URL Redirection

Allison Okomski and Khylei Neal

Secure Coding Final Project

Abstract

URL redirection is a technique used to redirect users from one website to another. Attackers can exploit web application vulnerabilities by injecting a script to redirect the user to a malicious site. If the application fails to validate input from untrusted sources, an attacker is capable of inputting a URL to redirect traffic from the trusted domain to a malicious page. These attacks can deceitfully redirect users to fraudulent websites to perform phishing and social engineering attacks. Cross-Site Scripting (XSS) is a major security threat to web applications and a common method to perform URL redirection. This study identifies effective and ineffective approaches to structuring and writing code for applications to minimize the security risks of insecure input handling. This project aims to showcase JavaScript-based URL redirection on a shopping website and propose measures to mitigate vulnerabilities associated with it.

Introduction

URL redirection is a critical component of web development, facilitating seamless navigation between web pages. However, it also presents security challenges, as malicious actors exploit vulnerabilities to redirect users to fraudulent websites and compromise personal information. Our research delved into various software tools and frameworks, seeking to create web applications with input fields lacking adequate input handling.

Our journey began with the construction of a shopping web application to demonstrate the concept and risks of URL redirection via JavaScript injection. Due to vulnerabilities in the code, our application was susceptible to JavaScript injection and potential XSS vulnerabilities. As we navigated through our experimentation phase with HTML, JavaScript, Notepad++, and Flask, we gained firsthand insight into the importance of

secure coding practices and the central role of understanding the tools and web frameworks you are using to help mitigate and plan for vulnerabilities.

Through our experiences, we emphasize the need for developers to prioritize security in all aspects of web development. By implementing effective input handling techniques and embracing secure coding practices, developers can safeguard user trust and thwart malicious activities. In our subsequent discussions, we will share our experiences, detailing both our failures and successes, and advocating for a holistic approach to web development security.

Background

Index.html:

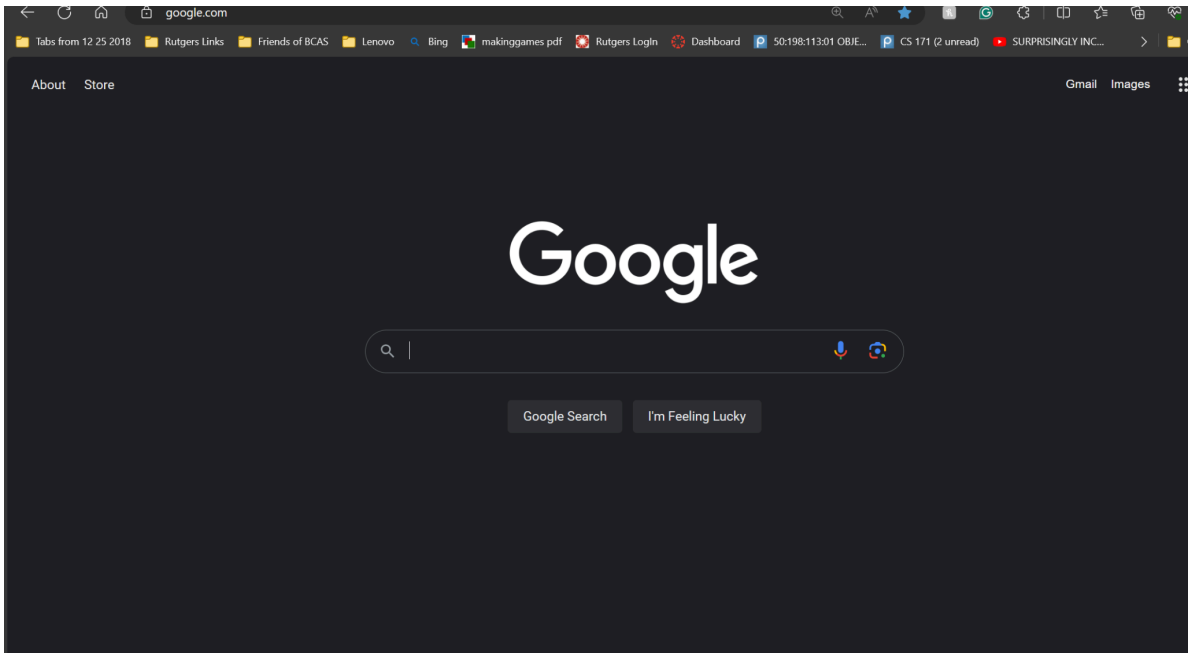
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF=8">
    <title>Shopping Website</title>
</head>
<body>
    <h1>Shopping Website</h1>
    <img src="tshirt.jpg" width="250" height="200">
    <h2>T-Shirt: $20</h2>
    <p>Quantity:</p>
    <input type="text" id="js-command">
    <button onclick="executeCommand()">Checkout</button>

    <script>
        function executeCommand() {
            var command = document.getElementById('js-command').value;
            eval(command); //vulnerable
        }
    </script>
</body>
</html>
```

Implementation of our Shopping Website:

We constructed a shopping web application to demonstrate the concept and risks of URL redirection via JavaScript injection. We created an “index.html” file that consists of a simple clothing website. The website displays a T-shirt product, a quantity field, and a checkout button that should bring users to the checkout page. Due to the use of the eval() function in our code, the quantity input field is susceptible to JavaScript code. When the “Checkout” button is selected, the eval() function “takes a string as input and executes it as JavaScript code” (Fluid Attacks), which can result in XSS vulnerabilities and URL redirection attacks. As a result of this poorly written code, the attacker can send a malicious request to be executed on the server. Due to the lack of sanitation for the user input, we are able to inject the following JavaScript code: “window.location.href='https://google.com';”.

Quantity:

window.location.href='https: Checkout



In this example, we are redirecting the page to google.com, but this XSS attack can redirect the page to any URL. The attacker can redirect users to a fabricated checkout page mirroring the authentic website and prompt users to enter their credentials and gain sensitive data. A typical checkout page requests the user's full name, address, phone number, email address, and credit card information. If an attacker successfully redirects the page and obtains the user's personal information, it can be extremely dangerous. This type of information is very sensitive as it can be used for identiy theft, financial fraud, and targeted attacks. In addition, URL redirection can be hard to detect because it "effectively bypasses traditional security measures that scan for known malicious URLS, as the initial link appears safe and originates from a trusted source" (Adriano). It is vital to ensure that your web application is not at risk of this harmful and stealthy attack on customers. From a business standpoint, URL redirection can result in a significant loss of income and trust from customers.

There are several different approaches to overcoming URL redirection attacks. It is essential for web applications to have as many safeguards in place to patch up any potential weaknesses. In our vulnerable shopping website example, the most effective way to diminish this issue is to “avoid using the eval function with user input as it can execute arbitrary code ” (Fluid Attacks). In addition, it is crucial to validate and sanitize all user input to protect against injection attacks and mitigate security risks. By filtering and assessing input, developers can build more secure and robust applications to defend against security threats and ensure positive user experience. Instead of using the hazardous function eval(), your code should use a parser to determine a string’s structure and ultimately extract the relevant information. Using a parser would break down the input and help identify “patterns, abnormalities, or potential threats within the input data” (KeepSolid). Using a parser in conjunction with validating user input will mitigate the risk of URL redirection and other malicious attacks greatly. Companies can also implement a whitelist approach to “only allow specific safe operations or functions” (Fluid Attacks) to be run on the system. Whitelisting is an additional security measure that can be utilized to further secure a web application. Each of these methods is one step closer to securing a web application and protecting the customers that browse the site.

Experiments

For our experiment, we attempted to use Flask, a built in web framework for Python, to exploit URL redirection within our web application. Flask was extremely simple and flexible, making it a good choice for us to develop our site successfully. However, as we started integrating Flask into our project and attempted to implement the URL redirection within the input field, we encountered an unexpected hurdle. Flask's security features automatically performed input checking and validation, which prevented us from implementing our project successfully. We had to do some research and debugging to finally figure out why our site would not redirect properly. While these security measures were a roadblock for us protecting our web application from common vulnerabilities, such as cross-site scripting (XSS) attacks and injection exploits, we learned a ton about our project and the direction we needed to go to complete it and some different frameworks that web developing tools utilize. Despite our efforts to bypass these security checks, Flask's inherent protections against malicious input ultimately restricted our ability to perform URL redirection within the framework. It is important to note that when implementing any web development project that you research the restrictions and safety protocols in place as it will allow for you to plan

accordingly in regards to how much or how little security is needed for the implementation.

Results

New and improved index.html file:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Shopping Website</title>
</head>
<body>
    <h1>Shopping Website</h1>
    <img src="tshirt.jpg" width="250" height="200">
    <h2>T-Shirt: $20</h2>
    <p>Quantity:</p>
    <!-- on input event listener to validate users input -->
    <input type="text" id="js-command" oninput="validateInput(this)">
    <button onclick="executeCommand()">Checkout</button>

    <script>
    // JavaScript functions
        function executeCommand() {
            var quantity = document.getElementById('js-command').value;
            // Logs to help discover odd behaviors or input
            console.log("Quantity:", quantity);
        }

        function validateInput(input) {
            var value = input.value;
            // Only accept integer values - parse characters using regex to convert to Int
            // replace non-digit chars ('\D') with an empty string
            var intValue = parseInt(value.replace(/\D/g, ''));
            // Update the input value with the sanitized integer value
            input.value = intValue || '';
        }
    </script>
</body>
</html>
```

In the new version of our website, we completely removed the use of the eval function and included validation input to minimize the risk of attack. The validateInput function was adjusted to ensure that only integer values are entered into the quantity field. The new version of this code utilizes the replace method with the regular expression (/\D/g, ‘ ‘) to remove all occurrences of non-digit characters. Then, the sanitized string is parsed

into an integer using the parseInt function. These implementations do not allow the user to enter any strings or scripts. In this case, we used regex and parseInt to block all inputs aside from integers, because an integer is the only necessary input for the quantity field. If an application needs the user to enter a string into the input field, the same practices of sanitization and parsing can be used to help prevent script injection and malicious strings.

Conclusion

Our exploration of URL redirection highlights the need for robust security practices in web development. Input handling is often overlooked when allowing user input and it often leaves a back door for bad actors to exploit and manipulate the code on the back end. As demonstrated in our revised website, leveraging techniques like regex and parsing tools for input sanitization can effectively thwart potential exploits. Overall, we found that a lot of web frameworks prioritize security to safeguard user trust and mitigate the impacts of malicious activities, and would not allow us to inject java script into the code. We researched a number of different web frameworks to find which frameworks had this built in security and which didn't for the purpose of our project. Overall, we found that even though some softwares and frameworks have built in security, it is always important to code with security in mind. We always want to view security as priority rather than just an added feature.

Works Cited

Adriano, Kevin. “Trusted Domain, Hidden Danger: Deceptive URL Redirections in Email Phishing Attacks.” *Leading Managed Detection and Response*, Trustwave Holdings, Inc., 29 Jan. 2024, www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/trusted-domain-hidden-danger-deceptive-url-redirections-in-email-phishing-attacks/#:~:text=Redirection%20effectively%20bypasses%20traditional%20security,to%20track%20the%20destination%20URL. Accessed 29 Apr. 2024.

"Flask Documentation." Flask.palletsprojects.com, Flask Documentation, flask.palletsprojects.com/.

"Flask Security Best Practices." Real Python, realpython.com/flask-security-best-practices/.

“Inappropriate Coding Practices - Eval Function: Fluid Attacks Documentation.” Fluid Attacks *In Java*, 18 Sept. 2023, docs.fluidattacks.com/criteria/fixes/java/143/#:~:text=The%20vulnerability%20lies%20in%20the,without%20any%20validation%20or%20sanitization. Accessed 29 Apr. 2024.

“Parser.” *What Is Parser - Cybersecurity Terms and Definitions*, KeepSolid Inc, www.vpnunlimited.com/help/cybersecurity/parser. Accessed 29 Apr. 2024.