Processing and storing artifacts

Artifact analysis training material

November 2014

European Union Agency for Network and Information Security www.enisa.europa.eu

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Authors

This document was created by Lauri Palkmets, Cosmin Ciobanu, Yonas Leguesse, and Christos Sidiropoulos in consultation with DFN-CERT Services[1] (Germany), ComCERT[2] (Poland), and S-CURE[3] (The Netherlands).

Contact

For contacting the authors please use cert-relations@enisa.europa.eu

For media enquires about this paper, please use press@enisa.europa.eu

Acknowledgements

ENISA wants to thank all institutions and persons who contributed to this document. A special 'Thank You' goes to Todor Dragostinov from ESMIS, Bulgaria.

[1] Klaus Möller, and Mirko Wollenberg
[2] Mirosław Maj, Tomasz Chlebowski, Krystian Kochanowski, Dawid Osojca, Paweł Weżgowiec, and Adam Ziaja
[3] Michael Potter, Alan Robinson, and Don Stikvoort

Legal notice

Copyright Notice

Table of Contents

Main Objective	Present the trainees various methods of potentially malicious artifacts acquisition methods with emphasis on artifacts collected through spam monitoring. Teach how to set up spam collecting environment and artifacts repository. The exercise also provides knowledge how to modify and patch created system to better suit environment needs.	
Targeted Audience	The exercise is dedicated to CERT staff involved in new threats detection and analysis. The exercise should be also helpful to CERT staff involved in malicious artifacts analysis as it presents how to create and use artifacts repository.	
Total Duration	4.5 hours	
Time Schedule	Introduction to the exercise	0.5 hours
	Task 1: Spam trap configuration and usage	1.0 hours
	Task 2: General methods for building the storage for artifacts	1.5 hours
	Task 3: Spam content analysis methods	1.0 hours
	Summary of the exercise	0.5 hours
Frequency	It's advised to organise this training when new team members who are involved in threat detection or malicious artifact analysis join a CERT.	

1 General description

The aim of this exercise is to show participants different methods of collecting, sorting and storing artifacts. During the exercise trainees will obtain artifacts from spam emails, and then store them in the configured storage.

In the first phase, participants will configure Shiva honeypot[4], which will be used to collect unwanted electronic mail. Next, students will test the spam trap by starting the provided script. If everything is working, participants will create and test a simple artifacts repository based on the Viper[5] project. Then students will learn how to modify Viper and Shiva code to extend their functionality.

In the second phase, when Shiva and Viper are configured, students will start a script to generate spam messages. Then students will carry out analysis of the received e-mails.

In this exercise students will learn:

a) How to configure a spamtrap based on Shiva honeypot?

b) How to create an artifacts repository using Viper?

c) How to extend Shiva and Viper functionality?

d) How to analyse spam messages collected by Shiva?

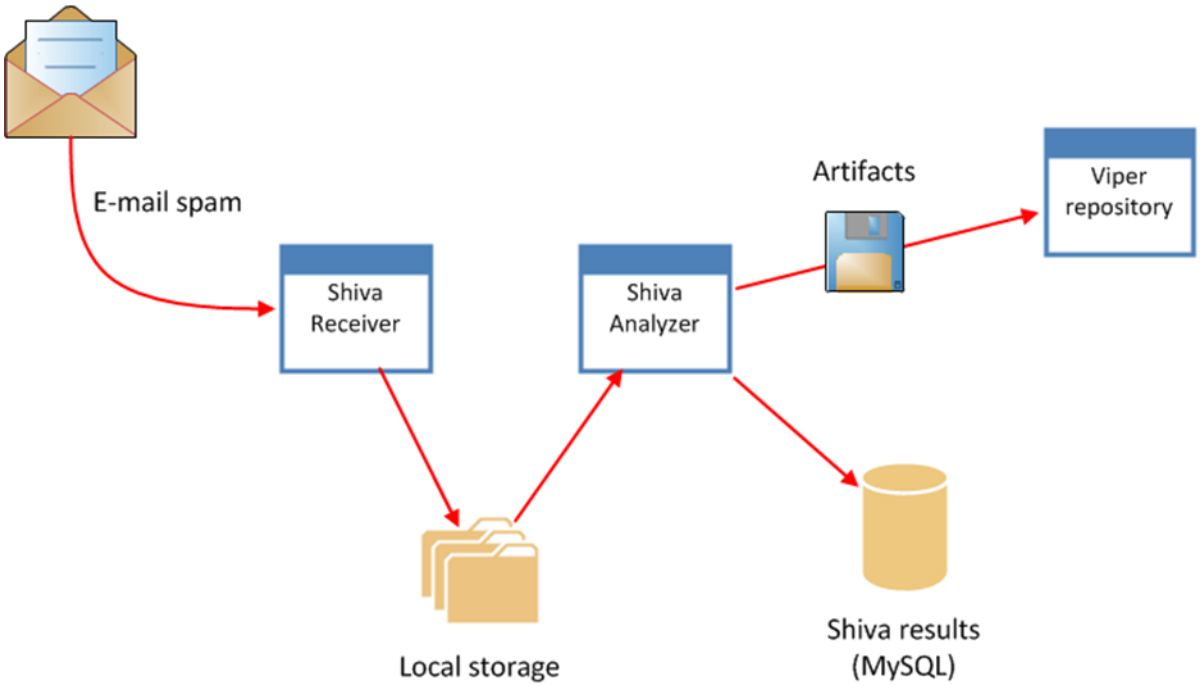

Figure 1: Architecture of the system which will be built during this exercise.

[4] https://github.com/shiva-spampot/shiva
[5] http://viper.li/

2 Task 1 – Spamtrap configuration and usage

In this step the participants install and configure Shiva honeypot[6], which is a high interaction SMTP honeypot specifically designed for spam collection and analysis. Shiva consists of two primary modules: shivaReceiver and shivaAnalyzer. The first one acts as a typical SMTP server allowing to receive and store e-mails containing spam. The second module performs preliminary spam analysis to detect similar messages (based on fuzzy hashing[7]) as well as extracting any attachments or uniform resource locators (URLs) contained in the spam messages.

2.1 Installing Shiva dependencies

Shiva honeypot is a Python project using Lamson Python Mail Server as a backend. Shiva also depends on an Exim4 mail server to relay e-mails (not used in the exercise) and a MySQL database to store the results.

First stop InetSim service:

Stopping INetSim

```
$ sudo service inetsim stop
```

Next install basic Shiva dependencies required by its installation script:

Installing Shiva dependencies

```
$ sudo apt-get install g++ make automake autoconf python-dev python-
virtualenv exim4-daemon-light libmysqlclient-dev libffi-dev
```

Then install ssdeep 2.10 from packages specially built for this exercise – default ssdeep version in the Ubuntu repository is too old and doesn't work with Shiva.

ssdeep 2.10 installation

```
$ cd /home/enisa/enisa/packages/extra
$ sudo dpkg -i libfuzzy* ssdeep*
```

Install MySQL database which is used by Shiva to store the analysed spam e-mails. When asked for a new password please provide password "enisa". It will be needed later.

MySQL database installation

```
$ sudo apt-get install mysql-server
```

[6] https://github.com/shiva-spampot/shiva
[7] http://jessekornblum.com/presentations/htcia06.pdf

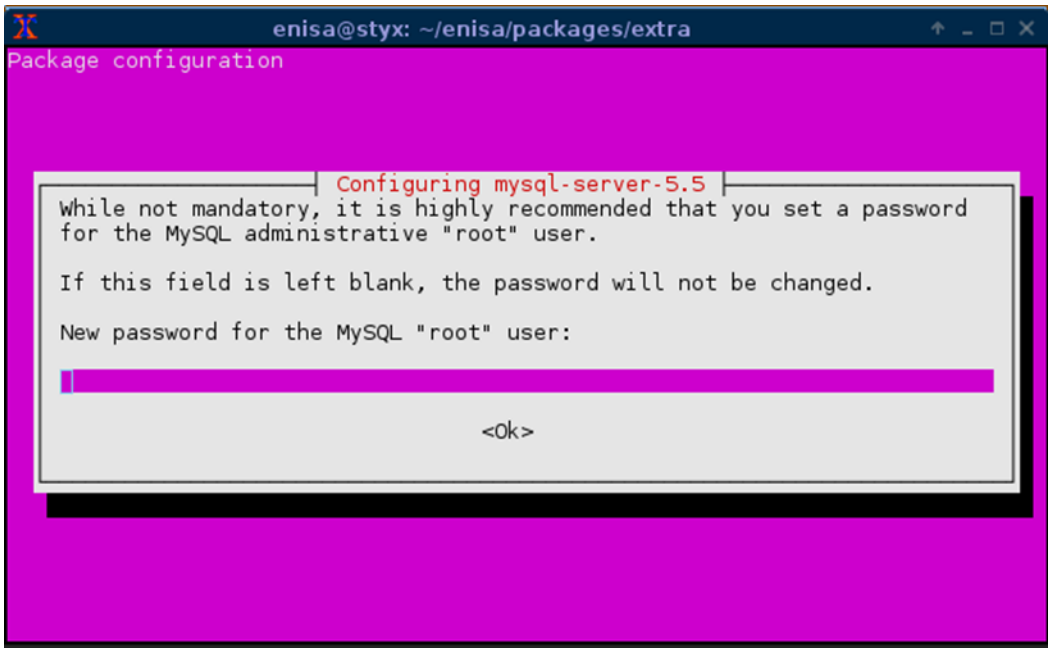

Figure 2. Setting root password for MySQL database [enisa].

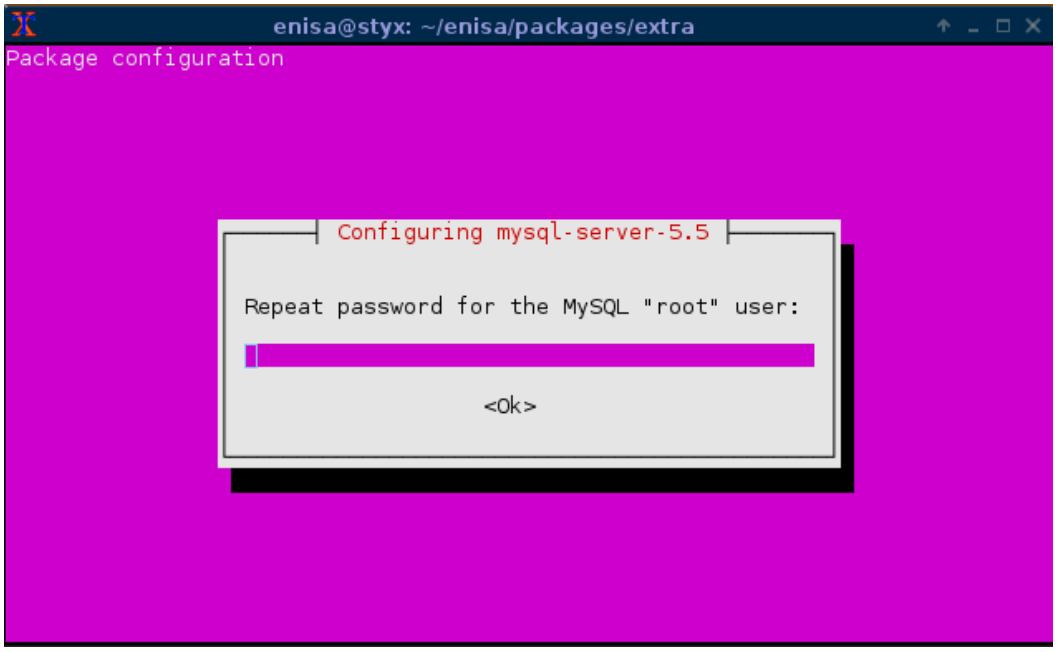

Figure 3. Confirming MySQL root password [enisa].

At the time of the writing of this document, Shiva honeypot doesn't provide any GUI interface to view and analyse stored results. All results are stored in the database. To make viewing the results easier, install phpMyAdmin. When asked to automatically reconfigure the web server, choose apache2 (select with space). When asked for a new phpMyAdmin password, set the password to "enisa".

phpMyAdmin installation

```
$ sudo apt-get install phpMyAdmin
```

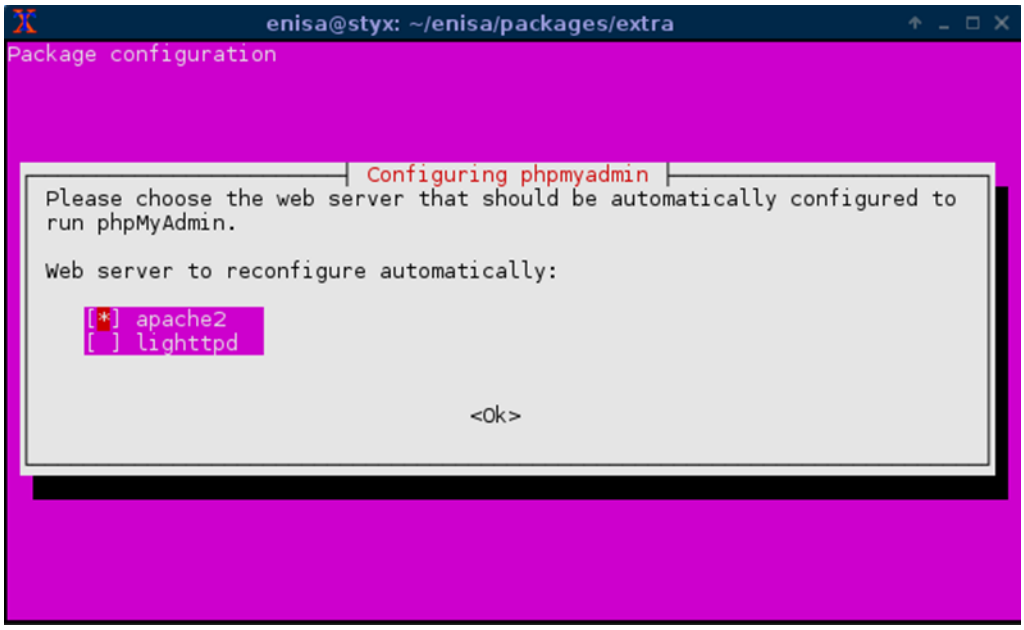

Figure 4. phpMyAdmin configuration - choosing web server [apache2]

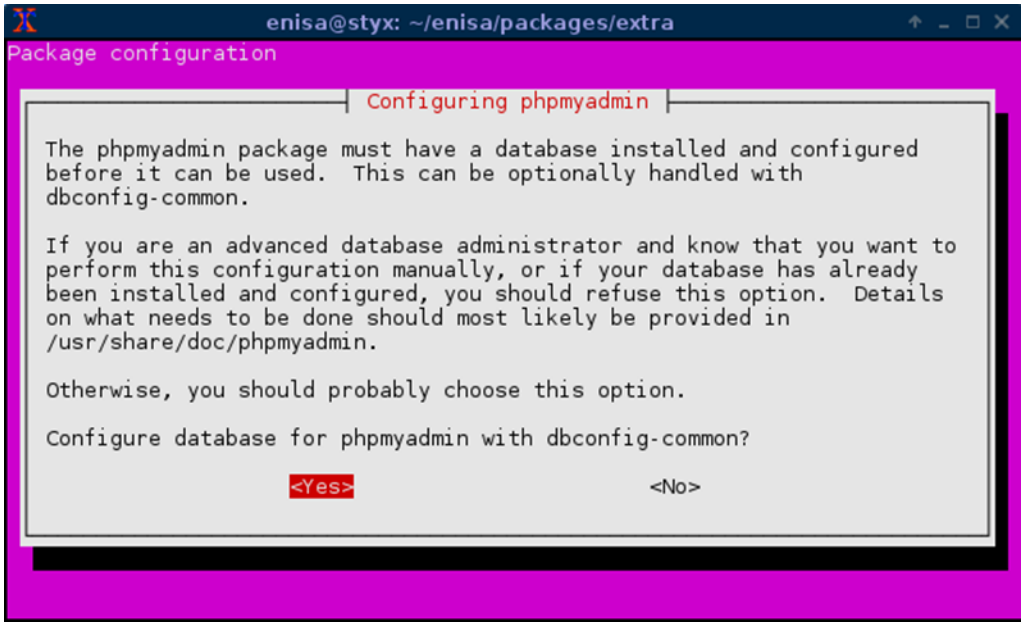

Figure 5. phpMyAdmin configuration - automatic database configuration [Yes]

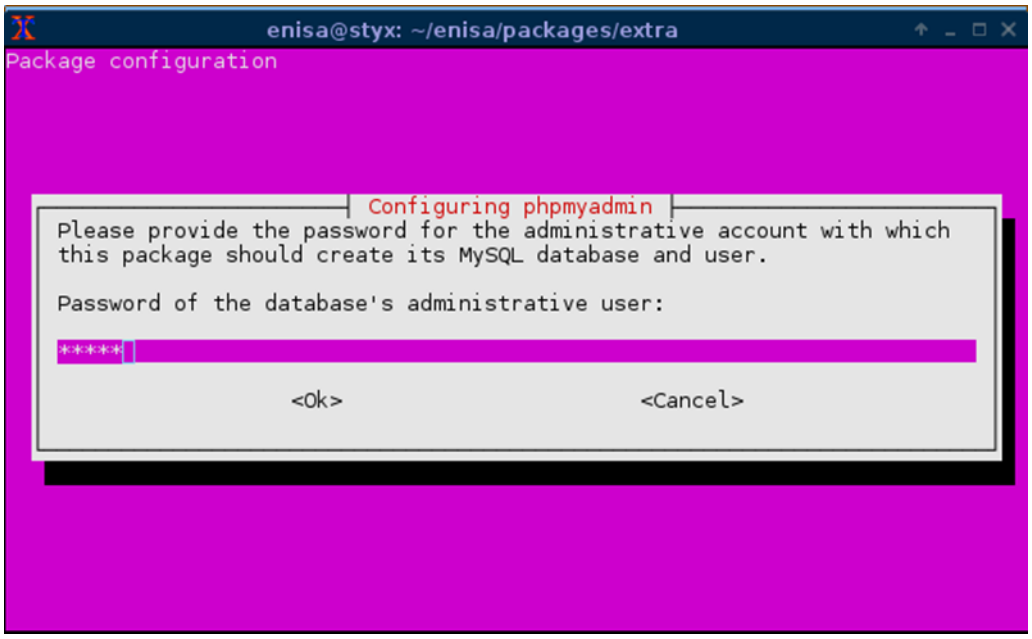

Figure 6. phpMyAdmin configuration – providing MySQL database root password [enisa]

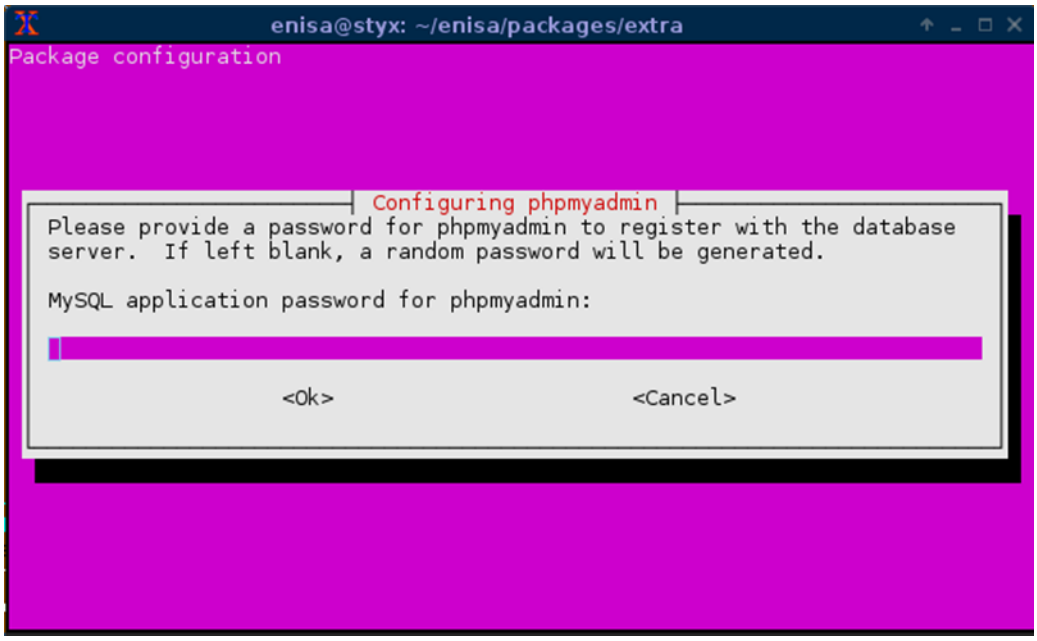

Figure 7. phpMyAdmin configuration – setting up phpMyAdmin root password [enisa]

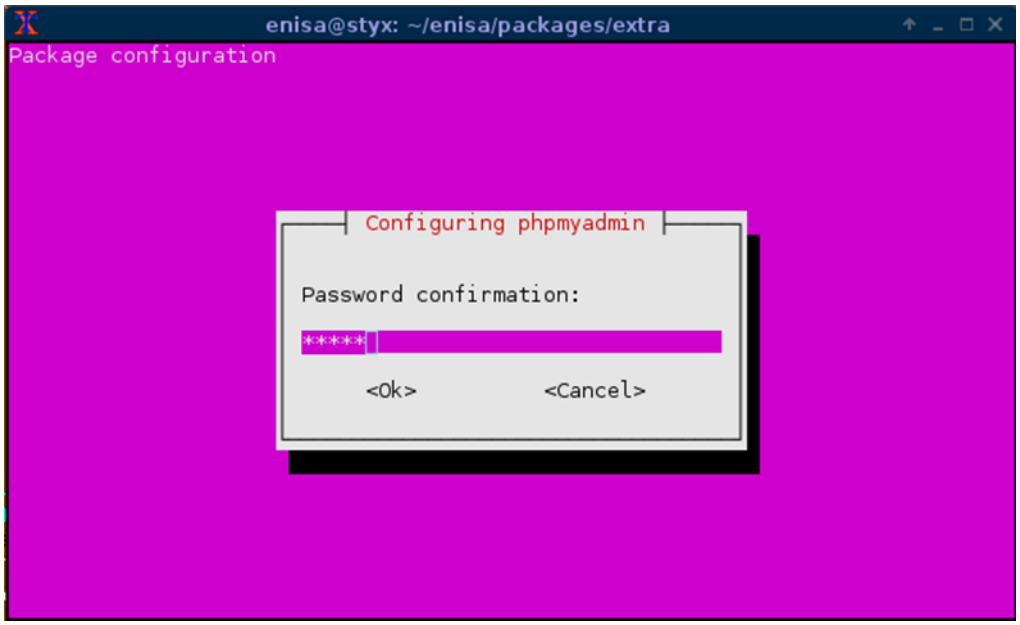

Figure 8. phpMyAdmin configuration – confirming phpMyAdmin password

Then configure Apache webserver to listen only on eth2 interface (192.168.56.10). In this way, no unauthorized person nor any malware running on the Winbox machine would be able to access the local phpMyAdmin instance.

Changing apache2 listen address

```
$ cd /etc/apache2
$ sudo sed -i 's/^Listen .*/Listen 192.168.56.10:80/' ports.conf
$ sudo apachectl restart
```

Then check if phpMyAdmin is working by starting a web browser in the host/native system and going to the address http://192.168.56.10/phpmyadmin. To login to phpMyAdmin use root username and the previously provided phpMyAdmin password (enisa).

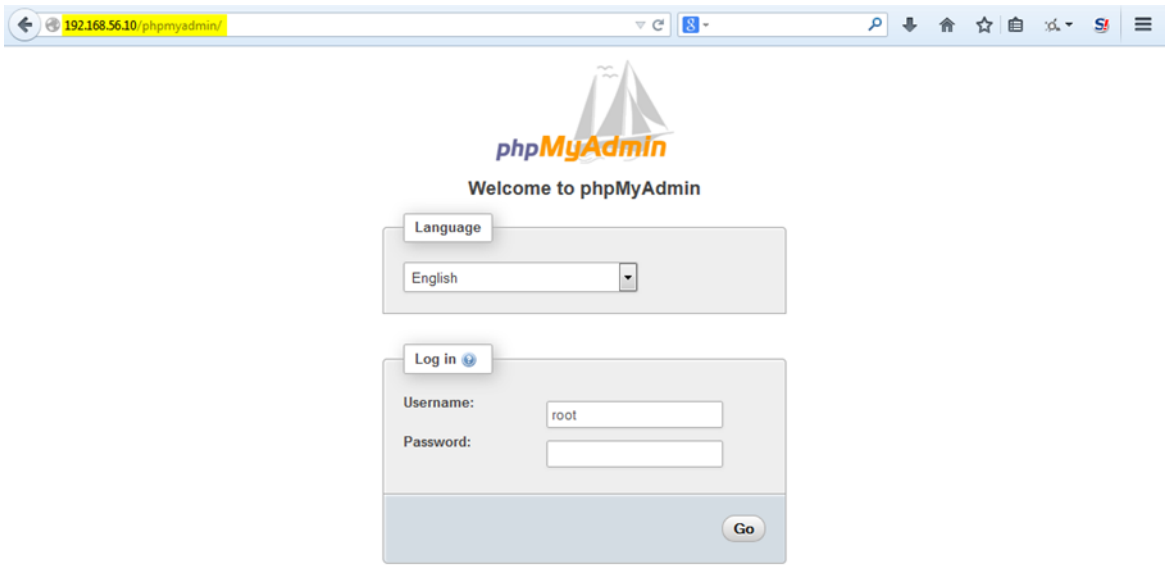

Figure 9. phpMyAdmin login screen (http://192.168.56.10/phpmyadmin/)

PhpMyAdmin is a graphical user interface frontend to the MySQL database where Shiva honeypot stores the results. PhpMyAdmin allows to manage the database as well as to view the data stored in the database. The students will use it later in the exercise to view results created by the Shiva spam honeypot.

After successfully installing all dependencies, the InetSim can be started again. InetSim and Apache2 will now listen on two separate interfaces (10.0.0.1, eth1 – InetSim and 192.168.56.10, eth2 – Apache2).

Starting INetSim

```
$ sudo service inetsim start
```

2.2 Installing Shiva honeypot

Copy and unpack Shiva source code to /opt/ directory and then start the installation. You need to issue chmod +x command to installation files prior the installation

Copying Shiva code

```
$ cd /opt/
$ sudo cp -a /home/enisa/enisa/ex2/source/shiva .
$ sudo chown -R enisa:enisa shiva
$ cd shiva
$ ./install.sh
```

During the installation, you will be asked whether to store analysed data in the database – answer 'Yes'.

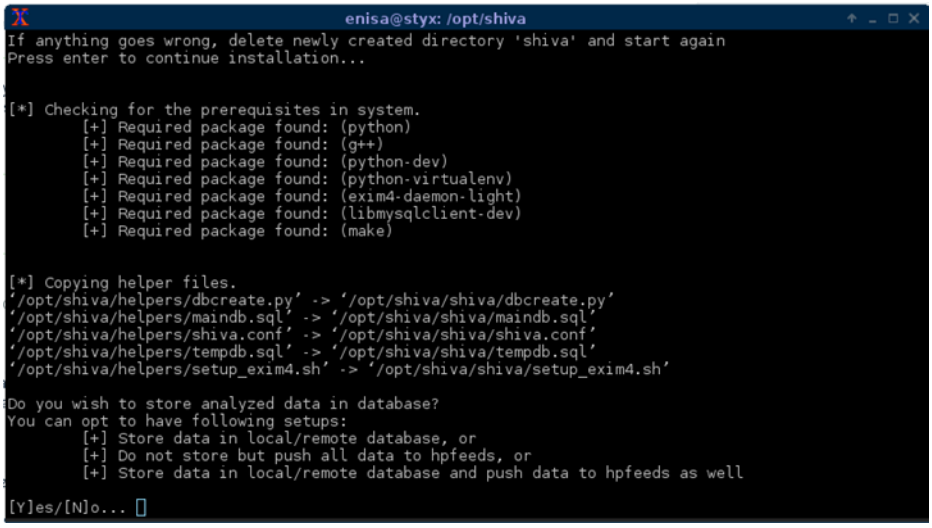

Figure 10. Shiva installation - database question [Yes]

Depending on the machine and system resources, this process might take up to a few minutes. If everything goes fine, you should see a message informing you that the installation is complete and you can start using the Shiva honeypot.

Message informing about successful Shiva installation

```
[+] Setting up Shiva Analyzer done!

[*] Creating necessary folders and updating configuration
files.....

[+] All done - phew!!!. Refer to User Manual to further customize
exim MTA, shiva.conf configuration file and starting honeyp0t
```

In case of any errors during the installation, remove the newly created shiva directory (/opt/shiva/shiva), resolve any problems and start installation script again.

2.3 Shiva configuration

After installation, go to the newly created shiva directory and open the shiva.conf configuration file with your favourite editor (vim, nano).

Shiva configuration

```
$ cd /opt/shiva/shiva
$ $EDITOR shiva.conf
```

Change the listening host and port of the shivaReceiver module ([receiver] section). It's an address on which the main SMTP process will be listening for incoming spam messages. For the purpose of the exercise you can leave 127.0.0.1 as a listening host but otherwise it should be set to the external IP address.

shiva.conf

```
[receiver]
listenhost : 127.0.0.1
listenport : 25
```

Disable spam relaying. In normal situation (as it was explained in exercise introduction) a user might decide to relay certain spam messages. By default Shiva allows a limited number of e-mails to redirect

in each time period. For the purpose of the exercise, relaying should be disabled; we don't want to relay any spam messages to the outside world.

shiva.conf

```
[analyzer]
relay : False
```

Set the scheduler time to 5 minutes. This is a time period at which scheduler starts analysing new e-mails and then pushes results to the database. In a normal case situation, to optimise performance, you may consider setting this time longer. The scheduler shouldn't be set to less than 4 minutes due to certain race conditions in Shiva source code.

shiva.conf

```
[analyzer]
scheduler time : 5
```

Configure database access in the [database] section. By default Shiva honeypot uses MySQL database and created two instances of databases. One for temporary results (ShivaTemp) and one for the final database (Shiva).

shiva.conf

```
[database]
localdb : True
host : 127.0.0.1
user : root
password : enisa
```

Disable additional notifications and the hpfeeds sharing feature.

shiva.conf

```
[hpfeeds]
enabled : False

[notification]
enabled : False
```

After saving the configuration file, the last step is to setup DB scheme and reconfigure local mail transfer agent (MTA) service (exim4). It can be done with dbcreate.py and setup_exim4.sh scripts.

Setting up DB and exim4

```
$ cd /opt/shiva/shiva
$ python2 ./dbcreate.py
confpath:  /opt/shiva/shiva/../shiva/shiva.conf
Temporary database created.
Main database created.
$ sudo ./setup_exim4.sh

 * Stopping MTA for restart    [ OK ]

 * Restarting MTA              [ OK ]
```

2.4 Running Shiva honeypot

Shiva consists of two distinct modules: shivaReceiver and shivaAnalyzer. The first one is responsible for receiving spam while the second one does some basic spam analysis and stores the results in the

database. In normal operations it's best to run both modules - except in situations where you collect spam in distributed environments or where certain hosts are intended to collect spam only while other hosts perform analyses.

To start shivaReceiver:

Starting shivaReceiver

```
$ sudo su
# cd /opt/shiva/shiva/shivaReceiver/
# source bin/activate
(shivaReceiver)# cd receiver/
(shivaReceiver)# lamson start
(shivaReceiver)# deactivate
# exit
```

Next start shivaAnalyzer:

Starting shivaAnalyzer

```
$ cd /opt/shiva/shiva/shivaAnalyzer/
$ source bin/activate
(shivaAnalyzer)$ cd analyzer/
(shivaAnalyzer)$ lamson start
(shivaAnalyzer)$ deactivate
```

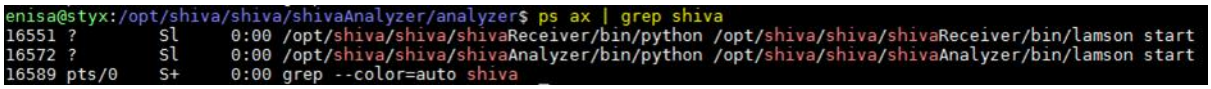

Figure 11. Checking if Shiva processes are running.

2.5 Testing Shiva honeypot

To test if Shiva was properly configured and is working, a special script should be used. This script will send a test e-mail to the Shiva local port and then students will view in logs if a new message was processed and was correctly added to the database.

First open two additional console windows in your host system and connect in both of them to Styx using ssh. If you are familiar with the screen tool, instead of opening two new windows you can start the screen and open two new tabs.

Connecting to Styx VM with ssh on Host-Only port

```
$ ssh enisa@192.168.56.10
```

Then, in the first additional window, view Shiva Receiver logs:

Viewing Receiver logs

```
$ cd /opt/shiva/shiva/shivaReceiver/receiver
$ tail -f logs/lamson.log
```

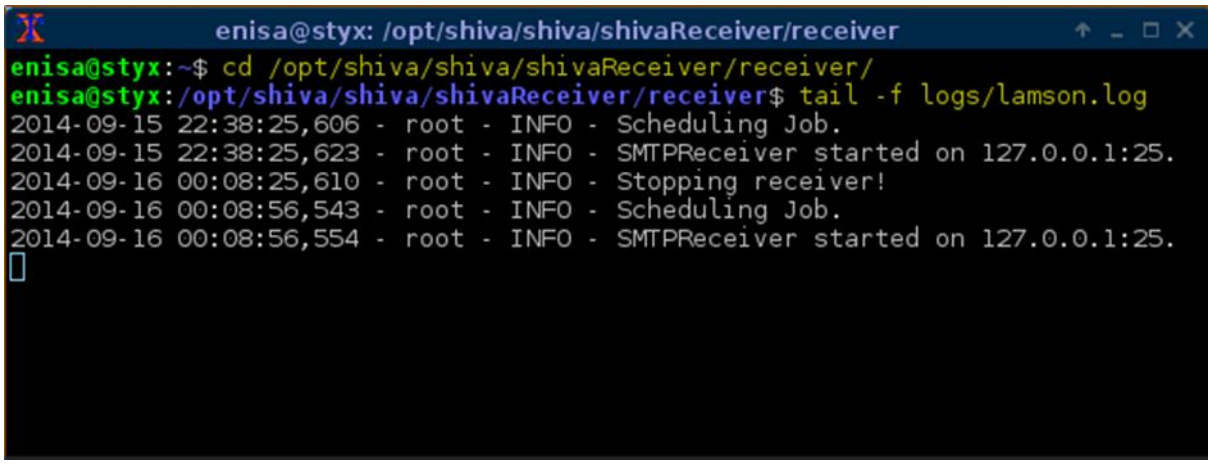

Figure 12. Viewing shivaReceiver logs.

In the second additional window, view Shiva Analyzer logs:

Viewing Analyzer logs

```
$ cd /opt/shiva/shiva/shivaAnalyzer/analyzer
$ tail -f logs/lamson.log
```

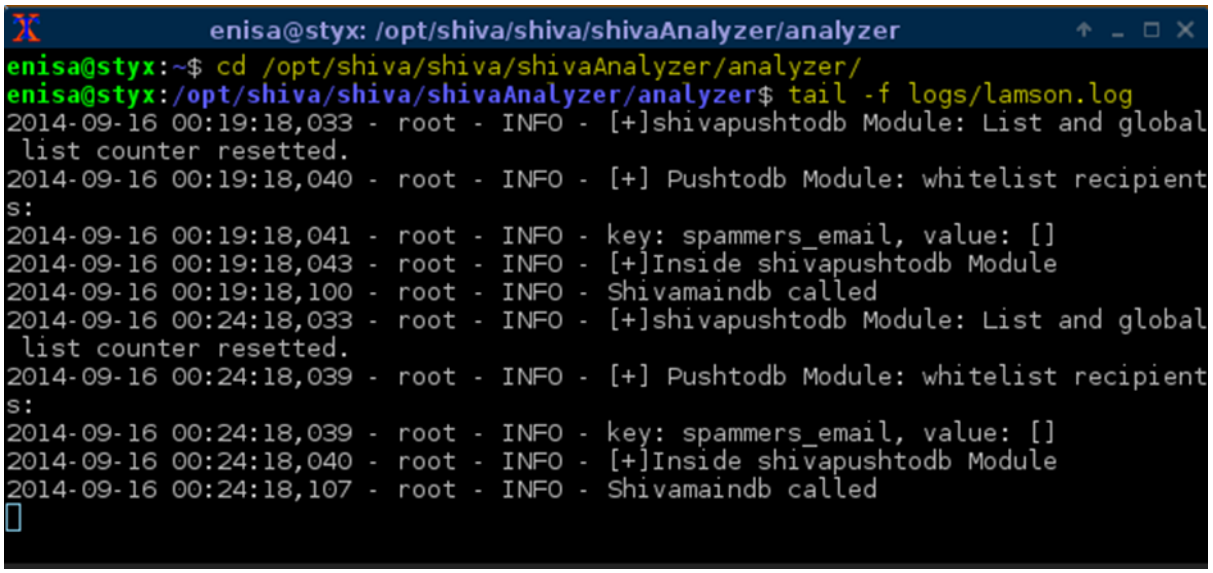

Figure 13. Viewing shivaAnalyzer logs.

When the preview of Receiver and Analyzer logs is open, start the test-shiva script in the primary window to send the test e-mails.

Sending test e-mails

```
$ /home/enisa/enisa/ex2/scripts/spam-script/test-shiva
```

At the same time, observe the Receiver and Analyzer logs. In both windows there should appear information about new messages being processed.

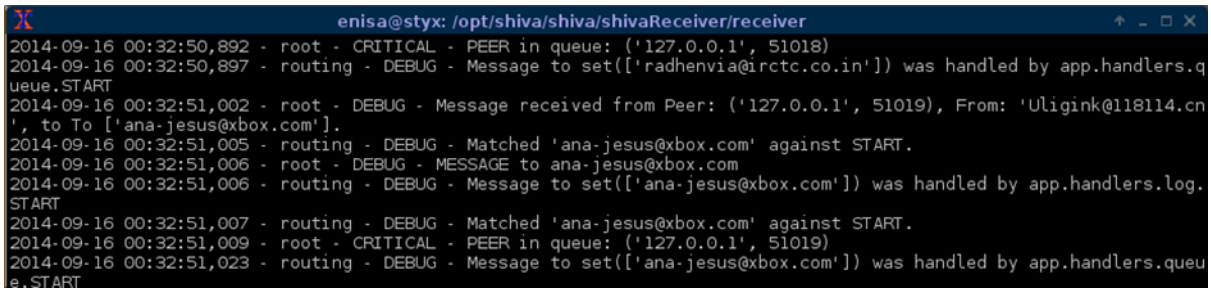

Figure 14. shivaReceiver logs informing about new messages being received.

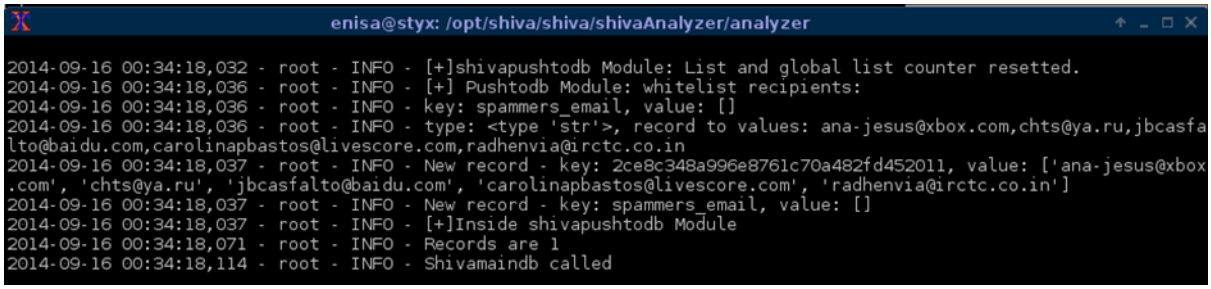

Figure 15. shivaAnalyzer logs informing about new messages being received and analyzed.

Wait until you see messages in the Analyzer log about the shivamaindb module being called and new records being pushed to the database (up to 5 min – schedulertime). Then you have to wait till shivamaindb finishes work and pushes records to the main DB. In this exercise it should take no more than 30s, normally up to 3.5min.

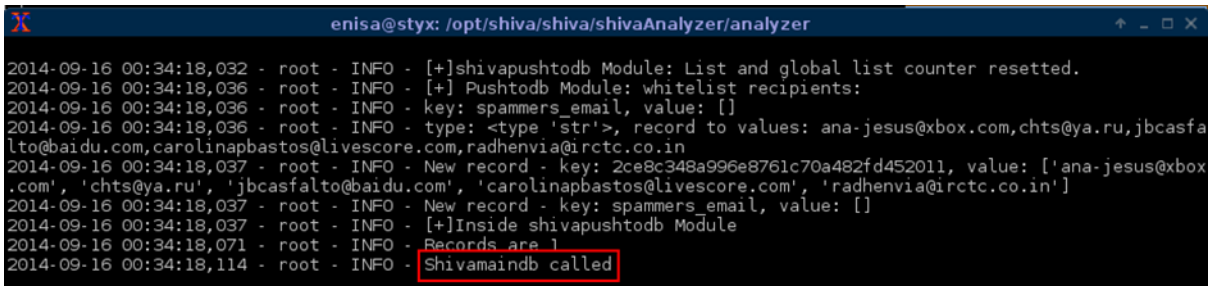

Figure 16. Information about shivamaindb module start in shivaAnalzyer logs.

Next log in to the phpMyAdmin at address http://192.168.56.10/phpmyadmin (root: enisa) and browse to the Spam table in the Shiva database.

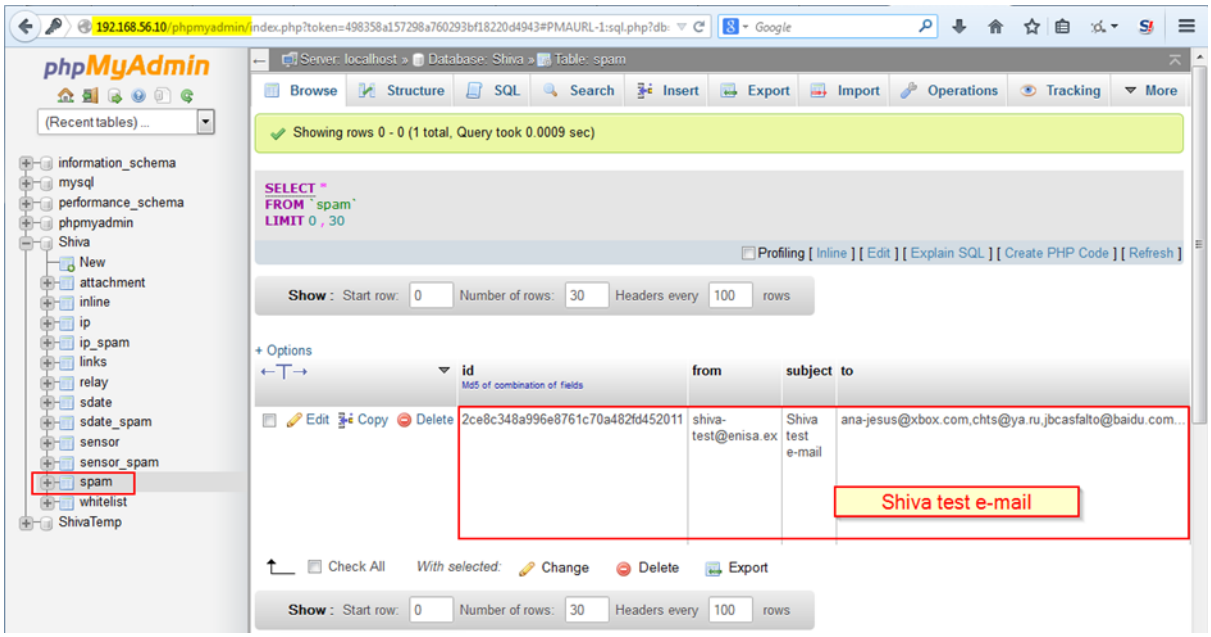

Figure 17: Viewing Shiva test e-mail in phpMyAdmin.

The live view of the Receiver and Analyzer logs in this step in two additional windows wasn't required. Logs could be also viewed in a single window afterwards (after sending test e-mails). The two additional windows were used to better visualize how the Shiva honeypot is working.

3 Task 2 – Building storage for the artifacts

In this task, participants will set up storage for the received artifacts such as malware samples or suspicious URLs. Samples storage will be based on the Viper project[8] which is a tool intended to ease organization and collection of malware samples. Viper organises samples in separate projects which can be used to represent samples associated with different campaigns or obtained from different sources.

One of the advantages of Viper is that it can be easily customised – users can write their own scripts performing certain analyses on the samples. In this exercise, participants will write a simple Viper module allowing automatic upload of certain samples to the analysis VM.

At the end of the task, participants will also apply patches to Viper and the previously configured Shiva extending Viper API functionality and allowing automatic uploads of binary samples caught by Shiva to Viper.

Connecting to Styx VM with ssh on Host-Only port

```
$ ssh enisa@192.168.56.10
```

3.1 Installing Viper

Viper installation is very simple. To install Viper just copy its source code to the install directory, install all the requirements and it should be ready to use. No further configuration is needed.

Viper installation

```
$ cd /opt
$ sudo cp -a /home/enisa/enisa/ex2/source/viper .
```

[8] http://viper.li/

```
$ cd viper/
$ sudo pip install -r requirements.txt
```

3.2 Running and using Viper

Viper can be started either in the global workspace (anonymous) or in the named workspace (called *project*). The general idea behind project workspaces is to allow users create separate distinct groups of malware samples. A malware sample is visible only within the workspace of the project to which it was added.

To start Viper go to its directory and simply run ./viper.py. Please note it's necessary to first switch to Viper's directory because Viper tries to read its database relative to the current working directory.

Starting Viper

```
$ cd /opt/viper
$ ./viper.py -p enisa-test
```

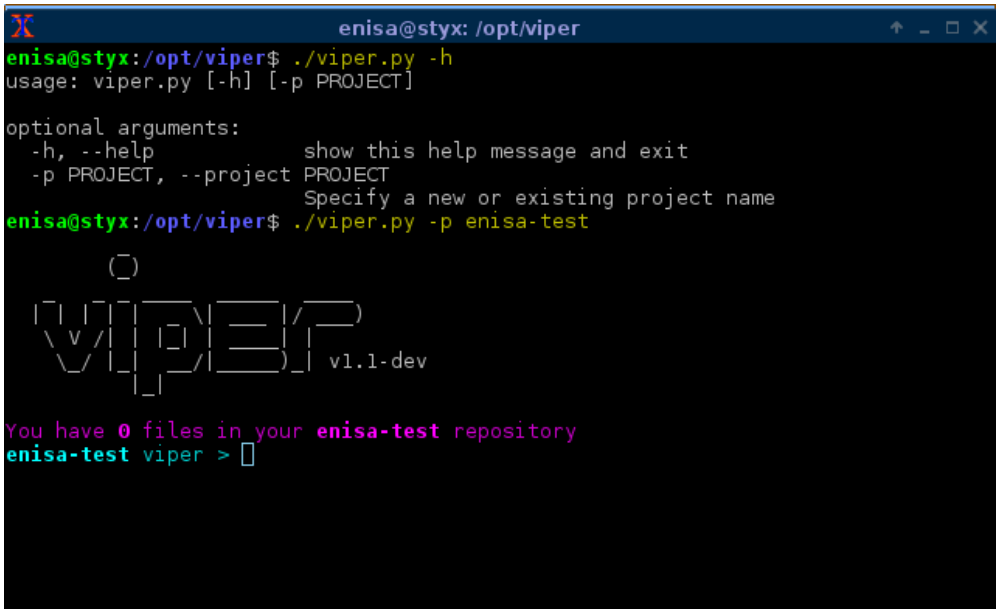

Figure 18. Main Viper prompt

After starting Viper, the user is presented with a prompt where he can type various commands. All available commands can be listed by typing *help*. Commands are divided into two groups: commands and module commands. Normal commands are used to manage samples and the repository (adding new samples, finding samples, adding notes, adding tags, etc.). Module commands are used to perform various analyses on specific samples such as checking file type, extracting strings and imports, scanning for Yara signatures or sending a sample to cuckoo analysis.

Next, open in Viper new sample (/home/enisa/enisa/ex2/samples/putty.exe) using the open command.

```
enisa-test viper > open -h
usage: open [-h] [-f] [-u] [-l] [-t] <target|md5|sha256>

Options:
        --help (-h)     Show this help message
        --file (-f)     The target is a file
        --url (-u)      The target is a URL
        --last (-l)     The target is the entry number from the last find command's results
        --tor (-t)      Download the file through Tor

You can also specify a MD5 or SHA256 hash to a previously stored
file in order to open a session on it.

enisa-test viper > open -f /home/enisa/enisa/ex2/samples/putty.exe
[*] Session opened on /home/enisa/enisa/ex2/samples/putty.exe
enisa-test viper putty.exe >
```

Figure 19. Opening putty.exe test sample in Viper

After opening a sample a new session is created. The sample itself isn't stored in the local repository yet. To store it in the repository, the student must use the *store* command:

```
enisa-test viper putty.exe > store -h
usage: store [-h] [-d] [-f <path>] [-s <size>] [-y <type>] [-n <name>] [-t]

Options:
        --help (-h)     Show this help message
        --delete (-d)   Delete the original file
        --folder (-f)   Specify a folder to import
        --file-size (-s)        Specify a maximum file size
        --file-type (-y)        Specify a file type pattern
        --file-name (-n)        Specify a file name pattern
        --tags (-t)     Specify a list of comma-separated tags

enisa-test viper putty.exe > store
[+] Stored file "putty.exe" to /opt/viper/projects/enisa-test/binaries/a/b/c/c/abcc2a2d828b1624459cf8c4d2c
cdfdcde62c8d1ab51e438db200ab3c5c8cd17
[*] Session opened on /opt/viper/projects/enisa-test/binaries/a/b/c/c/abcc2a2d828b1624459cf8c4d2ccdfdcde62
c8d1ab51e438db200ab3c5c8cd17
enisa-test viper putty.exe >
```

Figure 20. Permanently storing putty.exe sample in Viper database

To list all open sessions use the *sessions* command. Each session is associated with a single opened file.

```
enisa-test viper putty.exe > sessions -h
usage: sessions [-h] [-l] [-s=session]

Options:
        --help (-h)     Show this help message
        --list (-l)     List all existing sessions
        --switch (-s)   Switch to the specified session

enisa-test viper putty.exe > sessions -l
[*] Opened Sessions:
+---+-----------+----------------------------------+---------------------+---------+
| # | Name      | MD5                              | Created At          | Current |
+---+-----------+----------------------------------+---------------------+---------+
| 2 | putty.exe | 7a0dfc5353ff6de7de0208a29fa2ffc9 | 2014-08-20 15:34:39 | Yes     |
+---+-----------+----------------------------------+---------------------+---------+
enisa-test viper putty.exe >
```

Figure 21. Listing sessions in Viper

All samples in the repository can be listed using the *find* command. The student can further narrow search results using various search criteria such as file name, file type, md5 sum, tags, etc. No regular expressions or wildcards are possible in the current version.

```
enisa-test viper putty.exe > find -h
usage: find [-h] [-t] <all|latest|name|type|mime|md5|sha256|tag|note> <value>

Options:
        --help (-h)     Show this help message
        --tags (-t)     List tags

enisa-test viper putty.exe > find all
+---+-----------+----------------------+----------------------------------+------+
| # | Name      | Mime                 | MD5                              | Tags |
+---+-----------+----------------------+----------------------------------+------+
| 1 | putty.exe | application/x-dosexec | 7a0dfc5353ff6de7de0208a29fa2ffc9 |      |
+---+-----------+----------------------+----------------------------------+------+
enisa-test viper putty.exe > find type exe
+---+-----------+----------------------+----------------------------------+------+
| # | Name      | Mime                 | MD5                              | Tags |
+---+-----------+----------------------+----------------------------------+------+
| 1 | putty.exe | application/x-dosexec | 7a0dfc5353ff6de7de0208a29fa2ffc9 |      |
+---+-----------+----------------------+----------------------------------+------+
enisa-test viper putty.exe > find type pdf
enisa-test viper putty.exe >
```

Figure 22. Using find command in Viper

The user can also add notes or tags to each sample. Notes can be used to report interesting findings about the sample or just to state its origin. Tags can be used to further organize various types of samples (e.g. by malware family, by its origin, etc.).

```
enisa-test viper putty.exe > notes -a
Enter a title for the new note: Sample source
[*] New note with title "Sample source" added to the current file
enisa-test viper putty.exe > notes -l
+----+---------------+
| ID | Title         |
+----+---------------+
| 1  | Sample source |
+----+---------------+
enisa-test viper putty.exe > notes -v 1
[*] Title: Sample source
[*] Body:
This sample was downloaded from http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe

enisa-test viper putty.exe >
```

Figure 23. Adding and viewing notes.

```
enisa-test viper putty.exe > find all
+---+-----------+----------------------+----------------------------------+------+
| # | Name      | Mime                 | MD5                              | Tags |
+---+-----------+----------------------+----------------------------------+------+
| 1 | putty.exe | application/x-dosexec | 7a0dfc5353ff6de7de0208a29fa2ffc9 |      |
+---+-----------+----------------------+----------------------------------+------+
enisa-test viper putty.exe > tags -a pe32,clean,putty
[*] Tags added to the currently opened file
[*] Refreshing session to update attributes...
[*] Session opened on /opt/viper/projects/enisa-test/binaries/a/b/c/c/abcc2a2d828b1624459cf8c4d2ccdfdcde62c8d1ab51e43
8db200ab3c5c8cd17
enisa-test viper putty.exe > find all
+---+-----------+----------------------+----------------------------------+-------------------+
| # | Name      | Mime                 | MD5                              | Tags              |
+---+-----------+----------------------+----------------------------------+-------------------+
| 1 | putty.exe | application/x-dosexec | 7a0dfc5353ff6de7de0208a29fa2ffc9 | pe32, clean, putty |
+---+-----------+----------------------+----------------------------------+-------------------+
enisa-test viper putty.exe >
```

Figure 24. Adding and viewing tags.

Now you should be familiar with basic repository management. Next, take your time and experiment with various module commands. List of all commands that are available after typing *help*. Each command has separate help info available with '-h' option.

Finding all strings matching hostname or IP address:

```
enisa-test viper putty.exe > strings -H
 - openssh.com
 - openssh.com
 - putty.projects.tartarus.org
 - putty.projects.tartarus.org
 - 0.0.0.0
 - lysator.liu.se
 - ssh.com
 - ssh.com
 - openssh.com
 - projects.tartarus.org
 - www.chiark.greenend.org.uk
 - 0.0.0.0
 - 6.0.0.0
 - schemas.microsoft.com
enisa-test viper putty.exe >
```

Figure 25. Finding strings matching IPs and hostnames.

Printing list of PE32 sections with its sizes, RVA and entropy:

```
enisa-test viper putty.exe > pe sections
[*] PE Sections:
+--------+---------+-------------+-------------+--------------+
| Name   | RVA     | VirtualSize | RawDataSize | Entropy      |
+--------+---------+-------------+-------------+--------------+
| .text  | 0x1000  | 0x55fb1     | 352256      | 6.67533118296 |
| .rdata | 0x57000 | 0x1b62a     | 114688      | 6.09870674855 |
| .data  | 0x73000 | 0x58c4      | 8192        | 2.40561023997 |
| .rsrc  | 0x79000 | 0x3b90      | 16384       | 3.87147274489 |
+--------+---------+-------------+-------------+--------------+
enisa-test viper putty.exe >
```

Figure 26. Viewing PE sections and their entropy of an executable file.

To exit Viper use the command `*exit*`.

3.3 Writing a Viper module

In this step we will write a Viper module allowing to directly upload samples to the analysis VM (Winbox). To send samples we will use the FTP protocol and the FTP server already installed on the Winbox. All samples will be uploaded to ftp://10.0.0.2/sample/ where 10.0.0.2 is assumed Winbox IP address.

First go to Viper's modules directory and create a new module file.

Creating Viper module

```
$ cd /opt/viper/modules
$ $EDITOR lab-send.py
```

Then write the following code:

lab-send.py

```
import re
import getopt
import ftplib

from viper.common.out import *
from viper.common.abstracts import Module
from viper.core.session import __sessions__

DEFAULT_HOST = '10.0.0.2'

class LabSend(Module):
    cmd = 'lab-send'
    description = 'Sends the file to the analysis VM (by ftp)'
    authors = ['DO']
```

```
def run(self):
    if not __sessions__.is_set():
        print_error("No session opened")
        return

    def usage():
        print("usage: lab-send [-h] [-H=host]")

    def help():
        usage()
        print("")
        print("Options:")
        print("\t--help (-h)\tShow this help message")
        print("\t--host (-H)\tSpecify an host (default: 10.0.0.2)")
        print("")

    try:
        opts, argv = getopt.getopt(self.args, 'hH:', ['help', 'host='])
    except getopt.GetoptError as e:
        print(e)
        usage()
        return

    host = DEFAULT_HOST

    for opt, value in opts:
        if opt in ('-h', '--help'):
            help()
            return
        elif opt in ('-H', '--host'):
            if value:
                host = value

    try:
        # Opening file
        fh = open(__sessions__.current.file.path, 'rb')
        fname = __sessions__.current.file.name
        #sanitize name
        fname = re.sub(r'[\\\/:\*\?"<>\|]', '_', fname)

        # Connecting to the ftp
        ftp = ftplib.FTP(host)
        ftp.login()
        ftp.cwd('sample')
        res = ftp.storbinary('STOR {}'.format(fname), fh)
        ftp.quit()
    except Exception as e:
        print_error("Unable to send sample to the VM: {}".format(e))
        return

    print(res)
```

In this code we assume that the Winbox IP address is 10.0.0.2 (DEFAULT_HOST). If it's different it should be changed accordingly or given as a command argument (--host) each time the script is run. Also please remember this is a Python code and consistent indentation matters (i.e. tabulators shouldn't be mixed with the spaces, it's best to use 4 spaces as an indentation).

In case of any problems the code can be copied from **/home/enisa/enisa/ex2/files/viper/lab-send.py**.

Using provided lab-send.py code (alternative)

```
$ cd /opt/viper/modules
$ cp /home/enisa/enisa/ex2/files/viper/lab-send.py .
```

Now when you start Viper (in *enisa-test* project space) and list all commands (*help*) lab-send script should be visible among module commands.

```
| ida          | Start IDA Pro                                |
| idx          | Parse Java idx files                         |
| image        | Perform analysis on images                   |
| jar          | Parse Java JAR archives                      |
| lab-send     | Sends the file to the analysis VM (by ftp)   |
| office       | Office Document Parser                       |
| pdf          | Extract PDF Stream Information               |
| pe           | Extract information from PE32 headers        |
| reports      | Online Sandboxes Reports                     |
```

Figure 27. New lab-send module visible in the output of help command.

To test the script **first restore and start Winbox machine** (snapshot dedicated to static and dynamic analyses) and open in Viper previously uploaded putty.exe sample. Then try to send putty.exe to the Winbox.

```
enisa viper > find name putty.exe
+---+-----------+----------------------+----------------------------------+-------+
| # | Name      | Mime                 | MD5                              | Tags  |
+---+-----------+----------------------+----------------------------------+-------+
| 1 | putty.exe | application/x-dosexec | 7a0dfc5353ff6de7de0208a29fa2ffc9 | shiva |
+---+-----------+----------------------+----------------------------------+-------+
enisa viper > open -l 1
[*] Session opened on /opt/viper/projects/enisa/binaries/a/b/c/c/abcc2a2d828b1624459cf8c
4d2ccdfdcde62c8d1ab51e438db200ab3c5c8cd17
enisa viper putty.exe > lab-send
226 Successfully transferred "/sample/putty.exe"
enisa viper putty.exe >
```

Figure 28. Sending putty.exe sample to Winbox machine.

To be sure if sample was successfully uploaded, go to the Winbox and check if there is a putty.exe file in c:\analyses\sample.

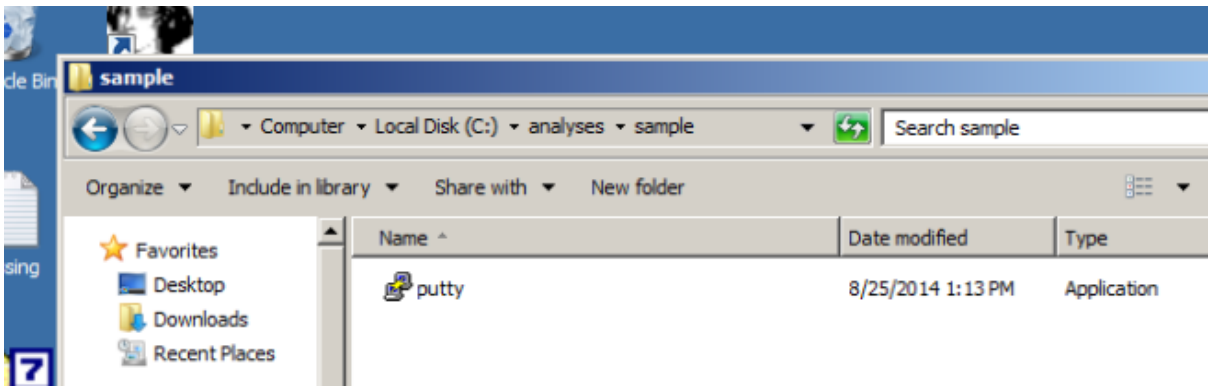

Figure 29. Checking if sample was uploaded to the Winbox machine.

3.4 Patching Viper API and building upload script

Viper provides a simple HTTP API allowing to perform basic operations such as adding new binary files, downloading samples, finding samples or listing tags.

In this step, students will apply a simple patch to Viper API to extend its functionality—adding new samples from the URLs and starting the API in the context of a specific project (by default the API starts

only in unnamed anonymous workspace). Then students will write a utility to add new samples to the Viper directly from the Linux command line.

Patching Viper API

```
$ cd /opt/viper
$ patch api.py < /home/enisa/enisa/ex2/files/viper/api.patch
```

Now start the Viper API in the background in the context of the 'enisa' project:

Starting patched API in the background

```
$ cd /opt/viper
$ nohup ./api.py -P enisa &
$ cat nohup.out
Bottle v0.12.7 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.
```

Next, we will write a script to add samples to Viper directly from the Linux console. This script will be using the Viper API to upload samples, so it's necessary for the Viper API to be running in the background.

Creating viper-upload script

```
$ cd /lab/bin
$ $EDITOR lab-viper-upload
$ chmod +x lab-viper-upload
```

lab-viper-upload

```
#!/usr/bin/python

import os
import sys
import argparse
import requests
import urlparse

VIPER_API='http://127.0.0.1:8080/'

def viper_upload_bin(path):
    """Uploads binary file to Viper"""

    upload_url = urlparse.urljoin(VIPER_API, 'file/add')
    tags = 'enisa,bin,pe32'

    try:
        response = requests.post(upload_url,
                                 files={'file': open(path, 'rb'),
                                 'filename':os.path.basename(path)},
                                 data={'tags': tags})
    except IOError as e:
        print("Error: IOError: {}".format(e))
        exit(1)
    except requests.exceptions.ConnectionError:
        print("Error: Connection error. Is Viper API running at {}?".format(VIPER_API))
        exit(1)

    print response.content


def viper_upload_url(url):
    """Adds sample from URL to Viper"""

    upload_url = urlparse.urljoin(VIPER_API, 'url/add')
    tags = 'enisa,url'
```

```
    try:
        response = requests.post(upload_url,
                                 data={'tags': tags, 'url': url})
    except requests.exceptions.ConnectionError:
        print("Connection error. Is Viper API running at {}?".format(VIPER_API))
        exit(1)

    print response.content


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Adds samples to Viper database")
    parser.add_argument('-p', '--path', type=str, action='store', help="path to the file")
    parser.add_argument('-u', '--url', type=str, action='store', help="url to be added")
    args = parser.parse_args()

    if args.path:
        viper_upload_bin(args.path)
    elif args.url:
        viper_upload_url(args.url)
    else:
        parser.print_help()
```

The script can be also copied from /home/enisa/enisa/ex2/files/viper/lab-viper-upload.

Using provided lab-viper-upload code (alternative)

```
$ cd /lab/bin
$ cp /home/enisa/enisa/ex2/files/viper/lab-viper-upload .
```

To test if the script is working correctly, try to add to the Viper pscp.exe sample from /home/enisa/enisa/ex2/samples/ directory.

Testing viper-upload script

```
$ cd /lab/bin
$ ./lab-viper-upload -p /home/enisa/enisa/ex2/samples/pscp.exe
{
    "message": "added"
}
```

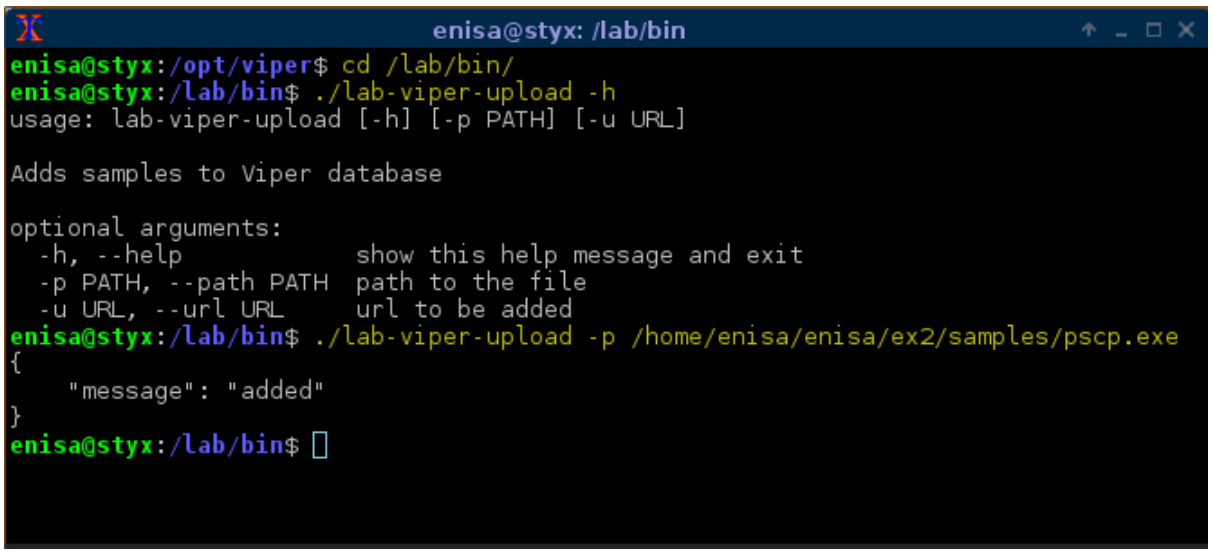

Figure 30. Adding to Viper pscp.exe sample with lab-viper-upload script.

Then you can start Viper (from /opt/viper path) and check if the sample was successfully added.

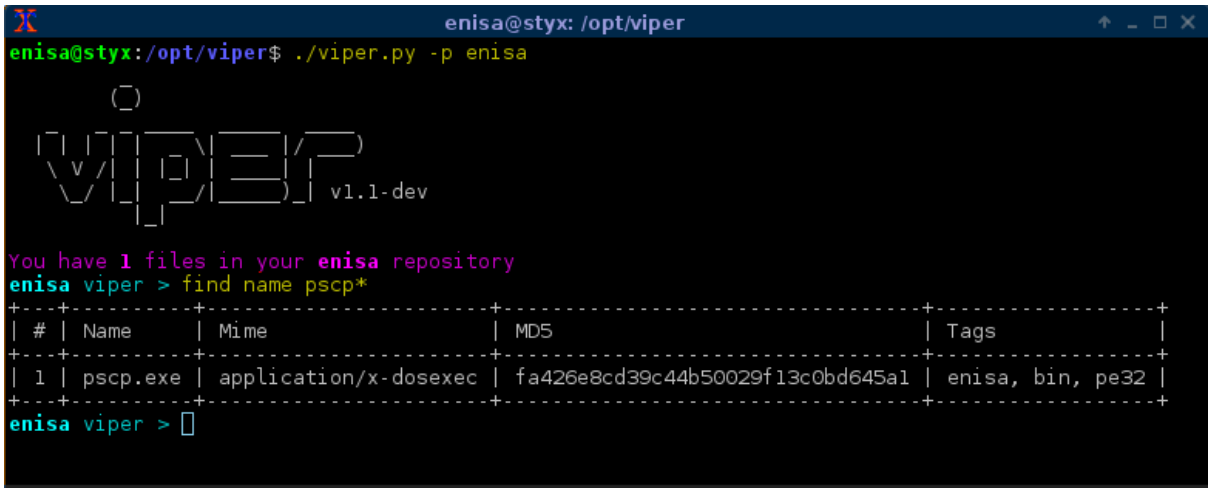

Figure 31. Checking if pscp.exe was successfully added to Viper.

3.5 Patching the Shiva honeypot

At this point we will patch the Shiva honeypot. After applying this patch whenever a new binary attachment is received by Shiva honeypot it will be automatically uploaded to Viper.

The Shiva patch is using Viper API to upload files. Consequently to make it work, the Viper API must be running and listening on the address http://localhost:8080/.

First install the Python *requests* module for shivaAnalyzer.

Installing requests for shivaAnalyzer

```
$ cd /opt/shiva/shiva/shivaAnalyzer
$ source bin/activate
(shivaAnalyzer)$ pip install requests
(shivaAnalyzer)$ deactivate
```

Then copy the Shiva viper module and apply the patch to shivapushtodb.py.

Applying patches for shivaAnalyzer

```
$ cd /opt/shiva/shiva/shivaAnalyzer/lib/python2.7/site-
packages/lamson
$ patch shivapushtodb.py <
/home/enisa/enisa/ex2/files/shiva/shivapushtodb.patch
$ mkdir viper
$ cd viper/
$ touch __init__.py
$ $EDITOR upload.py
```

Content of the *upload.py* script.

```
import os
import sys
import requests
import urlparse
import logging

VIPER_API='http://127.0.0.1:8080/'

def upload_bin(path, filename):
    upload_url = urlparse.urljoin(VIPER_API,'file/add')
    tags = 'shiva'
```

```
    response = requests.post(upload_url,
                             files={'file': (filename, open(path, 'rb'))},
                             data={'tags': tags})

    if response.status_code == 200:
        logging.info("[+] New sample uploaded to Viper: %s" % filename)
```

Alternatively upload.py can be copied from /home/enisa/enisa/ex2/files/shiva/viper/ path.

Using provided viper Shiva module (alternative)

```
$ cd /opt/shiva/shiva/shivaAnalyzer/lib/python2.7/site-
packages/lamson/viper
$ cp /home/enisa/enisa/ex2/files/shiva/viper/* .
```

After applying the new patch, shivaAnalyzer must be restarted.

Restarting shivaAnalyzer

```
$ cd /opt/shiva/shiva/shivaAnalyzer
$ source bin/activate
(shivaAnalyzer)$ cd analyzer
(shivaAnalyzer)$ lamson stop
(shivaAnalyzer)$ lamson start
(shivaAnalyzer)$ deactivate
```

4 Task 3 – Spam content analysis methods

In this task, participants run a special script to generate e-mail spam messages which will be delivered to the previously configured spamtrap. Messages might contain malicious attachments and some of them, malicious links. After receiving the spam messages, students will analyse their content and try to identify spam campaigns.

4.1 Sending spam messages

To send spam emails use the provided script.

Sending spam

```
$ /home/enisa/enisa/ex2/scripts/spam-script/send-spam
Spam 15/50 sent (30.0%)
```

After the script completes, type the following to view shivaAnalyzer logs. There might be some "CRITICAL" lines in the logs - ignore them, this is normal behaviour when processing attachments.

Viewing shivaAnalyzer logs

```
$ cd /opt/shiva/shiva/shivaAnalyzer/analyser
$ tail -n 100 -f logs/lamson.log
```

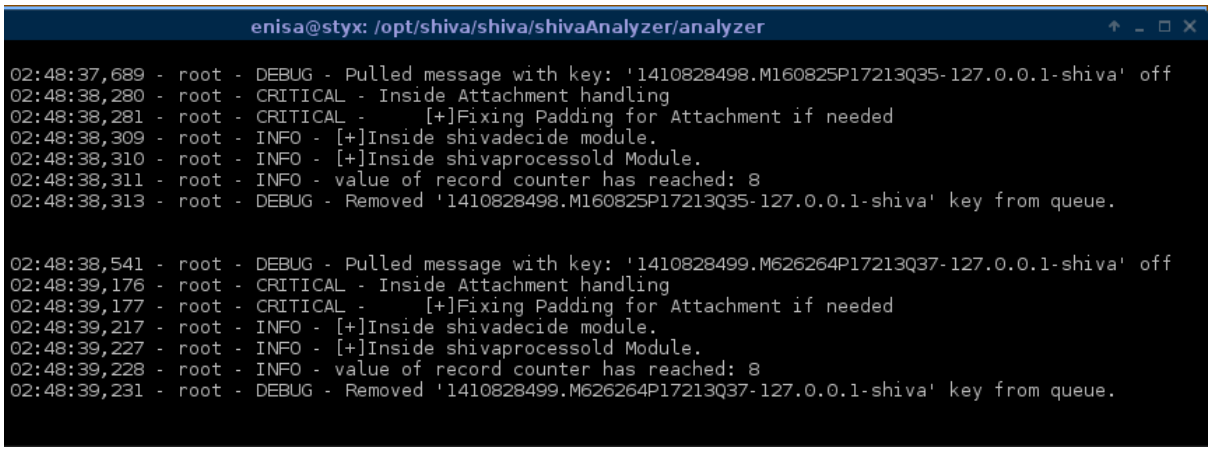

Figure 32. ShivaAnalyzer log informing about new samples being analysed - some of them containing attachments.

Wait until scheduler processes all new e-mails and adds them to the database (information about *shivamaindb* being called). In this exercise it could take up to 5 minutes (scheduler time value from Shiva configuration file).

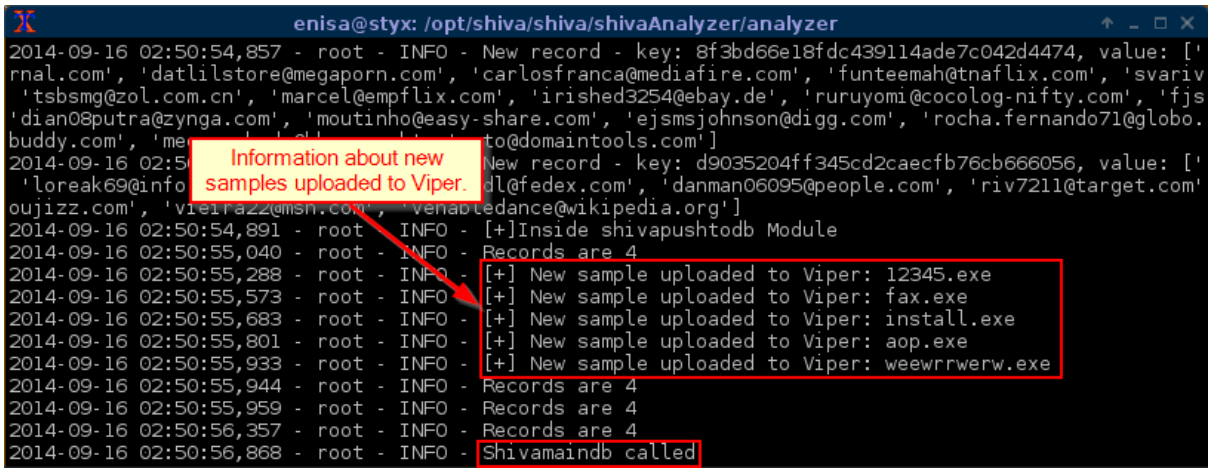

Figure 33. Shiva scheduler processing new samples and uploading them to Viper database.

Exercise (extra):

To receive more spam messages, run *send-more-spam* script.

Sending more spam

```
$ /home/enisa/enisa/ex2/scripts/spam-script/send-more-spam
```

Then by analysing received messages (viewing message content in the database as described in the next step) try to answer the following questions:

1. **Name a few social engineering techniques used in spam campaigns.**
   - Messages appearing to be sent by well-known companies and financial institutions (e.g. Wells Fargo, Citibank).
   - Messages appearing to be sent by local machines (printers, scanners, fax machines).
   - Message content suggesting it's important to the sender (e.g. invoice message, tax refunds).
   - Messages with attached executables appearing to be some documents with .pdf extension (e.g. message.pdf.exe).

2. **Can you identify a few distinct campaigns? What are their distinguishing features?**

Campaign with messages appearing to be sent by local printer/voice machines.

Subjects:

- Scanned Image from a Xerox WorkCentre
- Scan from a Xerox WorkCentre
- Scanned from a Xerox Multifunction Device
- Voice Message from Unknown (985-668-7888)

All messages with those subjects had similar mail headers (below) and the recipient domain was the same as sender domain. Messages had executable attachments with the name <name>.pdf.exe.

Fragment of message headers

```
X-MS-Has-Attach: yes
X-MS-Exchange-Organization-SCL: -1
X-MS-TNEF-Correlator:
<X66AF23LP311YV9Q1D1F44WLA61NIXG84M2BB7@example.com>
X-MS-Exchange-Organization-AuthSource: FRZK163AFS3MTID@example.com
X-MS-Exchange-Organization-AuthAs: Internal
X-MS-Exchange-Organization-AuthMechanism: 01
X-MS-Exchange-Organization-AVStamp-Mailbox: MSFTFF;6;0;0 0 0
X-Priority: 3 (Normal)
```

Campaign with messages appearing to be sent by Wells Fargo.

Subjects:

- RE: Account docs
- Documents - WellsFargo

The content of the messages suggests they were sent by a Wells Fargo employee who is asking the recipient to open important documents attached to the message.

Fragment of message content

```
For more details please check the attached documents.

Cory Rowell
Wells Fargo Advisors
817-347-4173 office
817-987-9964 cell
Cory.Rowell@wellsfargo.com
```

4.2 Checking spam messages in the database

To view the spam database go to phpMyAdmin (http://192.168.56.10/phpmyadmin) and login using the previously chosen password (root:enisa).

Then select the Shiva database where all final results are stored.

In the `spam` table you can find all distinct spam messages that were observed.

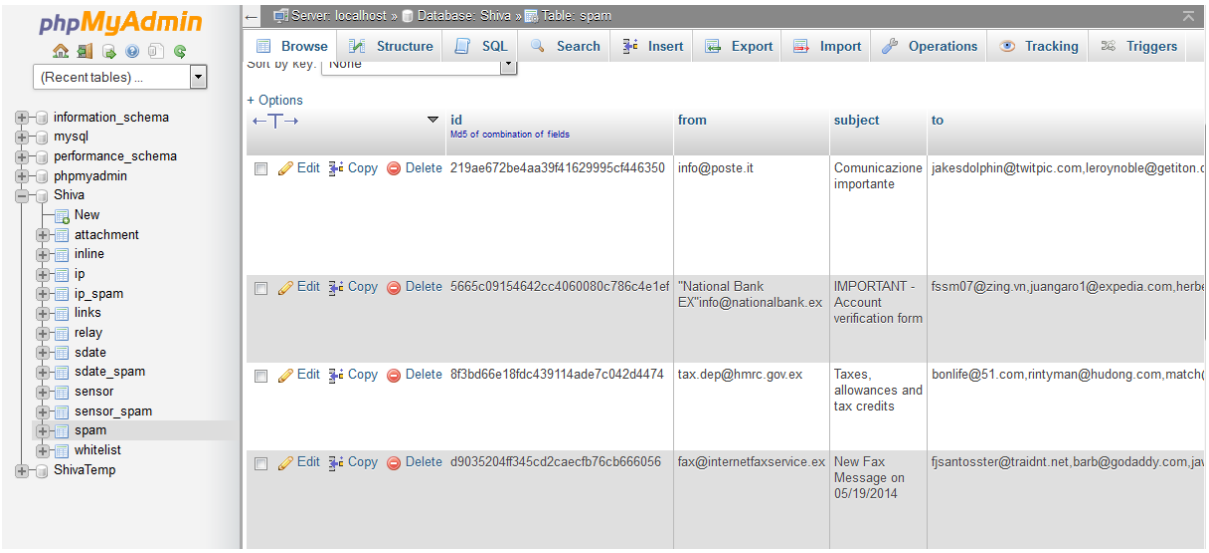

Figure 34. Table `spam` containing spam messages observed during spam campaign.

In the `sdate` table you can view spam campaign timings and the total number of observed messages for each campaign.

	id	date	firstSeen First Occurance of Spam	lastSeen Last Occurance of Spam	todaysCounter
Edit Copy Delete	1	2014-09-15	2014-09-15 01:46:26	2014-09-15 01:50:27	12
Edit Copy Delete	2	2014-09-15	2014-09-15 01:50:26	2014-09-15 01:50:26	17
Edit Copy Delete	3	2014-09-15	2014-09-15 01:50:27	2014-09-15 01:50:27	10
Edit Copy Delete	4	2014-09-15	2014-09-15 01:50:29	2014-09-15 01:50:29	12

Figure 35. Table `sdate` - spam campaigns timings and totals.

The `links` table lists all observed links in spam messages:

	id	date	hyperLink	spam_id
Edit Copy Delete	1	2014-09-15	https://service.ringcentral.com/?rdr=/tellfriend/	d9035204ff345cd2caecfb76cb666056
Edit Copy Delete	2	2014-09-15	http://service.ringcentral.com/picture/email/banne...	d9035204ff345cd2caecfb76cb666056
Edit Copy Delete	3	2014-09-15	http://nationa1bank.ex/accouns_verify/?43A34511111	5665c09154642cc4060080c786c4e1ef

Figure 36. Table `links` - urls found in spam messages.

The `attachment` table lists all observed attachments in various spam campaigns.

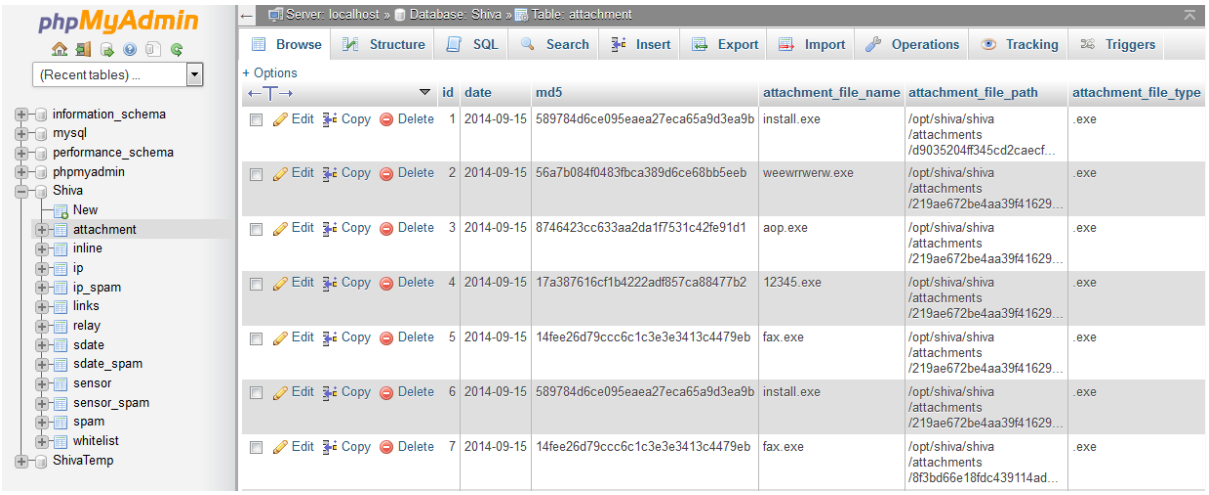

Figure 37. Table `attachments` - attachments found in received messages.

In the table `ip` there are source IP addresses from which spam messages originate. For this particular exercise, all messages were sent from localhost so there will be no external IP addresses in this table.

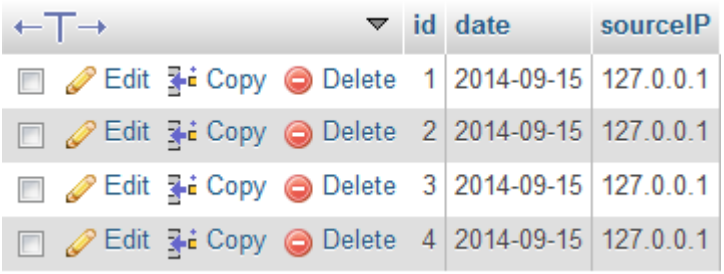

Figure 38. Table `ip` - source ip addresses of spam messages.

4.3 Checking raw spam

To view raw spam messages go to /opt/shiva/shiva/rawspams directory. In this directory there should be one file for each distinct spam message present in `spam` table in the database.

Viewing Shiva raw spam messages

```
$ cd /opt/shiva/shiva/rawspams
$ ls
```

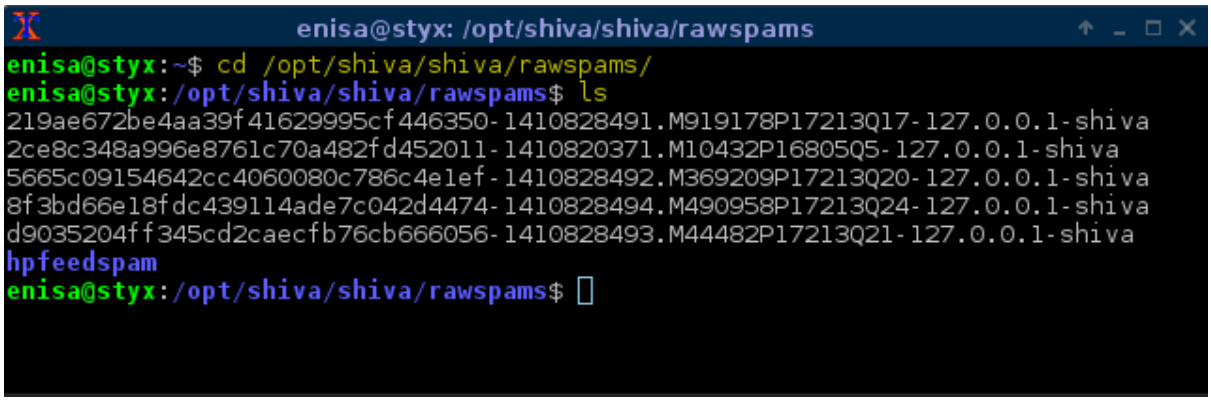

Figure 39. Viewing distinct raw spam messages.

Each file contains a slightly pre-processed spam message generated by shivaAnalyzer and includes e-mail headers (but not SMTP headers), the message body, and any attachments it contains.

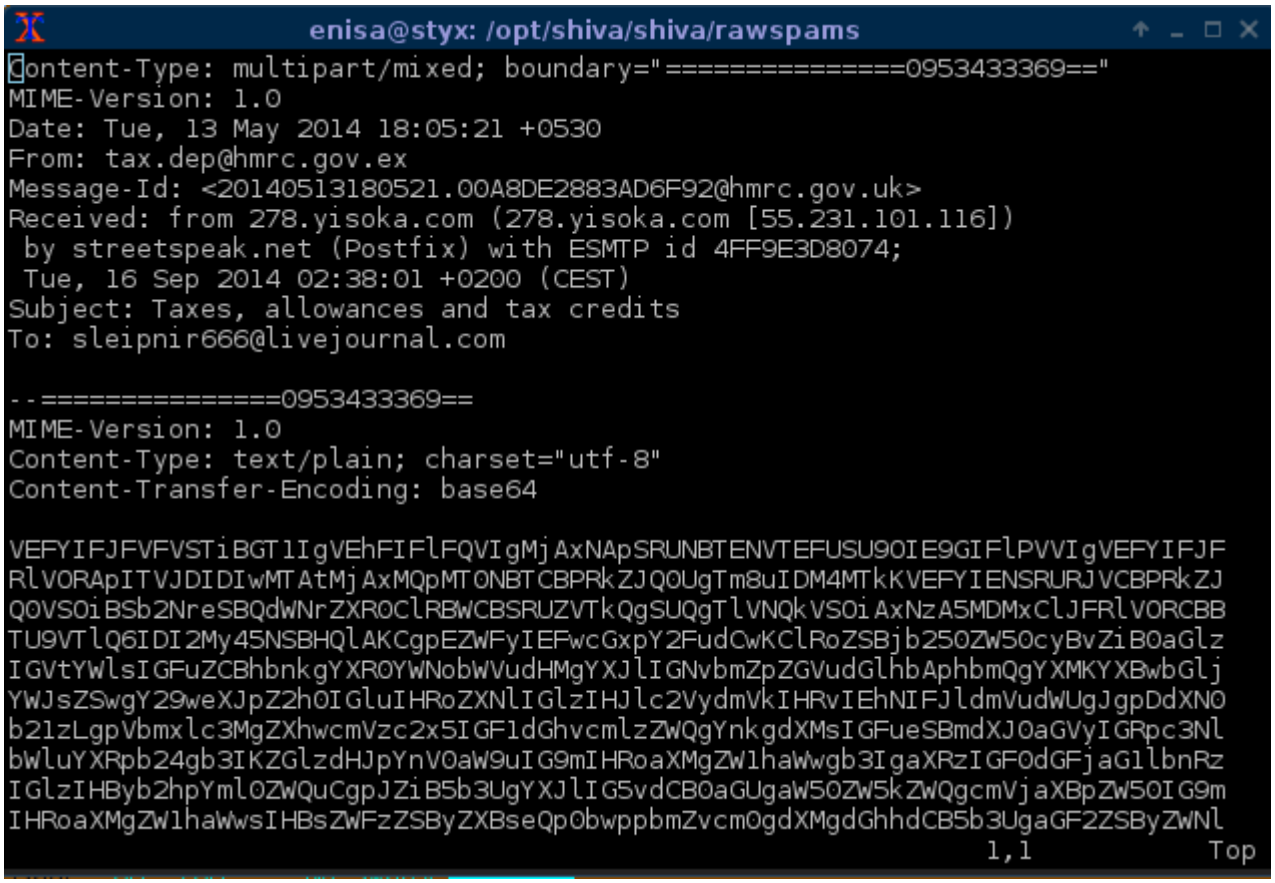

Figure 40. Fragment of example raw spam message file.

Additionally, extracted attachment files can be found in the /opt/shiva/shiva/attachments directory.

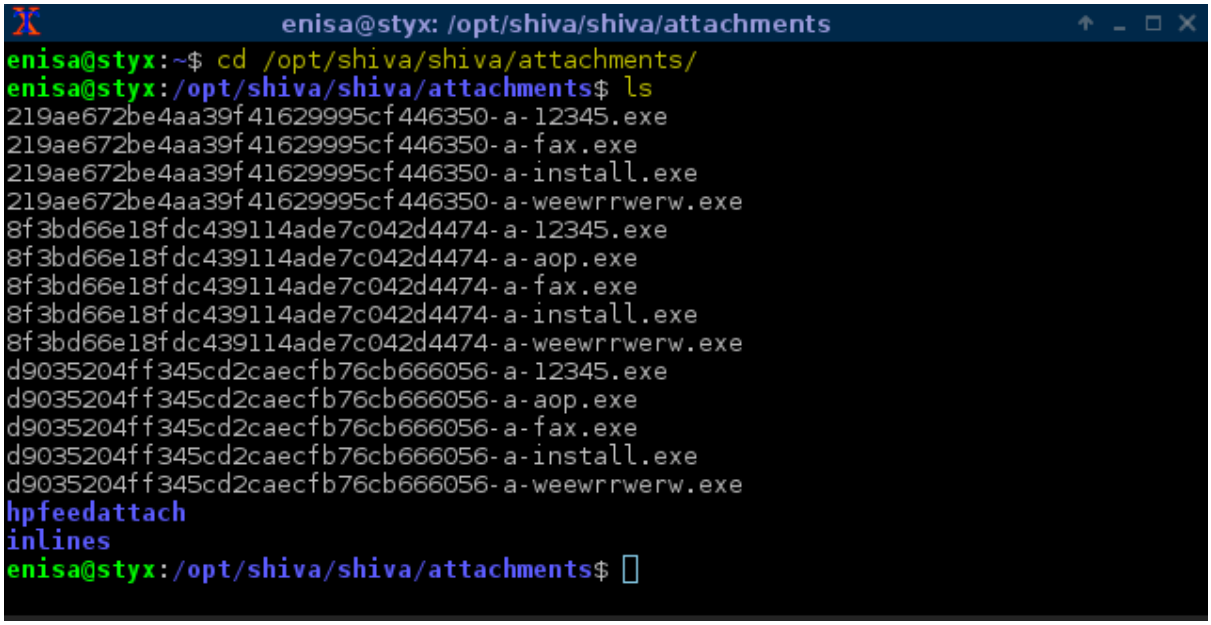

Figure 41. Viewing extracted attachments.

4.4 Checking spam in Viper

Next, students should check if the extracted attachments were successfully added to the Viper database.

Go to the Viper directory and start Viper in the context of the *enisa* project:

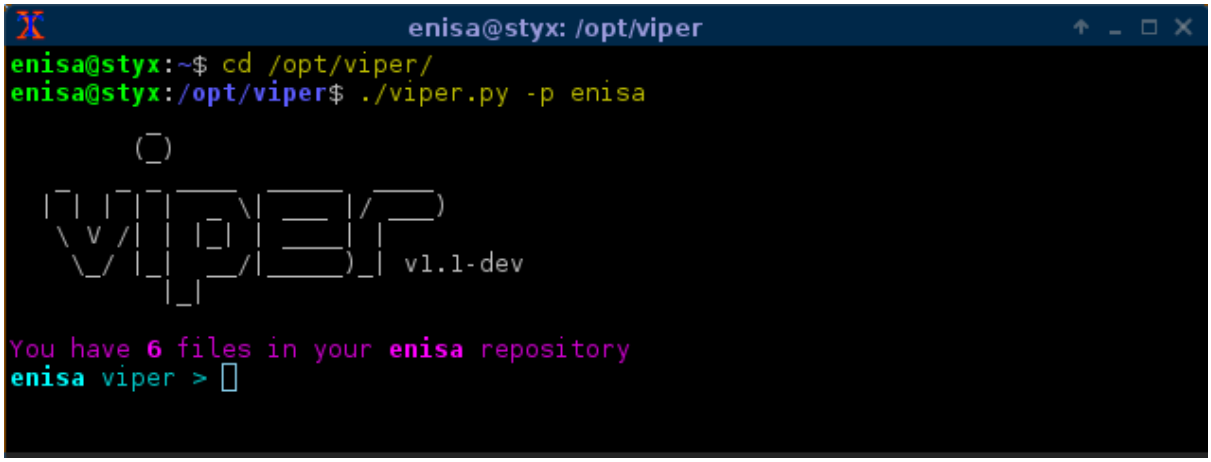

Figure 42. Checking if samples were uploaded to Viper.

Next try to find files with the tag 'shiva':

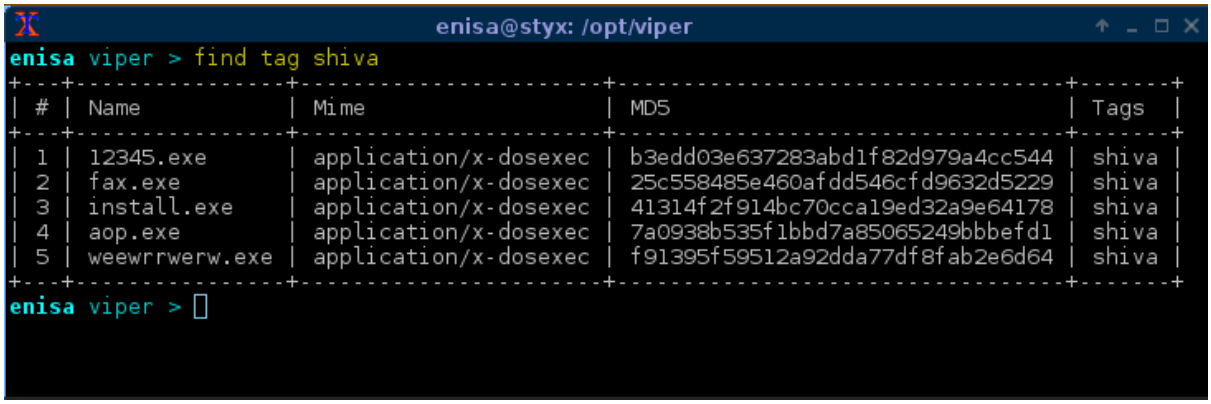

Figure 43. Finding samples with the 'shiva' tag in Viper.

In the result, there should be a list of samples obtained from spam messages and extracted by the Shiva honeypot. Those samples will be used in later exercises focusing on the malware analysis. Please note that the final list of samples might differ from the list presented on the screenshot.

5 Exercise summary

In this exercise, students had the opportunity to configure an e-mail honeypot Shiva, acting as a spam trap for any incoming messages. From the perspective of a spam bot, all e-mail messages sent to Shiva were seemingly delivered or relayed - while in reality they were stored and processed by shivaAnalyzer module scanning for any attachments or suspicious urls.

Next, students set up a simple artifacts repository based on the Viper project. Viper not only allowed students to store and manage samples (adding notes, tags, etc.) but also to perform some preliminary sample analysis. Additionally, thanks to Viper's modular architecture, students were able to extend Viper functionality with an additional module.

After completing the installation and configuration of the Shiva honeypot and Viper repository, the students started a special script generating spam messages and sending them to local spam trap. This simulated spam campaign and allowed students to obtain artifacts that will be used in the later exercises for the malware analysis.

6 Bibliography

1. ENISA - Honeypots training material http://www.enisa.europa.eu/activities/cert/support/exercise/files/Honeypots_CERT_Exercise_Handbook.pdf (accessed 15. October 2014)
2. Proactive detection of security incidents II – Honeypots http://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-of-security-incidents-II-honeypots/at_download/fullReport (accessed 15. October 2014)
3. M3AAWG Best Current Practices For Building and Operating a Spam trap http://www.maawg.org/sites/maawg/files/news/M3AAWG_Spamtrap_Operations_BCP-2013-10.pdf (accessed 15. October 2014)
4. Malware Analysis: Environment Design and Architecture http://www.sans.org/reading-room/whitepapers/threats/malware-analysis-environment-design-artitecture-1841 (accessed 15. October 2014)
5. Spam Honeypot with Intelligent Virtual Analyzer https://github.com/shiva-spampot/shiva (accessed 15. October 2014)
6. The Honeynet Project https://github.com/shiva-spampot/shiva (accessed 15. October 2014)
7. Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients https://www.syssec.rub.de/media/emma/veroeffentlichungen/2012/08/07/MonkeySpider-Sicherheit08.pdf (accessed 15. October 2014)
8. Malware Sample Sources for Researchers http://zeltser.com/combating-malicious-software/malware-sample-sources.html (accessed 15. October 2014)
9. Fuzzy Hashing http://jessekornblum.com/presentations/htcia06.pdf (accessed 15. October 2014)
10. VIPER Time to do malware research right http://viper.li/ (accessed 15. October 2014)

ENISA
European Union Agency for Network and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office
1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu