An IDE Extension for Secure Web Development

MSc Research Project
MSc in Cybersecurity

Dnyanesh Mahajan
Student ID: x21151270

School of Computing
National College of Ireland

Supervisor: Apurva Vangujar

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dnyanesh Mahajan
Student ID:	x21151270
Programme:	MSc in Cybersecurity
Year:	2023
Module:	MSc Research Project
Supervisor:	Apurva Vangujar
Submission Due Date:	14/08/2023
Project Title:	An IDE Extension for Secure Web Development
Word Count:	6075
Page Count:	22

Signature:	
Date:	18th September 2023

An IDE Extension for Secure Web Development

Dnyanesh Mahajan
x21151270

Abstract

The research explores the vulnerabilities and security challenges faced by WordPress applications. WordPress sites are frequently targeted by attackers, taking advantage of vulnerability in the third-party plugins or themes. The research conducts analysis of the WordPress plugins with their common vulnerabilities. The study highlights the importance of secure coding practices, offering a set of guidelines and best practices for developers. This research paper introduces an Integrated Development Environment (IDE) extension to help secure coding practices and mitigate vulnerabilities in stage of WordPress plugin development. The IDE extension is developed for Visual Studio Code IDE, and it is developed using TypeScript and VS code libraries. The application provides real time feedback for developers to follow secure coding practises.

1 Introduction

In an increasingly digital world, technology is changing rapidly and the web is becoming more sophisticated and complex. In such an evolving industry, the need for secure web development has become crucial. Websites and web applications are now essential for businesses, content creators, and individuals and WordPress stands out as one of the most widely used platforms among the content management systems available. However, these applications are a common target for attackers as a result of their popularity. Being exposed to such vulnerabilities and security breaches highlights the need for a secure WordPress ecosystem.

To address this need, the current research paper works on improving WordPress security by exploring the various vulnerabilities found within the WordPress plugin ecosystem. After careful consideration of the Wordpress ecosystem, an IDE extension has been created to enhance security measures in the development process of WordPress plugins. The primary focus of the proposed solution is on mitigating identified security risks and promoting best practices in the development.

WordPress, with its user-friendly interface and plugins library, has become an easy platform for developers and content creators to build feature-rich websites and web applications. However, this has resulted in increased security concerns, ranging from simple code vulnerabilities to more complex attacks like Cross Site Scripting. The consequences of security breaches can have major impacts, such as data leaks compromising user privacy and many more.

The objective of this research is to analyse the existing security issues in the WordPress plugin environment and propose a practical approach to support security during the plugin development process. The focus of this research is on the development of an IDE

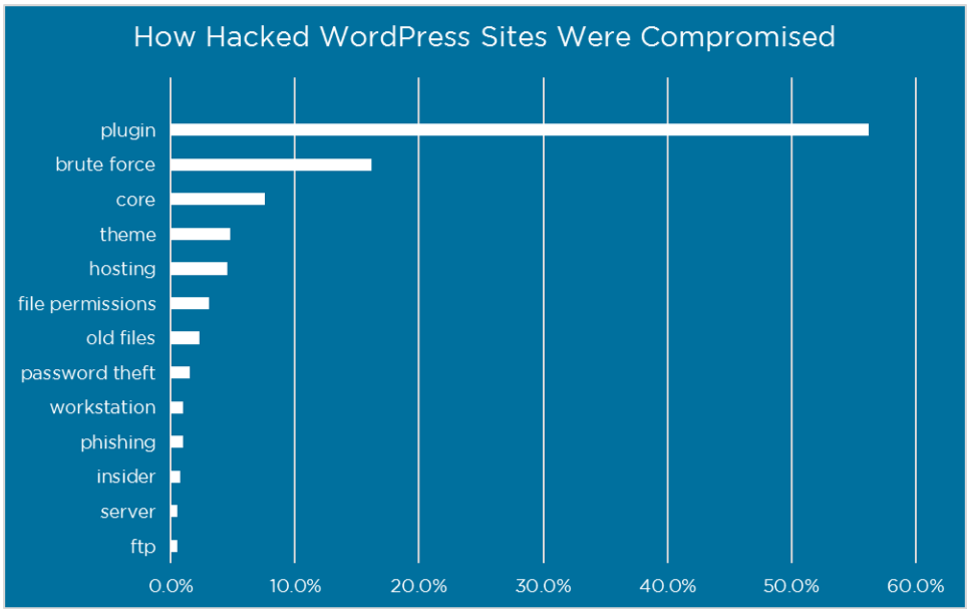


Figure 1: Stats of how WordPress get compromised, Source: *How Attackers Gain Access to WordPress Sites* (n.d.)

extension to be integrated with the widely used IDE Visual Studio Code (VS Code). This will enable developers to write secure WordPress plugins while development of the plugins itself.

This extension will enable developers to receive real-time suggestions according to WordPress security standards and function. It will help them to identify potential vulnerabilities in the early development lifecycle. In addition to this, the extension will work as a static code analysis to provide timely feedback to developers. It will promote a security-first mindset among developers and help establish an environment of secure WordPress plugin development.

The following sections of this research paper will dive deeper into the security challenges faced within the WordPress ecosystem, present an in-depth analysis of the proposed IDE extension, and its impact on improving the overall security of plugins within the ecosystem.

1.1 Research Question

What measures can be taken to support developers in following security standards into the coding stage of WordPress plugin development?

1.2 Objective

The objective of this research is to make the development of WordPress plugins more secure. Developing this IDE extension will help developers to follow security standards defined by WordPress which will save time and cost spent on security testing. This paper will explore the existing security issues in the WordPress plugin environment, their Impact on WordPress site, Importance of secure coding, the research focused on the development of an IDE extension, designed to integrate with the Visual Studio Code (VS Code) IDE. This will contribute towards knowledge and security of WordPress plugins,

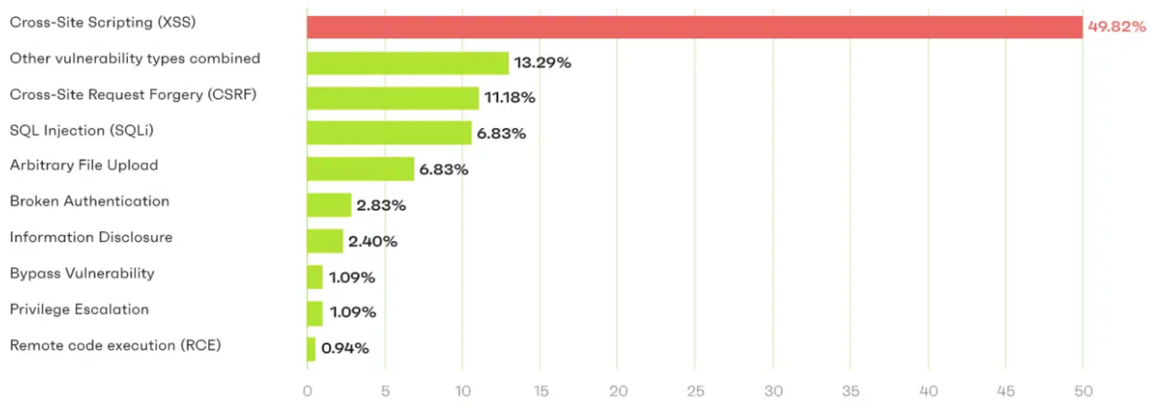


Figure 2: List of vulnerabilities in WordPress plugins, Source: *WordPress Hacking Statistics* (n.d.)

leading to best practices for developers.

2 Related Work

WordPress, WordPress Security, Secure Coding, IDEs, and Extensions are the foundation of this research. Discovered a research gap between them; this research and solution is novel, so to understand research more clearly related work is divided into the following sections

2.1 WordPress And WordPress Security. 2.2 IDE and IDE Extensions. 2.3 Secure Coding. 2.4 IDE and IDE extensions for secure coding. and 2.5 Summary

2.1 WordPress and WordPress Security

This paper (Abhilash Kumar; 2021) provides an overview of WordPress as a content management system (CMS) and discusses its key features. The paper guides new users to make use of WordPress effectively. The paper begins with an introduction to WordPress and its increasing popularity among some other open-source CMS platforms like Drupal and Joomla. The authors underlines the importance of CMS to update the websites without any requirement of previous knowledge of web development tools in this paper. It then continues to explore the main characteristics of WordPress. It is described as an all-in-one content manager and publisher which allows administrators to modify content and publish it in real time. It has the ability to be easily extended by extra features with plugins which makes it attractive for developers. Along with providing the services for free it has compatibility with the MySQL Database for managing the database of the application. The simplicity of managing content through the WP dashboard is remarkable as it enables even the non-technical users to create and manage the content.

The paper Jose-Manuel Martinez-Caro (2018)discusses the vulnerability of WordPress websites, they mention that WordPress is widely used and therefore a popular target for hackers, with about 95.62% of hacked websites in 2021 being WordPress-based. The paper also highlights those vulnerabilities from third-party code, such as plugins and themes, a significant threat to WordPress websites. Regularly updating WordPress, plugins, and themes is also recommended to reduce vulnerabilities and enhance website security.

The paper(Jiahuei Lin; 2023) explores the co-evolution of plugins with WordPress and peer plugins. It addresses security issues related to this co-evolution. It highlights the quick fixes which lead to compatibility problems, and potential security vulnerabilities resulting from it. The importance of resolving such issues earlier to mitigate security risks is highlighted in this study. It further analyses modifications to tools and WordPress APIs in various release versions and possible security concerns arising from it. It mentions alterations to the core features and APIs and impact it has on the overall security status of the WordPress platform and its plugins. In general, the paper's findings indirectly underline the importance of addressing compatibility and API-related security concerns in order to create a safer and more secure WordPress ecosystem.

The paper (Teemu Koskinen; 2012) focuses on analysing the vulnerability of plugins in the official WordPress plugin directory while evaluating the correlation between user ratings and security. It demonstrates the popularity of the online content management platform with over 73 million websites developed on the platform with 300+ million monthly page views. The large number of plugins available in the official WordPress plugin directory is cited as one of the reasons for its huge popularity. The researchers applied static analysis to assess the security of the plugins with help of an open-source tool RIPS which supports detecting various vulnerability types. The paper discusses the types of vulnerabilities commonly found in PHP, which include remote code execution, cross-site scripting (XSS), SQL injection (SQLI), PHP configuration, and file system attacks.

(Oslien Mesa; 2018) paper focuses on understanding vulnerabilities in plugin-based web systems. The paper follows an exploratory study that aims to analyse the main types of vulnerabilities caused by plugins in web-based systems, their impact, and the common security-related topics discussed among the WordPress developers community.

It provides some background on plugin-based web systems and their importance. It states the rapid growth of the Internet and web technologies as a cause leading to the adoption of plugin-based development.

The researchers performed an in-depth investigation to analyse vulnerabilities related to plugins in web systems, with a specific focus on WordPress. The researchers examined the presence, seriousness, intricacy, and duration of vulnerabilities resulting from the wide range of plugins offered for WordPress. In addition, the researchers examined the most frequently discussed security-related subjects among developers on Q&A websites. Key findings of the study revealed Cross-site Scripting (XSS), SQL Injection, and Cross-site Request Forgery (CSRF) to be the most common vulnerabilities in WordPress plugins.

Considering the various types of alterations needed and the number of files to be modified, the researchers analysed the complexity involved in resolving vulnerabilities. This analysis discovered addressing vulnerabilities to be a challenging and demanding effort. Furthermore, the study has shown that vulnerabilities resulting from plugins can go unnoticed for a significant period before they are recognised and resolved. The study mentioned the average duration for vulnerabilities to be addressed in code to be 653 days.

It shows the importance of the knowledge of vulnerabilities in plugins and significance of tools that allow to build secure plugins to new and experienced developers alike.

The paper (Daniel T. Murphy; 2021) examines the ability of the security scanner plugins to find vulnerabilities in third-party plugins for WordPress. Although there is a wide variety of choice in security tools, the majority of them are complex static analysis tools that may not be very user-friendly for beginners. Hence, the primary objective of this study is to investigate tools that are user-friendly and easily accessible, with a particular focus on the Word fence security plugin.

The methodology in the study involved analysing 11 security scanner plugins on a WordPress test site which were linked to 51 known vulnerable plugins. A reputable Exploit Database was used to extensively search WordPress which has been mentioned to hold a significant market share of 60.3%. This ensures the reliability of the study and its potential for replication. This study found that plugins for WordPress security scanning have the capability to identify weaknesses in plugins developed by third-party sources.

The findings exposed the security concerns present in the WordPress plugin ecosystem and the challenges that new users might face. It established the importance of having security tools that are user-friendly and easy to understand which also has a well-documented methodology and results, and can be considered a valuable resource for enhancing the security of WordPress websites.

The paper (Ruohonen; 2019) explores the connection between the popularity of WordPress plugins and the number of vulnerabilities they have. The paper argues against the common belief of popular and current plugins are less likely to be exploited. Instead, it suggests that their popularity could make them more appealing to attackers who are looking to discover new vulnerabilities.

The main question of the study is if more users results in finding more vulnerabilities. The paper uses data from the WordPress Vulnerability Database (WPVDB) which includes information about 1,657 plugins and their 2,629 vulnerabilities. Additionally , it also includes information from the National Vulnerability Database (NVD) to provide a more complete understanding of the topic. The results show that there is a connection between plugins with more users and a higher number of vulnerabilities. This suggests that plugins that are not popular do not provide much motivation for finding vulnerabilities.

The study notes that there may be limitations in the data, particularly from WPVDB. However, it argues that using vulnerability counts as a metric is a suitable approach for examining incentives, rather than assessing software security. This study suggests that more empirical research should be conducted at the code level. A thorough examination of the code and an understanding the difficulties involved in maintaining software in order to improve its security is required. The evidence-based studies are significant in understanding software vulnerabilities in this way. This emphasises the need for further investigation in this field.

The paper (Hannes Trunde; 2015) studies the security weaknesses of WordPress websites, especially problems related to SQL injection attacks. The researchers looked at 199 known SQL injection issues and found that automatic scanning tools are unable to detect many of these problems. They believe manual checks are required to fully test and secure a site rather than completely relying on automated tools. The paper suggests that to better protect WordPress sites, a mix of both methods and special attention to user inputs is necessary along with following safe coding practices.

2.2 IDE and IDE Extensions

The paper (Aniqua Z. Baset; 2017) research shows An Integrated Development Environment (IDE) is an application that offers a wide range of tools and features to assist in software development including a source code editor, compiler or interpreter, and A debugger which helps identify and fix errors or bugs in the code. IDE extensions also referred as plugins or add-ons, can be installed in an IDE to improve its functionality. extensions offer additional functionalities like code analysis, code completion suggestion, version control integration, and support for specific programming languages or frameworks. IDE plugins are tools that assist developers in identifying security vulnerabilities during the process of coding itself. Developers are permitted to examine security flaws in their code directly from their IDE. There are various plugins that can be used for security checks. These plugins are compatible with various IDEs including Eclipse, IntelliJ IDEA, Visual Studio, and Netbeans. The research shows several widely used IDE plugins such as Checkmarx CxSAST, Fortify, Veracode, Codepro AnalytiX, SensioLabsInsight, and SSVChecker. The above mentioned plugins provide developers with real-time security analysis and feedback to enhance the security of their code while detecting vulnerabilities associated with input validation.

2.3 Secure Coding

Paper (Arafa Anis; 2018) and (Musa Bala Shuaibu; 2017)research shows Secure coding is the practice of developing computer software in a way that guards against the accidental introduction of security vulnerabilities. By following secure coding practices, the security of the applications can be significantly enhanced by developers . This study states that the security of applications can be achieved by effectively mitigating vulnerabilities such as input validation and output sanitization with use of secure coding practices. Additionally, It can effectively prevent the occurrence of common vulnerabilities like SQL injection and cross-site scripting (XSS) attacks. By following the principle of least privilege and implementing defence in depth, developers can decrease the potential attack surface of their applications. By utilising parameterized queries and encrypting sensitive data, the integrity of the data can be effectively maintained and any unauthorised access or tampering can be detered.

Developers can effectively deal with security concerns and reduce vulnerabilities through integrating secure coding practises into the software development life cycle (SDLC). By adopting this approach, a more secure application can be developed right from the start of the development, instead of treating security as an afterthought.

2.4 IDE and IDE extensions for secure coding

The papers Aniqua Z. Baset (2017) Jingyue Li (2019),Kire Jakimoski (2022) shows IDEs (Integrated Development Environments) and IDE extensions play a crucial role in promoting secure coding practices. They provide developers with tools and features that help detect and prevent security vulnerabilities in their code. Here are some ways in which IDEs and IDE extensions assist in secure coding: Security Analysis: IDEs can integrate static analysis tools that scan the code for security flaws and vulnerabilities. These tools analyze the codebase and provide real-time feedback on potential security issues, such as input validation vulnerabilities, command injection, cross-site scripting, and SQL injection 1b. This in-situ security analysis helps developers identify and address security flaws early in the development process. Code Review: IDEs often include features for code review, which allow developers to collaborate and review each other's code for security vulnerabilities. Code review helps identify potential security weaknesses and provides an opportunity for developers to share best practices and suggest improvements. Security Guidelines: IDEs can provide built-in support for secure coding guidelines and best practices. They can highlight potential security issues and suggest secure alternatives or code patterns that adhere to secure coding practices. Secure Code Templates: IDEs can offer pre-defined code templates that follow secure coding practices. These templates can help developers write secure code by providing them with a starting point that incorporates security measures. Security Testing: IDE extensions can integrate with security testing tools, such as vulnerability scanners or penetration testing frameworks, to automate security testing processes. These extensions can help developers identify and fix security vulnerabilities by running security tests directly from the IDE. Education and Training: IDEs can provide educational resources, tutorials, and documentation on secure coding practices. They can offer contextual help and tooltips that explain security concepts and provide guidance on secure coding techniques.

2.5 Summary

In conclusion, Following secure coding can help to prevent vulnerability. this research found a gap between WordPress plugin secure development and tools for secure coding practices. As this research provides a solution that will help developers to follow secure coding.

Table 1: Literature review summary table

Index	**Research Paper**	**Findings**
1	WordPress: A Multi-Functional Content Management System (Abhilash Kumar; 2021)	WordPress is a widely used content management system. Most popular CMS than others, Wide range of customizable plugins and themes.
2	Vulnerability Assessment with Network-Based Scanner Method for Improving Website Security (Laksmiati; 2023)	It highlights the need for regular vulnerability scanning of WordPress sites to reduce exploit risks.
3	Optimization of Secure Coding Practices in SDLC as Part of Cybersecurity Framework (Kire Jakimoski; 2022)	Secure coding techniques in the standard SDLC will help avoid costs and delays caused by improperly identifying security issues. It highlights the rising number of software flaws and the need for secure coding throughout the software development life cycle.
4	Systematic Mapping Study on Security Approaches in Secure Software Engineering (Rafiq Ahmad Khan; 2021)	Secure software engineering (SSE) is still in its early stages.

5	Plugins to Detect Vulnerable Plugins: An Empirical Assessment of the Security Scanner Plugins for WordPress (Daniel T. Murphy; 2021)	Plugins are vulnerable, rather than the core of WordPress. Many WordPress users don't secure their websites. While numerous security scanner plugins are available for WordPress, their efficiency in identifying plugin vulnerabilities is inconsistent.
6	Is Secure Coding Education in the Industry Needed? An Investigation Through a Large Scale Survey (Tiago Espinha Gasiba; 2021)	Many software developers are not aware of secure coding guidelines. Increasing awareness of security is important to improve compliance with secure coding guidelines. The survey had 194 participants from different industries. In the industry, secure coding standards compliance is not audited. Highlight the need secure coding practices.
7	Evaluation of Open-Source IDE Plugins for Detecting Security Vulnerabilities (Jingyue Li; 2019)	IDE plugins have limitations and a difference between claimed and confirmed coverage of vulnerabilities. Many vulnerabilities like injection and broken access control are covered by most plugins, while others are ignored.
8	A Demand-Side Viewpoint to Software Vulnerabilities in WordPress Plugins (Ruohonen; 2019)	WordPress plugins with large installation bases are more likely to have multiple vulnerabilities. Using multiple and wide range on plugins can cause more vulnerability. Highlights a need of code-level validation is needed to better understand vulnerabilities and fix issues.
9	Detectors for Intent ICC Security Vulnerability with Android IDE (Xianyong Meng; 2018)	Developers often overlook security vulnerabilities due to time constraints.
10	Securing Web Applications with Secure Coding Practices and Integrity Verification (Arafa Anis; 2018)	Security in web applications is often ignored during development, leading to vulnerabilities like cross-site scripting, injection attacks, and code tampering on the client side. To protect against security attacks, it's important to add security features and practise secure coding.
11	A Comparative Study of Web Content Management Systems (Jose-Manuel Martinez-Caro; 2018)	The study compared the web content management systems (WCMS) Joomla! WordPress, and Drupal. WordPress is the most efficient WCMS in terms of performance.

12	Understanding vulnerabilities in plugin-based web systems (Oslien Mesa; 2018)	The most common types of vulnerabilities caused by WordPress plugins are Cross-site Scripting, SQL Injection, and Cross-site Request Forgery. The average time it takes to fix a vulnerability caused by a plugin is 653 days. Plugin developers lack knowledge of secure programming, which can result in common vulnerabilities.
13	Analysis and development of an online knowledge management support system for a Community of Practice (Moeketsi Mafereka; 2017)	The study compares Drupal, Joomla, and WordPress as Content Management Systems (CMSes) for developing and maintaining online services to support a Knowledge Management System (KMS) WordPress scored highest in terms of creating discussion forums, uploading documents, and creating and editing user profiles.
14	Web application development model with security concern in the entire life cycle (Musa Bala Shuaibu; 2017)	Many existing web application development models do not adequately address security concerns throughout the entire development life cycle. Inculcating security considerations at each stage of the web application development life cycle can improve the security of web applications. Existing web applications often lack security considerations throughout the development life cycle, leading to vulnerabilities and potential attacks. The research paper provides a solution to the security challenges faced by web applications by emphasizing security considerations throughout the development life cycle.
15	IDE Plugins for Detecting Input-Validation Vulnerabilities (Aniqua Z. Baset; 2017)	There is a low adoption rate of security plugins that check for vulnerabilities. Documentation on the supported vulnerability checks is not publicly available for several plugins, making it difficult for developers to compare and choose which plugins to use.
16	Detection of Wordpress Content Injection Vulnerability (Maruf Hassan Md; 2017)	WordPress content injection vulnerability is a common issue, particularly in versions 4.7.0 and 4.7.1. The vulnerability allows unauthorized users to modify the content of posts or pages. Among the examined WordPress web applications, approximately 34% were found to still contain the content injection vulnerability.

17	Quality of WordPress Plug-Ins: An Overview of Security and User Ratings (Teemu Koskinen; 2012)	There is a weak non-linear correlation between user ratings and the number of vulnerabilities in WordPress plugins. Over half of the plugins analysed passed the static analysis with no vulnerabilities detected. The quality of security among individual plugins is inconsistent. Manual review or static analysis is necessary to ensure the security of plugins before using them on a WordPress site.
18	Impact of secure programming on web application vulnerabilities (Blerim Rexha; 2015)	The majority of web developers lack experience and knowledge in web security, leading to vulnerabilities in web applications. There is a correlation between the factors identified in penetration testing and the data gathered from the survey, highlighting the importance of secure programming techniques.
19	WordPress Security White Paper (*WordPress Security White Paper*; n.d.)	Highlights WordPress core is secure. WordPress is a widely used content management system, powering over 23% of the top 10 million websites. WordPress follows the OWASP Top 10 list of common security vulnerabilities and has measures in place to mitigate these risks.

3 Methodology

The aim of the research is to help the developer to follow secure coding standards at the stage of development to Implement a solution of IDE extension, which is designed to help developers with real-time security best practices. Analysed common vulnerabilities in the plugin and WordPress secure coding standards to mitigate those vulnerabilities.

- Identification of common vulnerabilities and their impact.
- WordPress security coding standards.

3.1 Common Vulnerabilities

Research Papers Daniel T. Murphy (2021),Oslien Mesa (2018),Paulo Nunes (2017),Maruf Hassan Md (2017),Laksmiati (2023),Marie Vasek (2016),Ruohonen (2019),Teemu Koskinen (2012)identified some common vulnerabilities in WordPress plugins. These vulnerabilities can cause security risks to WordPress-powered websites. Here are the common vulnerabilities:

Cross-Site Scripting (XSS): XSS vulnerabilities are prevalent in WordPress plugins. They allow attackers to inject malicious scripts into web pages viewed by users, leading to unauthorized actions or data theft. XSS was found to be the most common vulnerability type, accounting for 43.72% of the vulnerabilities.

SQL Injection (SQLi): SQLi vulnerabilities allow attackers to manipulate database queries, potentially gaining unauthorized access to sensitive data or modifying the database. SQLi accounted for 20% of the vulnerabilities in the analyzed plugins.

Cross-Site Request Forgery (CSRF): CSRF vulnerabilities enable attackers to trick authenticated users into performing unintended actions on a website. CSRF vulnerabilities were found in the analyzed plugins.

WP_DEBUG True: Setting WP_DEBUG to true in WordPress is essentially turning on the debugging mode. While this is incredibly useful during the development phase, it can introduce several security concerns like exposing sensitive information and it can be a potential attack vector.

wp_ (Default table prefix): The default table prefix "wp_" in WordPress installations can cause a security concern. With the default prefix, database table names are predictable, which makes it easier for attackers to design SQL injection attacks that target your database. They are able to run scripts intended to exploit or extract data from particular tables by knowing the standard table names.

These vulnerabilities can have severe consequences, including data breaches, unauthorized access, and website defacement. It is important to note that the prevalence of these vulnerabilities may vary across different plugins and versions.

Regarding statistics, one study analysed 322 WordPress plugins and discovered 860 vulnerabilities, with XSS being the most common vulnerability type. Another study found that over 73% of WordPress installations in the Alexa Top 1 Million had vulnerabilities that could potentially be detected using automated tools.

To address these vulnerabilities, it is crucial to regularly update plugins, use reputable plugins from trusted sources, and follow security best practices. The WordPress Security Team works to identify and resolve security issues in the core software and provides recommendations for plugin and theme authors. However, it is important to note that no security scanner plugin analysed in one study was capable of sufficiently detecting and flagging plugin vulnerabilities.

3.2 WordPress Coding Standards

WordPress has defined specific security coding standards. The purpose of these guidelines is to make sure that developers follow the most effective methods that help avoid common vulnerabilities, such as Cross-Site Scripting (XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF). By following to these standards, developers can protect their applications but also improve the overall security environment of WordPress-powered websites. The standards act as a guide for developers, showing them how to write, sanitise, and validate code in a way that strengthens their applications against security threats.

1. **Cross-Site Scripting (XSS):**

   When a application allows data to be entered e.g. Comments, attackers can input malicious code. If the site then displays this data without checking, browsers can execute this code, leading to XSS attacks. The function esc_html() converts potentially harmful characters into their safe, displayable counterparts. When an application allows data to be entered (like in a comment), attackers can input malicious code. If the site then displays this data without checking, browsers can execute this code, leading to XSS attacks. The function `esc_html()` converts potentially harmful characters into their safe, displayable counterparts. Example `<` becomes `&lt;`. This ensures that the data is displayed as text and not executed as code.

2. **SQL Injection (SQLi):**

   Attackers can try to "inject" SQL code into queries. This can reveal, alter, or delete data from database. Use of prepare() process the data ($id in this case) as a string rather than executable SQL. This means attackers can't insert malicious SQL commands.

3. **Cross-Site Request Forgery (CSRF):**

   Nonces are unique tokens generated for specific actions and are valid for a short time. They ensure that the action being taken is genuine and not something an attacker tricked a user into. Example: use function of wp_create_nonce() to create nonce and use wp_verify_nonce() function to verify that nonce.

4. **WP_DEBUG True:**

   Debug mode is useful for the development stage, showing all errors. However, in live sites, these error messages can reveal system details or sensitive information. By setting WP_DEBUG to false, error messages aren't publicly displayed. This keeps system details and potential vulnerabilities hidden from prying eyes.

5. **wp_ (Default table prefix):**

   If attackers know default table names in your database, they can craft specific attacks. Altering the default prefix makes it harder for attackers to guess your table names. This simple change can deter many automated attacks targeting default configurations.

4 Implementation

To implement VS code IDE extension, VS code provides Extension API, which allows to add functionality with Extension. Implement extension require VS Code itself , Extensions are written in TypeScript or JavaScript, and it requires NPM (Node Packaging Module) which helps to install required libraries and Yeoman. This extension is written in TypeScript.

Yeomen give a primary extension file folder structure to maintain standard methods. It will install all required NPM models or libraries. For example: vs code library. The folder structure of the extension has a src folder and it contains extensions.ts, which is the main file of the extension.

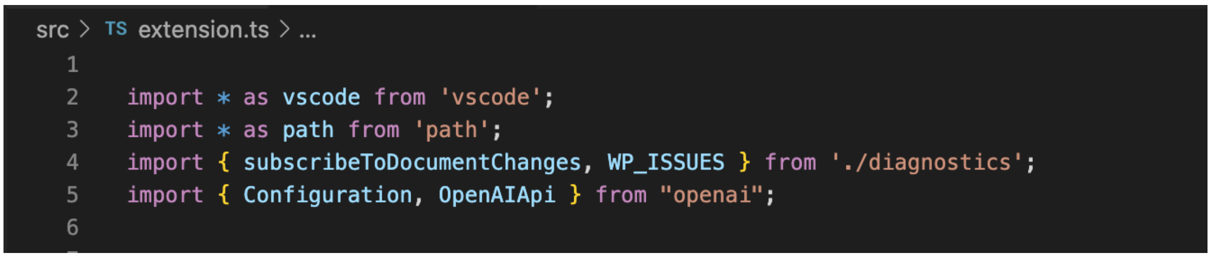
4.1 Code explanation

Extension.ts:

1. **Imports:**

   vscode: This module provides access to the Visual Studio Code extensibility API. It allows the extension to interact with the VS Code editor, register commands, create web views, manage diagnostics, and perform many other tasks.

   `subscribeToDocumentChanges, WP_ISSUES`: These are imported from the 'diagnostics.ts' file. The subscribeToDocumentChanges function is a helper that sets up event listeners to refresh diagnostics when a document changes. The constant 'WP_ISSUES' is used in the context of diagnostic management.



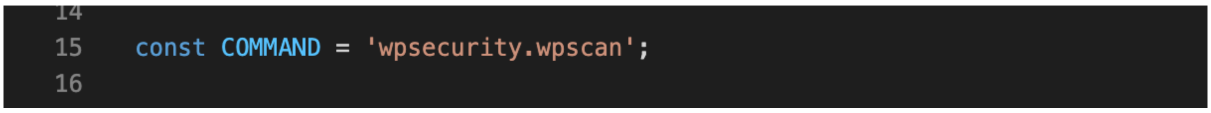
```
src > TS extension.ts > ...

import * as vscode from 'vscode';
import * as path from 'path';
import { subscribeToDocumentChanges, WP_ISSUES } from './diagnostics';
import { Configuration, OpenAIApi } from "openai";

```

Figure 3: Imported libraries

   Configuration, OpenAI Api: These are related to the integration with the OpenAI API. The 'Configuration' class is used to set up API configuration, and `OpenAIApi` is used to interact with the OpenAI service.

2. **OpenAI Configuration:** The OpenAI API is initialized with API key.

3. **Command Definition:** The constant `COMMAND` serves as an identifier for the command that, when executed, will trigger the WordPress security scan functionality of the extension.



```
const COMMAND = 'wpsecurity.wpscan';
```

Figure 4: Extension command

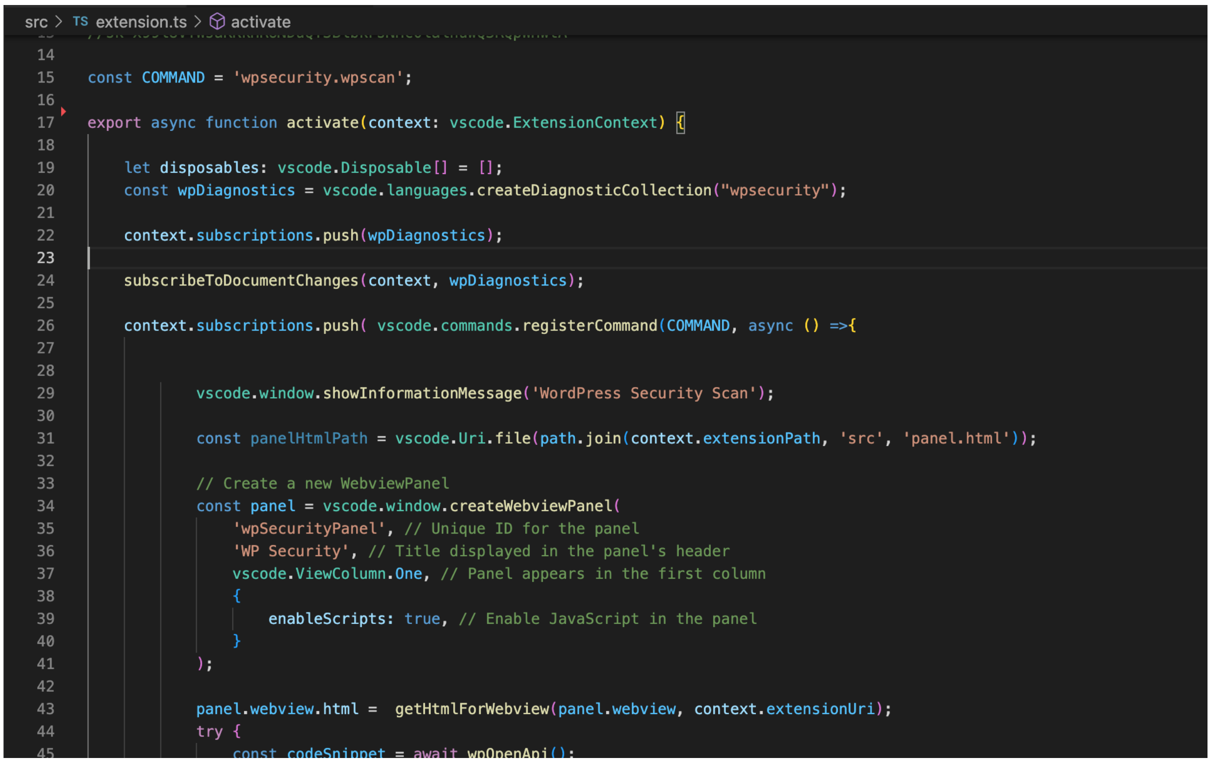
4. **Activation Logic:** The `activate` function is the main entry point for the extension. This function is called once when the extension is activated.

- A new diagnostic collection named `wpsecurity` is created. This collection will store all the diagnostic messages (warnings/errors) related to WordPress security vulnerabilities.

The extension subscribes to various events:

1. Changes in the active text editor.

2. Changes in any text document.



```
const COMMAND = 'wpsecurity.wpscan';

export async function activate(context: vscode.ExtensionContext) {

    let disposables: vscode.Disposable[] = [];
    const wpDiagnostics = vscode.languages.createDiagnosticCollection("wpsecurity");

    context.subscriptions.push(wpDiagnostics);

    subscribeToDocumentChanges(context, wpDiagnostics);

    context.subscriptions.push( vscode.commands.registerCommand(COMMAND, async () =>{


            vscode.window.showInformationMessage('WordPress Security Scan');

            const panelHtmlPath = vscode.Uri.file(path.join(context.extensionPath, 'src', 'panel.html'));

            // Create a new WebviewPanel
            const panel = vscode.window.createWebviewPanel(
                'wpSecurityPanel', // Unique ID for the panel
                'WP Security', // Title displayed in the panel's header
                vscode.ViewColumn.One, // Panel appears in the first column
                {
                    enableScripts: true, // Enable JavaScript in the panel
                }
            );

            panel.webview.html =  getHtmlForWebview(panel.webview, context.extensionUri);
            try {
                const codeSnippet = await wpOpenApi();
```

Figure 5: Logic after extension activation

3. Closing of any text document.

A command is registered (using the identifier from the `COMMAND` constant) that, when triggered, will initiate the WordPress security scan. When the command is executed, an information message is shown, and a webview panel is created. The specifics of what is displayed in the webview aren't entirely visible in the provided snippet, but based on the function name, it's likely to be a security report or scan results.

diagnostics.ts

1. **Imports vs code:** Provides access to the Visual Studio Code extensibility API, allowing the extension to interact with the editor, manage diagnostics, and more.

2. **2.Constants:** `WP_SECURITY`: A unique identifier for the WordPress-specific diagnostic messages.

   `WP_ISSUES`: An array that will store detected WordPress-related security issues in the scanned documents. Each item in this array represents a potential vulnerability in the code.

3. **Diagnostic Refresh Logic:**

   The `refreshDiagnostics` function is the primary function responsible for analysing the provided document for potential security vulnerabilities. This function is expected to be called whenever the document content changes.

   Unused Nonce Forms: This section of the function identifies forms in the WordPress PHP code that lack a nonce field. Nonces are cryptographic tokens used in

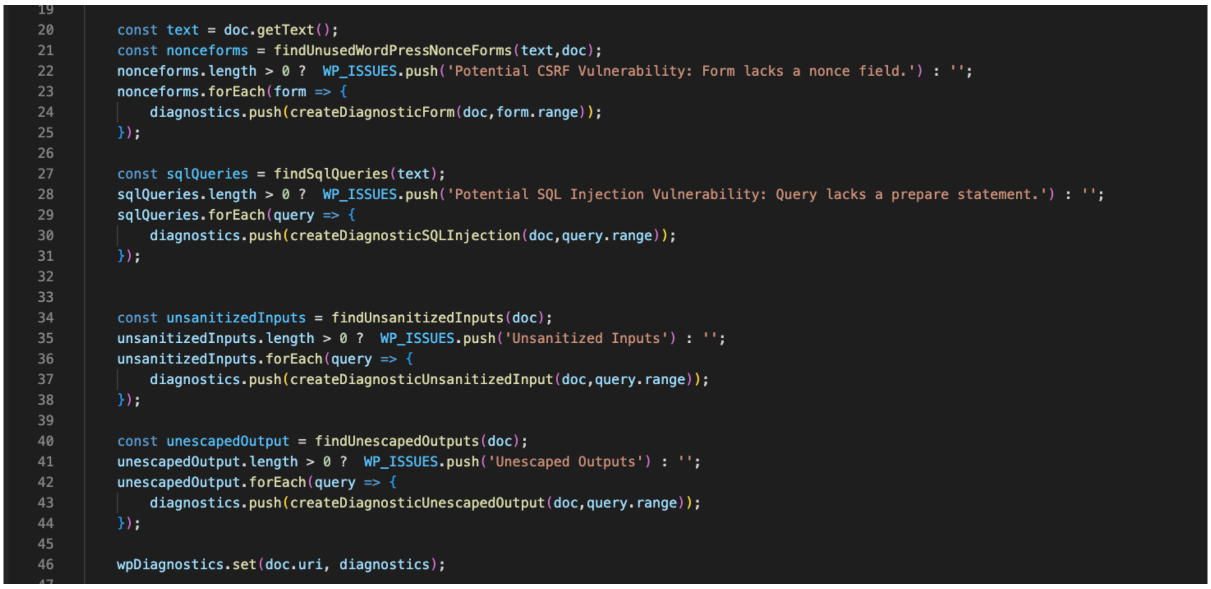
WordPress to verify the origin and intent of requests. If a nonce is missing, it can lead to vulnerabilities like Cross-Site Request Forgery (CSRF).

SQL Queries Without Prepare Statement: This section identifies SQL queries that lack a ‘prepare‘ statement, potentially leading to SQL injection vulnerabilities. The ‘prepare‘ function in WordPress is used to safely substitute placeholders in SQL queries with user-provided data.

After detecting potential vulnerabilities, they are added to the `wpDiagnostics` diagnostic collection.

4. **Diagnostic Detection Functions:**

   `refreshDiagnostics` function calls functions `findUnusedWordPressNonceForms` to find forms which are not using nonce field, `findSqlQueries` check database queries are using prepared statement or not, `findUnsanitizedInputs` to check user input is sanitized or not, all the functions perform the actual code scanning to identify potential issues. These functions will use patterns, possibly regular expressions, to search for the described vulnerabilities in the document content.



```
const text = doc.getText();
const nonceforms = findUnusedWordPressNonceForms(text,doc);
nonceforms.length > 0 ?  WP_ISSUES.push('Potential CSRF Vulnerability: Form lacks a nonce field.') : '';
nonceforms.forEach(form => {
    diagnostics.push(createDiagnosticForm(doc,form.range));
});

const sqlQueries = findSqlQueries(text);
sqlQueries.length > 0 ?  WP_ISSUES.push('Potential SQL Injection Vulnerability: Query lacks a prepare statement.') : '';
sqlQueries.forEach(query => {
    diagnostics.push(createDiagnosticSQLInjection(doc,query.range));
});


const unsanitizedInputs = findUnsanitizedInputs(doc);
unsanitizedInputs.length > 0 ?  WP_ISSUES.push('Unsanitized Inputs') : '';
unsanitizedInputs.forEach(query => {
    diagnostics.push(createDiagnosticUnsanitizedInput(doc,query.range));
});

const unescapedOutput = findUnescapedOutputs(doc);
unescapedOutput.length > 0 ?  WP_ISSUES.push('Unescaped Outputs') : '';
unescapedOutput.forEach(query => {
    diagnostics.push(createDiagnosticUnescapedOutput(doc,query.range));
});

wpDiagnostics.set(doc.uri, diagnostics);
```

Figure 6: Functions list

Features:

- In detail analysis report with a solution using OpenAI.
- Immediate Warnings As soon as the developer writes code, it’ll highlight if there’s a potential issue.
- Easy to Understand: It clearly points out the problems and explains them, making it easier for coders to fix them.
- Checking Unfiltered Inputs: The extension checks if the code might be missing steps to sanitize user inputs. If it finds issues, it’ll highlight them and suggest ways to fix them using WordPress methods.

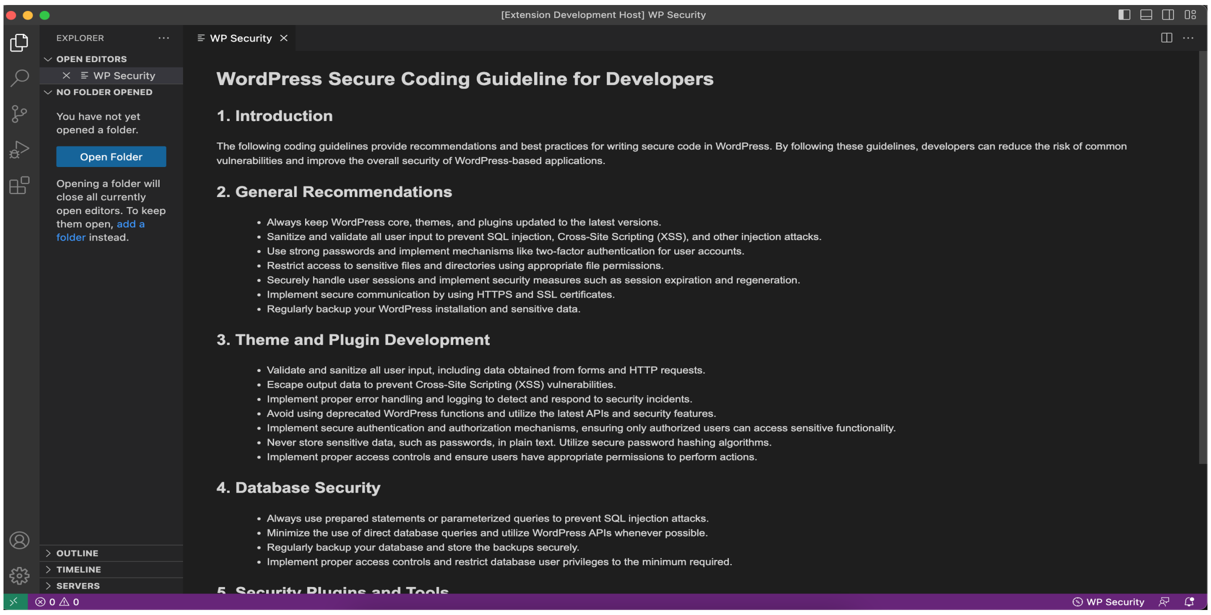
- Making Sure Forms are Safe: It checks if forms have security checks called nonce fields, which make sure the requests are legit. If a form misses these checks, it highlights the whole form.
- Looking for Unprotected Outputs: The extension checks where data might be shown without necessary safety steps. It can lead to Cross-Site Scripting (XSS). It highlights the line of code.
- Checking SQL Queries: It looks at SQL queries to make sure they're using prepared statements, which prevents SQL attacks.

Instant Feedback: Every time the code changes, the tool rechecks it, making sure developers get feedback right away. This helps catch and fix problems early on. Working with OpenAI: The extension integrated with OpenAI, to give better feedback on security issues. It does not share code with OpenAI.

How It Looks: Any issues found will be highlighted in the code editor. They come with clear descriptions. Developers can hover over the issues to get more details and advice on fixing them.

5 Evaluation

The Evaluation of this extension is to find vulnerabilities in the PHP source code file. There are many cases according to vulnerability. The extension is targeting the 4 most common vulnerabilities. In all cases, While writing code or opening an existing PHP file it will scan the code and highlight the line of code, with a hover on that line of code, It popup a small window with feedback information.



You have not yet opened a folder.

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

WordPress Secure Coding Guideline for Developers

1. Introduction

The following coding guidelines provide recommendations and best practices for writing secure code in WordPress. By following these guidelines, developers can reduce the risk of common vulnerabilities and improve the overall security of WordPress-based applications.

2. General Recommendations

- Always keep WordPress core, themes, and plugins updated to the latest versions.
- Sanitize and validate all user input to prevent SQL injection, Cross-Site Scripting (XSS), and other injection attacks.
- Use strong passwords and implement mechanisms like two-factor authentication for user accounts.
- Restrict access to sensitive files and directories using appropriate file permissions.
- Securely handle user sessions and implement security measures such as session expiration and regeneration.
- Implement secure communication by using HTTPS and SSL certificates.
- Regularly backup your WordPress installation and sensitive data.

3. Theme and Plugin Development

- Validate and sanitize all user input, including data obtained from forms and HTTP requests.
- Escape output data to prevent Cross-Site Scripting (XSS) vulnerabilities.
- Implement proper error handling and logging to detect and respond to security incidents.
- Avoid using deprecated WordPress functions and utilize the latest APIs and security features.
- Implement secure authentication and authorization mechanisms, ensuring only authorized users can access sensitive functionality.
- Never store sensitive data, such as passwords, in plain text. Utilize secure password hashing algorithms.
- Implement proper access controls and ensure users have appropriate permissions to perform actions.

4. Database Security

- Always use prepared statements or parameterized queries to prevent SQL injection attacks.
- Minimize the use of direct database queries and utilize WordPress APIs whenever possible.
- Regularly backup your database and store the backups securely.
- Implement proper access controls and restrict database user privileges to the minimum required.

5. Security Plugins and Tools

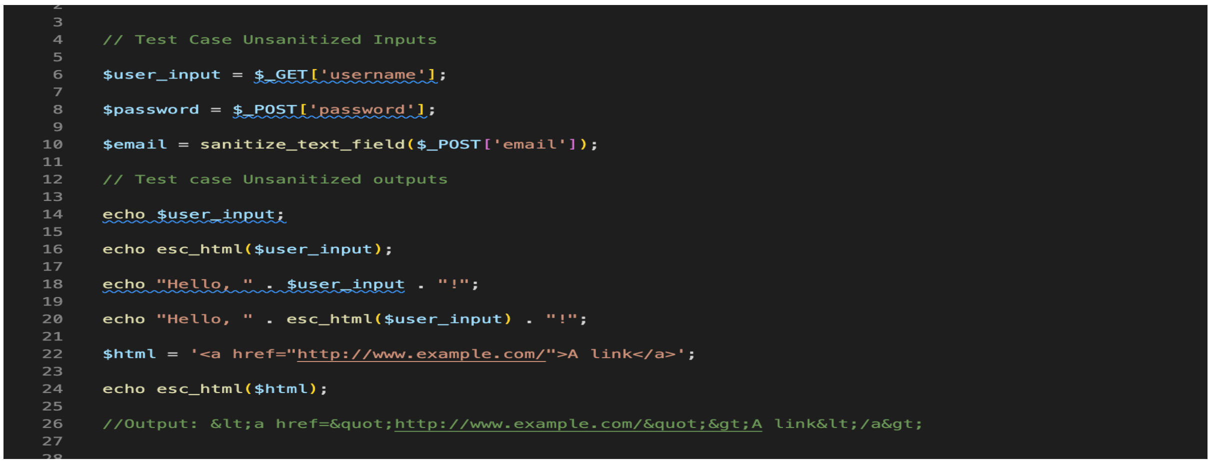
⊗ 0 ⚠ 0

Figure 7: Opening Window of Extension

To give an idea about WordPress Secure Coding, added an opening window to the extension, which can be helpful for developers.

5.1 Case Study 1: Cross Site Scripting XSS

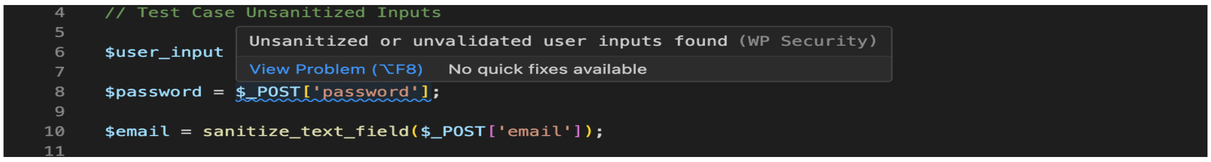
Cross-site scripting (XSS) is the most common vulnerability, the main reason to cause XSS is to trust user input and not sanitise it and not escape output securely. WordPress security provides a range of functions to avoid this.



```
 3
 4  // Test Case Unsanitized Inputs
 5
 6  $user_input = $_GET['username'];
 7
 8  $password = $_POST['password'];
 9
10  $email = sanitize_text_field($_POST['email']);
11
12  // Test case Unsanitized outputs
13
14  echo $user_input;
15
16  echo esc_html($user_input);
17
18  echo "Hello, " . $user_input . "!";
19
20  echo "Hello, " . esc_html($user_input) . "!";
21
22  $html = '<a href="http://www.example.com/">A link</a>';
23
24  echo esc_html($html);
25
26  //Output: &lt;a href=&quot;http://www.example.com/&quot;&gt;A link&lt;/a&gt;
27
```

Figure 8: Unsecured Code of Cross-Site Scripting

Here line numbers 6,8, 14 and 18 are highlighted by extension, input from the form is passed to a variable as it is. It means the code is trusting user inputs but the user can give malicious input. To Sanitize user input WordPress has provided many functions `sanitize_text_field()` is one of them. On line number 8 and 14 output is directly printed, content can be anything, if the user passes the script to input and that is printed on the browser can cause cross-site scripting.

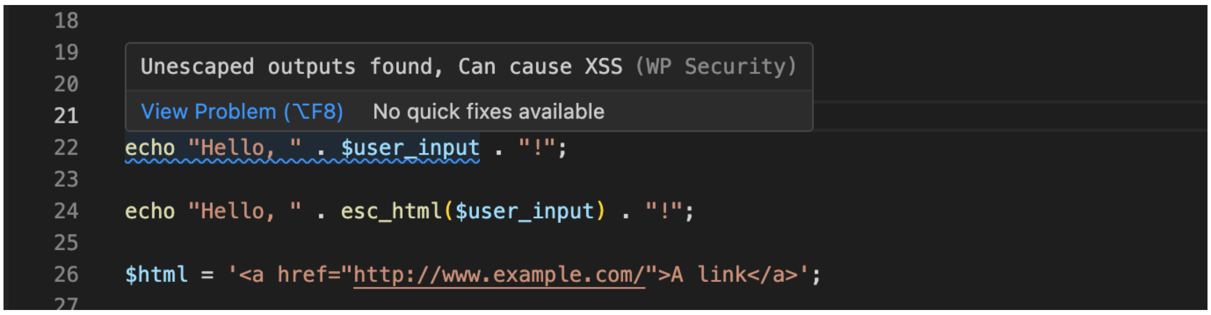


```
 4  // Test Case Unsanitized Inputs
 5
 6  $user_input
 7
 8  $password = $_POST['password'];
 9
10  $email = sanitize_text_field($_POST['email']);
11
```

```
Unsanitized or unvalidated user inputs found (WP Security)
View Problem (⌥F8)   No quick fixes available
```

Figure 9: Highlighting Unsanitized Input

Hovering over the highlighted line of code, the extension will give a warning message

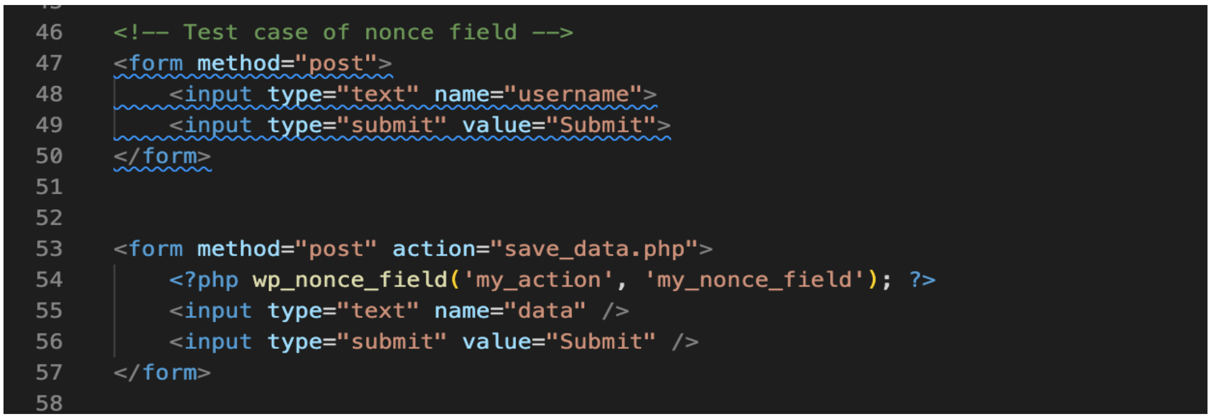


```
Unescaped outputs found, Can cause XSS (WP Security)
View Problem (⌥F8)    No quick fixes available
echo "Hello, " . $user_input . "!";

echo "Hello, " . esc_html($user_input) . "!";

$html = '<a href="http://www.example.com/">A link</a>';
```

Figure 10: Unescaped Output

5.2 Case Study 2: Cross-Site Request Forgery CSRF

To prevent CSRF attacks, WordPress has given a token method that is known as the Nonce field. In HTML form, the developer must need to use a nonce field. Line numbers 47 to 50, is an example of an unsecured HTML form highlighted because it's not using nonce field. Where line number 53 to 57, the HTML form is using nonce field so that is not highlighted.



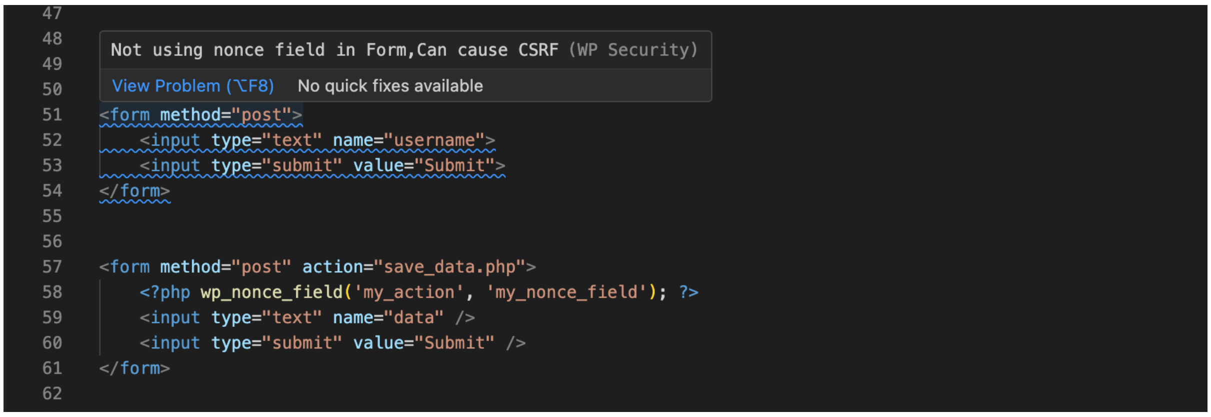
```
46  <!-- Test case of nonce field -->
47  <form method="post">
48      <input type="text" name="username">
49      <input type="submit" value="Submit">
50  </form>
51
52
53  <form method="post" action="save_data.php">
54      <?php wp_nonce_field('my_action', 'my_nonce_field'); ?>
55      <input type="text" name="data" />
56      <input type="submit" value="Submit" />
57  </form>
58
```

Figure 11: Nonce Field

Hovering over the highlighted line of code, the extension will give a warning message

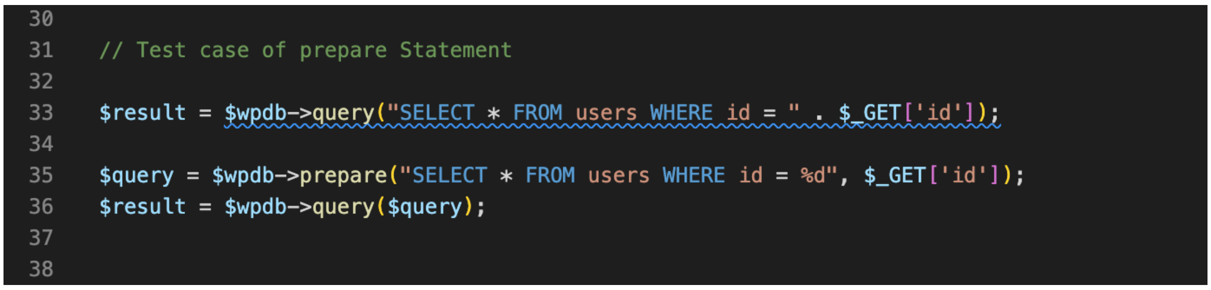
5.3 Case Study 3: SQL Injection

SQL injection is still the top vulnerability in WordPress vulnerabilities, SQL injection can be prevented by sanitizing input instead of passing variables directly to query, a developer can use WordPress predefined prepare statement. On line number 33 in the SQL query, the variable is passed directly which can cause SQL injection



```
Not using nonce field in Form,Can cause CSRF (WP Security)
View Problem (⌥F8)    No quick fixes available
<form method="post">
    <input type="text" name="username">
    <input type="submit" value="Submit">
</form>


<form method="post" action="save_data.php">
    <?php wp_nonce_field('my_action', 'my_nonce_field'); ?>
    <input type="text" name="data" />
    <input type="submit" value="Submit" />
</form>
```

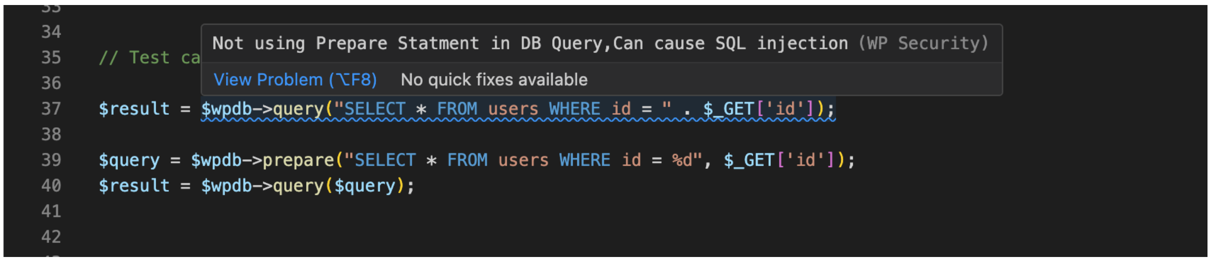
Figure 12: Highlighting Nonce field in Form



```
30
31  // Test case of prepare Statement
32
33  $result = $wpdb->query("SELECT * FROM users WHERE id = " . $_GET['id']);
34
35  $query = $wpdb->prepare("SELECT * FROM users WHERE id = %d", $_GET['id']);
36  $result = $wpdb->query($query);
37
38
```

Figure 13: Prepare statement SQL Injection

on line number 35, the variable is passed to prepare function but prepare function process the data as a string rather than executable SQL. This means attackers can't insert malicious SQL commands



```
// Test ca
Not using Prepare Statment in DB Query,Can cause SQL injection (WP Security)
View Problem (⌥F8)    No quick fixes available

$result = $wpdb->query("SELECT * FROM users WHERE id = " . $_GET['id']);

$query = $wpdb->prepare("SELECT * FROM users WHERE id = %d", $_GET['id']);
$result = $wpdb->query($query);
```

Figure 14: Highlighting unsecured SQL Query

6 Conclusion and Future Work

In conclusion, WordPress core is secure, but the plugin ecosystem brings vulnerabilities. Web developers lack knowledge of secure coding which results in vulnerable plugins. To avoid those vulnerabilities developers must follow secure coding from the coding stage. IDE and IDE extensions play a major role in secure coding.

The developed VS Code extension "WordPress Security Scan" is helping to tackle vulnerabilities, which cover the most common vulnerabilities. Tackling those vulnerabilities improves the security of WordPress applications. Integrating OpenAI for security reports, give in detail solutions and the impact of the vulnerabilities and which helps developer to understand them easily. It is giving results as expected and prevents common security vulnerabilities, this extension is easy to use, and it is beneficial for WordPress plugin developers.

6.1 Future Work

At present, the extension addresses 5 to 6 security controls. In the future, it can aim to analyse a wide range of security vulnerabilities caused by various coding practices. This analysis will enhance the capabilities of the existing extension. While the extension currently scans and analyses code using the regular expression method, it's important to recognise that every developer has a different approach to coding. To understand complex coding styles, need more deep analysis of coding styles.

VS code marketplace allows publishing extensions, publishing extensions can be very helpful for WordPress developers. Feedback from developers will directly help to improve the extension.

References

Abhilash Kumar, Aman Kumar, H. H. S. A. K. (2021). Wordpress: A multi-functional content management system, *2021 10th International Conference on System Modeling &amp Advancement in Research Trends (SMART)*, IEEE.
URL: *https://doi.org/10.11092Fsmart52563.2021.9675311*

Aniqua Z. Baset, T. D. (2017). Ide plugins for detecting input-validation vulnerabilities, *2017 IEEE Security and Privacy Workshops (SPW)*, IEEE.
URL: *https://doi.org/10.11092Fspw.2017.37*

Arafa Anis, Mohammad Zulkernine, S. I. C. L. C. C. (2018). Securing web applications with secure coding practices and integrity verification, *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, IEEE.
URL: *https://doi.org/10.11092Fdasc2Fpicom2Fdatacom2Fcyberscitec.2018.00112*

Blerim Rexha, Arbnor Halili, K. R. D. I. (2015). Impact of secure programming on web application vulnerabilities, *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*, IEEE.
URL: *https://doi.org/10.11092Fcgvis.2015.7449894*

Daniel T. Murphy, Minhaz F. Zibran, F. Z. E. (2021). Plugins to detect vulnerable plugins: An empirical assessment of the security scanner plugins for wordpress, *2021 IEEE/ACIS 19th International Conference on Software Engineering Research, Management and Applications (SERA)*, IEEE.
URL: *https://doi.org/10.11092Fsera51205.2021.9509274*

Hannes Trunde, E. W. (2015). Wordpress security, *Proceedings of the 17th International Conference on Information Integration and Web-based Applications &amp Services*, ACM.
URL: *https://doi.org/10.11452F2837185.2837195*

How Attackers Gain Access to WordPress Sites (n.d.). `https://wordpress.org/about/security/`.

Jiahuei Lin, Mohammed Sayagh, A. E. H. (2023). The co-evolution of the wordpress platform and its plugins, *ACM Transactions on Software Engineering and Methodology* **32**(1): 1–24.
URL: *https://doi.org/10.11452F3533700*

Jingyue Li, Sindre Beba, M. M. K. (2019). Evaluation of open-source ide plugins for detecting security vulnerabilities, *Proceedings of the Evaluation and Assessment on Software Engineering*, ACM.
URL: *https://doi.org/10.11452F3319008.3319011*

Jose-Manuel Martinez-Caro, Antonio-Jose Aledo-Hernandez, A. G.-P. R. S.-I. M.-D. C. (2018). A comparative study of web content management systems, *Information* **9**(2): 27.
URL: *https://doi.org/10.33902Finfo9020027*

Kire Jakimoski, Zorica Stefanovska, V. S. (2022). Optimization of secure coding practices in sdlc as part of cybersecurity framework, *Journal of Computer Science Research* **4**(2): 31–41.
URL: *https://doi.org/10.305642Fjcsr.v4i2.4048*

Laksmiati, D. (2023). Vulnerability assessment with network-based scanner method for improving website security, *Journal of Computer Networks, Architecture and High Performance Computing* **5**(1): 38–45.
URL: *https://doi.org/10.477092Fcnahpc.v5i1.1991*

Marie Vasek, John Wadleigh, T. M. (2016). Hacking is not random: A case-control study of webserver-compromise risk, *IEEE Transactions on Dependable and Secure Computing* **13**(2): 206–219.
URL: *https://doi.org/10.11092Ftdsc.2015.2427847*

Maruf Hassan Md, Kaushik Sarker, S. B.-H. S. M. (2017). Detection of wordpress content injection vulnerability, *International Journal on Cybernetics &amp Informatics* **6**(5): 1–15.
URL: *https://doi.org/10.51212Fijci.2017.6501*

Moeketsi Mafereka, S. W. (2017). Analysis and development of an online knowledge management support system for a community of practice, *Proceedings of the 2017 International Conference on Information System and Data Mining*, ACM.
URL: *https://doi.org/10.11452F3077584.3077604*

Musa Bala Shuaibu, R. A. I. (2017). Web application development model with security concern in the entire life-cycle, *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, IEEE.
URL: *https://doi.org/10.11092Ficetas.2017.8277849*

Oslien Mesa, Reginaldo Vieira, M. V. V. H. S. D.-E. C. M. K. C. L. (2018). Understanding vulnerabilities in plugin-based web systems, *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1*, ACM.
URL: *https://doi.org/10.11452F3233027.3233042*

Paulo Nunes, Iberia Medeiros, J. F. N. N. M. C.-M. V. (2017). On combining diverse static analysis tools for web security: An empirical study, *2017 13th European Dependable Computing Conference (EDCC)*, IEEE.
URL: *https://doi.org/10.11092Fedcc.2017.16*

Rafiq Ahmad Khan, Siffat Ullah Khan, H. U. K. M. I. (2021). Systematic mapping study on security approaches in secure software engineering, *IEEE Access* **9**: 19139–19160.
URL: *https://doi.org/10.11092Faccess.2021.3052311*

Ruohonen, J. (2019). A demand-side viewpoint to software vulnerabilities in wordpress plugins, *Proceedings of the Evaluation and Assessment on Software Engineering*, ACM.
URL: *https://doi.org/10.11452F3319008.3319029*

Teemu Koskinen, Petri Ihantola, V. K. (2012). Quality of wordpress plug-ins: An overview of security and user ratings, *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, IEEE.
URL: *https://doi.org/10.11092Fsocialcom-passat.2012.31*

Tiago Espinha Gasiba, Ulrike Lechner, M. P.-A. D. M. (2021). Is secure coding education in the industry needed? an investigation through a large scale survey, *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE.
URL: *https://doi.org/10.11092Ficse-seet52601.2021.00034*

WordPress Hacking Statistics (n.d.). `https://colorlib.com/wp/wordpress-hacking-statistics/`.

WordPress Security White Paper (n.d.). `https://wordpress.org/about/security/`.

Xianyong Meng, Kai Qian, D. L. P. B. (2018). Detectors for intent icc security vulnerability with android ide, *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, IEEE.
URL: *https://doi.org/10.11092Ficufn.2018.8436802*