Vulnerabilities in WebApplications

In the Scope of WordPress 6.7.x Plugins and Themes

Bennett Poulin

Gabriel Urbaitis

CS544 Intro to CyberSecurity

Dr. Afsah Anwar

May 10, 2025

Introduction

Our goal was to study vulnerabilities in web applications. Specifically in the context of the popular Content Management System (CMS), WordPress. Wordpress is used by millions of websites, and is an easy way for non-technical people to be able to create and administer web content. Its popularity has made it a large target for vulnerabilities and exploits. We will study these exploits, see if any are reproducible, what we can learn from them, and how they can be mitigated.

We aim to further explain the susceptibilities we are exploring, their applicability and landscape. We will demonstrate the kind of breadth and capacity these problems encompass. We shall also enumerate our approach and why we chose reproducing in a static and safe environment rather than finding these liabilities in the wild. We will also specify the numerous difficulties we had in exploring these vulnerabilities. Additionally we intend to discuss any successes we had in analysis. Finally, we will conclude with the outcomes our findings have led to in how to better be safe when using the WordPress Software.

After reading our paper you should have a good idea how to set up a safe testing environment to conduct your own research. We will describe our process of finding vulnerabilities such as reflected XSS and SQL Injection. Additionally we will discuss in great detail common complications we encountered such as missing plugins, barriers to using found plugins and WordPress' built-in defenses. Finally we'll relay our successful testing of exploits that you can try yourselves.

Problem Statement

Despite WordPress Core being relatively secure, its third-party plugins introduce a vast attack surface. Many of these plugins are developed with little supervision, inconsistent coding standards, and poor documentation, leading to a high volume of vulnerabilities such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL Injection. The problem is worsened by the fact that many site administrators install plugins with barely any vetting, which leaves their sites exposed to attacks that can compromise data, users, or even full system control. Our goal is to assess whether these vulnerabilities are reproducible in a controlled environment, evaluate how they can be chained or escalated, and understand the limitations of WordPress's built-in defenses.

Problem Relevance

WordPress is easily the most popular CMS out there today. Being free to install and open-source adds to its following. WordPress boasts being the backbone for over a third of the internet "It is also the platform of choice for over 43% of all sites across the web" (WordPress, n.d.). Even if their claim is only partially correct, that's still millions and millions of sites across the internet. This prevalence definitely makes it a more ripe target for malactors on the web.

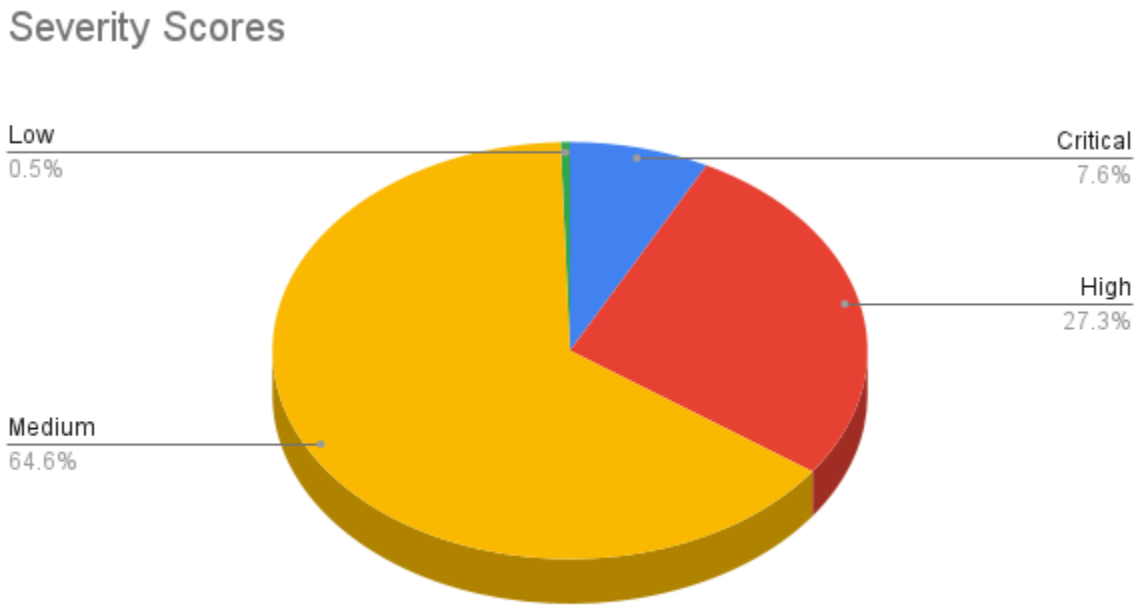
Of this multitude of sites using a WordPress backbone are numerous prominent ones. Many governments, large corporations, and universities use WordPress in some fashion.

Whitehouse.gov, state.gov, and nasa.gov are all WordPress sites. Corporations such as Walt Disney and Sony use WordPress in some of their public facing sites. The New York Times uses WP for their press site, not the newspaper site. Harvard University uses WP for their main site. MIT, Stanford, Cornell, and Princeton utilize WordPress for various student and departmental websites. The University of New Mexico uses WordPress for its Press site https://www.unmpress.com/. With this many targets all on one platform it is easy to see why attackers would choose this application to focus their efforts on.

Problem Scope

There are many sites that disseminate information on WordPress and its security statuses. We used solidwp.com to gather some statistics concerning the magnitude of vulnerabilities in WordPress (SolidWP, 2025). Solidwp.com puts out a weekly list of disclosed vulnerabilities in three sections: core, plugins, and themes. Each vulnerability lists the affected section, the Vulnerability Type, the Severity Score (low to critical,) and a link to the associated CVE. Plugins and themes that haven't been removed by WordPress will also include a link to the wordpress.org page of the listed plugin or theme.

By scraping the website for 52 weeks worth of data between 2025-01-01 and 2025-01-10 we found 9496 individual vulnerabilities. Impressively only 6 of these citations were attributed to the Section core. This can be interpreted that the core of WordPress is more peer reviewed and inspected before its release. The following chart shows that severity scores were distributed in such a way that Critical and High scores shared more than a third.



Low scores were the least represented with half a percentile. Similarly for Sections, plugins were about ninety six percent of the reported vulnerabilities, and themes took most of the

remaining numbers. The largest Vulnerability Type by number of incidents was by far Cross Site Scripting (XSS) with 4554 incidents. Broken Access control, Cross Site Request Forgery (CSRF), and SQL Injection were the next most prevalent respectively. With almost twelve and half vulnerabilities being reported daily on average, one can exactly see the immensity and pervasiveness of the problem.

Method

To avoid any potential legal or morality issues our study was limited to previously disclosed vulnerabilities being studied in a closed and controlled environment. The following sections will enumerate setting up our environments, how we tested the vulnerabilities, and what approaches and tools were used. Outlining our process in such a way that we intend our efforts to be reproducible depending on the availability of the afflicted softwares in their unaltered form.

Setting Up Environments

The two researchers each used their own environments with strikingly similar structures. One used an ARM computer as the attacker and an AWS Virtual Machine (VM) running Ubuntu as the web server. The second setup used two VMs administered by VirtualBox, one as the attacker and the other as the web server. By utilizing the two environments we were able to probe twice the number of plugins and themes.

Environment One

Environment One used a two-host architecture combining a local attacker machine with a cloud-hosted WordPress server. The attacker machine was a MacBook running macOS 14 on Apple Silicon. It was used to serve malicious HTML pages, monitor traffic using browser developer tools, and replay HTTP requests with command-line tools like curl and wget. For CSRF testing, a simple Python web server (python3 -m http.server 8000) was used to host created payloads locally. The browser-based testing environment provided insight into cross-origin behavior, session cookies, redirect behavior, and frame-related security headers.

The target machine was an AWS EC2 instance running Ubuntu 22.04 LTS. This instance hosted the WordPress installation and was publicly accessible via its assigned Elastic IP. The server was configured with Apache, PHP 8.1, and MariaDB, and hardened using mysql_secure_installation and UFW rules to allow only HTTP and HTTPS traffic. WordPress was manually installed in /srv/www/wordpress, and its file permissions were configured for compatibility with the web server. Several vulnerable plugins were installed manually from .zip archives due to write permission issues affecting WP-CLI. These included teachPress (CVE-2025-1320), GlobalPayments WooCommerce (CVE-2025-22767), and EZ SQL Reports (CVE-2025-2319). This setup replicated a plausible production-like WordPress environment while allowing full control and logging access for vulnerability testing.

Environment Two

The second environment used two Virtual Machines with VirtualBox to administer them. The first VM was the attacker. Kali Linux was chosen for the attacker's machine's Operating System as it comes preloaded with many useful tools and capabilities. The VM specific for VirtualBox is available on their website kali.org. Additional tools such as XSStrike were added as needed, but they were few and far between. A Natted Network was added in VirtualBox so the two machines could communicate.

The web server was built on Fedora Workstation 41. This image was chosen for convenience and similarity to the RedHat Enterprise Linux (RHEL) environment. Apache was installed and enabled and the firewall was punched through to allow for both HTTP and HTTPS. PHP version 8.3 and its dependencies were then installed as well as mariadb-server which is MySQL's free clone spinoff. The software binary mysql_secure_installation was then run to perform the initial hardening of the Database Management System (DBMS).

On top of this second VM, the web server, WordPress was installed. Fedora has some excellent documentation on the process (Fedora Project, 2024). Basically WordPress core was downloaded and extracted to the web server's root. A database was created in the MySQL clone. At this point one can complete the installation by navigating to the localhost in the browser and following the on screen instructions, plugging in the values for the database name, user, and password.

Execution and Tools

Browser Tools

We used built-in developer tools in Firefox and Chrome to inspect forms, monitor network requests, and verify whether CSRF and XSS attempts were actually triggering the expected behavior. This helped us confirm when POST requests went out or when injected scripts were reflected or blocked.

Programming Languages

We used HTML and JavaScript to build test pages for CSRF and XSS attacks. We also used Python to run a lightweight local web server, which made it easy to serve these files locally while testing as an authenticated admin. The setup allowed us to test our attacks in a controlled environment without uploading anything directly to the WordPress server.

SQLMap

SQLMap is a tool for automating probing for possible SQL Injection vulnerabilities. This tool was incredibly useful to our efforts. It is conveniently included in the kali linux image for VirtualBox. According to sqlmap.org

> It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and

executing commands on the operating system via out-of-band connections. (SQLMap, n.d.).

Using the tool was incredibly easy and aided in being able to determine if a plugin had a viable exploit without having to spend massive amounts of time crafting individual payloads. Providing credentials was easy for attacks that required authenticated users. It took a long time to run depending on the set depth parameter, but it consistently provided an incredible amount of information.

XSStrike

XSStrike was not included in the Kali Linux VM, but installing was a simple git clone command with an installation of an included python environment. It was similar to SQLMap in that it was a command line tool which then pointed to a web server to try and find vulnerabilities, but differed in some key aspects. Including credentials such as cookies required editing a file every time as it could not read from a stored file inexplicably. The tool would bring up a nano instance during the command runtime and you would have to paste and save your key or cookie, which became tedious rapidly. Adding headers and post payloads to the attack was also exhausting. Many forms needed files in order to validate and process correctly. This required adding the binary to the command line command as an argument, which was messy at best.

BurpSuite

BurpSuite was the Swiss Army knife of pentesting tools. Its proxy allows for capturing of data passing to and from the web server with ease. The repeater functionality aided crafting an attack payload and refining it slowly without having to reset everything everytime. URL encoding is an important aspect to testing these vulnerabilities. BurpSuite had this service available ensuite, with keyboard shortcuts for added ease. Mapping an entire site, listing all the pages and html forms is just the click of a few buttons. BurpSuite was by far the most useful tool we found in our exploration of these vulnerable plugins and themes.

Challenges

Despite using vulnerabilities that were previously disclosed, we faced many challenges in our exploration. A good portion of plugins and themes were no longer available and had been scrubbed from the wordpress.org website. Each of the available plugins and themes came with its own learning curve as to its administration and functionality. Though some plugins and themes had vulnerable code, many were not viable due to WordPress' built-in defenses. Barriers such as language and paywall also kept us from exploring many of the vulnerabilities listed. We will discuss each of these complications further in the following sections.

Finding Viable Vulnerabilities

To find good vulnerabilities to test, we went through recent CVEs listed on sites like SolidWP and cve.org. We looked for plugins with known issues like CSRF, XSS and SQL

injection, and checked if they still had downloadable versions available. Some plugins were removed or locked behind licenses, so just finding one we could actually test was its own challenge.

Unfamiliarity with Modules

WordPress administrators and content managers generally work with a few sets of modules and become fairly familiar with them over a period of time. We did not have that luxury. Every plugin we were to investigate was a crash course in learning a new software set. Akin to the first time one sees Microsoft's Excel worksheet. Aiding to this difficulty is a lack of conformity for plugins, poor documentation, and lack of consistency. Each of these individually is surmountable, together they added significant time to the project that we did not have. It was a real gamble to see which plugins and themes would pay off in our research.

There seemed little rhyme or reason to how new plugins were displayed, the interface to administer them, or how they were implemented on the site. Some plugins would add a menu item to the sidebar, and some would add them to the top navigation. Still others would do neither and only add themselves to the general plugin administration menu.

Finding out how to use the plugin once you found its location was another laborious task. A number of plugins would show complex forms and cryptic checkboxes and radio buttons with little explanation for what they did. Others had very little in the way of customization or administration.

Documentation for these plugins was as hit or miss as the administration interfaces. Most plugins had some form of documentation on the WordPress.org site, but some were just a few paragraphs and very few were detailed with useful instructions. Finding good documentation went a long way to actually letting us see if an attack was going to be viable or not.

Once a module was customized to expose the possible attack vector, it had to be implemented on the site somehow. WordPress uses shortcodes in the form of [short_code_name] on the page's wysiwyg in order to display some predefined code snippet or functionality. You could select from a long list of these shortcodes, but it was difficult at best for us novices to choose the correct plugin functionality from a large list of just names. Again, the documentation was irregular. Some would list their shortcodes, how to use them, and what they did, while others simply did not.

In short, being completely unfamiliar with each plugin we were trying to use led to harsh difficulties and extreme delays. The modules themselves were inconsistent in the manner of display and application. The documentation was either sparse or brilliant and may or may not have included how to utilize the plugin or theme in the site with shortcodes. All these factors took time away from actually testing the vulnerabilities and were the majority of the learning process in each angle of attack explored.

WordPress Built in Defenses

WordPress uses things like nonces and session checks to stop fake requests, which made our CSRF attacks harder. Even when we copied how the real forms looked, they didn't

work unless the right token (nonce) was there. We also noticed that WordPress blocks its login page from loading in iframes, which made silent CSRF attacks trickier to track.

Although the tools we used found many vulnerabilities, crafting viable payloads remained difficult. For example, several XSS vulnerabilities were found by XSStrike, though they could not produce any vectors that were usable, and neither could we. Some of the SQL Injections we found would be passed directly to the database as per the code… but in practice did not behave that way. Though not explicitly stated these appeared to be the actions of WordPress parameterizing queries and stripping script tags independent of the module code. Why some modules we investigated were affected by this behaviour and not others remains an interesting topic and will be explored in future work.

Complications in Setup

One of the biggest problems we ran into was getting the right plugin versions. Most of the recent versions of the plugins were already patched, so we had to search for older releases that still had the vulnerabilities. Sometimes there were no clear download links, so we had to dig through archives. Even when we managed to download what we thought were vulnerable versions, some of them had already been patched too, which made testing unreliable and more time-consuming.

Language Barriers

Technology has aided greatly in global communication and understanding. Nevertheless, language and translation barriers remained an issue in our research. Translation tools can still only go so far, especially when the subject matter is unfamiliar even when presented in one’s native tongue.

One example of this was our investigation of the WPCOM Member plugin containing a severity score of Critical. The CVE for this plugin told us exactly which field was vulnerable to SQL Injection. Installing the plugin using the slug was no problem and the codebase was even in english. However, the admin interface was all in Chinese Hanzi characters. We turned to the documentation to see if it could help us administer the plugin to find that it was also in Hanzi. We used several translation sites, including Firefox's built-in page translation. That allowed us to progress a little, but we were ultimately unsuccessful in finding the endpoint with the vulnerable form field.

Paywall Barriers

While WordPress is free, that isn’t necessarily true for its extensions. Any plugin can charge a fee for use and many have a model that distributes a freemium version with a premium upgrade. While these pay plugins were easy enough to avoid as the initial vulnerable plugin – due to the complications outlined in the Unfamiliarity with Modules – it was less easy to identify paid plugins as a dependency for a different plugin that we were investigating.

One example of this difficulty was exploring the WP-Recall plugin. This plugin seemed to claim being a one-stop shop for registration and ecommerce needs. Four CVEs were listed on the March 12, 2025 vulnerabilities report from solidwp.com (SolidWP, 2025) ranging from Broken Access Control to SQL Injection and XSS. One of these was lack of validation for members registering with a phone number.

With the aid of another (paid) plugin, this module could log users in using SMS messages. The plugin had a recurrent cost of fifty dollars. The idea was that if an attacker knew the phone number of an admin user, they could register a new account with the same phone number. Logging in with the SMS option would give both the previous account and the new one Admin privileges, elevating the new account of the attacker. We were unfortunately not able to test this further since the SMS login plugins we found all cost money.

Findings

We looked at a few WordPress plugins with known problems to see if we could get the attacks to work ourselves. The ones we tried were TeachPress, Thumbnail Carousel Slider, GlobalPayments WooCommerce, and EZ SQL Reports. Some of the attacks worked, and some didn't, but each one helped us learn more about how these kinds of exploits show up in real websites.

TeachPress

TeachPress is a plugin designed for academic use. It handles publications, courses, and student registration, and is often used in university websites. It appeared on SolidWP's vulnerability list for April 2025 with a medium-severity CSRF vulnerability (CVE-2025-1320).

This vulnerability listing gave us a clear starting point. According to the advisory, the plugin allowed arbitrary actions to be performed by an authenticated administrator if they were tricked into clicking a malicious link. This is the textbook definition of a Cross-Site Request Forgery (CSRF). We gathered more data by inspecting the plugin source code and quickly identified several areas of functionality—most notably, SQL report creation—where user input was processed without nonce validation.

After setting up the plugin on a vulnerable WordPress instance, we created a malicious HTML file (csrf.html) with a hidden form that would auto-submit when the page loaded. This form sent a POST request to admin.php?page=elisqlreports&action=save, mimicking the format of legitimate report creation requests. It included parameters like report_name, query, and a hidden submit value.

After opening the file while logged in as an admin, we were able to confirm that the request went through and the new report appeared in the plugin interface. All of these reports were created without further user interaction, verifying that the CSRF was successful.

However, we were not able to chain this attack with others, like the XSS in GlobalPayments WooCommerce, because the teachPress plugin relied on a WordPress nonce (tp_nonce) for most sensitive actions. We attempted to steal this token through a stored XSS, but we couldn't reach the correct settings page due to permission restrictions.

This shows that while CSRF was exploitable in isolation, chaining it required access to or leakage of a valid nonce, which is something that WordPress tries hard to protect. We learned that many plugins only protect high-risk operations and forget to secure lower-visibility features like report generation, which still carry real risks in the hands of a determined attacker.

Thumbnail Carousel Slider

Carousels and sliders are web components that rotate through images and usually have controls for selecting the next image or stopping progression. Carousels have lost some popularity in recent years due to poor user experience and accessibility issues (ACS Creative, 2025). The plugin Thumbnail Carousel Slider (TCS) is one of the available WordPress plugins found on the list of vulnerabilities listed on solidwp.com's vulnerability list for March 19, 2025.

This vulnerability listing gave us valuable information and we investigated using the procedure outlined in the Method section of this paper. TCS was listed as a SQL Injection and given a Severity Score of High. Following the linked CVE to cve.org we gathered more data about the exposure. Cve.org listed the specific parameter we were looking for, but nothing about how the module worked, or which form/page the parameter was attached to.

After some trial and error we were able to get a vulnerable version of the plugin installed and enabled. Then creating several sliders we noticed a few that used the GET parameter 'id'. All of these forms were located in the admin interface and were not public facing despite the CVE stating otherwise "This makes it possible for unauthenticated attackers to append additional SQL queries into already existing queries that can be used to extract sensitive information from the database (CVE, 2025)." While there may have been a public facing aspect to the vulnerability, all attempts we published did not expose a matching GET parameter.

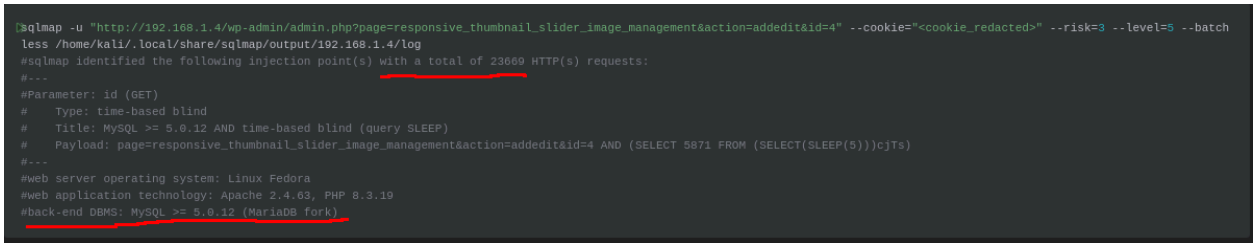
The CVE pointed us to the direct line of code where the vulnerability was visible. You could in fact see the GET parameter being passed directly into the query. In the following figure you can see the plugin get the GET parameter on line 1326.

```
1322        <?php if(isset($_GET['id']) and $_GET['id']>0)
1323            {
1324
1325
1326                $id= $_GET['id'];
1327                $query="SELECT * FROM ".$wpdb->prefix."responsive_thumbnail_slider WHERE id=$id";
1328                $myrow  = $wpdb->get_row($query);
1329
1330                if(is_object($myrow)){
1331
1332                    $title=$myrow->title;
1333                    $image_link=$myrow->custom_link;
1334                    $image_name=$myrow->image_name;
1335
1336                }
1337
1338            ?>
```

The next line, 1327 the id is being put into the query without any validation whatsoever. This is exactly the kind of mistake that a novice developer will make. Without any kind verification on

WordPress' end as to the kind and quality of plugins that the community provides, it's up to independent researchers to find and disclose this kind of vulnerability.

After collecting related information and exploring the modules functionality we continued the exploration. We pointed the SQLMap tool at our server. We needed to provide a cookie since all areas with the parameter ID were restricted to users with the content editors role. This limits the scope of exposure and reduces the risk to persons already trusted unless chained with another vulnerability such as broken access control. SQLMap used nearly twenty four thousand queries on our testing server and was able to locate and identify the vulnerability as shown in the following figure.



Once the vulnerability was confirmed, we turned on MySQL's native logging so we wouldn't have to try and figure out printing the queries to the page. Tailing the logs we were able to see every transaction between the browser requests and the database. Passing the following URL to the browser:

```
http://192.168.1.4/wp-admin/admin.php?page=responsive_thumbnail_slider_image_management&action=addedit&id=3' OR 1=1; UPDATE wp_options SET option_value = 'Vulnerable' WHERE option_name = 'siteurl' LIMIT 1; --
```

We were able to see this output in the database logs: `15 Query SELECT * FROM wp_responsive_thumbnail_slider WHERE id=3' OR 1=1; UPDATE wp_options SET option_value = 'Vulnerable' WHERE option_name = 'siteurl' LIMIT 1; --`

However, despite this query displaying in the logs, we were unable to make any real changes to the database. This was most likely due to WordPress' built in security features not allowing multiple queries with parameterized queries covered in the challenges section of this paper.

WooCommerce

WooCommerce is one of the most widely used plugins for WordPress and serves as the foundation for many of the ecommerce sites built with the platform. Given its importance, we wanted to see if any related plugins had serious vulnerabilities that we could explore. We came across a plugin titled GlobalPayments WooCommerce, which had a known stored XSS vulnerability listed under CVE-2025-22767. According to the listing, the plugin did not properly sanitize or escape user input in certain payment method fields. This kind of vulnerability is dangerous because if JavaScript can be injected into these fields, it could execute in the browser of any admin or user who later views the page.

After finding the vulnerability listing, we downloaded an older, vulnerable version of the plugin and installed it on our WordPress test server. We began by exploring the plugin's various settings and inputs, particularly the GlobalPayments options that are exposed through the WooCommerce payment methods interface. We attempted to inject payloads like <script>alert(1)</script> into these fields in places like payment method names, descriptions, or

metadata inputs. However, one of the first challenges was that most of the GlobalPayments payment types were disabled by default and required additional setup. In many cases, the plugin blocked access to those pages entirely unless a valid license key was provided. This severely limited our ability to reach and test the inputs we were targeting.

We were able to get the script tags to save into some fields, and when we manually visited the WooCommerce settings page after saving, the script appeared in the HTML. But it did not execute. This suggested that either WordPress or the plugin was escaping the tags upon rendering or that the specific DOM location was not sensitive to script injection. So, while the stored XSS was technically present (and we could see the payload in the page source), we were unable to cause script execution in the admin interface.

The main takeaway from this attempt was that while the vulnerability exists and user input is not being properly escaped in some cases, exploitation depends heavily on the rendering context. We learned that for stored XSS to be practically exploitable, it is not enough to inject the script—it must also be rendered in a way that triggers execution. This effort also showed us the barriers posed by paid or restricted plugin features, which sometimes prevent full testing unless a license is purchased or circumvented. Even with partial access, we were able to validate some aspects of the vulnerability, though a full exploit chain (such as using the XSS to steal a CSRF nonce from another plugin) was not possible under these conditions.

EZ SQL Reports

EZ SQL Reports is a plugin that allows WordPress administrators to write and save custom SQL queries from inside the admin interface. This sounded like an ideal target for CSRF testing since it involves arbitrary database queries, which could be dangerous if an attacker could trick an admin into saving one.

The vulnerability listing for this plugin indicated that CSRF was possible when creating or saving reports. We used that information as our starting point. After installing the plugin and exploring its interface, we found the form used to save new reports. We confirmed that this form did not include a nonce or CSRF token. That gave us a potential opening.

We created a standalone HTML page containing a hidden form. This form auto-submitted the malicious query, “SELECT user_login, user_email FROM wp_users,” to the report creation endpoint, simulating a CSRF attack. While logged in as an admin, we clicked the page locally and observed the form submission through browser developer tools. We saw a POST request go out and received a 302 redirect in response, which usually signals success or rejection followed by a login check.

After some trial and error, we confirmed that the report was being created when we visited the CSRF page while authenticated. However, it only worked when the admin actually visited the page. It wasn’t a blind CSRF. This meant we couldn’t chain it with other vulnerabilities like stored XSS to automatically execute the CSRF. We also learned that WordPress’ X-Frame-Options setting blocked the login page from loading in our iframe, which made quiet background redirects more difficult to track.

In the end, we confirmed that EZ SQL Reports is vulnerable to CSRF, but only in a limited way: the admin must be logged in and must visit a malicious link. Still, this is a realistic threat model, especially when paired with phishing or other forms of social engineering.

Contributions

We reproduced several real-word plugin vulnerabilities in a safe environment, including working CSRF and SQL Injection attacks. WE showed how even outdated plugins still pose a risk if they skip nonce checks or rely on admin clicks. Our approach, using CVE listings, archived plugin versions and simple tools, made these tests reproducible for others interested in this research.

We also tested ways to chain attacks, like combining XSS and CSRF and learned that it's harder than one may assume. Even though we couldn't exploit everything fully, we documented what stopped us, including updated plugins, missing tokens, or WordPress headers that block risky behavior. This helped show where defenses succeed and where some plugins still need to improve.

Future Work

There were still a lot of plugins we didn't get to try. Some needed payment, didn't have older versions available, or just broke when we tried to install them. We stuck to what worked, but there's definitely more ground to cover. Future work could mean setting up a larger testbed, maybe with premium plugins or known dependency chains, to see if we can trigger more advanced or chained attacks. A more complete setup might also help test real-world scenarios, like what happens when multiple weak plugins interact.

We also didn't do much with automation or custom tools. Writing a scanner to flag missing nonces or insecure patterns in plugin code would speed things up and help catch low-hanging fruit early. Another direction might be trying to recreate more realistic user behavior, leaving tabs open, flipping between admin pages, to see if session-based or timing-sensitive attacks work better. Lastly, while we focused on local testing, finding vulnerabilities in the wild might have more practical use. But ethics and legality would definitely come into play there, so that would have to be approached with caution and proper disclosure.

Another aspect of focus for future work should be a stronger analysis of WordPress' core defense functionalities and why some modules with vulnerable code were not experiencing the effects of these vulnerabilities while others were. Deep diving into exactly what the core defenses do, how they behave independent of module code, and which vulnerabilities are not covered would be an attractive area for future study.

Conclusion

While we have shown that the problem of web application security, even when limited to WordPress, is large and attacks are ever advancing, it is possible to be reasonably safe in administering a WordPress website. While there is no such thing as a completely safe system these are some steps you can take to help to reduce risk and exposure from common

vulnerabilities that affect WordPress Installations. The following suggestions are a minimum and in no way guarantee a safe instance; however, they are highly recommended.

First to consider is the environment. For shared hosting or publication websites such as Squarespace or Wix many of these considerations have been made for you. Whole papers have been written on this subject so we will only cover a short list and you are encouraged to study the subject yourself before hosting any publicly accessible server. In self hosted web servers you need to be able to make sure that you're using up to date applications to not introduce vulnerabilities at the foundation. If you're using MySQL as your platform database management system, ensure that you run the mysql_secure_installation binary to perform the basics of hardening (Oracle, 2025). Additionally, ensure the principles of least privilege between databases, users, and applications; set connection error limits; rename root; and disable load data local infile (Bellon, 2024).

WordPress has taken a lot of measures to ensure that Core is as secure as possible. Once you branch out into publicly coded themes and plugins provided by the community you are exposed to less strict and rigorous coding standards. WordPress has some built in security measures to save you from the naive developer, but as we've shown these are not always fool proof methods. If you venture out into the extensible landscape of WordPress to enhance the look and feel of your site, guard your site users by vetting each theme and plugin individually. Traveling to sites such as https://solidwp.com/blog/category/wordpress-vulnerability-report/ and https://www.cve.org and search for the specific plugin or theme. Slugs are shorthand machine friendly titles that can better help you search for known vulnerabilities.

Each plugin or theme listed on wordpress.org is also likely to have an associated page, in the form of wordpress.org/category/slug for example: https://wordpress.org/plugins/embed-lottie-player/. These pages often have valuable information on the extensions installation, functionality, and development history. Reading the changelog listed there is a good way to get a sense of the author's knowledge, past vulnerabilities, how active the development is, and what features are more established. It is also recommended to do general search engine queries to see what additional information is available to you before you install and enable.

Above all else, update often. Updates can break site functionality creating a hardship for both end users and site owners. This can lead to tendencies for delay updating until the changes have been thoroughly tested. This crucial vulnerable time is a window for the entire internet to analyze publicly disclosed vulnerabilities and exploit them. By updating often you can minimize the exposure of these timeframes.

By performing the above steps and staying vigilant you can significantly reduce the risk of your WordPress installations. Again, no publicly available system is ever completely secure. However reducing your risk is paramount to having a good user experience for both your patrons and administrators.

References

ACS Creative. (2025, February 12). *Why the Homepage Carousel Trend is Falling Out of Favor in 2025*. ACS Creative. Retrieved May 3, 2025, from https://www.acscreative.com/insights/why-the-homepage-carousel-trend-is-falling-out-of-favor-in-2025/

Bellon, A. (2024, 1 6). *Computer Security Hardening*. sysnet.ucsd.edu. Retrieved May 3, 2025, from https://www.sysnet.ucsd.edu/~abellon/brain/computer/security/hardening

CVE. (2025, 03 15). *CVE-2019-25222*. CVE.ORG. Retrieved 5 3, 2025, from https://www.cve.org/CVERecord?id=CVE-2019-25222

Fedora Project. (2024, January 14). *Installing Wordpress CMS :: Fedora Docs*. Fedora Documentation. Retrieved May 4, 2025, from https://docs.fedoraproject.org/en-US/fedora-server/tutorials/wordpress-installation/

Firefox DevTools User Docs — Firefox Source Docs documentation. (n.d.). Firefox Source Docs. Retrieved April 12, 2025, from https://firefox-source-docs.mozilla.org/devtools-user/

Oracle. (2025). *MySQL :: MySQL 8.4 Reference Manual :: 6.4.2 mysql_secure_installation — Improve MySQL Installation Security*. MySQL :: Developer Zone. Retrieved May 3, 2025, from https://dev.mysql.com/doc/refman/8.4/en/mysql-secure-installation.html

SolidWP. (2025, March 12). *wordpress-vulnerability-report-march-12-2025*. SolidWP. Retrieved May 4, 2025, from https://solidwp.com/blog/wordpress-vulnerability-report-march-12-2025/#h-wp-recall-registration-profile-commerce-more

SolidWP. (2025, April 30). *WordPress Vulnerability Report*. SolidWP. Retrieved May 5, 2025, from https://solidwp.com/blog/category/wordpress-vulnerability-report/

SQLMap. (n.d.). *Introduction*. sqlmap: automatic SQL injection and database takeover tool. Retrieved May 10, 2025, from https://sqlmap.org/

WordPress. (n.d.). *About – WordPress.org*. WordPress.org. Retrieved May 4, 2025, from https://wordpress.org/about/