Web Application Vulnerabilities

We will use a tool called Damn Vulnerable Web Application (DWVA) to get some perspective and experience with some software vulnerabilities: Command Execution, SQL Injection, and Cross-Site Scripting.

1. First, Build your playground

1. VMware > Take a snapshot of your Windows VM & name it “Pre-setup” (FYI: this is your backup!)
2. Power VM
    - a. If you take a snapshot while the VM is on, then wait for the snapshot to complete (there’s a status bar near the bottom).
3. Via Windows Control Panel/Programs/Uninstall:
    - a. Uninstall XAMPP
    - b. Turn Windows Features on or off / unselect Internet Information Services (it’ll reboot)
4. Download, save & install XAMPP
    - a. UAC warning = OK
    - b. Select components: select Apache + MySQL + PHP only
    - c. Some Next’s to Finish install, then start XAMPP Control Panel
5. Start Control Panel → Pick Language
6. Start Apache & MySQL
7. Allow both services through both firewall Private & Public networks
8. Check localhost via Internet Explorer (should see default XAMPP splash page)
9. Navigate to c:\xampp\htdocs (default files xampp comes with)
10. Delete them all
11. Download & Save, then right-click to Extract all from DVWA zip file
12. Create a new folder called “DVWA” under c:\xampp\htdocs.
13. Copy all extracted DVWA-master directory’s files (but not directory itself) to the new c:\xampp\htdocs\DVWA folder you just created
14. Check 127.0.0.1 (spot check: should see DVWA directory)
15. Check 127.0.0.1/DVWA (spot check: should get error re:config.inc.php to configure … so spoiler …)
16. Rename said file by removing .dist portion only
17. Open, change & save config.inc file:
    - a. For blank db_password (mindful the =‘’; syntax)
    - b. Default security level -
        - i. $_DVWA['default_security_level'] = ‘low’;
    - c. Change recaptha settings -
        - i. $_DVWA['recaptcha_public_key'] = '6LdK7xITAAzzAAJQTfL7fu6I-0aPl8KHHieAT_yJg';

        ii. $_DVWA['recaptcha_private_key'] = '6LdK7xITAzzAAL_uw9YXVUOPoIHPZLfw2K1n5NVQ';

18. Via 127.0.0.1/DVWA , create database (spot check: should see logon screen after it is created)
19. Login to the DVWA application via admin:password (#Security…Pfft!)
20. XAMPP Control Panel → Apache’s Config / PHP.ini
    a. Via Alt+F, find & change ‘url_include’ from Off to On, then save & close
21. XAMPP Control Panel →Stop & start Apache service
22. XAMPP Control Panel / Config → Check Apache & MySQL for Autostart of modules
    a. When rebooting the VM, you’ll have to re-launch XAMMP by opening it. Then these start auto start. Of course, you could figure how to auto start applications XAMPP at start …
23. Refresh webpage, re-check Setup/Reset DB → The ‘allow_url_include’ = Enabled (green & happy!)
24. Sometimes the default security level doesn’t stick, so after logging in, check DVWA Security tab. You can change it to low here, also.
25. All systems go!

2. Command Execution

DISCUSSION: Whenever you hear or read that a compromise could lead to arbitrary command execution, we should pay close attention to the warning. This type of attack means that someone could execute a command within the context of the user running the vulnerable service or application. If it is a remote arbitrary command execution, then increase our awareness and monitoring of the affected service or application.

Objective: Learn how attackers can execute arbitrary commands via a web application and, potentially, gain a shell.

Executing commands requires knowing some of the common commands. In Windows, these are normally executed in the CMD window. In Linux, it’s a terminal window. In some environments, it’s refer to as the CLI (Command Line Interface)

Some of typical commands are:

- dir: directory listing for Windows
- ls: directory listing for Linux
- Date
- Hostname
- More: print out the contents of a file
- Find: find a text string in files
- cd: change directory (.. is one directory level up, and ../.. is 2 directory levels up)

Select the "**Command Execution**" module → This is an application that will ping an IP address. However, this application is super flawed and allows commands to be added after the address. <insert ominous music here…>

First, try it the way it was intended to work

To test it out, use it the way it was designed: enter your host machine IP address & click Submit.

You should see the results of the ping output. That lets you know the command works. Now we are going to try to inject something malicious. >:)

In Windows CMD line and Linux Bash, a double ampersand (&&) allows us to chain commands together which allows command execution.

Type the commands:

- **YOUR_IP && hostname**
  - **What do you see?** You should get the ping response and the hostname of the system running the application (i.e. 10.10.10.10 && hostname)

- **YOUR_IP && dir**
  - You should get the ping response, but also a directory listing from the server

- **YOUR_IP && systeminfo**
  - **What do you see?** You should see lots of juicy info. Review it to make sense of it.

Investigation:

Using what you know so far and information from networking in general, try to find the following.

- Find the MAC address of your playground.
- Find the original install time & date.
- Find BIOS version.

SUBMISSION: Take Snips/Screenshots of your inputs & outputs.

3. SQL Injection

OBJECTIVE: Demonstrate how to test for SQL injection vulnerabilities

SQL Injection Menu

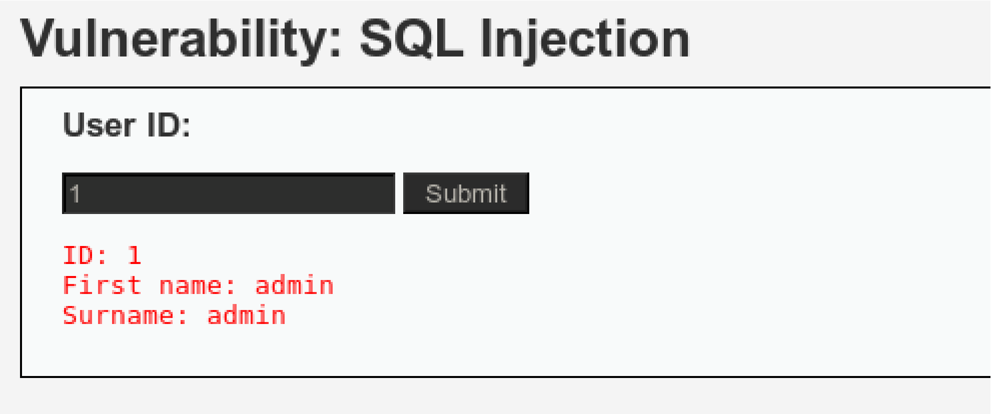
- Select "SQL Injection" from the left navigation menu.

Basic Injection

Extract First Name and Surname from ID Field
In order to exploit SQL injection vulnerabilities, we need to figure out how the query is built in order to inject our parameter in a situation that the query will remain true.

- Input "1" into the User ID: Text box & click Submit.

Note, the webpage (via its code) is **supposed to** print ID, First name, and Surname to the screen. Again, we are querying the database.



```
ID: 1
First name: admin
Surname: admin
```

This means that the query that was executed back in the database was the following:

SELECT First_Name,Surname FROM table WHERE ID=’1′;

We can see the query in the DVWA exercise, but in most cases we wouldn’t know so we are making a guess.

Testing for SQL Injection Vulnerabilities

ACHTUNG!

- ' => Test each field in a web form with a single quote to see if it generates an error.
  - If you get an error, then FYI … you're able to have some SQL injection fun
- ' or 1=1# => Test to see if this SQL code returns results.
  - What do you see? Amazing how this still works …

Note that digits in SQL queries don't contain single quotes around it unless it is being expressed as a string.

You should see the first and last name of all employees → Get a screen shot of that, as that may come in handy!

Through some more sophisticated methods, you can also get the username and password

Try this as the input:

- ' or 0=0 union select null, version() #

Interesting … at the end, we should see the database version running our DVWA → Snip that!

How about this?

- ' or 0=0 union select null, user() #

Pretty … what's account is DVWA running under? → Snip another clue!

Ok … now for a slightly more challenging query (all one line):

- ' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #

Are those … usernames AND password hashes? #Shiny! → Snip all day long!

Cool … Now we open a password hashcracker (or another online password cracker of your choice), and enter 2+ of the encrypted password hashes and then decrypt them.

- Get a screenshot of the decrypted passwords
- Challenge question because you want to dig deeper … What type of Checksum Hash algorithm are those passwords? (Hint: we just covered checksum hashes!)

SUBMISSION: Assemble all the snips showing the SQL Injections & outputs along the way in 1 document.

3. Cross Site Scripting (XSS)

DISCUSSION: Cross-site scripting attacks (XSS) can lead to malicious attacks where the user is redirected to a malicious site, download exploits in the background with malicious iframes, annoy users with endless pop-up messages, etc. These attacks are difficult to detect so programs like NoScript have been designed to disable scripting on websites. However, many sites require scripting for a good 'user-experience' so these tools can annoy some people so they just disable it. XSS attacks should be taken seriously because they can pose high-risk vulnerabilities to an organization or users visiting the organization's site. Even more problematic is that a large site like CNN may not be compromised, but one of their ad providers could be compromised and that has the appearance of CNN carrying a malicious payload, when it is a third-party.

Some web browsers, like Chrome are designed to prevent the impact of XSS attacks by not interpreting the HTML code as an HTML Entity. This will change the code:

<b>hello</b>

to

&lt;b&gt;hello&lt;/b&gt;

So the browser translates as a literal string of "<b>hello</b>" and not as HTML code.

OBJECTIVE: Demonstrate how XSS works by testing to determine if an application is vulnerable and then performing the attack.

Use the module "XSS reflected" on the left-hand menu.

Test for basic XSS vulnerability by entering the string:

<b>hi</b>

If the results display "hi" in bold, then there is a potential vulnerability and an opportunity to search for more treasure. ☺

Try an attack more interesting like a pop-up message by entering:

```
<script>alert(“attack”);</script>
```

However, in some browsers XSS has been disabled (boo!). But it might work if we disable pop-up under advanced settings. >:)

SUBMISSION: Assemble Snips/screenshots of your inputs and outputs.