CS2107 Self-Exploration Activity 10: Web Security

Notes:

In this Activity 10 about **web security**, you will perform the following:

1. To set up a web server running a **buggy web application** to be attacked in the 4 subsequent tasks;
2. To perform an **OS Command Injection** attack;
3. To perform a **SQL Injection** attack;
4. To perform **reflected** and **stored XSS** attacks;
5. To bypass an **anti-XSS input-sanitization** mechanism.

Note that you can also view the ***accompanying demo video*** about web security, which has been uploaded to our module's Multimedia on LumiNUS.

Task 1: Setting up a Server with a Buggy Web Application

For web attacks described in the 4 subsequent tasks below, you can use VirtualBox to set up and run the following 2 VMs:

- a VM for your **Ubuntu desktop**: as your **attack host**;
- a VM for a **web server** running a **buggy web app**: as your **target host**.

For the buggy web app, we can use the "***Damn Vulnerable Web App***" **(DVWA)**: http://www.dvwa.co.uk. It is a PHP/MySQL-based web app that is purposely made vulnerable for learning and testing purposes. You can follow its set-up instructions as given in https://github.com/digininja/DVWA. For a much

simpler set-up, however, you can simply download and utilize its ISO image file, DVWA-1.0.7.iso, which is cached at https://drive.google.com/file/d/1mN-zVSFWkXeUgsWF033qCVFXe-V280gN/view?usp=sharing.
The file's MD5 hash value is 9484d8e2154d4e01fbd742cd7c10affd.

For your required set-up, do **create the 2 needed VMs** in VirtualBox by following the steps given in https://docs.oracle.com/cd/E26217_01/E26796/html/qs-create-vm.html. We need to make sure that the 2 hosts can communicate with each other. For this, you can set the 2 VMs to use either the "*Internal Networking*", "*Host-only Networking*" or "*NAT Network*" **networking mode** in VirtualBox (see: https://blogs.oracle.com/scoter/networking-in-virtualbox-v2).

Once your set-up is ready, do run the 2 VMs. On your attack host, open **Firefox**, and enter the IP address of your target host in the URL bar. Using Firefox, do log into the DVWA app by using its default credential `username`="`admin`" and `password`="`password`". Then, click "DVWA Security" on the DVWA menu, and set the "Script Security Level" to "low".

Task 2: Performing an OS Command Injection Attack

Let's launch an **OS Command Injection** attack. In this attack, you provide an IP address to be pinged by the server, but additionally include an extra **OS/shell command** to be executed by the target server.

Do click "Command Execution" on the DVWA menu. You should be able to see its "Ping for FREE" service. In the textbox shown, enter some IP address to ping, e.g. "`192.168.1.8`", and notice the output of the conducted ping operation.

Now, enter the following string: "`192.168.1.8; cat /etc/password`". You should be able to see that, in addition to the given ping result, the content of the `/etc/password` file is additionally shown. This tells us that the server also run your injected OS command "`cat /etc/password`".

Task 3: Performing a SQL Injection Attack

Let's now do a **SQL Injection** attack. Do click "SQL Injection" on the DVWA menu. You can try entering some names into the shown User ID textbox, such as "`Alice`" and "`Bob`". Notice that, upon receiving a user ID that does not exist in its user database, the app just outputs nothing.

Now, enter "**`CS2107'`**" as your input. Notice that the input string contains a single quote ("`'`") character. You should be able to see that, due to this character, the server gives a SQL syntax error message.

Exploit this vulnerability by entering "**`CS2107' OR 1=1; --`** ". Note the space character " " after the "`--`" characters as it may be needed by the used database. Because of our attack, you should be able to see **all records** in the table.

Task 4: Performing Reflected and Stored XSS Attacks

Let's now do **XSS attacks**, both reflected and stored XSS attacks. For the **reflected XSS** attack, do click "XSS reflected" on the DVWA menu. You can enter some names into the shown name textbox, and notice how the web app **reflects back** your entered name strings in its corresponding response pages. Let's launch the attack by entering your following attack Proof-of-Concept (PoC) string:

"`<script>alert(”XSS-ed”)</script>`"

As the string is reflected back intact to web client, the entered script gets executed by your browser. This shows that the web app is subject to reflected XSS attacks.

Now, let's us do the **stored XSS** attack. For that, do click "XSS stored" on the DVWA menu. Similarly, you can put the same attack PoC string into the Message textbox. Upon your message string posting and follow-up message string display in the returned webpage, the entered script gets executed by your browser.
This demonstrates that the web app is subject to stored XSS attacks.

Task 5: Bypassing an Anti-XSS Input-Sanitization Mechanism

DVWA actually employs some **counter measures** to prevent various web attacks, including the XSS attacks described in Task 4. Let's now see how DVWA running with a **higher security level** can deal with our reflected XSS attack.

First, click "DVWA Security" on the DVWA menu, and set the "Script Security Level" to "**medium**". Some counter measures are now activated.

Now, do select "XSS reflected" on the DVWA menu, and enter again your following attack Proof-of-Concept (PoC) string in the shown name textbox:

"`<script>alert(″XSS-ed″)</script>`"

Notice that, from the returned webpage, your string has been sanitized by the server and turned into the "`alert(″XSS-ed″)`" string. You can guess that the tags "`<script>`" and "`</script>`" are removed by the app's input sanitization.

Can you bypass the app's anti-XSS input-sanitization mechanism? Yes! You can try entering the following attack string variant instead:

"`<Script>alert(″XSS-ed″)</Script>`"

Now, you can see that the script string can bypass the mechanism, get reflected back to your browser and then gets executed by your browser. This shows that the applied mechanism fails to sanitize both tags "`<script>`" and "`</script>`" written with different letter case!

(***Note***: You can try setting the "Script Security Level" to "high", and see how you can still bypass the app's applied anti-XSS input-sanitization mechanism. This part is more challenging, and is left to you as an extra exercise. *Have fun!)*