.

2012-08-22 Version 4.2.2.6219**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]The Packet Sniffer SDK traffic capture library was updated. TCP connection reconstruction performance is now higher, memory consumption was significantly reduced.
- [+]Comments can now be added to individual packet filter rules.

Captured objects analysis:

[+]Added the check for physical memory availability. If the amount of available memory is below 50 MB, the cache with the capture results is step by step written to the disk.

[-]An error was fixed with unpacking large archives which resulted in increased memory consumption (the error with unpacking GZIP data in HTTP queries).

Delivering analysis results to consumer system:

[-]An error was fixed with the format of exception messages.

Configuration console:

[-]Errors were fixed in DeviceLock EtherSensor service startup, stop and restart configuration.

2012-08-14 Version 4.2.1.6196

Data sources and objects capture

EtherSensor EtherCAP service:

[*]The Packet Sniffer SDK traffic capture library was updated: TCP connection reconstruction performance is now higher.

[*]Protocol parsing is now faster. Detected TCP sessions for each monitored network adapter are now analyzed in multiple threads simultaneously (the number of threads is equal to the number of CPU * 2).

[*]XMPP protocol parsing algorithm was updated.

[-]An error was fixed which resulted in an exception during the processing of SSL connections.

[-]An error was fixed with unpacking LZ1 attachments in Lotus Notes.

[-]An error was fixed in the processing of PCAP files with packets larger than 1514 bytes.

[-]An error was fixed in BPF filter generation.

Captured objects analysis:

[*]Disk quotas at EtherSensor Analyser startup are now analyzed faster.

[*]The transmission channel for reconstructed objects between the EtherSensor EtherCAP and EtherSensor Analyser services was made 4 times wider.

Please note:

This increases the bandwidth and makes message detection and analysis faster, but hardware requirements to DeviceLock EtherSensor have also increased: 1 GB of RAM is now required. Take this into account when deploying DeviceLock EtherSensor to virtual machines.

[+]Added processing the scheme (GET/PUT/POST) ftp://xxx.xx.xx/, which allows to process/capture files sent/received via FTP OVER HTTP

[+]The following detectors were updated: CV (hh.ru, job-mo.ru, job.ru, rabota.ru, job50.ru, jobsmarket.ru, rabota.mail.ru, rosrabota.ru, superjob.ru), facebook.com, hotmail.com, linkedin.com, livejournal.com, livejournal.ru, loveplanet.ru, mail.ru, my.mail.ru, mamba.ru, meebo.com, moikrug.ru, myspace.com, odnoklassniki.ru, phpbb, pochta.ru, rambler.ru, smsmms, twitter.com, vkontakte.ru, yahoo.com, yandex.ru.

[+]The \data\temp directory for temporary files is now forced to purge at DeviceLock EtherSensor startup.

[-]An error was fixed in the MIME parser: there was a problem with finding boundaries made of '-' characters.

[-]An error was fixed with unpacking data in HTTP queries and responses.

Delivering analysis results to consumer system:

[+]The save-zip option was added to the FTP and FileDrop transport profiles which can be used to compress objects to ZIP files.

Logging:

[+]Functions were added to monitor health and operation of the EtherSensor LotusTXN service.

Configuration console:

[+]The Lotus Notes Transaction Log message extraction service (EtherSensor LotusTXN) can now be managed.

- [+] DeviceLock EtherSensor license and version files can now be added to diagnostic reports.
- [+] EtherSensor LotusTXN service performance counters are now displayed.
- [+] Yahoo protocol counters are now displayed.
- [-] An error was fixed which resulted in memory leaks.
- [-] Errors were fixed in DeviceLock EtherSensor service startup, stop and restart configuration.

2012-06-20 Version 4.2.0.6046**Data sources and objects capture****EtherSensor EtherCAP service:**

- [+] Protocol parsing was added for Yahoo.

EtherSensor LotusTXN service:

- [+] Limited testing was started for the Lotus Notes Transaction Log message extraction service (EtherSensor LotusTXN). This service can be used to monitor and extract messages from the Lotus Notes Transaction Log and pass them for further analysis to the EtherSensor Analyser message detection and analysis service.

Please note:

The service has not been tested for full compatibility with the Lotus Notes Transaction Log "linear" transaction log style.

2012-04-19 Version 4.1.5.5923**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*] The Packet Sniffer SDK traffic capture library was updated. TCP connection reconstruction performance is now higher.

2012-04-16 Version 4.1.4.5917**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*] The Packet Sniffer SDK traffic capture library was updated. TCP connection reconstruction performance is now higher.
- [-] An error was fixed with the interception of proxied connections which use the CONNECT HTTP method. Connections were processed incorrectly when the proxy server required authentication.

EtherSensor ICAP service:

- [-] An error was fixed with unpacking compressed (GZIP) queries.

Captured objects analysis:

- [*] Interaction protocol was updated for the DeviceLock EtherSensor agent.
- [*] Analysis service configuration was updated (interaction settings for DeviceLock EtherSensor agents).
- [*] Email address detection algorithm was modified for web messages.
- [+] The following detectors were updated: CV (hh.ru, job-mo.ru, rabota.mail.ru, zarplata.ru), google.com (Web GTalk support was added), facebook.com, myspace.com, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms, ukr.net, vbulletinboard, vkontakte.ru, yandex.ru.

Logging:

- [-] An error was fixed with log archiving.

Configuration console:

- [*] Display of web message processing counters was updated.

2012-03-27 Version 4.1.3.5862

- [-] The installer for the previous version contained an incorrect version of agent.server.dll.

2012-03-26 Version 4.1.2.5859**Data sources and objects capture****EtherSensor ICAP service:**

- [-]An error was fixed in the processing of HTTP queries with non-transparent proxying. Sometimes it resulted in a failure to process queries with attachments.

Captured objects analysis:

- [-]An error was fixed in the processing of HTTP queries with non-transparent proxying. Sometimes it resulted in queries with attachments failing to be processed.

Configuration console:

- [*]Analysis service configuration was updated: the DeviceLock EtherSensor agent server can now be enabled or disabled. The agent server is disabled by default.
- [-]An error was fixed with the display of ICAP server counters in the configurator (performance monitor).

2012-03-19 Version 4.1.1.9899**Data sources and objects capture****EtherSensor EtherCAP service:**

- [-]An error was fixed with Lotus Notes message processing. Sometimes message and metadata strings were processed incorrectly.

EtherSensor ICAP service:

- [+]An "ICAP protocol synchronous" mode was added. In this mode, the ICAP server first receives the entire request from the ICAP client, and sends the response only after it is received, even in case of large requests (for example, a an ISO image or another large file download/upload). If the synchronous mode is disabled, the ICAP server works in a streaming mode.

This means that the server does not wait until the entire request is received, but starts sending the response as soon as possible. From a user viewpoint, this means no delay in a file transfer. This is important if multimedia traffic (such as video streaming) is passed via ICAP.

Before that, a video stream would be sent from the ICAP proxy to a user only after it was fully downloaded to the ICAP proxy, then to the ICAP server, and then fully received back from the ICAP server. The use of the synchronous mode is required by some ICAP clients that mostly use ICAP to work with antiviruses: to make a decision on what response they send to the ICAP client, antiviruses normally need to receive the entire object (irrespective of its size) first.

Captured objects analysis:

- [+]Built in a UDP server to process messages from DeviceLock EtherSensor agents. If DeviceLock EtherSensor agents are installed and working on the workstation in the organization network, they can deliver the information on user-created connections to the sensor.

The EtherSensor Analyser service uses this information to identify the user during message interception, and to add message owner attributes to messages with the help of headers:

X-Sensor-UID: 0e515c8c-61eb-11e1-a529-000c29ff0707

X-Sensor-UID-UserName: CN=Administrator,CN=Users,DC=bigbrother,DC=foo

X-Sensor-UID-ComputerName: WS325-LOCK.bigbrother.foo

- [+]Support for a new license was added. Versions 4.1 and higher of DeviceLock EtherSensor support licenses with subscription to updates.

[-]An error was fixed with Lotus Notes message processing. Sometimes message and metadata strings were processed incorrectly.

Logging:

- [*]The performance of DeviceLock EtherSensor log archiving was improved, required disk space was reduced.
- [+]Lotus Notes message processing statistics can now be saved.
- [+]Support for a new license was added. Versions 4.1 and higher of DeviceLock EtherSensor support licenses with subscription to updates.

Configuration console:

- [*]EtherSensor ICAP service configuration functions were updated.
- [*]EtherSensor Analyser service configuration functions were updated.

2012-01-25 Version 4.0.14.9561

Data sources and objects capture**EtherSensor EtherCAP service:**

- [+]An algorithm was added to calculate connection timeout based on the number of monitored connections.
- [+]Network adapter can now be reconnected if it was disabled and then enabled again (e.g. via the Windows management console) during EtherSensor EtherCAP operation.

Captured objects analysis:

- [*]Email address parsing algorithm was updated.
- [*]Detectors are now loaded faster.
- [+]LZH attachments (IBM Lotus Notes format) can now be unpacked.
- [+]Counters were added for messages removed by the license unit.
- [+]The following detectors were updated: CV (careerist.ru, hh.ru, job-mo.ru, job50.ru, rabota.mail.ru, rabota.ru, rosrabota.ru, zarplata.ru), facebook.com, hotmail.com, linkedin.com, livejournal.com, mail.ru, mamba.ru, my.mail.ru, rambler.ru, pochta.ru, smsmms (beeline.ru, mts.ru, skylink.ru), vkontakte.ru.
- [-]An error was fixed with Lotus Notes message processing.

Configuration console:

- [+]A validator was added for HTTP object filters and message filters. The filter can now be checked for errors.
- [+]Functions were added for loading and saving individual logs.
- [+]Functions were added to view log contents (as plain text).
- [+]You can now search the log for data (in plain text mode).
- [+]You can now go to a certain log line (in plain text mode).
- [+]Display of Lotus Notes message statistics was added.
- [+]Counters were added for messages removed by the license unit.
- [-]A diagnostics error was fixed which occurred during log loading and display.

2011-12-23 Version 4.0.13.9421

Captured objects analysis:

- [*]The processing algorithm for large HTTP queries was updated. The HTTP filter now processes large queries without loading them into memory. Previously it could result in overconsumption of memory. This change is closely related to the use of filters similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<filter name="HTTP filter" version="1.0">
  <table name="main">

    <rule enabled="1">
      <comment>
        The rule stops processing HTTP objects whose
        request or response size is greater than 100 megabytes.
      </comment>
      <match>
        <c name="size" op="gt" value="100M"/>
      </match>
      <action name="drop" />
    </rule>

    <rule enabled="true">
      <action name="accept" />
    </rule>

  </table>
</filter>
```

[+]The following detectors were updated: yandex.ru.

2011-12-19 Version 4.0.12.9403

Captured objects analysis:

[+]Large attachments to Lotus Notes messages packed with the LZ1 algorithm can now be unpacked.

2011-12-19 Version 4.0.11.9393

Data sources and objects capture

EtherSensor EtherCAP service:

- [*]MRA protocol processing efficiency was improved.
- [*]MSN protocol processing efficiency was improved.
- [*]Saving problem traffic to PCAP files was made more efficient.

EtherSensor ICAP service:

- [-]An error was fixed in IsTag header processing.

Delivering analysis results to consumer system:

- [*]The algorithm of processing locked transport profiles was updated.

Logging:

- [*]Names were changed for directories where DeviceLock EtherSensor statistics is accumulated.

Configuration console:

- [*]Encoding is now specified explicitly in HTML files of the configurator console.
- [-]Errors were fixed in the perfmon utility.
- [-]An error was fixed in the bugreport utility which resulted in the utility freezing.

2011-11-29 Version 4.0.10.9285

Captured objects analysis:

- [+]The following detectors were updated: facebook.com, mail.ru, pochta.ru, rambler.ru, smsmms (mts.ru), yandex.ru (incoming and outgoing messages of the WEB agent are now detected), CV (rabota.ru).

Configuration console:

- [-]Errors were fixed in the perfmon utility.

2011-11-25 Version 4.0.9.9249

Data sources and objects capture

EtherSensor EtherCAP service:

[-]An error was fixed in the service stopping process which sometimes resulted in freezing.

[-]An error was fixed with MSN protocol detection.

Captured objects analysis:

[+]The following detectors were updated: mail.ru (incoming and outgoing messages of the MRA WEB agent are now detected), my.mail.ru, odnoklassniki.ru, vkontakte.ru, CV (job-mo.ru).

[-]An error was fixed with MSN protocol processing.

Delivering analysis results to consumer system:

[+]Information about the transport profile used to send the message is now logged.

Logging:

[-]An error was fixed in the EtherSensor Transfer service startup process which sometimes resulted in freezing.

Configuration console:

[*]The bugreport utility interface was updated.

[-]Errors were fixed in the bugreport utility.

2011-11-23 Version 4.0.8.9215**Captured objects analysis:**

[-]An error was fixed in decoding X-Sensor-Ldap... headers

Delivering analysis results to consumer system:

[*]Log entry format was made more detailed.

Logging:

[*]Log entry format was made more detailed.

Configuration console:

[*]The bugreport utility interface was updated.

[-]Errors were fixed in the bugreport utility.

2011-11-18 Version 4.0.7.9171**Captured objects analysis:**

[*]The following detectors were updated: blogger.com, facebook.com, hotmail.com, linkedin.com, mail.ru, my.mail.ru, meebo.com, mspace.com, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms (megafon, mts, skylink), twitter.com, vkontakte.ru, yahoo.com, CV (careerist.ru, hh.ru, job-mo.ru, job50.ru, rabota.mail.ru, rabotavgorode.ru, rabotavia.ru, rosrabota.ru).

[-]Errors were fixed which made the EtherSensor Analyser service crash when interception results were being processed.

Delivering analysis results to consumer system:

[*]Log entry format was made more detailed.

Logging:

[*]Log entry format was made more detailed.

Configuration console:

[*]Now the bugreport utility collects all the active log entries (previously it only collected criterror, error, and warning entries).

2011-11-10 Version 4.0.6.9037**Data sources and objects capture****EtherSensor EtherCAP service:**

[*]The Packet Sniffer SDK traffic capture library was updated. Errors were fixed with TCP connection reconstruction.

Captured objects analysis:

[-]Errors were fixed which made the EtherSensor Transfer service crash (very rarely) when interception results were being processed.

Delivering analysis results to consumer system:

- [*]The structure and update algorithm of EtherSensor Transfer service counters were modified.

2011-09-29 Version 4.0.5.8903**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]The Packet Sniffer SDK traffic capture library was updated. 64 bit drivers were not signed in the DeviceLock EtherSensor 4.0.4.8875 distribution, so no traffic was captured in Vista/Win7/Win2008 (x64). This error did not appear in most installations (Win2003 x64).

Captured objects analysis:

- [+]Decoding was added for the LMBCS encoding (Lotus Multibyte Character Set). The following encodings are supported: latin-1, greek, hebrew, arabic, cyrillic, latin-2, turkish, thai, unicode-16. Now addresses, subject, body, attachment names and all other headers are decoded completely.

2011-09-22 Version 4.0.4.8875**Data sources and objects capture****EtherSensor EtherCAP service:**

- [+]Detection and reconstruction of Lotus Notes sessions was added.

Captured objects analysis:

- [+]Lotus Notes message detector was added.

2011-09-15 Version 4.0.3.8813**Captured objects analysis:**

- [+]The following detectors were updated: CV (rabotavgorode.ru, rosrabota.ru, careerist.ru, hh.ru, job-mo.ru, rabota.mail.ru, rabotamedikam.ru, zarplata.ru), facebook.com, my.mail.ru, mail.ru, smsmms (megafon, mts, skylink), yahoo.com (incoming mail).

Logging:

- [-]An error was fixed which resulted in memory leaks.

2011-08-18 Version 4.0.2.8709**Data sources and objects capture****EtherSensor EtherCAP service:**

- [+]Statistical processing of the SSL protocol was added: statistics are accumulated for SSL connections, and results are generated with SSL connection lists. You can disable this feature in the configuration (ethcap.xml file) by doing the following:

```
<Protocol enable="false" name="ssl" />
```

- [-]An error was fixed which sometimes resulted in exceptions at service startup.

Captured objects analysis:

- [+]The following detectors were updated: odnoklassniki.ru, pochta.ru, rambler.ru, smsmms, twitter.com, vkontakte.ru, yahoo.com, yandex.ru.
- [+]Now all messages that reach the stage of sending to consumers are marked with the X-Sensor-RawSource-Type header. This is to help the consumer (for example, a message archiving system) to know which initial "raw" captured data were used to get the final message. The following values are currently possible:
 - HttpGetRequest: The message source is an HTTP GET REQUEST
 - HttpPostRequest: The message source is an HTTP POST REQUEST
 - HttpPutRequest: The message source is an HTTP PUT REQUEST
 - FtpFile: The message source is an FTP file
 - SmtPEml: The message source is an SMTP EML
 - Pop3Eml: The message source is a POP3 EML
 - IcqContactList: The message source is an ICQ Contact List

IcqMessageList: The message source is an ICQ Message List
IcqFile: The message source is an ICQ File
IcqLoginInfo: The message source is the ICQ Login Info
MraUserInfo: The message source is the MRA user info
MraContactList: The message source is an MRA Contact List
MraMessageList: The message source is an MRA Message List
MraFile: The message source is an MRA File
MsnContactList: The message source is an MSN Contact List
MsnMessageList: The message source is an MSN Message List
MsnFile: The message source is an MSN File
XmppContactList: The message source is an XMPP Contact List
XmppMessageList: The message source is an XMPP Message List
XmppFile: The message source is an XMPP File
IrcMessageList: The message source is an IRC Message List
IrcFile: The message source is an IRC File
SkypeVersionRequest: The data source is the Get Last version request to ui.skype.com
SslSessionsList: The data source is a list of SSL sessions

Please note:

Certain mailing systems use POST requests to read incoming mail. In this case, interception result for such message will contain the X-Sensor-RawSource-Type header: HttpPostRequest.

- [*] Now all messages which are ready to be transported are labeled with the X-Sensor-LicOption header. This header can have the following values:

WebMail
WebSocial
Email
IM
FT
WebMailRead

- [*] From and To format was updated for WebMail, WebSocial, and WebCV messages.
- [+] Empty message bodies are now checked for and removed for WebMail, WebSocial, and WebCV messages.
- [+] Results are generated to include SSL connection lists.

Delivering analysis results to consumer system:

- [*] Memory consumption was optimized for message delivery.

Configuration console:

- [*] Management interface user interface was updated.
- [*] Integration with the DeviceLock EtherSensor help system was implemented.
- [+] HTTP filter statistics is now displayed.

2011-07-15 Version 4.0.1.8529

Data sources and objects capture

EtherSensor EtherCAP service:

- [*] Traffic processing efficiency is now higher. Incoming HTTP traffic can now be intercepted due to lower resource consumption of DeviceLock EtherSensor and the execution environment.

Captured objects analysis:

- [*] Processing of all available HTTP GET requests from Ethernet and ICAP is guaranteed. Explanation: earlier, the analysis was only available for messages sent from the workstations via the web interface, but now we can also analyze all incoming messages.
- [*] The server can now monitor logins, passwords and downloaded files for GET requests.
- [+] HTTP prefiltering was added to discard obvious junk. This helps to reduce the load on the analysis system. It can also be used for debugging.

Logging:

- [+] You can now select encodings for DeviceLock EtherSensor messages sent to log files or syslog servers; thus, another obstacle to using common log analyzers is removed.

Configuration console:

- [*] The configurator was completely rewritten: this is now a standard Windows application (mconsole.exe file from the distribution kit) instead of an MMC console used previously.
- [+] DeviceLock EtherSensor statistics is now accumulated for integration with external monitoring systems.

2011-06-07 Version 3.0.29.8081**Configuration console:**

- [-] An error was fixed in the bugreport utility which resulted in incorrect report generation.

2011-06-06 Version 3.0.28.8059**Captured objects analysis:**

- [+] The method of detecting IDs from Referer for vkontakte.ru was improved for version 3.x.
- [-] An error was fixed with message generation in WebMail (!generic detector).
- [-] An error was fixed with message generation in IM (IRC and XMPP detectors).
- [-] Errors were fixed with message address filtering.
- [-] An error was fixed with message header generation. The X-Sensor-Smtp-Helo header was generated instead of X-Sensor-Smtp-From.

Configuration console:

- [-] An error was fixed with displaying perpetual licenses.
- [-] Errors were fixed with reading license files from the management console.

2011-04-29 Version 3.0.27.7583**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*] The most probable sources of exceptions for the ftp, icq, irc, msn, mra, smtp protocols were removed.

Captured objects analysis:

- [+] The following service detectors were updated: mail.ru, moikrug.ru.
- [-] An error was fixed which resulted in the EtherSensor Analyser service freezing at startup.

Configuration console:

- [*] The old MMC-based version of the configuration/management subsystem was replaced with a WinFrom application.

2011-03-14 Version 3.0.25.6931**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*] Data processing by protocol parsers is now more reliable. The most probable sources of exceptions for the ftp, icq, irc, msn, mra, smtp protocols were removed.

- [*] Exception handling logic during TCP session reconstruction was modified.

Captured objects analysis:

- [+] The following service detectors were updated: CV (careerist.ru, superjob.ru, zarplata.ru), livejournal.com, facebook.com, mail.ru, myspace.com, odnoklassniki.ru, vkontakte.ru, yandex.ru.
- [-] An error was fixed with caching reconstructed "raw" objects. This error resulted in interception result processing being stopped.
- [-] An error was fixed with the processing of results of reconstructed object filter which raised an exception.

[-]An error was fixed with retrieving DNS names by IP addresses.

[-]An error was fixed with EML address detection.

Delivering analysis results to consumer system:

[-]An error was fixed with address generation (from, to, cc, bcc) in EML envelopes in the SMTP protocol when data is being transmitted to external consumers.

Logging:

[*]Resource consumption was reduced for processes used in DeviceLock EtherSensor statistics maintenance.

2011-02-10 Version 3.0.24.6617

Data sources and objects capture

EtherSensor EtherCAP service:

[-]An error was fixed with HTTP session processing which occurred for header line sizes over 4Kb.

Captured objects analysis:

[+]Compatibility was added for the last version of the script which detects user name and host in HTTP traffic.

[-]An error was fixed with header name parsing for EML envelopes according to RFC-5322.

Logging:

[*]Resource consumption was reduced for the EtherSensor Watcher service.

2011-01-28 Version 3.0.21.6411

Captured objects analysis:

[*]Configuration was updated to version 3.1, a block of parameters was added to detect KPPSU and KPPSH fields (eSafeUser, eSafeHost) in WebMail messages.

[+]The following detectors were updated: CV (careerist.ru, jobsmarket.ru, rabota.mail.ru, rabota.ru, superjob.ru), facebook.com, mail.ru, myspace.com, odnoklassniki.ru, vkontakte.ru.

Logging:

[-]An error was fixed with XML format violation when saving statistics.

[-]An error was fixed: the DumpTime parameter was not updated in top.hostname.XXX.xml when data was accumulated.

2010-12-29 Version 3.0.20.6277

Data sources and objects capture

EtherSensor EtherCAP service:

[*]Recognition validity was improved for the XMPP/Jabber protocol.

Captured objects analysis:

[*]Message processing (detection, filtering) is now faster. Now up to N intercepted objects can be detected and filtered simultaneously. N is calculated as follows: $N = \text{CPU cores} \times 2$.

[+]Updated the following detectors: CV (careerist.ru, hh.ru, job-mo.ru, rabota.mail.ru, rabota.by, rabotavia.ru, superjob.ru, zarplata.ru), facebook.com, livejournal.com, mail.ru, mail.ru-social, mamba.ru, moikrug.ru, myspace.com, odnoklassniki.ru, pochta.ru, vkontakte.ru, yandex.ru.

Logging:

[*]Now the EtherSensor Watcher service maintains data processing statistics and accumulates data in the [INSTALLDIR]\data\statistics directory. Accumulated information is rotated every 2 months. Daily statistics are accumulated in a separate folder inside the [INSTALLDIR]\data\statistics directory. A separate set of files is generated every hour:

top.clients.XXX.xml:

The number of connections established by the client, and the total size of data transmitted over all the sessions created by this client.

top.detectors.XXX.xml:

The number of detected messages.

top.hostname.XXX.xml:

The number of HTTP connections per hour.

Configuration console:

- [*]The bugreport utility can now open reports directly from ZIP archives.
- [*]You can now generate new and detect existing MINIDump files of DeviceLock EtherSensor services.
- [*]You can now detect PCAP files with packets interception of which raised a data processing exception.

2010-12-02 Version 3.0.19.6031**Captured objects analysis:**

- [-]An error was fixed in message filters with the check-md5 status check.

Configuration console:

- [-]An error was fixed which occurred during report generation by the bugreport utility.

2010-10-25 Version 3.0.18.6027**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]Recognition validity was improved for the MRA (MRIM) protocol. The new protocol version was released, supported by new clients.
- [+]The "FROZEN" state of the EtherSensor EtherCAP service can now be detected and rectified. The service is in the FROZEN state when it has stopped traffic interception due to internal errors or an attack. The packets the processing of which resulted in the FROZEN state of the service are saved to a PCAP file in the \log\pcaps directory.
- [-]An error was fixed in the MRA(MRIM) protocol parser which resulted in the EtherSensor EtherCAP service freezing.

EtherSensor ICAP service:

- [*]The string collection buffer size was increased to 32K (it was 8K according to the standard, but the size of individual strings was sometimes over 9.5K).
- [*]HTTP commands longer than 7 characters were previously not processed; now this length is increased to 32 characters.
- [*]Service configuration was also updated:

```
<?xml version="1.0" encoding="utf-8"?>
<IcapConfig version="3.0">
  <SensorId>icap-01</SensorId>
  <Network max_connections="4000">
    <ListenAddress address="0.0.0.0:1344" />
  </Network>
  <Icap>
    <Preview enabled="false" size="4096" />
    <Allow204 enabled="true" />
    <RawLog enabled="false" path=".\\raw-log" />
    <RequestLog enabled="false"
      http_enabled="true"
      channel="ICAP-REQUEST" />
    <AlwaysOk enabled="true" />
    <Header name="X-Client-IP" enabled="true" />
    <Header name="X-Server-IP" enabled="false" />
  </Icap>
</IcapConfig>
```

New tags added:

AlwaysOk

The AlwaysOk tag is nested within the Icap tag and enables the "always ok" mode. In this mode, the ICAP server responds with code 204 "No modifications" to any errors detected in the ICAP protocol on the client side.

The enabled attribute of the AlwaysOk tag specifies whether the "always ok" mode is active:

enabled="true" - the mode is active.
enabled="false" - the mode is inactive.

If the AlwaysOk tag is omitted, this mode is assumed to be disabled by default.

If the mode is active, the 204 response code is sent in return to errors even if it is prohibited by Allow204.

You should only enable this mode in rare cases when you are absolutely sure about what you are doing, because it may result in unpredictable failures in the operation of the ICAP client providing traffic.

RequestLog

The RequestLog tag is nested within the lcap tag and defines the error logging mode settings for ICAP and HTTP query protocols processed by the ICAP server. This log stores errors which may occur when ICAP client and server communicate, and are related to incorrect data formats sent by the ICAP client, misuse of the ICAP protocol, etc.

The enabled attribute of the RequestLog tag specifies whether the error logging mode is enabled for the ICAP and HTTP protocols:

enabled="true" - the logging mode is active.
enabled="false" - the logging mode is inactive.

The http_enabled attribute of the RequestLog tag specifies whether the error logging mode is enabled for HTTP queries:

enabled="true" - the logging mode for HTTP queries is active.
enabled="false" - the logging mode for HTTP queries is inactive.
http_enabled="true" is assumed by default (if the attribute is omitted).

The channel attribute of the RequestLog tag specifies the internal system name for the HTTP query logging channel. This parameter must always be channel="ICAP-REQUEST" and may only be modified on the direct instruction of the DeviceLock EtherSensor developer.

Header

The Header tag controls extended headers of the ICAP protocol. This tag can be used to notify the ICAP client whether the server supports the specified header. This can be used to allow or disallow the ICAP client to send the corresponding header to the server.

The enabled attribute of the Header tag specifies whether support for this header is enabled:

enabled="true" - the header is supported.
enabled="false" - the header is not supported.

If the tag is omitted, the header is assumed to be supported by default.

The name attribute of the Header tag specifies the name of the extended header.

The following extended ICAP headers are supported:

X-Client-IP

X-Server-IP
X-Client-Username
X-Subscriber-ID
X-Authenticated-User
X-Authenticated-Groups

All the headers are assumed to be supported by default (if the Header tags are omitted in the configuration file).

Captured objects analysis:

- [*]Message filtering was made faster.
- [+]The following detectors were updated: CV (careerist.ru, hh.ru, job.ru, job50.ru, jobsmarket.ru, rabota.mail.ru, rabotavgorode.ru, superjob.ru, zarplata.ru), facebook.com, livejournal.com, mail.ru, mail.ru-social, mamba.ru, moikrug.ru, meebo.com, myspace.com, odnoklassniki.ru, pochta.ru, smsmms (mts), twitter.com, vkontakte.ru, yandex.ru.
- [-]An error was fixed with url-encoded check.
- [-]An error was fixed with the Content-Type MIME header parsing.

2010-10-01 Version 3.0.17.5713

Data sources and objects capture

EtherSensor EtherCAP service:

- [*]TCP connection reconstruction performance is now higher in the Packet Sniffer SDK library.
- [-]An error was fixed with generating large IP filters.

Captured objects analysis:

- [+]The following detectors were updated: CV (careerist.ru, hh.ru, job-mo.ru, job.ru, jobsmarket.ru, rabota.ru, superjob.ru, zarplata.ru; services added: job50.ru, rabotavgorode.ru, rabota.mail.ru), moikrug.ru, facebook.com, livejournal.com, loveplanet.ru, mail.ru, mail.ru-social, mamba.ru, meebo.com, myspace.com, odnoklassniki.ru, rambler.ru, smsmms, taba.ru, twitter.com, vkontakte.ru, yandex.ru.

Configuration console:

- [+]A report can now be generated with DeviceLock EtherSensor counter values and details.
- [+]The first release of the bugreport utility which can collect operation information for the DeviceLock EtherSensor system and the EtherSensor Updater automatic update service and send it to the DeviceLock EtherSensor developer for analysis.

2010-09-14 Version 3.0.16.5573

Captured objects analysis:

- [*]Name generation logic for reconstructed objects was updated. The "unknown" name is now generated for the "image/*" Content-Type and unknown file names.<subtype>. I.e. the name will be unknown.jpeg for image.jpeg, unknown.gif for image/gif, etc.
- [+]The following detectors were updated: CV (careerist.ru, funkyjob.ru, hh.ru, job-mo.ru, job.ru, jobsmarket.ru, rabota.ru, superjob.ru, zarplata.ru), hotmail.com, moikrug.ru, facebook.com, linkedin.com, livejournal.com, mail.ru, mail.ru-social, mamba.ru, myspace.com, odnoklassniki.ru, rambler.ru, smsmms, twitter.com, vkontakte.ru, yandex.ru.
- [-]An error was fixed with envelope generation for MSN messages and contact lists.

Delivering analysis results to consumer system:

- [-]An error was fixed with incorrect encoding of message subjects and other headers.

Configuration console:

- [-]A logical error was fixed with setting and reading disk quota values.

2010-08-27 Version 3.0.13.5377

Data sources and objects capture

EtherSensor EtherCAP service:

[-]An error was fixed with protocol analyzer generation, which resulted in the crash of the EtherSensor EtherCAP service.

Captured objects analysis:

[+]Message decoding based on the Content-Transfer-Encoding property was added for messages only containing an attachment and being not MIME-encoded.

[+]The following detectors were updated: CV (zarplata.ru), moikrug.ru, facebook.com, mail.ru, mail.ru-social, mamba.ru, odnoklassniki.ru, rambler.ru, twitter.com, vkontakte.ru, yandex.ru.

[-]An error was fixed in the DNSBL action in filter rules.

[-]An error was fixed with the processing incorrect messages without bodies.

[-]An error was fixed in the EML parser: there were problems with MIME decoding with transport encoding other than binary.

Delivering analysis results to consumer system:

[+]Transport thread pool was implemented. Now there are up to 10 threads (instead of 1 thread) which send analysis results to consumer systems, depending in the data size of reconstructed objects. This makes the EtherSensor Transfer service significantly faster.

2010-07-26 Version 3.0.12.5119

Captured objects analysis:

[+]The following detectors were updated: CV (careerist.ru, hh.ru, job-mo.ru, job.ru, rabota.ru, superjob.ru, zarplata.ru), moikrug.ru, facebook.com, loveplanet.ru, mail.ru, mail.ru-social, mamba.ru, meebo.com, odnoklassniki.ru, twitter.com, vkontakte.ru, yandex.ru.

[-]An error was fixed in the filter rule condition which used MD5 to find message duplicates.

2010-07-19 Version 3.0.11.4969

Captured objects analysis:

[+]A filter rule condition was added to find message duplicates by MD5.

[+]The following detectors were updated: chanboard, CV (hh.ru, job-mo.ru, job.ru, rabota.ru, superjob.ru, zarplata.ru), moikrug.ru, myspace.com, my.mail.ru, facebook.com, taba.ru, twitter.com.

[-]An error was fixed with X-Sensor-Ldap-Hostname, X-Sensor-Ldap-User header detection.

2010-07-26 Version 2.0.17.4175

Captured objects analysis:

[+]The following detectors were updated: facebook.com, loveplanet.ru, mail.ru, mail.ru-social, mamba.ru, odnoklassniki.ru, vkontakte.ru, yandex.ru.

Configuration console:

[-]An error was fixed in the MMC management console which resulted in an exception when interception result transport rules were being edited.

2010-07-23 Version 2.0.16.4170

Captured objects analysis:

[+]The following detectors were updated: accounts, facebook.com, loveplanet.ru, mail.ru, mail.ru-social, mamba.ru, meebo.com, myspace.com, odnoklassniki.ru, pochta.ru, twitter.com, vkontakte.ru, yandex.ru.

Delivering analysis results to consumer system:

[+]Transport thread pool was implemented. Now there are up to 10 threads (instead of 1 thread in the previous version) which send analysis results to

consumer systems, depending in the data size of reconstructed objects. This makes the EtherSensor Transfer service significantly faster.

2010-07-09 Version 2.0.15.4019**Captured objects analysis:**

- [+] X-Sensor-Ldap-Hostname and X-Sensor-Ldap-User headers are now generated in the EML envelope if the User-Agent query field contains the eSafeHost and eSafeUser fields.
- [+] The following detectors were updated: accounts, blogger.com, facebook.com, fileupload, !generic, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, meebo.com, myspace.com, odnoklassniki.ru, plaxo.com, pochta.ru, smsmms, twitter.com, vkontakte.ru, yandex.ru.
- [+] The following detectors were added: moimir.mail.ru.

2010-05-18 Version 2.0.14.3656**Captured objects analysis:**

- [+] Generation of the following EML envelope headers was implemented from the data provided by the ICAP server:
 - X-Sensor-Icap-Client-Username
 - X-Sensor-Icap-Subscriber-Id
 - X-Sensor-Icap-Authenticated-User
 - X-Sensor-Icap-Authenticated-Group

2010-05-06 Version 2.0.13.3595**Delivering analysis results to consumer system:**

- [-] An error was fixed with EML envelope processing in the SMTP client of the transport service.

2010-04-30 Version 2.0.12.3570**Delivering analysis results to consumer system:**

- [-] An error was fixed with the BDAT command processing in the SMTP client of the transport service.

2010-04-28 Version 2.0.11.3551**Captured objects analysis:**

- [+] Processing of binary messages (Exchange 2010) was added. Now all binary messages are received as attachments.
- [+] The following detectors were updated: facebook.com, linkedin.com, livejournal.com, loveplanet.ru, mamba.ru, myspace.com, odnoklassniki.ru, twitter.com, vkontakte.ru, yandex.ru.

2010-03-09 Version 2.0.11.2767**Data sources and objects capture****EtherSensor ICAP service:**

- [*] Compliance with the updated draft-stecher-icap-subid-00.txt.
 - When the ICAP client sends the X-Subscriber-ID, X-Authenticated-User, X-Authenticated-Groups headers, they are now saved as data file properties in the passport: icap-x-subscriber-id, icap-x-authenticated-user, icap-x-authenticated-group correspondingly.
 - If the X-Authenticated-Groups contains multiple groups, each of them is saved as a separate icap-x-authenticated-group property which contains only one group.
- [+] New debug counters were added:
 - \ICAP\Request\POST
 - The number of POST requests passed by ICAP to REQMOD.

 - \ICAP\Request\PUT

The number of PUT requests passed by ICAP to REQMOD.

\ICAP\Request\Open

The number of files opened to save POST/PUT in REQMOD.

\ICAP\Request\Open-active

The number of files currently open to save POST/PUT in REQMOD.

\ICAP\Request\Closed-and-save

The number of files closed with the "save to cache" status (they must be in the cache).

\ICAP\Request\Closed-and-delete

The number of files closed with the "remove from cache" status (due to some errors).

- [+] Filtering for raw-log was implemented in REQMOD. Requests to Host: icap.health.check result in the removal of raw-log files for this TCP session. The filter is currently hard-coded.
- [+] Now if the ICAP client sends the X-Client-Username header, it is saved as a data file property in the icap-x-client-username passport. The header may be sent by the SQUID server if it has user authentication configured. Configuring `icap_send_client_username` on/off in `squid.conf` file. Other clients may also send this header, but we don't know it for sure.

2010-01-28 Version 2.0.10.2766

Captured objects analysis:

- [-] An error was fixed in the SMTP parser which was introduced in the 2.0.9.2741 release due to changes made to capture the Exchange server traffic.

2010-01-26 Version 2.0.9.2741

Captured objects analysis:

- [+] The following detectors were updated: chanboard, facebook.com, hotmail.com, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, meebo.com, myspace.com, nextmail.ru, odnoklassniki.ru, plaxo.com, pochta.ru, rambler.ru, smsmms, twitter.com, vkontakte.ru, yahoo.com, yandex.ru.
- [+] The following detectors were added: The accounts detector: it detects user registration events on remove web services. In addition to logins and passwords, it collects additional details submitted by the user during at registration (addresses, emails, names, phones, nicks, descriptions, etc.). You can use this detector to determine the resources visited by network users at the level of intercepted authentication events for these resources.

2009-09-03 Version 2.0.5.2500

Captured objects analysis:

- [+] The following detectors were updated: facebook.com, hotmail.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, meebo.com, myspace.com, nextmail.ru, odnoklassniki.ru, pochta.ru, rambler.ru, smsmms, twitter.com, vbulletin, vkontakte.ru, yandex.ru, icq.
- [-] Errors were fixed in the HTTP protocol parser.
- [-] Errors were fixed in the MRA protocol parser.

2009-08-03 Version 2.0.4.2360

Captured objects analysis:

- [+] The following detectors were updated: chanboard, facebook.com, hotmail.com, linkedin.com, livejournal.com, loveplanet.ru, mail.ru, mamba.ru, meebo.com, myspace.com, nextmail.ru, odnoklassniki.ru, rambler.ru, smsmms, twitter.com, ukr.net, vkontakte.ru, wordpress.com, yandex.ru.

[+]The following detectors were added: gazup.com, ifolder.ru.

2009-05-31 Version 2.0.0.2300

Captured objects analysis:

[+]New detectors:

facebook.com

New social network user registration event.

Social network user profile update event.

Events transmitted by social network users.

File uploads (downloads) by social network users.

linkedin.com

New social network user registration event.

Social network user profile update event.

Events transmitted by social network users.

File uploads (downloads) by social network users.

meebo.com

Messages exchanged by meebo.com users.

myspace.com

New social network user registration event.

Social network user profile update event.

Events transmitted by social network users.

File uploads (downloads) by social network users.

plaxo.com

New social network user registration event.

Social network user profile update event.

Events transmitted by social network users.

File uploads (downloads) by social network users.

twitter.com

New social network user registration event.

Social network user profile update event.

Events transmitted by social network users.

File uploads (downloads) by social network users.

wordpress.com

Messages posted by users.

mail.ru

Detection of instant online messages exchanged by mail.ru users via MRA.

2009-02-26 Version 1.0.8

Data sources and objects capture

EtherSensor EtherCAP service:

[*]TCP session reconstruction performance is now higher in the Packet Sniffer SDK library.

Captured objects analysis:

[*]Processing of new messages in WebMail and WebSocial was added.

[+]Detection and reconstruction of ICQ and MRA objects (currently only for contact lists and text messages) was added.

Delivering analysis results to consumer system:

[-]An error was fixed with sending messages to consumers over SMTP in the DeviceLock EtherSensor client.

Logging:

[+] Statistics of results sent to consumers by modules was added (the !translag.log file).

2008-10-29 Version 1.0.7.7

Data sources and objects capture

[+] The 32 bit version of DeviceLock EtherSensor was added.

Delivering analysis results to consumer system:

[-] An error was fixed which resulted in duplicate messages sent to consumers and incorrect counters displayed in log files.

2008-10-27 Version 1.0.7

Data sources and objects capture

EtherSensor EtherCAP service:

[*] Packet Sniffer SDK traffic capture library version 5.0. This version increases traffic capture and connection reconstruction performance by 8-10 times compared to the previous version 4.x.

Captured objects analysis:

[*] Performance of modules which process objects from the odnoklassniki.ru and vkontakte.ru services is now higher.

[+] The following odnoklassniki.ru domains were added: odnoklassniki.ru, odnoklasniki.ru, odnoklassniki.ua, odnoklassniki.kz.

[+] Modules were added to implement reconstruction of HTTP objects for the following services: loveplanet.ru, mamba.ru, livejournal.com.

loveplanet.ru domains added:

loveplanet.ru, alovely.ru, datelove.ru, dating.wmj.ru, flirt.me-to-you.info, flirt2008.ru, loveplanet-vip.ru, lovemir.com, love.efremov.net, love.livedate.ru, love-planet.ru, loveplanet.ru, loveplanet.ru, loveplanet-vip.ru, lovemoskva.ru, lov1.ru, loveopen.ru, lovedrom.ru, lovepeace.ru, mos-love.ru, planet.sbrn.ru, dating-loveplanet.ru, dating-znakomstva.ru, lovo.ru;

mamba.ru domains:

4love.ru, date.datinglove.ru, dating.freetime.com.ua, explore.ru, flirtru.ru, flirt77.ru, fastlove.ru, facelink.ru, greatlove.ru, holiday.ru, iloveyou.ru, jdu.ru, jzzz.info, lovemy.com.ua, love.alfa-beta.ru, love.azlyrics.ru, love.butt-head.ru, love.girlzzz.info, love.lovz.ru, love.mail.ru, love.rambler.ru, love-kiss-dating.ru, love.neolove.ru, love.pautinka.ru, love.primochka.ru, love.gay.ru, love.ignio.com, love.lesbi.ru, love.russianchat.ru, lov.ru, lovedate.ru, mamba.ru, mambo.ru, mheart.ru, romantica.ru, search.all4love.ru, svidanka.ru, vstret.ru, znakomstva.lt, znakomstva.lv, znakomstva.odnoklassniki.ru, younglover.ru, lovedosug.ru, lubovy.ru;

livejournal.com domains:

livejournal.com, livejournal.ru.

Logging:

[+] Extended the set of counters that show DeviceLock EtherSensor performance indicators. The counters are displayed in real time in !capstat.log, !sysstat.log, !transtat.log log files.

[+] The log entry was made more detailed for the SMTP transport which sends analysis results to consumer systems.

Configuration console:

[+] The crashreport.dll module was added which generates exception reports when DeviceLock EtherSensor services crash. The exception report is saved to the log\crashrpt directory. The exception report consists of two files: 1) application minidump; 2) execution environment report.

- [+]The DeviceLock EtherSensor installer enables automatic startup for services in case of a crash or a exception.
- [-]An error was fixed with a long startup of DeviceLock EtherSensor services after an operating system restart.

2008-07-28 Version 1.0.5**Captured objects analysis:**

- [+]MD5 Hash is now calculated for reconstructed objects to identify duplicates.
- [+]The following detectors were updated: mail.ru, rambler.ru, yandex.ru, pochta.ru, google.com, yahoo.com, ukr.net, odnoklassniki.ru, vkontakte.ru, squirrel-mail, file-upload, hotmail.com, nextmail.ru, newmail.ru, phpBB.

Delivering analysis results to consumer system:

- [+]The X-Sensor-Object-MD5Hash header was added to messages sent to consumers over SMTP.
- [-]A logic error was fixed with sending messages to consumers over SMTP.

2008-07-28 Version 1.0.4**Captured objects analysis:**

- [+]The following detectors were updated: mail.ru, rambler.ru, yandex.ru, pochta.ru, google.com, yahoo.com, ukr.net, odnoklassniki.ru, vkontakte.ru, squirrel-mail, file-upload, hotmail.com, nextmail.ru.
- [+]The following detectors were added: newmail.ru, phpBB.

Configuration console:

- [+]Logging settings can now be configured.

2008-07-25 Version 1.0.2**Captured objects analysis:**

- [+]The following detectors were added: hotmail.com, nextmail.ru.

Delivering analysis results to consumer system:

- [-]An error was fixed with the "Sensor-Id" header.

2008-07-16 Version 1.0.1**Data sources and objects capture****EtherSensor EtherCAP service:**

- [*]Memory consumption was optimized for traffic capture.

Delivering analysis results to consumer system:

- [+]The following headers were added for EML generation to be sent to the consumer over SMTP:
 - MailFrom/RcpTo
 - Via/X-Forwarded
 - Date
 - Network-If-Id (you can now determine exactly the network interface from which the reconstructed object was captured).
- [+]The header with the size of the reconstructed object.

Logging:

- [+]Log archiving was added to save disk space.

Configuration console:

- [+]The configurator MMC console was implemented.