Allegheny College
Department of Computer Science

Senior Thesis

SeQure: Cross-Site Scripting Vulnerability Tool for WordPress Form Plugins

by

Alexis Caldwell

ALLEGHENY COLLEGE

COMPUTER SCIENCE

Project Supervisor: **Douglas Luman**
Co-Supervisor: **Oliver Bonham-Carter**

22 July 2022

Abstract

XSS vulnerabilities, despite being one of the older web exploits, continue to plague modern web applications. This odes to programmers not properly screening inputs to programs – in this case, web forms. The content manager system, WordPress, for example operates roughly 43% of web applications across the Internet. Within WordPress, it has distinct plugins to handle contact form information which includes the ability to take in information such as a name, a phone number, email address, etc. However, not all versions of these plugins are deemed safe. There are many versions of WordPress form plugins that allow attackers to enter in cross-site scripting, or otherwise known as XSS, scripts into these input forms making the attacker able to provide malicious information, downloads, and/or plugins to the user. To avoid XSS attacks within form plugins on WordPress, the plugin SeQure allows website users to be able to check their form plugins to make sure that they are safe for further use.

Table of Contents

List of Figures

List of Tables

Introduction

Making a website can be very difficult for many people. However, a content management system called WordPress has made building a website easier for the general public. WordPress provides a wide variety of functions, including the capacity for users to establish blogs and the use of store owners to establish eCommerce websites. There are countless applications of WordPress that a user can do with their WordPress website.

Nevertheless, that sounds intriguing, yet the use of WordPress' capabilities are not always protected correctly. Many WordPress features have downloadable components called plugins that expand the functionality of the website. Even though the plugins that are highlighted seem secure, not all variations of a given plugin are regarded as secure.

This consists of the usage of third-party plugins and downloads that are permitted by WordPress, so users can change anything that is useful to them. Potential attackers can use this opening to discover vulnerabilities through a WordPress plugin on a website. The harm to the owner of the website could be permanent if an attacker is able to design a vulnerability and launch a successful assault against it.

This study will focus on the Cross-Site Scripting (XSS) attack, one of the many assaults that can be launched against a WordPress plugin. A website owner may incur significant expenses as a result of having a vulnerable plugin. A malicious download or some functionality being disabled could have a significant negative impact on a user's operating system and the owner's business or organization. For WordPress users, this is a significant danger, and this study will focus on those risks.

WordPress

WordPress is a content management system that allows users to build non-interactive and interactive websites for their business or own personal use. It is utilized by 43.2% of all websites on the Internet. WordPress is also GPLv2 licensed, which implies that anybody can use or modify the WordPress software without charge. Users of WordPress can create an unlimited number of pages, articles, and products, as well as benefit from flexible scheduling options, managed website security, core and plugin updates, automated backups, and a host of other features.

Honing in on the plugin functionality within WordPress, programmers have created a wide variety of plugins tailored to customer demands and needs. These plugins include handling security, eCommerce, blogs, widgets, Google analytics, contact forms, and so on. These plugins are created to make it simpler for the general public to design and create dynamic websites.

The substantial issue that needs to be taken into account is that not every person will consider inspecting the particulars of the plugin code that they are utilizing. Users may often assume that if the plugin is available for downloadable use, then the plugin is safe, right? Although this may be true in some cases, not all plugins are equal to the fact that they are updated frequently and regularly checked for vulnerabilities. A majority of the active WordPress plugins are, in fact, updated rather frequently containing plenty of different and safe versions a user can download.

A helpful aspect WordPress has to attempt to combat an older and/or a potential vulnerable plugin version staying active is the ability of alerting the website owners that a plugin needs to be updated via their home dashboard. In spite of the alert, the website owner still has the option whether to update their plugin or not since plugin updates are not automatic. This research will be centered around contact form plugins WordPress makes available to users to download that could potentially harmful for WordPress user.

We must first define threats and vulnerabilities in relation to this technology and the research that underlies it before we can start talking about the harms of a vulnerable website and the threats that could come with it. Any situation or occurrence that has the potential to harm an IS through unauthorized access, data deletion, disclosure, alteration, and/or denial of service is referred to as a threat[5].

Contrarily, a vulnerability is defined as a weak point in an IT system that can be used by an attacker to launch a successful attack. Attackers will try

to exploit any of them, frequently combining one or more, to reach their end aim. They might arise from defects, features, or human error[4].

90% of all WordPress vulnerabilities, according to WPScan, are located in plugins. As a result, plugins are the most typical point of entry for hackers to infect WordPress sites. They are also expanding. The number of WordPress plugin vulnerabilities increased by 142% in 2021, according to statistics from Risk Based Security[10].

To fully highlight the harm plugins can ensue, a look into a popular form plugin named, Contact Form 7 (CF7) should be discussed. CF7 is a contact form plugin that has over five million installations. Users are able to dynamically adjust this contact form to be able to provide user inputs to specific fields determined by the website's wonder. The public can choose from a number of versions of this plugin, and as of right now, CF7 is using version 5.7.5.1, which was most recently updated three weeks ago. Although the plugin is regularly updated, 17.1% of users are using a version lower than 5.1.

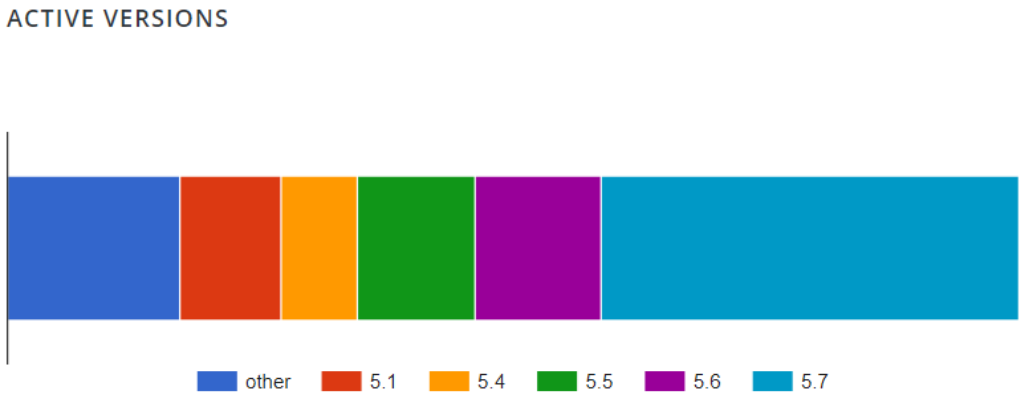


Figure 1: Contact Form 7 Statistics

Unknowingly to some users, there was a necessity for an upgrade due to several versions of version 4.0 having flaws. That indicates that a vulnerable version of this form is being used by 855,000 people. Another example is Ninja Forms with one million installation in total and 11% of those users are utilizing an older and potentially vulnerable version of that form which

means 110,000 users have the potential to have a vulnerability within their website.

Among these potential flaws, cross-site scripting (XSS), a type of cyberattack, accounted for 52% of plugin vulnerabilities in the first half of 2021[10].

Cross-Site Scripting (XSS)

Simply described, cross-site scripting (XSS) involves the insertion of malicious JavaScript scripts or payloads in order to distribute harmful files, plugins, and other media content to target a web application. The term "payload" has a similar definition when referring to malicious scripts because it refers to malicious code that harms the victim who is being targeted.

In the event that a XSS attack is being executed, the client-side or server-side attacks are the attacker's two possible points of entry in an XSS assault. When processing is client-side, it happens on the user's computer. It necessitates that browser executes the scripts locally, without utilizing any server-side processing. Processing that is done "server-side" happens on a web server, meaning, it is not visible to the users.

If organizations and businesses that run online websites have the ability to display content from users or untrusted sources without the required escaping or validation, they risk opening the door for XSS assaults creating the option and ability for attackers to plant harmful programs into user's operating systems as well as the potential to gain access to user's passwords. When discussing cross-site scripting attacks, they are typically divided into three groups: reflected, stored, and document object model (DOM) attacks.

Reflected XSS

When an application receives data in an HTTP request and includes that data inadvertently in the instant response, it is known as reflected cross-site scripting (or XSS). To put it simply, an HTTP request, or hypertext transfer protocol request, is the structural transferral of data over the internet. So when the reflected attacks takes place, the inputted payload is displayed outside of the web server because the data was supplied directly to the server as part of the request.

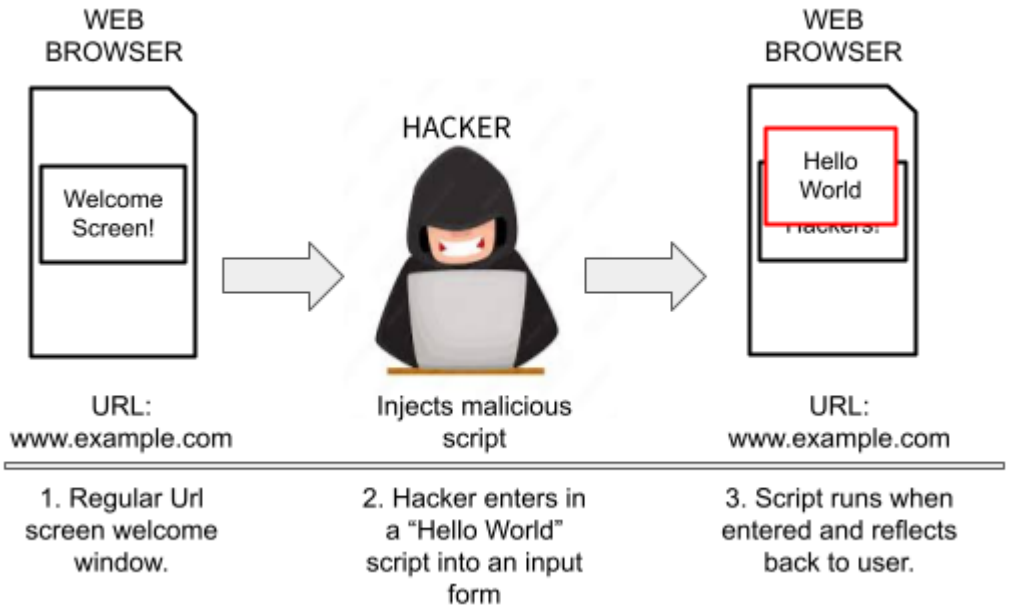


Figure 2: Reflected XSS

Following Figure 1, the web browser has a default "Welcome Screen!" opening page. As an example script, let's use this simple XSS reflection script to go along with the image above:

```
<script>alert("Hello World")</script>
```

The script, which the attacker can type on the page, is then performed and mirrored back to the screen as an alert, which the user can see making this a client-side attack. Now, it doesn't seem to represent a real threat with this example script; however, if a more well-planned script were to be entered, the attacker would have the ability to "perform any action within the application that the user is capable of performing," "view any information that the user is capable of viewing," "modify any information that the user is able to modify," and many other options (PortSwigger). The impact of a successful reflected attack could mean the opportunity to completely compromise the end-user or victim.

Consider a website for online banking that enables customers to sign in and view their accounts. Users can use the website's search box to look up

specific transactions in their account history. However, the website's search bar does not effectively sanitize user input.

The search query parameter of a malicious link made by an attacker could contain JavaScript code. The JavaScript code is run on the victim's browser when the victim clicks on this link. This code has the potential to steal the victim's session cookie, giving the attacker access to the victim's account and the ability to carry out unwanted transactions.

The reflected XSS attack in this instance gave the attacker the ability to steal sensitive data (the victim's session cookie) and carry out harmful tasks (unauthorized transactions) on the victim's behalf. Financial loss for the victim and harm to the online banking website's reputation could result from this.

Stored XSS

When an application receives user-supplied input from an unreliable source and unintentionally incorporates that data in successive HTTP answers, the result is known as stored XSS, sometimes known as "persistent attacks." Stored XSS attacks take place server-side, so the user cannot see them. Attacks that are stored have an impact on the website owner's database, and the owner must entirely clean the database to remedy a stored XSS assault. For example when a stored XSS script is entered onto a host website, the web application receives the user input from the source and stores it in the website's database. This type of XSS is alarming because whenever a user reloads their browser the script will continue to run because the stored script has not been removed from the database.

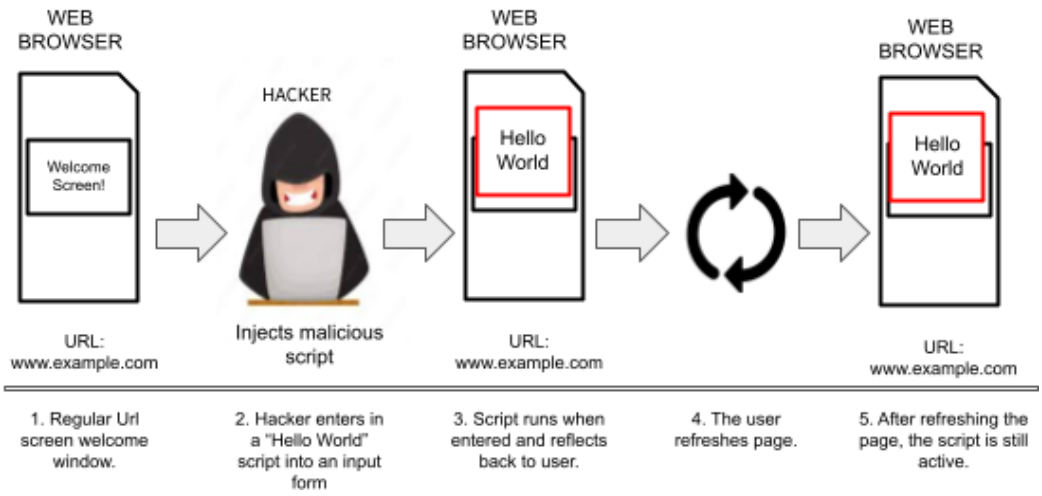


Figure 3: Stored XSS

Figure 2 outlines the process of stored XSS on a web application. The hacker enters in a script and it is able to be executed on the browser screen. The script for this example is demonstrated below:

```
<script src="http://malicious.com/exploit.js"> </script>
```

The script in this instance is made to steal a visitor's session cookie, giving the attacker access to the user's account. this enables them to purchase items on behalf of the user and provides them access to credit card and other personal data (Moradov). Regardless if the user is able to notice the script, the script is now embedded within the website's database. Any visitor who now goes to this website, the script will be able to run even if the user refreshes the page. The risks of a stored script being inside of a database is that the attack could "obtain sensitive information stored in the user's account or in their browser", "steal the user's credentials", "hijack the user's session and perform actions on their behalf" along with many other malicious alternatives that could negatively affect a web application. Mordadov states in his article of "Stored XSS", that "Stored XSS attacks are even more significant in websites that require authentication. When an authenticated user visits a page with stored XSS, attackers are usually able to hijack their session and perform actions on their behalf. On some websites, such as those of financial

or medical institutions, this can result in financial loss or exposure of highly sensitive data."

Moradov highlights the great negative impact that a successful stored XSS attack could mean for a business and company. The website's owner would have to invalidate all sessions and clean the database where the injected script is stored, otherwise, the script will stay in affect and continue to plague the website.

In the latter months of 2015 and the beginning of 2016, eBay had a significant XSS vulnerability. The website employed a `URL` parameter that sent visitors to various sites on the platform, but the parameter's value was not verified. This made it possible for attackers to insert harmful code into a page.

Due to this stored vulnerability, attackers were able to take full control of eBay seller accounts, offer products at a lower price, and steal payment information. Attackers actively utilized it to modify eBay listings for expensive goods like automobiles. Although the vulnerability was subsequently fixed by eBay, recurrent attacks persisted until 2017[8].

Document Object Model (DOM)

Last but not least, document object model, or "type-0" XSS, is an attack in which the original client side script modifies the DOM status of the victim's browser and changes it to cause the client side code to run erroneously [1]. To put it simply, the DOM of a webpage is a tree of objects. The "head" of the HTML document is visible in this image as the root element. Two child elements that include the "head" and "body," and so forth, branch out of the root element. It consists of the hierarchy of HTML elements.

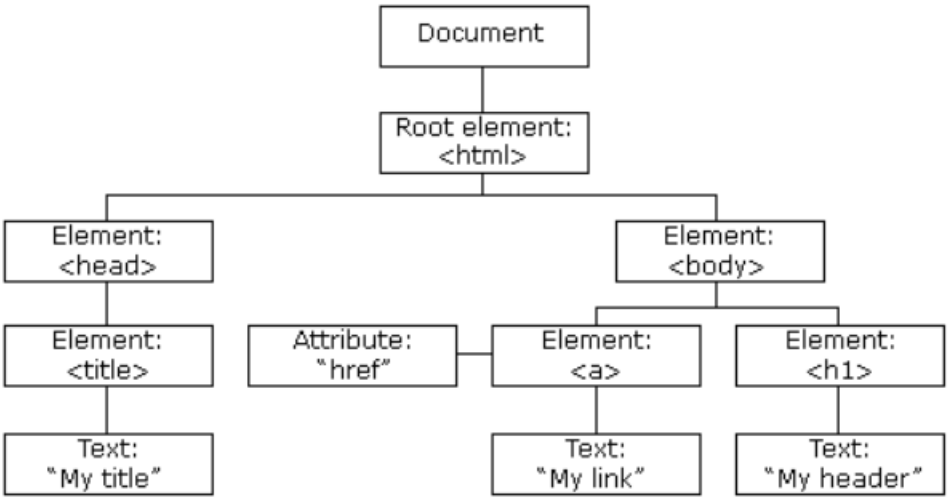


Figure 4: DOM Tree [2]

So, a DOM attack modifies the environment, but "the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment"[1]. To put in other words, the attacker's malicious changes to the DOM environment are incorporated when the browser runs the client-side JavaScript code present on the page. This may result in the code operating differently than the developer intended, which may cause a variety of problems.

The malicious JavaScript that was injected from the source is ultimately executed at the reflection point known as the sink (or assisted in execution) which is why attackers typically use the website's URL. They are often either JavaScript functions that allow for the direct execution of JavaScript or locations on the DOM or other browser objects that can modify and activate code.

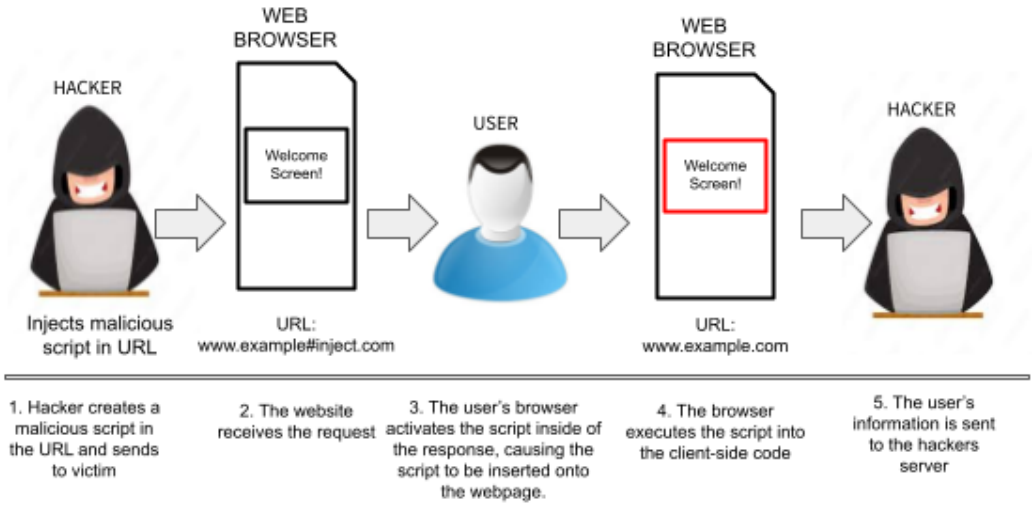


Figure 5: DOM XSS

```
www.example.com.html#context=<script>Function(variable)</script>
```

The aforementioned example script shows how a hacker can insert a harmful script inside of a URL. Following receipt of this URL, the victim's browser makes an HTTP request to the website "www.example.com" and obtains the static HTML. The browser begins creating the page's DOM and adds the injected URL to the document URL attribute. Then, the script is run by the browser once it has parsed the HTML page and extracted the malicious material from the document URL property. Next, it gets added by the browser to the page's raw HTML body. The JavaScript code is executed by the browser after being located in the HTML body [1].

Consider a social media platform where individuals can publish messages on their profile pages. Users of the website have access to a tool that lets them utilize unique fonts and colors in their postings. However, the website's font and color selection options do not adequately sanitize user input.

A malicious post that uses JavaScript code in the font or color picker options could be made by an attacker. The JavaScript code is run in the victim's browser when they visit this post. This code has the potential to change the page in a malicious manner or steal the victim's session cookie or other private data.

In this instance, the DOM-based XSS attack made it possible for the

attacker to carry out illicit deeds without having to include code into the server response. The victim's browser was the only place where the code was really run. Various sorts of harm, like identity theft, illegal access to user data, or other fraud, could result from this.

Harms of XSS on WordPress

The basic idea behind a WordPress XSS attack is to insert malicious scripting code into a weak website, for instance by posting a remark in a forum that drives users to a bogus website (phishing) or stealing data (cookies).

The majority of the time, XSS enables an attacker to acquire the cookies of a victim and use them to hijack the victim's session on the server. Information theft, including the theft of email addresses, IP addresses, and even credit card numbers, might result from this[9].

Prevention

Website owners face significant risk if their website has a cross-site scripting vulnerability. Depending on the type of script being supplied, reflected, stored, and DOM XSS can all be equally dangerous to users. Website developers must make sure that the input forms they are employing have special cybersecurity safeguards to prevent XSS attacks in order to prevent these types of attacks from occurring on a website.

In order to avoid harms like identity theft, fraud, the ability to steal payment information, financial loss, and many other things, it is crucial for developers to take precautions against XSS attacks, such as properly sanitizing user input and using output encoding techniques to prevent malicious code from being executed on the client-side.

Data Sanitization and Escaping

Cybersecurity measures that should be taken are correctly sanitizing input forms and providing input validation such as escaping and encoding characters to increase security on web applications from becoming vulnerable to an XSS attack.

In order for data to be given to a subsystem, it must first be cleaned up or "sanitized" to ensure that it complies with all applicable standards.

As part of data sanitization, it is also important to make sure that sensitive information is not exposed or leaked when output across a trust boundary. Sanitization entails deleting all characters from a value to make it "safe," unlike encoding and escape. To generalize, it is the process of safeguarding, cleaning, and filtering input data.

Sanitization can take place both before and after input (input sanitization), or it can take place before the data is sent across a trust boundary (output sanitization). To continue, verifying that input data fits within the desired range of acceptable program input is the process of validation. This necessitates that inputs abide by the class or subsystem's input invariants as well as requirements for type and numeric range. [15]

Escaping is the process of protecting output by removing extraneous information, such as erroneous HTML or script tags, to stop this information from being interpreted as code. An escape sequence is created by converting a control character to it. As discussed in an "Integrating the Escaping Technique in Preventing Cross Site Scripting in an Online Inventory System" written by Jasmin Caliwag and other colleagues, she illustrates the example of, "a < symbol may be converted to < so that the characters following the < are not interpreted as an XML tag instead of XML content." To elaborate XML, is a tool for storing and moving data that is independent of hardware and software. Much like HTML it is a markup language designed to carry the data whereas HTML if more focused on the design of displaying data.

Caliwag goes on in the article to provide an example of an escape function. [7]

```
<?php
function escape($string) {
 return htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
}
```

The code snippet above illustrates a PHP function that accepts a string as an argument and escapes all special characters before returning the string. The escaping is done by the function using the built-in PHP method `htmlspecialchars()`. Special characters such, >, &, and " are converted to their relevant HTML entities by the `htmlspecialchars()` function. The function is instructed to convert both single and double quotes to the appropriate HTML entities via the `ENT_QUOTES` parameter. The character encoding to be used is specified via the UTF-8 argument which is an encoding system for Unicode.

Now, if a user inputted this attack:

```
Hello Divya<script>alert(1);</script>!
```

The alert will be reflected back to the website's on the client side. With the PHP code, coded into the input field, the script will return as such:

```
Output
Hello Divya&script&alert(1);&/script&!
```

Now that the PHP scripted into the code, the input field has been sanitized for certain characters. When properly and fully implemented, these cybersecurity measures can reduce the likelihood that a user input form may be vulnerable to cross-site scripting.

Motivation

As an open source, third-party content management system, WordPress should be treated just as seriously in terms of security as the WordPress website as a whole. If the entrance point of the attack is a plugin, a successful XSS attack threatens the website owner in addition to being extremely destructive to the site's visitors. Anything from displaying false information to covertly installing malware onto the user's operating system could fall under this category. WordPress users need to be aware of the security risks posed by form plugins. To be clear, not everyone is interested in computer security, so it is doubtful that the code now in use has been checked for bugs.

Incorrect input validation or sanitation can make it more likely for an XSS script to be sent successfully when using earlier versions of form plugins. The goal of this tool and the research is to draw attention to a current issue in the WordPress plugin market. SeQure is a tool that helps WordPress users stay vigilant and check their plugins for security flaws.

Goals of the Project

An in-depth research and study of plugin vulnerabilities in WordPress is the aim of this project. To assist myself in this study, will be a tool of my creation called, SeQure. The plugin itself is a plugin that checks contact input form

plugins for cross-site scripting vulnerabilities. The tool will be able scan a web page to find specific HTML tags to search for any input forms that are on the page. Next, it enters in its own malicious scripts to see if the input forms will allow the malicious characters or symbols that are entered in. If the script is able to be sent back to the owner, then the owner of the website is left with two choices: update the plugin and retry this method or to get rid of the plugin completely if it is its most updated version.

This research, also, focuses on the popular versions of contact form plugins. Because they have the most installations, users should be able to know whether or not their plugin is vulnerable. As the project continues, other form plugins may be used for research but to start, the focus are on the more popular plugins. Overall, the primary goal of this project and this research is to be able to increase the security around form plugins for users in WordPress.

Ethical Implications

Many ethical issues are raised by this undertaking and research of XSS. The tool must be able to carry out its stated purpose, provide accurate information without deceiving users, be operational, and last but not least, it cannot transform into something it is not.

Correctness, Factual, and Usability

The tool itself must be capable of carrying out its stated functions. SeQure must be able to scrape through an HTML page, discover input values, input script values into the input forms, and push the submit button to determine whether the script may send to the owner or not. If my instrument is unable to perform the aforementioned tasks, then lying and misleading advertising become one of the most serious ethical issues with my product. Users do not desire products that do not fulfill their described functions, and developers have little incentive to release a product if it is not ready for general usage and fulfills its stated functions.

My tool must not only be accurate for the activities I am advertising, but it must also be able to communicate accurate information to my users and be easy to use. A useless tool ties up with the moral implication that users are using a useless tool that does them absolutely no good. In order

for consumers to boost security within the plugins on their websites, my tool must operate properly.

Integrity

The integrity of my tool would have to be the most significant ethical implication. For instance, an outside developer should not be able to use my cross-site scripting tool! My tool has the capacity to change from a safe feature to a very damaging one if altered. Users can tinker with the code to accomplish any additional tasks they require because WordPress is an open-source content management system. Because of this, it is possible to alter the code and functionality of my tool. The plugin and website users would be seriously threatened if the code were altered by an attacker. Several comprehensive cybersecurity measures must be implemented in an effort to prevent this so that is not the outcome.

Related work

While plugins are one of WordPress' greatest strengths, they also pose a large security concern when it comes to web application attacks. If properly implemented, being able to protect against web app attacks by utilizing certain security measures will contribute to the overall security of the functionality of the online applications. These concerns will be addressed, serving as both the primary emphasis of this background investigation, as well as a focus on SeQure-comparable plugin tools and how those tools have performed while actively serving as WordPress security scanner plugins.

Defending against Web Application Attacks

To reiterate, the three types of cross-site scripting attacks are non-persistent (also known as reflected) XSS, persistent (also known as stored) XSS, and DOM-based XSS. Non-persistent XSS attacks happen when a user's input is evaluated instantly by server-side application logic and results in a dynamically generated response that is ultimately presented by the user's browser without sufficient sanitization. As a result, in a non-persistent XSS attack, the injected code is immediately included into the content that is delivered back to a user rather than being saved on the server. On the other hand, malicious code is kept on the server indefinitely in persistent XSS attacks. Once again entering the application's execution cycle, the injected server-side code finally becomes a component of the content that is provided to the user as part of a subsequent response. DOM-based XSS attacks change a webpage's DOM in some way. An HTML document is viewed by the DOM as a tree structure, with each node representing a different section of the content. Programmatically, each item can be accessed and changed, and any noticeable modifications may then be seen in the browser.

The most harmful web attacks, such Cross-Site Scripting and SQL Inser-

tion, take use of flaws in web applications that could accept and process data of questionable origin without sufficient validation or filtering, enabling the injection and execution of dynamic or domain-specific language code. Despite the numerous defenses that have been suggested over the past 15 years, these assaults have consistently topped the lists of several security bulletin providers. Despite the numerous countermeasures that are being offered, attackers continue to come up with innovative ways to get through defenses. For instance, more than 20 potential countermeasures against SQL injection attacks were available as early as 2006. Since then, the amount has doubled, and according to researchers, the frequency of SQL injection assaults has been rising rapidly in recent years [12].

Developers of web-based infrastructures should use coding methods that involve a defense-in-depth strategy, considering that each security measure has a chance of failing, to generate code free from vulnerabilities. During implementation, it is crucial to use a strategy that relies on multiple layers of defensive mechanisms since sometimes a single precaution or defense is not enough to prevent security flaws.

As a first line of defense, input validation reduces the input domain of an application by operating solely on the values supplied by the user. This style of defense focuses on stopping execution when a user enters a value outside the allowed domain or requiring the input parameters to fall within a specified valid domain. Starting with normalizing the inputs to a standard character set and encoding is recommended for Web applications. The software must then apply filtering techniques to the normalized inputs in order to eliminate any that contain values outside of the acceptable range. By using this technique, Web applications that do input validation using positive pattern matching or positive validation might prevent several issues[6].

A second layer of security is necessary to address input validation's flaws. In light of this, each attack type focuses on a hotspot, or a portion of the application's code that is especially susceptible to a certain kind of flaw. The proper usage of parameterized commands is an additive defense against injection vulnerabilities. In this case, the developer defines the structure of the commands by using placeholders to indicate the instructions' variable values. Once the application has connected the proper values to the command, the command interpreter can use them in the future without tampering with the command's structure.

In any event, users should not receive information about application exceptions or other information that could be used to launch additional attacks.

This can be avoided by validating a process' output before it is sent out. Another illustration of output validation is when a security program checks the output of an application for crucial data, like a credit card number, and substitutes it with asterisks before delivering it to the intended recipient. A sort of output validation called encoding protects against XSS vulnerabilities. Depending on where the data ends up in the webpage after being delivered to the browser, it should be encoded using either HTML or percent encoding. In this approach, even harmful characters used in XSS assaults become harmless, yet the encoding still maintains the sense of the message [6].

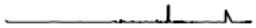
Plugin-based Web Systems

The internet and web technologies are becoming a crucial component of contemporary society. Web applications may now store, exchange, and retrieve a significant quantity of information, some of which may be sensitive, thanks to recent advancements in web technology and the Internet's quick spread. The concept behind plugin-based programming is that plugins provide extra functionality to supplement the software's basic functionalities. Additionally, this quick expansion gave developers the chance to adopt software product line engineering approaches like plugin-based development. It is unrealistic to foresee all conceivable plugin installations, interactions, and combinations in a specific web application given that WordPress allows millions of developers around the world to create their own plugins. This led to the 2018 article, Understanding Vulnerabilities in Plugin-based Web Systems written by author, Oslien Mesa and colleagues to record research about plugin-based vulnerabilities.

The study had 5 research questions they wanted to explore although by concentrating on study questions one, two, and four, they were able to learn about the primary categories of security vulnerabilities, their level of importance, and the durability of a susceptible plugin.

- RQ1: What are the main types of security vulnerabilities caused by WordPress plugins?
- RQ2: How critical are the security vulnerabilities caused by WordPress plugins?
- RQ4: How long does a vulnerability survive in the WordPress plugins code?

Types of Security Vulnerabilities

Type	Total	Time of Introduction	Frequency
Cross-site Scripting (CWE-79)	397	Implementation	
SQL Injection (CWE-89)	166	Implementation / Operation	
Cross-Site Request Forgery (CWE-352)	120	Implementation	
Path Traversal (CWE-22)	48	Implementation	
Management of Permissions (CWE-264)	28	Design	
Improper Input Validation (CWE-20)	21	Implementation	
Code Injection (CWE-94)	20	Implementation	
Information Exposure (CWE-200)	19	Implementation	
Unrestricted Upload (CWE-434)	11	Implementation	

[11]

The findings of their analysis revealed nine separate vulnerabilities that are brought on by WordPress plugins, as seen in the image above, along with information on how frequently each vulnerability happened and when each type was first discovered. The only vulnerabilities that had more than 100 successful insertions were cross-site scripting (XSS), structured query injection (SQL), and cross-site request forgery, and they were all introduced to the plugins at implementation.

The findings also indicate that certain types of vulnerabilities can fluctuate in terms of frequency throughout time. There are four types of vulnerabilities: persistent vulnerability, which appears frequently over time (P1); occasional vulnerability, which appears occasionally over time (P2); once occasional vulnerability but now lays "dormant," which refers to a vulnerability that appeared briefly before almost disappearing and has not appeared repeatedly since (P3); and (iv) occasional vulnerability that has become predisposed (P4). Notably, the bulk of the plugin-related vulnerabilities in WordPress fall under patter P1, implying that 76.31% of these vulnerabilities continue to exist over time and that security is not getting better.

Critical Vulnerabilities

In order to determine how "critical" these vulnerabilities were, they also assessed the vulnerability severity using the Common Vulnerability Scoring System (CVSS) model, which offers a qualitative measure of severity, and

information from the National Vulnerability Database (NVD), a government repository of standards-based vulnerability database, bulletins.:

Table: Common Vulnerability Scoring System Model

* Integrity:
  * None - there is no impact on integrity;
  * Partial - it is possible to make modifications to files or gain access to application information; however, either the attacker has no control over what can be modified or the attacker's scope is limited;
  * Complete - there is a total compromise of application integrity.
* Availability.
  * None - no impact on web application availability;
  * Partial - reduction in performance, or partial interruptions in available reso
  * Complete - stop of affected resources.
* Confidentiality.
  * None - no impact on web application confidentiality;
  * Partial - considerable informational disclosure;
  * Complete - total information disclosure.
* Complexity.
  * Low - does not demand expertise;
  * Medium - requires a little expertise;
  * High - demands high expertise

Type	Complexity			Integrity			Availability			Confidentiality		
	L	M	H	N	P	C	N	P	C	N	P	C
Data Processing Errors (CWE-19)	1	-	-	-	1	-	1	-	-	-	1	-
Improper Input Validation (CWE-20)	12	9	-	1	18	2	9	10	2	5	14	2
Information Exposure (CWE-200)	19	-	-	19	-	-	19	-	-	-	19	-
Path Traversal (CWE-22)	46	2	-	43	4	1	44	4	-	3	43	2
Security Features (CWE-254)	1	-	-	-	1	-	1	-	-	1	-	-
Access Controls (CWE-264)	23	5	-	5	21	2	13	13	2	11	15	2
Improper Access Control (CWE-284)	6	-	-	2	4	-	4	2	-	3	3	-
Improper Authorization (CWE-285)	2	-	-	-	2	-	2	-	-	2	-	-
Improper Authentication (CWE-287)	4	1	-	-	5	-	1	4	-	1	4	-
Cross-Site Request Forgery (CWE-352)	2	117	1	2	117	1	7	112	1	5	114	1
Unrestricted Upload of File (CWE-434)	11	-	-	-	10	1	3	7	1	3	7	1
Files Accessible to External Parties (CWE-552)	1	-	-	1	-	-	1	-	-	-	1	-
Link Following (CWE-59)	1	-	-	1	-	-	1	-	-	-	1	-
Open Redirect (CWE-601)	-	1	-	-	1	-	1	-	-	-	1	-
Command Injection (CWE-77)	2	-	-	-	1	1	-	1	1	-	1	1
OS Command Injection (CWE-78)	1	-	-	-	1	-	-	1	-	-	1	-
Cross-site Scripting (CWE-79)	1	388	8	-	397	-	397	-	-	397	-	-
SQL Injection (CWE-89)	160	6	-	-	166	-	-	166	-	-	166	-
Code Injection (CWE-94)	14	6	-	-	18	2	-	18	2	-	18	2
NOINFO	11	-	-	-	8	3	4	4	3	4	4	3
OTHER	16	16	1	2	29	2	8	23	2	2	29	2
Total	334	551	10	78	804	15	516	365	14	437	442	16

Figure 6: How critical the vulnerabilities are according to the CVSS model. L means Low, M means Medium, H means High, N means None, P means Partial, and C means Complete.

Overall, their findings imply that while a small number of vulnerabilities expose web applications to significant security threats, the majority of flaws have the potential to result in data breaches. Furthermore, fewer flaws (less than 2%) may have made the entire application susceptible to harmful attacks[11].

In accordance with the findings, 516 (57.65%) vulnerabilities do not jeopardize availability. However, the reports indicate that 365 (40.78%) vulnerabilities have the ability to adversely influence the web application's performance or partially make resources unavailable. When we take confidentiality into account, we can also see findings that are similar. A sizable portion of vulnerabilities, or 437 (48.82%), do not compromise the web application's confidentiality whereas 442 (49.38%) allow an attacker to access some system files. Only 16 (1.78%) of the reports had a vulnerability that could lead to a security breach and provide an attacker access to sensitive data. [11]

Longevity of Vulnerable Plugins

The fourth and last query, "How long does a vulnerability persist in the WordPress plugins code," is now the emphasis. The survival period, which is calculated as the time from when the vulnerability was introduced to when it was fixed, is shown in the table below.

	Number of Days
Min	1
Max	1710
Mean	563.41
Median	470
Std. Dev.	1.34

Figure 7: Longevity of vulnerability.

According to the table, it might take 563.41 days on average for a vulnerability in the plugins' code to be fixed. The consumers may have unintentionally used a vulnerable plugin version for a full year, according to this. Users of WordPress are at danger because of this, especially since 40.78% of vulnerabilities, as was indicated in the section on significant vulnerabilities, have the potential to reduce the functionality of the online application[11].

Mesa and colleagues' research provided valuable information regarding the most prevalent vulnerability types, their severity, and their persistence for WordPress users. They claim that their research indicates that these widespread vulnerabilities keep emerging for a variety of causes, indicating that their frequency of occurrence does not appear to be decreasing with time. They contend that the primary explanation is that plugin developers lack sufficient expertise in safe programming, and that WordPress's greatest appeal, even to non-techies, is its simplicity.

WordPress plugin developers also have to keep in mind that WordPress itself also continuously updates. Minor releases take place as needed, while major improvements often happen twice or three times per year. Some hosting companies will automatically update your WordPress core or the user can manually upgrade it, depending on where your hosting website has available. Updates to your website should be made carefully. Minor upgrades shouldn't create any problems, but significant updates might bring bigger changes, such the addition of new features or the removal of existing ones.

Although the WordPress development teams make a lot of effort to make upgrades backwards-compatible, it is crucial that you verify everything before making it live [16].

There can be features on an owner's website that are no longer supported. A plugin that is currently active might not work with the most recent WordPress version. Having those out-of-date plugins installed all the time is risky since the longer they go without an update, the easier it will be for hackers to access your website or online application.

As a result, in order to maintain functionality, plugins need to get regular updates from WordPress core in addition to bug and security fixes. Without that component, the plugins could be fully updated, but if WordPress itself is not updated, the functionality of the entire plugin may be at danger, creating a new vulnerability for WordPress users[16].

Security Scanner Plugins

It is important to realize that my tool is not the first plugin to scan other plugins for security flaws. In WordPress' plugin marketplace there are other more security scanner plugins available, including Defender Security, Security Ninja, JetPack, WordFence, and All in One. The fact that these plugins exist emphasizes the requirement for plugins to be able to identify insecure plugins. In "Plugins to Detect Vulnerable Plugins", many tests were run to examine how many of these security plugins actually execute their jobs in order to assess how effective these security scanner plugins are[1].

In the article, it stated that "the security of the third-party plugins is not guaranteed by WordPress itself, and indeed, 92% of the vulnerabilities found in WordPress powered websites are detected by third party plugins"[14]. Being aware of this major problem, WordPress plugin marketplace offers a range of security scanning programs to meet this lack of experience by its everyday users.

To conduct this type of experiment, they needed to be aware of the appropriate plugins for the job. Deciding which plugins should be used in the tests, the study was conducted using a set of criteria. The plugin must:

- Claim to address plugin security
- Must have 10,000+ active installs
- Must be available in the WordPress.org plugin repository

- Must offer a free tier for users on a budget

Murphy, Zibran, and Eishita ultimately decided to evaluate the effectiveness of 11 total WordPress security scanners, based off of the criteria, to see how well they perform their respective tasks.

TABLE I
SECURITY SCANNER PLUGINS EXAMINED IN THIS STUDY

Security Scanner Plugin	Version	# of Active Installs
Jetpack [8]	6.8	5,000,000+
WordFence Security [13]	7.4.12	3,000,000+
iThemes Security [7]	7.9.0	1,000,000+
All In One WP Security & Firewall [3]	4.4.4	900,000+
Bullet Proof Security [4]	4.3	60,000+
Security Ninja [2]	5.111	10,000+
Titan Anti-spam & Security [12]	7.2.1	100,000+
Sucuri Security [11]	1.8.24	700,000+
Defender Security [6]	2.4.5	40,000+
Cerber Security [5]	8.7	200,000+
SecuPress [10]	1.4.12	30,000+

Figure 8: The Security Scanners used in this study.

WordPress plugins that had been tagged with know vulnerabilities over the "past two years" are located using the well-known Exploit Database kept by Offensive Security[14]. Below are the most common vulnerabilities found in 51 vulnerable plugins that were sourced from the Exploit Database[14].

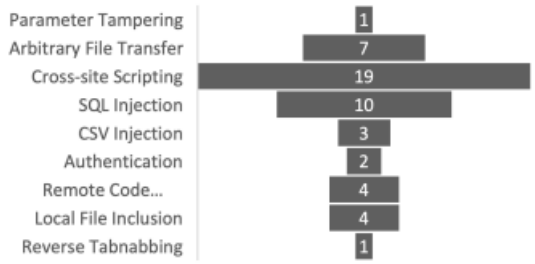


Figure 9: WordPress plugins' most common vulnerability types.

As we can see based off of the the figure, cross-site scripting was found in 37% of vulnerable plugins. With this information, Murphy and his colleagues began experimenting the security scanner plugins on the vulnerable

plugins. After conducting the experiment, "Surprisingly, six out of those 11 security scanner plugins completely failed in detecting any of the 51 vulnerable plugins" [14]. This means that even while website owners may be employing security scanner plugins, it's possible that they aren't finding all of the plugin vulnerabilities. This demonstrates how critical it is to routinely check websites for vulnerabilities and to avoid relying exclusively on security scanner plugins to do the job.

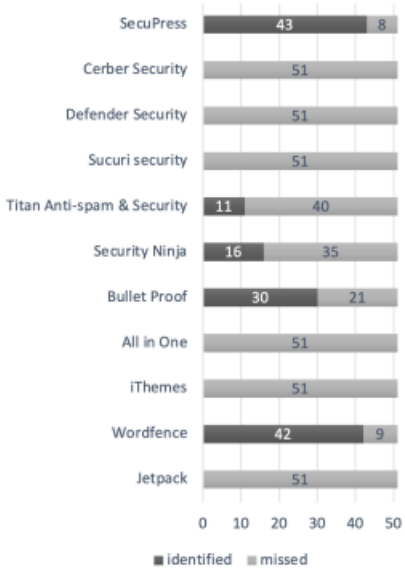


Figure 10: Performances of the Security Scanner Plugins.

The security scanners that failed the experiment require users to pay to upgrade their plugin. "The most alarming finding is that despite claiming to provide plugin security, six of the security tools (representing over 7,840,000 active installs) provide no capability for identifying vulnerable plugins under their free tier" [14]. This idea that a person needs to pay for a product to receive its full functionality is a trend within these kinds of plugins on WordPress. Not all users will pay to upgrade their plugin because they are lulled into a sense of false reality since the plugins are named a "security scanner" that the scanner will provide its utmost protection while downloaded onto their website. However, that is not the case when "free-tier provides basic functionality but requires and upgrade to a paid-tier to access additional features" [14].

In the final analysis of this study, it can be concluded that although WordPress offers a variety of security scanner plugins that not many of them are efficient enough to detect vulnerabilities. Six out of the eleven popular security scanner plugins failed this experiment by not finding any vulnerabilities within known vulnerable plugins. The study's most startling discovery is the lack of a single free WordPress security scanner plugin that can reliably identify and warn plugin vulnerabilities at the time of writing [14].

Impacts

For WordPress, there are a wide variety of security scanner plugins, and their efficacy can vary. Given that new security flaws are continually being identified, certain plugins might not be able to identify all vulnerabilities. Furthermore, certain plugins might not be updated, which could cause them to overlook more recent vulnerabilities. outdated plugins may interfere with the functionality of other parts of your website or web application, resulting in errors or strange behavior. Furthermore, outdated plugin versions could not have the most recent features or bug updates, which can harm the user experience.

With that being said, a WordPress website may be significantly impacted by vulnerable plugins as shown in tables above as well as security lapses, data loss, loss of website functionality, reputational harm, monetary loss, and many other potential effects. To reduce the possibility of these kinds of effects, WordPress users must give top priority to maintaining all plugins and the WordPress core up to date and frequently tested for vulnerabilities.

Although it can be hopeful, it is crucial to remember that no plugin can guarantee 100% security, thus having a thorough security plan in place for your WordPress website is always advised. This can involve employing a security plugin in conjunction with other safety precautions like regular backups, updating your WordPress core along with all plugins and themes, and using a strong and distinctive passwords for your admin account. To guarantee best efficiency and security, it is generally advised to keep all plugins on your website or web application up to date.

Method of approach

Project Design Outline

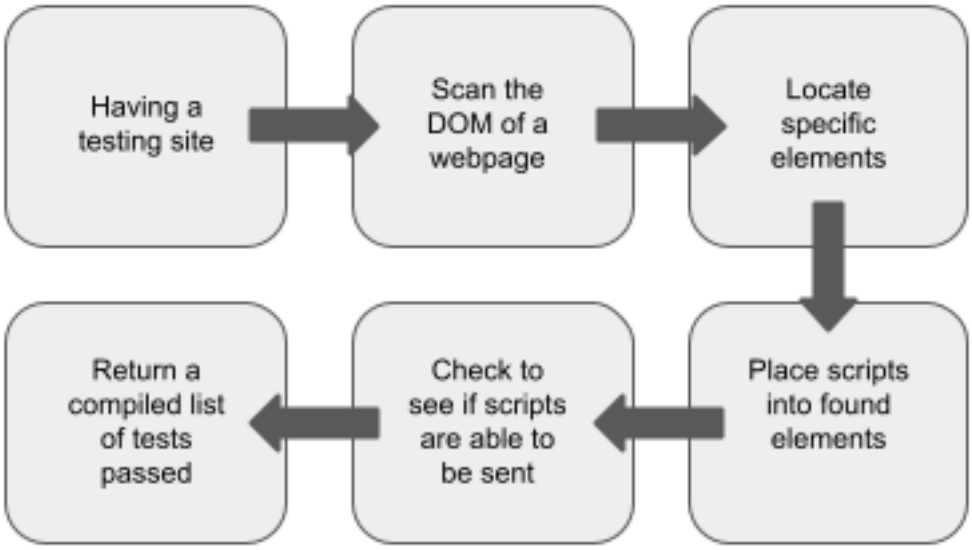


Figure 11: Technical Diagram

In general, the design plan called for the ability to execute tests on input forms. Knowing so, it was clear that specific actions were required to realize this concept.

1. Be able to have a testing site

As already mentioned, I'll be testing things, thus I'll need a test website to use for that. I can't use just any website since I need to be able to send and

receive programmed emails, and I can't send scripts to any website without permission. I must create a WordPress website with a contact form because my research focuses on the shortcomings of form plugins for WordPress.

2. Scan an HTML document

For this tool, I need to be able to retrieve data from a webpage and give it back to the user. In order to use these techniques to be able to scrape data from websites, I must employ other kinds of software.

3. Locate specific elements

I need to be able to find extremely particular HTML elements and attributes while scraping the website. I have to comprehend further about document object model (DOM) of a website to do this.

4. Place values in those found elements

I must be allowed to enter my own data into input fields. I'm making sure that the input fields have undergone thorough validation and sanitization. For testing purposes, different scripts will be entered into these input fields.

5. Check to see if a script can be sent or not

Once the values have been successfully entered into the input fields, the contact form itself produces a JavaScript pop-up, alerting the user sending the message whether or not the message was successfully delivered. If the message was sent successfully, it can be concluded that the fields were not adequately sanitized; nevertheless, if the message was unsuccessful in sending, the message had correctly sanitized and validated the input.

6. Return a compiled list of if scripts were sent or not

With each test that is run, I need to keep track of whether the entered scripts were sent successfully or not. The total number of tests that were performed will also be included.

In order to achieve these concepts, I had to utilize various software tools and programs to run this tool.

Software Tools

Document Object Model (DOM)

Previously discussed, the Document Object Model (DOM) is a hierarchical tree-like structure that represents the content, structure, and relationships of a web page. The DOM is created by the web browser when a web page is loaded and can be manipulated using JavaScript, allowing for dynamic and interactive web pages.

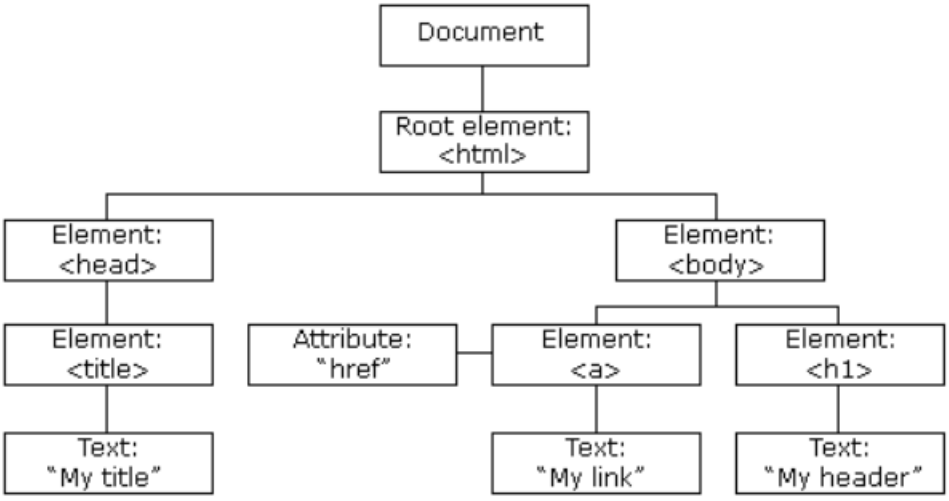


Figure 12: DOM Tree

DOM attributes are the properties of DOM elements that provide additional information about the element, such as its id, class, style, and src. Attributes can be accessed and manipulated using JavaScript, and they are often used to add dynamic behavior to elements, such as changing the style or content of an element based on user interaction.

An element can be uniquely identified using the "id," "class," "style," and "src" attributes. A web page's HTML code has attributes, each of which must have a different value.

Take this code snippet as an example:

```
<h1 id="example"> Example </h1>
```

In this snippet, the header has an id named: example. With this specific id the programmer is able to find headers easier if they have other headers with the `<h1>` tag.

Because of this, this tool will be able to locate items by understanding how a web page's DOM works and by leveraging other software tools to scrape this type of data for usage.

WordPress

WordPress is a content management system that enables users to create interactive and non-interactive websites for either their personal or professional usage. This project's objective is to take advantage of weak contact form plugins, which necessitates using WordPress to create the testing website. My ability to access numerous contact forms to test on will be made possible by building a WordPress website.

Making a WordPress website is essential to the work I'm doing for this tool. I had to create my own testing website to test these forms because my solution relies on the exploitation of contact form plugins. All I had to do to develop my website was add a page on which to place my form, and once that page was up, I could build my contact form.

Contact Forms

Plugins are a very useful addition to WordPress because they provide a user's website greater functionality. Focusing on contact forms specifically, contact forms enable website visitors to enter in specific types of information and send it back to the website's administrator. Information like name, phone number, email, are a few options contact forms have. Numerous forms, like Contact Form 7, Ninja Forms, Basic Forms, and many more, are available for owners to employ on their website.

Although they aid in functionality, contact form plugins can be harmful for website owners if an attacker is able to use the data by being susceptible to cross-site scripting (XSS) assaults. The plugin can be exploited, leaving the website insecure, if the contact form is not routinely updated and checked.

One of the more well-known form plugins, Contact Form 7 (CF7), will be the main topic of discussion for additional instances all throughout. This form has more than five million active installations with a rating of four stars overall. The plugin has many different versions with the most current version

type being 5.7.3. Unlike some plugins, this plugin gets updated and reviewed quite frequently[13].

Since CF7 is a popular tool with many different versions, it can be inferred that the reasons behind the upgrades is to better the plugin. Then again, plugins, like CF7, are also updated to fix major bugs or problems found in their plugin functionality. For example, due to improper input sanitization in some versions, CF7 is vulnerable to cross-site scripting. An attacker might take advantage of this flaw to run any script they wanted in the context of the compromised website in the browser of a careless user. As a result, the attacker may be able to launch additional attacks and steal cookie based authentication credentials. Version 4.0.1 of the plugin is vulnerable to this attack, and earlier versions can possibly be affected[3].

To this point, various versions of Contact Form 7 and others will be analyzed for correctly sanitizing input.

Programming Languages

Python

Python is a versatile and powerful coding language that is widely used in a variety of industries and applications. One of the main reasons for its popularity is its ease of use and readability. The syntax of Python is simple and straightforward, making it easy for developers of all skill levels to understand and write code.

Another advantage of Python is its versatility. It is a general-purpose language that can be used for a wide range of applications, such as web development, data analysis, machine learning, and artificial intelligence. This makes it a great choice for projects that require a combination of different skill sets.

Overall, because of its simplicity, adaptability, performance, and scalability, Python is a fantastic coding language for a variety of applications. It is a fantastic option for developers of all skill levels because to its open-source nature and vibrant community. The entire design and functionality of the tool will benefit greatly from the usage of Python in this project.

Selenium/Chromedriver

Selenium Python is a powerful tool that is commonly used for automated web testing. It is an open-source library that allows developers to automate browser interactions by simulating user interactions with a website or web application. Selenium Python is built on top of the Selenium WebDriver and provides a convenient way to write automated tests for web applications using Python.

One of the key advantages of using Selenium Python for testing is its support for multiple browsers. Selenium Python supports a wide range of browsers, including Chrome, Firefox, Edge, Safari, and Internet Explorer, which makes it a versatile tool for testing web applications across different platforms and devices.

Another major benefit of Selenium Python is its ability to simulate real user interactions. Selenium Python allows developers to automate browser interactions such as clicking buttons, filling out forms, and navigating through pages. This makes it possible to perform functional testing, regression testing, and end-to-end testing, which are essential for ensuring the quality and reliability of a web application.

Overall, Selenium Python is a powerful tool that can help developers automate web testing and ensure the quality and reliability of their web applications. Its support for multiple browsers, ability to simulate real user interactions, compatibility with various testing frameworks, and ability to perform cross-browser testing make it an ideal tool for web testing.

With this in mind, SeQure uses Chromedriver. ChromeDriver, a separate executable or standalone server, is used by Selenium WebDriver to run Google Chrome. Despite the fact that Selenium may launch its own testing browser, it is not user-friendly to simply observe browsers as they open and close (but in some cases could be). To combat this, selenium provides a `headless` option that enables it to function even without launching a browser.

```
options.headless = True
s = Service("C:/Users/Lex Caldwell/Downloads/chromedriver")
driver = webdriver.Chrome(service=s, options=options)
```

With the options equaling `True`, Selenium is able to continue to run without opening a browser. The second line in the code example illustrates the initialization of the Chrome webdriver. As it is initialized, the driver holds

two arguments. The first arguments is a `service` class that is responsible for the starting and stopping of chromedriver. Since the service needed is Chromedriver, a path should be provided in order for Chrome to open. The second argument runs the `headless` option.

To continue, the code needs to be able to find and open the browser itself. In order to do that, Selenium offers a `.get()` method that navigates to the particular website and wait till page load.

```
driver.get(www.example.com)
```

With this option available, the user will be able to enter in their specific URL that contains the contact form for them to test.

Locating Elements

One of the key features of Selenium Python is its ability to locate elements on a web page. This is an essential feature for automating web interactions, as it allows developers to locate and interact with specific elements on a web page, such as buttons, links, and forms.

There are several reasons why Selenium Python's ability to locate elements is particularly beneficial.

First, it allows developers to automate repetitive tasks and interactions on a web page. This can save time and increase efficiency by automating tasks that would otherwise have to be done manually.

Second, Selenium Python's ability to locate elements makes it easy to create robust and reliable automated tests. Developers can use Selenium Python to locate elements and interact with them in the same way that a human user would, which allows them to create tests that closely mimic real-world usage.

Lastly, Selenium Python's ability to locate elements allows for easy integration with other frameworks and libraries. Developers can use Selenium Python to locate elements and interact with them, and then use other frameworks and libraries to process and analyze the data that is extracted.

In conclusion, Selenium Python's ability to locate elements is a powerful feature that allows developers to automate web interactions, create robust and reliable automated tests, scrape data, and integrate with other frameworks and libraries. It makes web automation, testing, and data extraction more efficient and easy to perform.

Under those circumstances, SeQure has to be able to navigate the DOM of a certain contact forms that are embedded on a user's website. In addition to navigating the HTML itself, I also needed to be able to access specific elements within the HTML of the website. In my case, I am searching for input form/fields that users can type into. As discussed, selenium also provides a convenient way to locate elements on a webpage.

```
driver.find_element()
```

Previously stated, the `driver` element is the webdriver. Giving it the `find_element` class, allows Selenium to find the first element that matches and returns that value. Equally as important, selenium uses a `By` class which is used to specify which DOM attribute to locate. It can locate a variety of elements such as:

- ID
- NAME
- XPATH
- CLASS_NAME
  - etc

As discussed previously, the `ID`, `NAME`, and `CLASS_NAME` are attributes that can be accessed and manipulated. Different from being able to access those attributes, `XPATH` is a query language used to navigate and select elements in the DOM. XPATH uses a syntax similar to that of a file system path to select elements in the DOM. For example, you can use an XPATH query to select all fo the links in a web page, or to select a specific element based on its ID or CLASS.

With these options, I was able to search for specific input fields within a page by analyzing the DOM.

```
driver.find_element(By.NAME, 'example1')
```

As an illustration, selenium traverses the DOM looking for the DOM element `By.NAME` to find whether or not the element it is looking for exists. With this in mind, I am also able to `.send_keys()` to the input field that I am searching for. The `.send_keys()` option that selenium offers allows its users to send input into the desired field.

```
driver.find_element(By.NAME, 'example1').send_keys("example2")
```

It is important to note that in the code above that selenium is working in two steps. The first step is to find the element `By.NAME`, so it searches for the element `example1` in the DOM. If selenium finds the element, then it continues to the `.send_keys()` step. As soon as the driver finds the element it sends, in this case the string `"example2"` into the `example1` field that it found.

Code Example

With the help of these methods, I am able to browse around a webpage's DOM and look for particular HTML elements. The important thing to remember when discovering and locating mentioned items is that not everyone will name their contact forms using the default names, so the tests that are performed must be extensive to allow for the possibility that the element won't be located. For example, in Contact Form 7 labels for input forms come with default names.

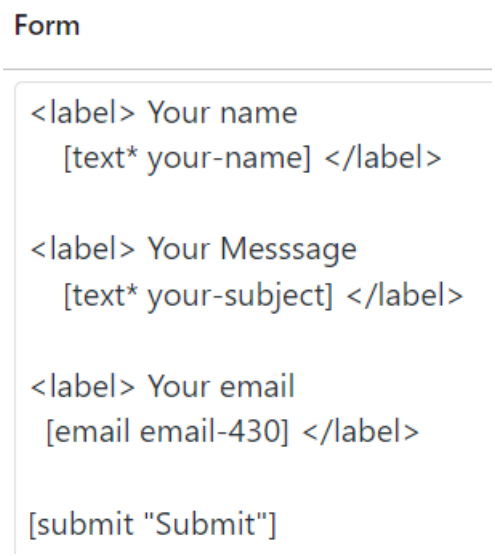


Figure 13: Contact Form 7 default screen.

Using the 'your-name' tag along with the HTML attribute it is provided, you may locate the "Your name" label in the DOM. The tests I've prepared

will feature a mix-and-match approach to try to account for any types of entries the website owner might have since the WordPress user can alter this form in a way that would benefit them.

Only two fields—name and email—could be required by the owner or the owner may have four input areas that ask for the user's name, message, email address, and phone number. Each of those potential changes will need to be taken into consideration in the tests I run.

In order to do that, Python has `try` and `except` blocks to take care of errors or in my case situations that may happen if a element is not found.

```
try:
  # the driver becomes initialized before this block
  driver.find_element(By.NAME, 'your-subject').send_keys("i.e.")
```

There are numerous `driver.find element` occurrences inside the try blocks to look for additional input fields. Selenium will continue looking for further potential items on the page if it is successful in locating the specified fields. If the aforementioned component is discovered, SeQure then looks for the `submit` button to determine whether the keys that were entered can actually be delivered.

```
button = driver.find_element(By.XPATH, "//input[@type='submit']")
```

The button uses the `XPATH` of the `find_element` to look for the button. This `find_element` differs from the others ones due to the fact Selenium could not find the button `By` other attributes. The next step is to click the button if it has been located. The user cannot manually push the submit button because the browser is automated. Once the element is found, Selenium can now click the submit button because it was able to locate the element.

```
driver.execute_script("arguments[0].click();", button)
```

Another feature of Selenium is that, with the use of the 'execute script' method, it can also carry out JavaScript instructions. The commands to be executed are passed as arguments to the method. The instruction to "click()" is delivered to the button field that Selenium had previously discovered. The submit button can now be activated by Selenium.

To determine if the message has been sent or not the user presses the submit button and a JavaScript message will appear shortly after clicking the button. The message below allows the user to know if their message was sent successfully or not.

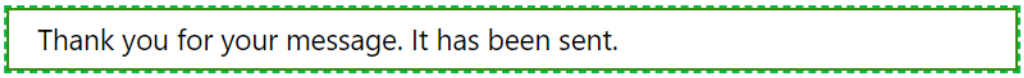


Figure 14: Contact Form 7 Successful Message.

Ethical Standards

The potential for this instrument to be used maliciously is something that it must ethically avoid in some ways. The code itself was incapable of causing harm in any way, shape, or form. The scripts that will be entered into these different contact forms, however, are. When used improperly or contrary to how it is intended, this technology has the potential to both be beneficial and problematic in terms of security.

Another ethical standard for this tool and method of approach would have to be using this tool as a "free to use" Python tool without a pay wall. When WordPress plugins only give partial security capabilities until users upgrade and pay for the premium version of the plugin, this can be problematic ethically. As a result, customers who cannot afford to purchase the plugin's premium version may become exposed to security threats. Additionally, it may put customers in a position where they must decide between purchasing the plugin's premium version and leaving their website open to attacks, which may be viewed as an unfair and exploitative tactic.

It's also critical to take into account the duty that plugin creators have to guarantee the security of their creations. If a plugin developer makes a product that is widely used but has security flaws, it is their duty to fix those flaws and guarantee that consumers are safe. Users' vulnerability may be exploited in an effort to make money rather than placing their security needs first by requiring them to purchase the premium edition of the plugin in order to access these security features.

In conclusion, it's critical that WordPress plugins prioritize user security requirements and guarantee that all users, regardless of ability to pay for a

premium version of the plugin, have access to the required security features. Failure to comply with this could be interpreted as an unethical activity that prioritizes profit before user protection.

Experiments

Experimental Design

My test configuration will consist of my tool and the form plugins Contact Form 7, Ninja Forms, and WPForms. These numerous contact forms have all been shown to have XSS attack-prone variations in the past. With this information, I will test this tool repeatedly using different scripts and payloads to see if the input fields these plugins give are appropriately sanitized or not.

Evaluation Metrics

Three key components will make up the evaluation metrics:

- Does the tool function as intended and successfully implement each payload?
- How many, if any, scripts are able to be sent back to the owner?
- If none of the scripts are able to be sent back to the owner, what does it mean?

These evaluation measures' primary criterion ensures that the tool itself and the code function as promised. If it fails, my tool could face serious ethical issues. The second statistic involves counting the total number of scripts that are successful compared to the total number of scripts that are unsuccessful. Continuing this point, it also has to do with why the scripts work, assuming that they do and the research behind why they do. For example, asking when scripting an XSS attack, are some characters more effective than others? The final metric relates to the project's research component. Does

the fact that none of the scripts function imply that the plugin code has not been properly sanitized?

After everything is said and done, a detailed analysis of escape characters will be performed, along with a comparison of how the various plugin versions responded to the payloads.

Testing

I'll be using 100 scripts altogether in a collection of XSS payloads to do these tests. Because the browser believes that the script is coming from a reliable source, it will grant access to any cookies, session tokens, or other sensitive data that the browser has saved and used with that website. These scripts are made up of many separate JavaScript codes. These scripts have the ability to completely change the content of HTML pages. These scripts also contain a variety of characters that are tested for input validation and sanitization within the scripts themselves. The scripts below serve as few examples:

```
"onclick=prompt(8)><svg/onload=prompt(8)>"@x.y
```

```
<svg onLoad svg onLoad="javascript:javascript:alert(1)"></svg onLoad>
```

```
`"'><img src='#\x27 onerror=javascript:alert(1)>
```

Now, these scripts that are seen above include symbols like `<`, `>`, `"`, `'`, `/`, and `@`. These characters are crucial because they are commonly used in XSS attacks frequently employ them in an effort to get around input validation and sanitization.

For instance, the single quotation character `'` is frequently used to end an attribute value in an HTML tag. An attacker can use a single quotation to escape the attribute value and insert their own code if they can successfully inject it into a website's input field. As a result, the attacker may be able to run arbitrary code on the website and steal user data or take over user sessions. Similar to how the single quotation can be used to insert malicious code into a website, the double quote character `"` can do the same. Additionally, it can be used to insert code and end an attribute value.

The beginning and conclusion of an HTML tag are indicated by the less-than `<` and greater-than `>` letters. An attacker can generate their own HTML

tags and inject their own code if they can insert a less-than character into a website's input field. Similar to using single and double quotes, this can let the attacker run any code they want on the page.

Last but not least, email addresses can be written in HTML code using the `@` character. This character could be used by a hacker to build a phony email address that appears to be real but is actually the attacker's email address. An attacker could, for instance, insert the following code into a text input field:

Because they can be exploited to insert malicious code into a website and provide an attacker the ability to run arbitrary code on the target website, these characters are regarded as "bad" for XSS. It is crucial to correctly sanitize input fields and encode user input to prevent the injection of these characters in order to prevent XSS attacks.

I will not be testing the power of the scripts but the sanitization of the input fields. If the fields are not correctly sanitized as they should be, then the scripts will have no problem getting sent to the website's owner and could present possible harm to the owner when they open up the email.

I'll be testing the following fields in my first test, as shown in the picture below.

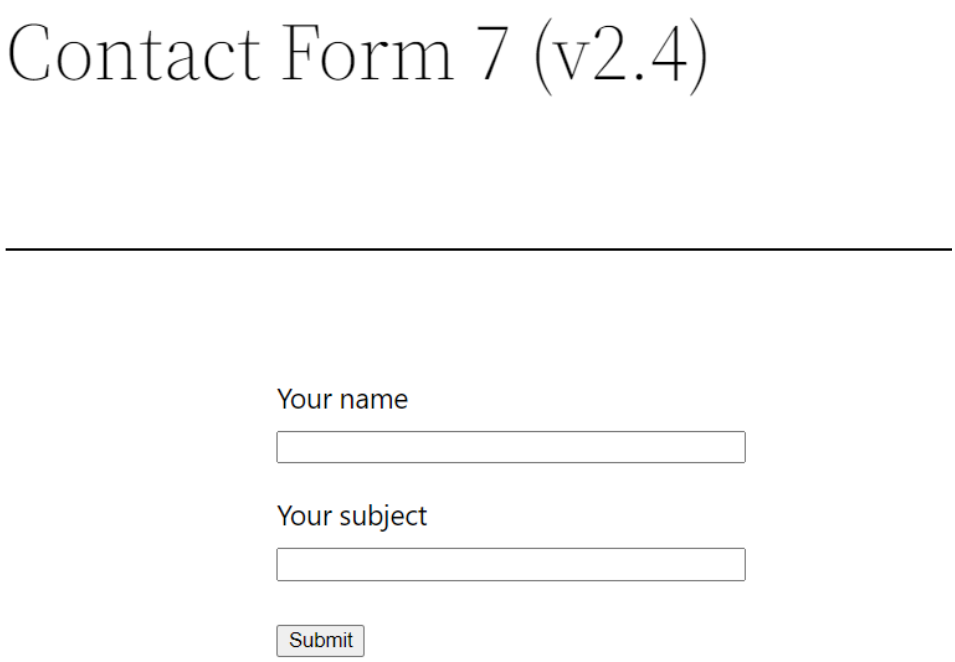


Figure 15: Experiment Base

I'll use a fictitious email account I've set up to test the scripts' functionality in getting sent back to the owner. This allows me to confirm whether the scripts are being sent or not by performing a secondary check. It is also good to note that since I only have to input fields to find utilizing Selenium to find these fields were made quite easy.

When deciding which versions to test, I concluded some outside research with known XSS vulnerable versions and wanted to add a more recent version of the plugin as well as an early version that also had known vulnerabilities.

The first plugin that will be tested is Contact Form 7 (CP7) and I will be testing versions:

- 2.4
- 4.0.1
- 5.0.1

With WPForms, I will be testing versions:

- 1.4.8
- 1.5.8.2
- 1.5.9

When it comes to Ninja Forms, I will be testing versions:

- 3.2
- 3.8
- 4.2

Results

Below are the results of nine total tests with 100 test scripts and three different versions per test.

Table 1: First tests:

Plugins	Version	Scripts that worked
Contact Form 7	2.4	0
WPForms	1.4.8	100
Ninja Forms	3.2	100

Table 2: Second tests

Plugins	Version	Scripts that worked
Contact Form 7	4.0.1	100
WPForms	1.5.8.2	100
Ninja Forms	3.8	100

Table 3: Final tests:

Plugins	Version	Scripts that worked
Contact Form 7	5.0.1	100
WPForms	1.5.9	100

Plugins	Version	Scripts that worked
Ninja Forms	4.2	100

All of the scripts passed the majority of the tests and were compatible with the majority of plugins and versions across the board. Each test was performed twice in order to confirm my findings and to ensure that all 100 scripts had been successfully executed when they had been.

We can see from the first table that Contact Form 7's initial test failed miserably since it did not allow for the sending of any emails. Due to an email configuration mishap none of the scripts were able to send an email resulting in the Selenium browser giving us a "Failed to send." notification by the plugin itself. I had assumed it had been because of certain email functionalities that version 2.4 of CF7 did not have and the later version did have. I compared and contrasted other email settings of the other versions I utilized and they were exactly the same. In the end, I had to infer that there was a problem within this old version's code that had not allowed emails to be sent to the owner whether it was a script or just plain text.

As we look at the scripts that had succeeded in being sent, a far bigger question arose. Why did every script get sent back to the owner?

Overall Analysis

Further investigation revealed that when sending an email through WordPress, the scripts are sent as PlainText, making it impossible for the owner to destroy the scripts by encoding them in any manner. PlainText is simply standard typed text that poses no threat to anyone, to put it simply. The words you are reading are an example of PlainText. Hence, yes, each script executed in turn, and there is no risk to the user in any way. This indicates that the specified input scripts are being successfully sanitized to prevent user harm.

Referring to my stated evaluation metrics, the tool did operate as expected and implement each payload in each field. The headless option was changed from `False` to `True` during the second round of testing in order to physically observe the payloads being implemented.

The second statistic indicated the amount of scripts that were able to be sent, if any were able too. As proved in the results section above, that all of the scripts did indeed function, albeit not exactly as I had anticipated. All

100 of the scripts in each test version (excluding CF7 version 2.4) were able to be sent directly to the owner.

Last but not least, only one version did not function; this is because, despite repeated attempts to try and run tests, there is a configuration mistake within that plugin that prevents me from sending emails with that version. The first test of Contact Form 7 is my outlier because no email was sent and no confirmation was received that it worked.

With the now known gap I had in my initial experimentation phase, all of the scripts should have worked since it poses no harm to the owner and the inputs that are being sent to the owner are sent in plain text. So, regarding to my last statistic, if none of the script are able to be sent back to the owner then there is an error on my end that has to deal with a configuration error or a plugin code error that I am not able to fix.

Threats to Validity

The degree to which an experiment or study accurately measures what it is intended to assess is known as validity, which is a key notion in research. Researchers must be mindful of potential validity threats that can have an impact on the precision and dependability of their findings when conducting an experiment. One such hazard is the failure of experiments, which can result in a variety of validity problems that affect the accuracy and applicability of the results.

The easiest way to comprehend risks to validity is that a hypothesis might be tested differently from how the researcher anticipated. This is distinct from the researcher not getting the outcome they expected. As a result of historical issues with XSS and plugins, my initial theory was that XSS scripts are bad for WordPress plugins since it is likely that they won't be properly sanitized and input validated. I had anticipated that some of the payloads would not function, and that this would be due to the positioning of some characters.

The possibility for confounding variables to be present in failed trials poses another danger to validity. Confounding variables are those that might affect the experiment's outcomes but are not taken into consideration in the design or analysis of the experiment. For instance, if a test of a novel teaching strategy does not yield the desired results, it may be because the study's design did not take into consideration the students' varied academic

backgrounds or preferred learning styles.

Confounding factors might be particularly difficult to find in failed trials since they might not be immediately obvious and could need more research. This can raise questions about the validity of the research findings since it may be difficult to determine whether a failed experiment was the result of poor experimental design or the presence of confounding variables.

Failed experiments might affect the research's external validity, which refers to how well the results can be applied to different populations or environments. If an experiment doesn't yield the desired outcomes, it could make people wonder if the conclusions can be applied to different situations or groups. For instance, it might not be obvious if a new intervention would be effective in other settings or populations if a trial testing its efficacy in one population did not yield the desired results.

The conclusion I came to is that when a plugin sends an email, it really sanitizes the input and modifies how it is sent to the website owner. I did not plan for the fact that the email will be sent in PlainText during this process. Although I didn't completely anticipate this result, this effort may also be useful in other ways that I haven't yet discovered. To add, failed experiments can impact pose a range of threats to validity that impact the reliability and generalizing of research findings. To address these threats, I should have been more vigilant in the experimental design, data collection, and analysis to consider the potential sources of error.

Reproducibility Details

These are the procedures for additional testing and how my results can be duplicated. For someone who is absolutely new to this procedure, becoming familiar with WordPress would be a fantastic detail to aid with this process. Also, in order to perform these automated tests, one must download their version of Chromedriver as well as Selenium in order to replicate the data I have (The code would have to change to the directory of where your Chromedriver is located).

1. Having a WordPress website

The process of replicating the aforementioned data begins with this. WordPress must be used to power the website, and form plugins must be supported. There is no need for additional adjustments.

2. Set up a temporary email address for the emails to be sent to

The user of this program can email the payloads to themselves to verify that they do, in fact, send to an email address by setting up a temporary email. This prevents the tool's user from receiving spam in their personal or employment email.

3. Download a certain version of a particular form plugin

Replicating the data above will require the user to have Contact Form 7, WPForms, and Ninja Forms to be downloaded onto the WordPress website. If further testing is warranted, then it would be a good idea to also download other form plugins other than the ones I used.

To add, within each plugins' settings, there is a specific setting for putting your fictitious email so that the payloads can actually be sent to you.

4. Have the same fields as the Experiments Base image above

- If you named the field differently, then the code would have to be changed slightly.

The fields that are provided in the tool are `Your name` and `Your subject`. Those are the two fields that I tested and to replicate the data above the fields should be exact or errors will occur.

5. Run the tool with whichever version and plugin that you have.

Lastly, would be running and testing the tool. With this steps, anyone should be able to replicate that data that I have provided from this experiment.

As this was the initial reproducibility list of things to do, these are a few extras that will have to taken into consideration when pushing the project and tool further:

6. Use a stable version of Chromedriver and Selenium

It is important to use a stable version of Chromedriver and Selenium in order to accurately replicate the results. The versions used during the initial experiment should be noted and matched in order to minimize any potential discrepancies in the results.

7. Ensure the website is up-to-date and secure

To minimize the possibility of other vulnerabilities affecting the experiment results, it is important to ensure that the WordPress website being used is up-to-date with the latest security patches and updates. This can help prevent any potential interference with the results from other security vulnerabilities.

8. Consider the limitations of the experiment

When conducting any experiment, it is important to consider the limitations of the methodology and the results. In this case, the experiment only tested a limited number of form plugins and only two specific input fields. The results may not be fully representative of all form plugins or input fields, and further testing may be necessary to fully understand the extent of the vulnerabilities.

9. Account for potential confounding variables

In any experiment, it is important to account for potential confounding variables that may affect the results. In this case, variables such as server configuration, network conditions, and other plugins installed on the website may affect the experiment results. It is important to keep these variables in mind when interpreting the results and to minimize their potential impact on the experiment.

10. Replicate the experiment multiple times

To ensure the validity and reliability of the results, it is important to replicate the experiment multiple times. This can help identify any inconsistencies or errors in the methodology or results, and can provide a more accurate understanding of the vulnerabilities present in the tested form plugins and input fields.

Although the aforementioned steps can help replicate an experiment's findings, it is vital to take into account any potential constraints, confounding variables, and other variables that might have an impact on the outcomes. Researchers can more properly understand the vulnerabilities contained in web input field form plugins and can build more efficient security methods to fight against these vulnerabilities by carefully accounting for these variables and repeating the experiment several times.

Conclusion

It is not unusual to run across failed experiments while employing XSS payloads to test WordPress form plugins. These unsuccessful experiments can be time-consuming, irritating, and even demoralizing. It is crucial to consider these failures as teaching moments and think back on what went wrong and how to make the experiment better.

The ethical ramifications of the studies must be taken into account in addition to the technical aspects of evaluating WordPress form plugins. Using XSS payloads in tests can have unfavorable effects, especially if the experiments are unsuccessful. This emphasizes how crucial it is to carefully organize and carry out tests in order to reduce potential hazards and guarantee ethical behavior.

Researchers should also think about how their trials might affect the greater WordPress community. False warnings from failed trials might cause unneeded stress and alarm among users. In order to effectively convey their findings and make suggestions for further research, scientists should emphasize both the triumphs and failures of their investigations.

The intricacy of WordPress form plugins is one of the main causes of failed trials in this situation. These plugins' various functionality, security features, and flaws can have an impact on the experiment's outcome. Furthermore, the complexity and potency of XSS payloads might vary, so what works on one plugin might not work on another. In order to assure compatibility and the likelihood that the intended outcome will be achieved, it is imperative to carefully choose the plugins and payloads.

Summary of Results

My investigation has shown that the WordPress contact form plugins Contact Form 7, WPForms, and Ninja Forms all have robust input validation and

sanitization methods in place to guard against cross-site scripting (XSS) attacks. The scripts are eliminated during the sanitization process when an attacker tries to inject a malicious script into the input field of these plugins, ensuring that the message sent to the website's owner is plain text rather than executable code.

This is an essential security feature since harmful scripts could be used by hackers to access important data without authorization, steal information, or even take over the website. However, the risk of XSS assaults is reduced and the website's security is improved by eliminating the scripts during the sanitization process.

Although these plugins have strong security safeguards in place, it is still crucial for website owners to maintain their plugins updated with the most recent security updates to make sure they are safeguarded against new security risks. To reduce the chance of security breaches, it's also essential to limit user privileges, use strong and unique passwords, and frequently backup website data.

I can say with certainty that the plugins for Contact Form 7, WPForms, and Ninja Forms are safe and effective at thwarting XSS assaults. To ensure the protection of sensitive data and information, it is crucial to keep in mind that website security is a constant process that necessitates monitoring and continual work.

WordPress Plugin and Email Security

Because WordPress is such a popular platform, having a strong and secure base is essential to preventing assaults. The first line of security against such assaults like cross-site scripting is having a secure platform. On top of this base, plugins can add security to the working environment they provide for users. WordPress offers a number of security precautions to guarantee the safety of a user's website.

Users can check the site health of their website by clicking on a link on the WordPress administrator's main page dashboard, for instance. This function looks for any design flaws and proposes ways to strengthen them, like updating a plugin or removing an inactive theme. The site's overall performance, user experience, and security will all benefit from these updates even though they may not be as urgent as a serious problem. Additionally, this functionality allows visitors to observe both successful and unsuccessful site tests.

WordPress offers a variety of security plugins in addition to the website testing function to assure the highest safety of its customers. The website is further protected by security plugins including Wordfence Security, Defender, iThemes Security, Jetpack, Security Ninja, and All In One WP Security and Firewall.

There is no need for sanitization in terms of email security because hazardous script tags like `<script>` will be displayed in plain-text emails that contain HTML information. To stop attacks like phishing or spoofing, the email system of the website must still be secured. Using a trustworthy email service provider with features like spam filtering, virus scanning, and two-factor authentication is one method to achieve this.

The security of the websites run by its users is generally ensured by a variety of security features and plugins offered by WordPress. Users can reduce the risk of attacks and keep their websites secure by adhering to suggested security procedures and keeping the platform current.

Future Work

This project's future work has a lot of potential directions. For instance, there are numerous different input fields that plugins may have, and it is possible to find and investigate what other types of contact form vulnerabilities exist. Malicious files may they be uploaded? The user can upload whatever file they want via a variety of contact forms. Are these contact forms susceptible to SQL injections (Structure Query Language) instead of cross-site scripting? Another direction this project can take is for the plugins to be able to supply a URL field as an additional field. Can URLs contain embedded scripts? If so, then a weakness in plugins known as a phishing vulnerability becomes available.

It is essential to consider the continuous evolution of technology and the new methods hackers are using to breach websites. Therefore, researchers can focus on exploring the possibility of new attack vectors and testing WordPress form plugins against these new attack vectors. Additionally, researchers can evaluate the effectiveness of existing security plugins in detecting and preventing these attacks.

Another potential direction this project could take is the development of new security measures to enhance the security of contact forms in WordPress. This could involve integrating advanced encryption mechanisms or

implementing machine learning algorithms to detect and block suspicious activity. Moreover, the project could explore the potential of blockchain technology in securing contact forms in WordPress.

Furthermore, it is important to consider the impact of third-party integrations on the security of WordPress contact forms. As many plugins rely on third-party APIs and services, it is important to evaluate the security of these integrations and ensure that they do not introduce new vulnerabilities into the system. Researchers can investigate the security practices of these third-party services and evaluate their effectiveness in preventing attacks.

Moreover, there is a need for more comprehensive user training on WordPress security practices. Many users may not be aware of the security risks associated with contact forms and the steps they can take to mitigate these risks. Therefore, researchers can develop educational materials, such as guides or tutorials, to help users understand the importance of contact form security and how to implement best practices to protect their websites.

The future work of this project has various potential directions that researchers can explore to enhance the security of contact forms in WordPress. The evolution of technology and the increasing sophistication of cyber attacks demand continuous efforts to identify new vulnerabilities and develop effective security measures. By evaluating the effectiveness of existing security plugins, exploring new attack vectors, and developing educational materials for users, researchers can contribute to improving the security of WordPress contact forms and preventing cyber attacks.

In the final analysis, while working with WordPress and its plugins, I noticed that the theme I had originally downloaded contains a page that users of my website can post to. A simple message is shown below.

Figure 16: Trial Post 1

Above shows my first message that I was able to post onto my site for everyone to see. I was able to post other comment like the one above as well.

Being curious with my current project and what it encompasses, I wanted to try out a simple script. I entered the script:

```
<script>alert("This script reflects!")</script>
```

When I posted the scripted comment, the image below popped up as an alert on my screen.

Figure 17: Scripted Comment

I pressed the `Ok` option to refresh the page to see if the script was still stored and when the page refreshed the image above popped up on my screen yet again. Now, there is a stored script on my website so, whenever a user

loads onto my comment page that script will run. Toying around with WordPress, I was able to find a part of my website that was vulnerable to XSS. The comment field on my current theme did not sanitize my input which allowed the script to work and be stored inside of the database. To add, my website is currently public so, during this journey with plugins other users were able to find my test website and comment root scripts and other links that I did not perceive to be trustworthy. With this kind of information, future work with this tool would be checking the sanitization of different WordPress provided themes.

To push that work another step, one could collect data on what kinds of scripts work versus which ones do not work. Do the payloads I have provided for this tool all work? If they do, then what measures can be taken to prevent this? If they do not, what are the differences between the scripts that do work and do not?

With all of these kinds of options as potential avenues for this work, future work for this tool and project has many different paths it could go down.

Project Impact

When testing WordPress Form plugins using XSS payloads, the effects of failed experiments might be felt by both the myself, the researcher, and the larger WordPress community. Failed experiments can, despite the possible drawbacks, benefit the larger WordPress community. Researchers can help to improve their approach by highlighting what doesn't work, which could result in more successful testing and eventually greater security for WordPress users.

Since my experiments did not turn out as I had hoped, the knowledge that they were unsuccessful has enabled me to look into alternative directions for future research on this project. It has also been helpful to know that the contact form plugins on my website do not accept cross-site scripting to function through sending emails. Yeah, my initial goal was to test these plugins to determine whether they had any flaws or whether I could insert scripts into email threads. Yet the outcome was different. The plugin does not allow XSS scripts to be transmitted as emails, which is the result that needs to be highlighted.

Researchers can stop others from making the same errors and maybe

squandering resources by recording failed experiments and sharing them with the WordPress community. The community may be able to better focus their efforts on regions that are more likely to yield results as a result, eventually saving time and effort.

It is important to remember that unsuccessful studies can also provide the researchers with worthwhile training opportunities. They can enhance their experimental design, discover potential sources of error, and fine-tune their methods by investigating why an experiment failed. Future testing may become more exact and thorough as a result of this.

Critical Reflection

Lack of sufficient testing and planning is the major factor in my failed trials. To find any potential problems or conflicts, it is crucial to thoroughly test the plugins and payloads before running the experiment. Furthermore, it is critical to comprehend the experiment's goals, methods, and anticipated results. Since these results were not anticipated it ultimately led to the failure of my experiments. Better and more careful planning can reduce the possibility of unsuccessful trials while ensuring that the experiment is focused and successful.

Also, it is critical to view failed trials with optimism. It is best to consider them as chances for growth and learning rather than as setbacks or failures. We can pinpoint areas for improvement and create more effective plans for subsequent experiments by investigating the causes of the failure. This strategy can advance our expertise and understanding in this area and, as a result, raise the likelihood that our tests will be successful.

The use of XSS payloads in experiments on WordPress form plugins inevitably resulted in failed tests. Even though they could be upsetting, it's crucial to look at them as learning opportunities and consider what went wrong and how to get better. We may improve our odds of success and become more efficient in this field by choosing appropriate plugins and payloads, thoroughly testing and preparing for the experiment, and maintaining a positive attitude toward failure. Realizing where I went wrong in this project has allowed me to look at other paths I could have taken and continue this project in the future.

Future Ethical Implications and Recommendations

The ethical ramifications of performing security analyses on WordPress Form plugins must be carefully taken into account. Although such testing is required to guarantee the security and protection of users, it can also present some hazards and raise ethical questions, particularly when trials go wrong.

The possibility for harm to users or website owners is one ethical problem. Failures in tests could expose websites to threats or harm users who depend on the security of these plugins. Researchers should make sure that their testing is controlled, ethical, and uses the proper precautions to prevent danger or damage in order to reduce this risk.

The possibility of unexpected repercussions is another another ethical worry. Even if the researcher had the best of intentions, the outcomes of these trials could be twisted or used against them. Researchers should be open and honest about their research's methodology, findings, and any possible risks in order to prevent such situations.

It is advised that researchers adhere to established practices for ethical research, such as gaining participants' informed consent and reducing risks and harm, to address these ethical considerations. Researchers should also think about different approaches that do not use potentially dangerous payloads for testing the security of WordPress Form plugins.

While assessing the security of WordPress Form plugins is crucial, it is equally important to think about the moral ramifications of such testing and take the required precautions to reduce any dangers to users and website owners.

False Sense of Security:

Users who assume they are secure because a specific vulnerability has been neglected or disregarded because of a failed experiment are said to have a false sense of security. As a result, there could be dangerous repercussions because attackers could take advantage of this weakness to access confidential data without authorization.

Furthermore, it's possible for security researchers to falsely identify a hole that doesn't exist, which might cause unneeded confusion and concern. Users might therefore take time- and money-consuming and needless measures to

secure their systems as a result.

Security researchers must carry out their studies in a safe and controlled setting, such as a sandbox or a test server, to prevent false positives and false negatives. When performing their research, they should also adhere to all applicable laws, regulations, and ethical standards. They can verify their findings are correct and that users can take the necessary steps to secure their systems by doing this.

Failed experiments have an ethical implication in that they can result in false negatives, in which vulnerabilities are overlooked because the studies did not yield the intended results. This puts users at danger and may give them a false sense of security. Failed trials may also provide false positives, in which security researchers mistakenly detect a flaw that does not exist, causing needless commotion and confusion. To lessen the possibility of unintended outcomes, it is crucial to carry out experiments in a secure and controlled environment, like a sandbox or test server. Also, scientists must make sure that all applicable laws, rules, and moral principles are followed in their research.

User Security:

Users may be less likely to take further security precautions to safeguard themselves against potential dangers if they believe that their website is safe against XSS attacks because they have installed particular plugins or have taken other security measures. False confidence can lead to complacency and make users less diligent in maintaining the security of their website, leaving them more open to future attacks.

Furthermore, users who find that researchers are testing XSS payloads on WordPress Form plugins could get perplexed or unsure about the security of their website. They could be concerned about whether the plugins they use are secure and whether they might be unwittingly exposed to security flaws. This lack of knowledge may cause undue worry or overreaction, both of which could be harmful in and of itself. Users might expend time and money trying to address fictitious security concerns, which could divert them from more pressing problems.

So it's essential to make sure consumers are aware of the goals and constraints of security testing. Users should be made aware of the advantages of security testing, which can help in locating and fixing security flaws. The accuracy of the results may be hampered by false positives and false nega-

tives that can come from testing. A false sense of security can be lessened by informing users of the value of a multi-layered security approach, which includes using trusted plugins, keeping software up to date, and following best practices.

Reputation Damage

A false sense of security brought on by unsuccessful experiments might have major repercussions for the researcher as well as the larger community. It can result in harm to one's reputation and a loss of credibility, which is something the scientific community is quite concerned about. Researchers run the risk of facing criticism for shoddy procedures or time and resource waste if they publish the results of unsuccessful experiments without describing the restrictions and the causes of the failure. This may harm their reputation and have an impact on their upcoming research plans.

Additionally, if the results are presented without the appropriate context, they can give WordPress or other platform users a false sense of security. Users may disregard the need for website security measures because they think the platform is totally secure, which can result in data theft and security breaches. It is crucial to acknowledge the constraints and shortcomings of experiments since this knowledge can result in better research techniques and, therefore, better community security measures.

It is vital to take into account alternative techniques and directions for future study in order to avoid the dangers of a false sense of security. For instance, to learn why a specific script functioned, the researcher could investigate WordPress themes and run trials. This strategy might help researchers better understand how WordPress themes are vulnerable, which would enhance users' security overall.

References

[1] Dom based XSS. *DOM Based XSS | OWASP Foundation*. Retrieved from https://owasp.org/www-community/attacks/DOM_Based_XSS#:~:text=DOM%20Based%20XSS%20(or%20as,in%20an%20%E2%80%9Cunexpected%E2%80%9D%20manner.

[2] HTML tag. *HTML img tag*. Retrieved from https://www.w3schools.com/tags/tag_img.asp

[3] WordPress plugin contact form 7 cross-site scripting (4.0.1). *Acunetix*. Retrieved from https://www.acunetix.com/vulnerabilities/web/wordpress-plugin-contact-form-7-cross-site-scripting-4-0-1/#:~:text=WordPress%20Plugin%20Contact%20Form%207%20is%20prone%20to%20a%20cross,context%20of%20the%20affected%20site.

[4] Understanding vulnerabilities. *NCSC*. Retrieved from https://www.ncsc.gov.uk/information/understanding-vulnerabilities#:~:text=A%20vulnerability%20is%20a%20weakness,to%20achieve%20their%20end%20goal

[5] Basics of computer science - threat. *Tutorials Point*. Retrieved from https://www.tutorialspoint.com/basics_of_computer_science/basics_of_computer_science_threat.htm

[6] Nuno Antunes and Marco Vieira. 2012. Defending against web application vulnerabilities. *Computer* 45, 02 (2012), 66–72.

[7] Jasmin A Caliwag, Roxanne A Pagaduan, Reynaldo E Castillo, and Walter Van J Ramos. 2019. Integrating the escaping technique in preventing cross site scripting in an online inventory system. In *Proceedings of the 2nd international conference on information science and systems*, 110–114.

[8] Admir Dizdar. 2023. XSS attack: 3 real life attacks and code examples. *Bright Security*. Retrieved from https://brightsec.com/blog/xss-attack/

[9] WordPress Security Expert. 2022. WordPress XSS attack (cross site scripting) - how to prevent? *WP Hacked Help Blog - WordPress Security Knowledgebase*. Retrieved from https://secure.wphackedhelp.com/blog/wordpress-xss-attack/

[10] Anna Fitzgerald. 2022. 7 WordPress plugins with recent vulnerability issues. *HubSpot Blog*. Retrieved from https://blog.hubspot.com/website/vulnerable-wordpress-plugins#:~:text=According%20to%20data%20from%20Risk,rose%20by%20142%25%20in%202021

[11] Oslien Mesa, Reginaldo Vieira, Marx Viana, Vinicius HS Durelli, Elder Cirilo, Marcos Kalinowski, and Carlos Lucena. 2018. Understanding vulnerabilities in plugin-based web systems: An exploratory study of wordpress. In *Proceedings of the 22nd international systems and software product line conference-volume 1*, 149–159.

[12] Dimitris Mitropoulos, Panos Louridas, Michalis Polychronakis, and Angelos Dennis Keromytis. 2017. Defending against web application attacks: Approaches, challenges and implications. *IEEE Transactions on Dependable and Secure Computing* 16, 2 (2017), 188–203.

[13] Takayuki Miyoshi. 2023. Contact form 7. *WordPress.org*. Retrieved from https://wordpress.org/plugins/contact-form-7/

[14] Daniel T Murphy, Minhaz F Zibran, and Farjana Z Eishita. 2021. Plugins to detect vulnerable plugins: An empirical assessment of the security scanner plugins for wordpress. In *2021 IEEE/ACIS 19th international conference on software engineering research, management and applications (SERA)*, IEEE, 39–44.

[15] Robert Seacord. Input validation and data sanitization. *Input Validation and Data Sanitization - SEI CERT Oracle Coding Standard for Java - Confluence*. Retrieved from https://wiki.sei.cmu.edu/confluence/display/java/Input+Validation+and+Data+Sanitization#:~:text=Sanitization%20may%20include%20the%20elimination,trust%20boundary%20(output%20sanitization).

[16] Melissa Smith. 2021. Website security: What updates should you be making to your WordPress site - and how often? *IMPACT Inbound Marketing Agency*. Retrieved from https://www.impactplus.com/blog/security-updates-wordpress-site#:~:text=Major%20upgrades%20usually%20happen%20two,automatically%20update%20your%20WordPress%20core.