

# **Sri Lanka Institute of Information Technology**



## **BSc. Honors in Information Technology Specialize in Cyber Security**

**Web Security – IE2062**

**February 2023**

**Individual Assignment**

**Whole Document**

Submitted by:

IT21226496

Gunasekara W.M.M

## Contents

<b>Introduction .....</b>	<b>3</b>
<b>OWASP Top 10 Vulnerabilities .....</b>	<b>5</b>
01 Injection .....	5
02 Broken Authentication .....	6
03 Sensitive Data Exposure .....	7
04 XML External Entities (XXE): .....	8
05 Broken Access Control: .....	9
06 Security Misconfiguration: .....	10
07 Cross-Site Scripting (XSS): .....	12
08 Insecure Deserialization .....	13
09 Using Components with Known Vulnerabilities: .....	14
10 Insufficient Logging and Monitoring: .....	16
<b>Bug bounty report making .....</b>	<b>17</b>
<b>Bug Bounty Reports .....</b>	<b>21</b>
Report 01- Outdated Version of jQuery Validation Plugin	<b>HIGH</b> ..... 21
Report 02-: Weak Ciphers Enabled in SSL Communication	<b>Medium</b> ..... 23
Report 03- HTTP Strict Transport Security (HSTS) Errors and Warnings	<b>Medium</b> ..... 24
Report 04- SSL/TLS Not Implemented	<b>Medium</b> ..... 26
Report 04-Password Transmitted over Query String	<b>Medium</b> ..... 27
Report 05-Phishing by Navigating Browser Tabs	<b>Low</b> ..... 28
Report 06-Cookie Not Marked as HTTP Only	<b>Low</b> ..... 30
Report 07- BREACH Attack Detected	<b>Medium</b> ..... 32
Report 08- Misconfigured Access-Control-Allow-Origin Header	<b>Low</b> ..... 34
Report 09- Cross-site Request Forgery (CSRF)	<b>Low</b> ..... 36
Report 10- Insecure Transportation Security Protocol - Support for TLS 1.0	<b>Low</b> ..... 38
<b>Challengers faced. ....</b>	<b>41</b>
<b>Conclusion.....</b>	<b>41</b>
<b>References .....</b>	<b>42</b>

## Introduction

A vulnerability rewards program (VRP), also referred to as a bug bounty program, is a crowdsourcing project that pays people for finding and reporting software defects. In addition to internal code audits and penetration testing, bug bounty programs are frequently started as a component of a company's vulnerability management approach.

Numerous software firms and websites conduct bug bounty programs, rewarding white hat hackers and software security researchers with financial rewards for reporting software flaws that might be exploited. For the company giving the prize to be able to replicate the vulnerability, bug reports must contain sufficient information. Payment amounts are often proportionate to the size of the company, the complexity of hacking the system, and the potential impact a defect could have on consumers.

Mozilla offered a \$3,000 flat-rate prize for bugs that met its standards, but Facebook has offered up to \$20,000 for a single bug report. Google paid \$700,000 to Chrome operating system bug reports in 2012, and Microsoft compensated \$100,000 to UK researcher James Forshaw for an attack vulnerability in Windows 8.1. Apple offered rewards in 2016 that may reach \$200,000 for a vulnerability in the iOS secure boot firmware components and up to \$50,000 for the execution of arbitrary code with kernel privileges or unauthorized iCloud access.

While ethical hacking may be a highly efficient way to uncover problems, some people have mixed feelings about these kinds of programs. Some businesses provide invitation-only, closed bug bounty programs to reduce possible risk. Apple, for example, has limited bug reward participation to a few hundred researchers.[1]

### Benefits for Ethical Hackers

A bug security bounty program benefits both ethical hackers, otherwise called white-hat hackers, and the organization that runs the program. These are some benefits for ethical hackers:

- In a bug security bounty program, skilled ethical hackers from a variety of backgrounds actively labor for corporations to pinpoint risks, flaws, and vulnerabilities that can be fixed.
- Ethical hackers receive compensation when they alert developers to a vulnerability.
- Hackers from all over the world are occasionally engaged by different companies to monitor bugs and report vulnerabilities in exchange for full-time pay.
- Hackers don't follow a set checklist; instead, they must examine the most recent and unpredictable hacks employed by online criminals, which forces them to become ever-more inventive.

### Benefits for the Organization

Now, here we discuss the benefits for the organization operating the bug security bounty program:

### **Improved Vulnerability Detection**

A bug security bounty program's primary advantage is that it helps a business identify and patch various application vulnerabilities. With a bug security reward program, a company has a better chance of recognizing flaws before they are used in attacks, protecting its brand and reducing the chances of high-value security breaches.

### **Realistic Threat Simulation**

With the use of a bug security reward program, a business pays bug trackers to simulate cyberattacks. They essentially have access to the same systems and knowledge of the organization. It suggests that vulnerability evaluations undertaken by bug security bounty trackers will be more feasible and realistic than a more coordinated engagement.

### **More Prominent Access to Talent**

Bug security bounty schemes can provide businesses with access to talent that would be hard to find and keep on staff. In comparison to what would be possible with a typical pen-test or vulnerability scan, a company may conduct vulnerability testing by more bug trackers with a wider range of skills and abilities.

### **Reduced Cost**

Paying a reward to find a vulnerability is unquestionably far less expensive than fixing a security event brought on by the same exposure. Surprisingly, even though prize prices might vary, the most expensive bounties are still less expensive than a data breach. An company can save money by only paying bug bounty trackers when they find something, which is another cost-saving measure. Additionally, it is still less expensive than paying for a comparable degree of cybersecurity testing through contractors who are paid by the hour regardless of what they find, or through internal cybersecurity testing.[2]

# OWASP Top 10 Vulnerabilities

## 01 Injection

Injection vulnerabilities can occur in a variety of situations, including operating system instructions, XML parsers, SQL, OS, LDAP, or NoSQL database queries. By introducing malicious code or unexpected data, attackers might take advantage of these weaknesses and change the way the program behaves.

Successful injection attacks can have serious repercussions, such as unauthorized data access, data modification, remote code execution, or even a system breach.

To prevent injection vulnerabilities, you should know to secure coding practices, such as:

1. 1. Validate all input obtained from users or outside sources. Use stringent whitelisting or pattern matching to make sure that only intended input is received.
  2. 2. Use parameterized or prepared statements with bound variables when creating database queries. This strategy helps to keep data apart from instructions or queries and stops attackers from inserting dangerous SQL code.
  3. 3. The concept of least privilege states that system or database user accounts should have the fewest privileges required to carry out their duties. Avoid doing regular tasks using privileged accounts.
  4. 4. User Input and Output Escaping: Before utilizing user input in command or query contexts, properly encode it. Additionally, when showing output to people, escape it to avoid accidental interpretation as code.
5. Secure Development Frameworks and Libraries: Make use of secure development frameworks and libraries that come with built-in defenses against injection attacks. Automatic input validation and query parameterization are two capabilities that these technologies frequently offer.
6. Consistent Patching and Updates: Make sure to apply the most recent security updates to all software libraries, frameworks, and components of your application. This procedure aids in reducing known vulnerabilities that could be used against you.
7. Security testing: Conduct routine security testing, including as penetration testing and vulnerability assessments, to find and fix injection vulnerabilities. Static code analysis software, dynamic application scanners, and human code reviews are a few tools that might be useful in this process.

You may dramatically lower the risk of injection vulnerabilities in your web apps by adhering to these procedures and remaining current with OWASP's most recent security recommendations.[3]

## 02 Broken Authentication

Another vulnerability in the OWASP Top 10 referred to as "Broken Authentication" concerns holes in the authentication and session management features of online applications. It happens when an application's authentication protocols are built improperly, enabling attackers to steal session tokens, user credentials, or even circumvent authentication entirely.

Here are some common causes and mitigation techniques for broken authentication vulnerabilities:

1. **Weak Credentials and Password Policies:** Set in place strong password regulations which require a minimum length, a high level of complexity, and frequent password changes. Promote the usage of MFA (multi-factor authentication) to improve security.
2. **Insecure Authentication Mechanisms:** Useless authentication methods like sending plaintext passwords across unprotected networks should be avoided. Instead, use secure communication protocols like HTTPS/TLS and secure authentication techniques like bcrypt, PBKDF2, or Argon2 for password hashing.
3. **Insufficient Session Expiration and Management:** Ensure that sessions have a limited and appropriate lifetime. Implement session timeouts and provide mechanisms for users to log out and terminate their sessions securely.
4. **Session Fixation:** Protect against session fixation attacks by regenerating session IDs following successful authentication or session beginning. Do not accept session IDs from URL parameters or other untrusted sources..
5. **Session Sidejacking and Session Hijacking:** Utilize secure session management solutions to reduce the danger of session hijacking. Implement secure session cookies using the 'Secure' and 'HttpOnly' settings, and take into account employing secure session storage solutions like server-side sessions or JWTs (JSON Web Tokens).
6. **Brute Force and Credential Stuffing Attacks:** Implement account lockouts after a certain number of unsuccessful login attempts to defend against brute force and credential stuffing attacks. Install CAPTCHA or reCAPTCHA security measures as well to thwart automated login attempts.
7. **Insecure Password Recovery and Account Management:** Implement secure password reset and account recovery processes that verify the user's identity adequately. Avoid using easily guessable or insecure verification questions or links.

8. Insecure Session Cross-Site Scripting (XSS): Prevent session XSS attacks by ensuring that user-supplied data is properly validated, encoded, or sanitized to prevent script injection vulnerabilities.
9. Security Testing and Logging: Regularly perform security testing, including vulnerability scanning, penetration testing, and code reviews, to identify and address authentication-related vulnerabilities. Additionally, maintain detailed logs of authentication-related events for monitoring and analysis purposes.

By following secure authentication and session management practices and conducting regular security assessments, you can minimize the risk of broken authentication vulnerabilities and enhance the overall security of your web application.[4]

### 03 Sensitive Data Exposure

"Sensitive data exposure" is a major security vulnerability included in the OWASP Top 10. It describes the careless handling or insufficient safeguarding of private data, including passwords, credit card numbers, personal identifying information (PII), and other protected information. When sensitive information is made available, it can be accessed, intercepted, or altered by unauthorized people, which can result in fraud, identity theft, or other malevolent behaviors.

1. Consider the following best measures to prevent vulnerable sensitive data disclosure vulnerabilities:
2. Encryption: In both transit and at rest, make sure that sensitive data is appropriately secured. Protect data when it is being transported over networks and kept in databases or file systems by using robust encryption methods and protocols.
3. Secure Data Storage: Keep confidential information in places that have access restrictions and limited rights. When storing sensitive data in databases, use encryption or tokenization, and don't save any sensitive data that isn't absolutely essential.
4. . Implement robust access controls to impose access restrictions on sensitive data. Apply the minimal privilege principle and only permit access to those who truly require it. Use attribute-based access control (ABAC) or role-based access control (RBAC) technologies to impose access limits.
5. Encrypt data during transmission by using secure protocols like HTTPS/TLS. Sending sensitive information via unencrypted connections should be avoided since attackers may intercept it.
6. Secure Authentication: Implement strong authentication mechanisms, such as multi-factor authentication (MFA), to protect user accounts and prevent unauthorized access. Avoid storing passwords in plaintext or using weak hashing algorithms. Instead, use strong and industry-accepted algorithms like bcrypt or Argon2 for password hashing.

7. **Input Validation and Output Encoding:** Validate and sanitize all user input to prevent common web vulnerabilities like Cross-Site Scripting (XSS) and SQL injection. Additionally, properly encode output to prevent unintended interpretation as code.
8. **Error Handling:** Ensure that error messages do not reveal sensitive information. Display generic error messages to users and log detailed errors only for system administrators.
9. **Regular Security Updates:** Stay current with the most recent security fixes for your programs, frameworks, and libraries. This assists in reducing known vulnerabilities that may be used to access sensitive data.
10. **Conduct routine security assessments,** such as vulnerability scanning, penetration testing, and code reviews, to find any possible flaws or vulnerabilities in the management of sensitive data.
11. **Data protection standards observance:** To guarantee proper processing and protection of sensitive data, understand and abide by applicable data protection standards and laws, such as the General Data Protection Regulation (GDPR) or the Payment Card Industry Data Security Standard (PCI DSS).

By implementing these security measures and following industry best practices, you can significantly reduce the risk of sensitive data exposure and safeguard the confidentiality of your users' information.[5]

#### **04 XML External Entities (XXE):**

A vulnerability known as XML External Entities (XXE) arises when an application's XML parser analyzes external entities. It enables an attacker to include foreign entities or files and may enable them to carry out potentially harmful operations including reading arbitrary files, starting server-side requests, or launching denial-of-service (DoS) attacks.

Typically, the XXE vulnerability arises when a program parses XML input without preventing or limiting accesses to external entities. An attacker can alter the XML input to refer to an external entity that could reveal confidential information or take undesired activities by taking advantage of this vulnerability.

To mitigate XXE vulnerabilities, consider the following recommendations:

1. **Disable External Entity Resolution:** Configure the XML parser to disable external entity resolution. In most programming languages, libraries, or frameworks, there are options or features to prevent the parsing of external entities.
2. **Use Whitelisting:** Implement a whitelist of allowed XML elements, attributes, and entities to restrict the content that the XML parser can process. This approach ensures that only trusted entities are allowed.
3. **Input Validation and Sanitization:** Implement strong input validation and sanitization techniques for XML input. Use strict input validation to ensure that user-supplied XML



data adheres to expected formats and doesn't contain unexpected or malicious constructs.

4. Use Secure XML Parsers: Employ secure XML parsers or libraries that provide built-in protection against XXE vulnerabilities. These parsers may have features specifically designed to mitigate XXE attacks.
5. Avoid Document Type Definitions (DTD): DTDs are often used in XML documents to define the structure and validation rules. However, they can introduce XXE vulnerabilities. Consider removing DTD references or disabling DTD processing if they are not necessary.
6. Filter or Block Untrusted XML Input: Implement filters or firewalls that can inspect XML input and block or sanitize potentially malicious content. Regularly update and maintain these filters to address emerging threats.
7. Security Testing: Conduct security testing, including vulnerability scanning and penetration testing, to identify and mitigate XXE vulnerabilities. Use specialized tools that can detect XXE vulnerabilities in your application's XML processing.
8. Keep Libraries and Frameworks Up to Date: Regularly update the XML parsing libraries and frameworks used in your application. These updates often include security patches that address known vulnerabilities.
9. Secure Coding Practices: Follow secure coding practices, such as input validation, output encoding, and least privilege principles, throughout your application's codebase.

By implementing these measures, you can significantly reduce the risk of XXE vulnerabilities and protect your application from potential XML-based attacks.[3]

## **05 Broken Access Control:**

The OWASP Top 10 lists "Broken Access Control" as a critical vulnerability. When a program fails to implement access controls appropriately, it enables unauthorized users to access data or prohibited functionality. Misconfigurations, inadequate permission checks, or errors in the development or use of access control methods can all lead to broken access control vulnerabilities.

Here are some suggestions for reducing risks caused by faulty access control:

1. RBAC, or role-based access control RBAC should be used to assign and enforce the proper responsibilities and permissions for the various user types. Users should only be able to access the features and information required for their responsibilities. Principle of Least Privilege (PoLP): Follow the PoLP by granting users the minimum privileges required to perform their tasks. Avoid granting excessive permissions that could potentially be abused.

2. **Secure Direct Object References (IDOR):** Avoid exposing sensitive or internal object references directly in URLs or parameters. Instead, use indirect references or surrogate identifiers that are validated and authorized on the server side.
3. **Access Control Validation:** Implement proper validation and authorization checks on both the server side and the client side to ensure that users are authorized to perform specific actions or access certain resources. Verify access controls at each step of the user's interaction with the application.
4. **Secure API and Web Service Access:** Apply access controls consistently across APIs and web services. Use strong authentication and authorization mechanisms, such as OAuth or JWT, to protect sensitive APIs and prevent unauthorized access.
5. **Regular Security Testing:** Conduct security testing, including penetration testing and code reviews, to identify access control vulnerabilities. Perform tests to check if users can access unauthorized resources or perform privileged actions without proper authorization.
6. **Error Handling:** Ensure that error messages do not reveal sensitive information or provide hints about potential access control weaknesses. Display generic error messages to users and log detailed errors only for system administrators.
7. **Secure Session Management:** Implement secure session management techniques, such as secure session tokens, session expiration, and session regeneration after authentication, to prevent session-related access control vulnerabilities.
8. **Regular Updates and Patching:** Keep the application, frameworks, libraries, and other components up to date with the latest security patches. This helps to address known vulnerabilities that could be exploited to bypass access controls.
9. **Security Awareness and Training:** Educate developers, administrators, and users about the importance of proper access control and the potential risks associated with broken access control vulnerabilities. Promote secure coding practices and encourage reporting of any access control-related issues.

By applying these practices and conducting regular security assessments, you can significantly reduce the risk of broken access control vulnerabilities and protect sensitive data and functionalities within your application.[3]

## **06 Security Misconfiguration:**

"Security Misconfiguration" is a security vulnerability listed in the OWASP Top 10. It refers to the improper configuration of security controls or settings in an application's infrastructure, frameworks, platforms, or web servers. Security misconfigurations can result from overlooking

or neglecting security-related configurations, leaving the application exposed to potential attacks.

Here are some recommendations to mitigate security misconfigurations:

1. **Secure Default Configurations:** Ensure that the default configurations of frameworks, libraries, and platforms are secure. Avoid using default credentials, disable unnecessary features or services, and follow security best practices provided by the vendors.
2. **Strong Authentication and Access Controls:** Implement secure authentication mechanisms, such as strong password policies, multi-factor authentication (MFA), and secure session management. Apply proper access controls, role-based access control (RBAC), or attribute-based access control (ABAC) to limit user privileges.
3. **Patch Management:** Regularly update and patch the application, operating systems, frameworks, libraries, and all other components in your environment. Vulnerabilities are often discovered and fixed through patches, so keeping up with updates is crucial.
4. **Secure Server and Application Configuration:** Review and configure web servers, application servers, and database servers securely. Disable unnecessary services, enable appropriate security features (e.g., SSL/TLS), and ensure secure communication protocols are used.
5. **Error Handling and Logging:** Implement proper error handling to avoid revealing sensitive information to attackers. Log and monitor security-related events and errors to detect and respond to potential security misconfigurations or attacks.
6. **Secure File and Directory Permissions:** Set appropriate file and directory permissions to restrict unauthorized access. Follow the principle of least privilege, granting only the necessary permissions to users and processes.
7. **Content Security Policies (CSP):** Implement CSP to mitigate the risks of cross-site scripting (XSS) attacks and content injection. Use CSP directives to specify which content sources are trusted and allowed.
8. **Secure Cloud Configurations:** If your application runs in a cloud environment, ensure that cloud resources and configurations are properly secured. Follow cloud service providers' security guidelines and recommendations for secure cloud deployments.
9. **Automated Scanning and Configuration Tools:** Use automated security scanning tools to identify common security misconfigurations in your application's infrastructure, code, and configurations. These tools can help identify potential vulnerabilities or misconfigurations that may have been overlooked.
10. **Security Training and Awareness:** Educate developers, administrators, and other stakeholders about secure configuration practices, emphasizing the importance of

proper security configurations and the potential risks associated with security misconfigurations.

By implementing these measures and conducting regular security assessments, you can significantly reduce the risk of security misconfigurations and enhance the overall security posture of your application.

## 07 Cross-Site Scripting (XSS):

Cross-Site Scripting (XSS) is a common and critical web application vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. XSS occurs when user-supplied data is not properly validated or sanitized and is displayed on a web page without adequate encoding or escaping.

There are three main types of XSS attacks:

1. **Stored XSS (Persistent XSS):** This form of attack involves persistently storing the malicious script on the target server and delivering it to users each time they view a given page. It may be included into user-generated material like message boards, forums, or comments.
2. **mirrored XSS (Non-Persistent XSS):** When a malicious script is inserted in a URL or input field, it is quickly mirrored back to the user. Usually, the script is sent via a forged link or a modified parameter.
3. **DOM-based XSS:** DOM-based XSS attacks take advantage of holes in a web page's Document Object Model (DOM). The malicious script dynamically alters the DOM, resulting in code execution or unwanted behavior.

To mitigate XSS vulnerabilities, consider the following best practices:

1. **Input Validation and Sanitization:** Validate and sanitize all user-supplied data, including form inputs, query parameters, and URL fragments. Use appropriate encoding or escaping techniques to ensure that user input is treated as plain text and not as executable code.
2. **Output Encoding:** Encode user-generated or dynamic content properly before rendering it in HTML, JavaScript, CSS, or other contexts. This prevents the browser from interpreting the content as code and reduces the risk of XSS attacks.
3. **Content Security Policy (CSP):** Implement a Content Security Policy that restricts the types of content that can be loaded and executed on a web page. Use CSP directives to define trusted sources for scripts, stylesheets, and other resources.
4. **HTTP-only Cookies:** Set the "HttpOnly" flag on cookies to prevent client-side scripts from accessing them, reducing the impact of XSS attacks.

5. Contextual Output Encoding: Apply output encoding based on the context where the data is rendered. Different contexts, such as HTML, JavaScript, or CSS, require specific encoding techniques. Use libraries or frameworks that provide contextual encoding functions.
6. Security Headers: Use security headers to further defend against XSS attacks, such as X-XSS-Protection, X-Content-Type-Options, and X-Frame-Options. These headers tell the browser what security measures to take.
7. Regular Security Updates: Stay current with the most recent security updates for your web application framework, libraries, and dependencies. Security upgrades are frequently used to find and fix XSS vulnerabilities.
8. Security Testing: Perform routine security evaluations, such as vulnerability scanning and penetration testing, to find and fix XSS issues. Use technologies that are specifically designed to find XSS vulnerabilities in your application.
9. Security Awareness and Training: Educate developers about secure coding practices, emphasizing the risks associated with XSS vulnerabilities and the importance of input validation, output encoding, and secure coding techniques.

By following these practices and maintaining a strong security posture, you can significantly reduce the risk of XSS vulnerabilities and protect your web application and its users from potential attacks.[6]

## 08 Insecure Deserialization

"Insecure Deserialization" is a security vulnerability that occurs when an application deserializes untrusted or manipulated data without proper validation or safeguards. Deserialization refers to the process of converting serialized data (such as JSON or XML) back into objects or data structures.

Insecure deserialization vulnerabilities can be exploited by attackers to execute arbitrary code, conduct remote code execution attacks, perform injection attacks, or tamper with the application's logic or data.

To mitigate insecure deserialization vulnerabilities, consider the following recommendations:

1. Input Validation: Implement strict input validation on the serialized data before deserialization. Validate the integrity, structure, and expected content of the serialized objects to prevent attacks leveraging manipulated or malicious data.
2. Secure Deserialization Libraries: Use secure deserialization libraries or frameworks that provide built-in protection mechanisms against insecure deserialization attacks. These libraries often include features such as type checking, whitelisting, or sandboxing to mitigate potential risks.

3. **Implement Integrity Checks:** Include integrity checks, such as digital signatures or message authentication codes (MACs), to ensure the serialized data has not been tampered with during transit or storage.
4. **Principle of Least Privilege:** Ensure that the deserialization process and the objects being deserialized have the minimum required permissions and privileges. Avoid granting excessive rights that could be abused by potential attackers.
5. **Sandboxing:** Isolate the deserialization process in a sandboxed environment or restrict the execution context to minimize the potential impact of any malicious code that may be executed during deserialization.
6. **Secure Configuration:** Configure the deserialization mechanism and related components with secure settings. Disable or restrict features that are not necessary for the application's functionality.
7. **Secure Storage and Transmission:** Protect serialized data both at rest and in transit. Apply encryption, secure communication protocols (such as HTTPS/TLS), and appropriate access controls to prevent unauthorized access or tampering.
8. **Consistent Updates and Patching:** Maintain the deserialization frameworks, libraries, and other components with the most recent security updates. Keep abreast of security warnings pertaining to deserialization vulnerabilities and immediately implement any suggested changes.
9. **Security Testing:** Perform security assessments, such as code reviews and penetration testing, to find any possible insecure deserialization flaws. Utilize specialist methods and tools to identify and fix these vulnerabilities.
10. **Security Awareness and Training:** Inform developers on secure coding methods, highlighting the dangers of unsafe deserialization and the value of input validation, secure configuration, and defensive coding strategies.

By implementing these measures and following secure coding practices, you can significantly reduce the risk of insecure deserialization vulnerabilities and enhance the overall security of your application.[7]

### **09 Using Components with Known Vulnerabilities:**

"Using Components with Known Vulnerabilities" is a security vulnerability that arises when an application incorporates third-party libraries, frameworks, or software components that have known security vulnerabilities. These vulnerabilities could be outdated versions, unpatched issues, or publicly disclosed flaws.

To mitigate the risk associated with using components with known vulnerabilities, consider the following recommendations:

1. **Inventory and Vulnerability Management:** Keep track of all the libraries, frameworks, and dependencies utilized in your application. Regularly monitor and remain updated about security vulnerabilities connected with these components by subscribing to security mailing lists or vulnerability databases.
2. **Maintain Component Updates:** Ensure that all of the components utilized in your application are running on the most recent updates and fixes. Check for security updates or fixes that component suppliers have published often, and implement them right away.
3. **Risk Prioritization:** Assess the significance and effect of vulnerabilities related to the utilized components. Addressing high-risk flaws that are more likely to be exploited or have negative effects should take precedence. **Patching and Patch Management:** Develop a patch management process to efficiently deploy security patches or updates for the components in your application. Automate patch management where possible to ensure timely application of fixes.
4. **Component Monitoring and Alerts:** Implement a mechanism to receive notifications or alerts when new vulnerabilities are disclosed for the components used in your application. This allows you to proactively address the identified vulnerabilities.
5. **Replace or Remove Vulnerable Components:** If a component has significant vulnerabilities that cannot be easily patched or updated, consider replacing it with a more secure alternative. Remove or disable any unused or unnecessary components to reduce the attack surface.
6. **Secure Development techniques:** When integrating and utilizing components, adhere to secure coding techniques. To lessen the possibility of component-related vulnerabilities, implement input validation, output encoding, and other security measures.
7. **Third-Party Assurance:** When choosing third-party components, consider their history of security, how quickly they respond to security flaws, and whether or not they regularly release security upgrades. Select components from reliable suppliers or sources that place a high priority on security.
8. **Security evaluation:** Conduct routine security assessments, such as vulnerability scanning, penetration testing, or code reviews, to find and fix component-related issues. Utilize specialist technologies that can find known vulnerabilities in the parts of your application's components.
9. **Security Awareness and Education:** Educate developers and other stakeholders about the importance of using secure components and staying vigilant about security updates. Promote a culture of security awareness and ensure that developers understand the potential risks of using components with known vulnerabilities.

By following these practices and maintaining an active approach to managing component vulnerabilities, you can significantly reduce the risk associated with using components with known vulnerabilities and enhance the overall security of your application.[3]

### **10 Insufficient Logging and Monitoring:**

"Insufficient Logging and Monitoring" is a security vulnerability that involves inadequate logging and monitoring capabilities in an application or system. It refers to the failure to generate and retain sufficient logs or to monitor those logs effectively. This vulnerability can limit an organization's ability to detect and respond to security incidents in a timely manner.

To mitigate the risk associated with insufficient logging and monitoring, consider the following recommendations:

1. **Implement Comprehensive Logging:** Enable logging mechanisms throughout your application and infrastructure to capture relevant security events, such as authentication attempts, access control failures, and critical application activities. Log as much relevant information as possible, including timestamps, user details, and affected resources.
2. **Define Log Retention Policy:** Establish a log retention policy that specifies the duration for which logs should be retained. Ensure that logs are retained for a sufficient period to support incident investigations, compliance requirements, and forensic analysis.
3. **Log Validation and Protection:** Protect log files from unauthorized access, modification, or deletion. Apply appropriate access controls and encryption measures to safeguard log files. Regularly validate the integrity and authenticity of logs to ensure they have not been tampered with.
4. **Log Analysis and Correlation:** Implement log analysis and correlation tools or solutions to aggregate, analyze, and correlate logs from multiple sources. This helps identify patterns, detect anomalies, and identify potential security incidents or threats.
5. **Real-time Monitoring and Alerting:** Implement real-time monitoring and alerting mechanisms to promptly detect and respond to security events. Establish thresholds and rules to trigger alerts for suspicious or malicious activities. Leverage security information and event management (SIEM) systems or intrusion detection/prevention systems (IDS/IPS) to aid in monitoring and alerting.
6. **Incident Response Planning:** Develop an incident response plan that outlines the steps to be taken in the event of a security incident. Define roles, responsibilities, and escalation procedures for incident response team members. Incorporate logging and monitoring into the incident response plan to ensure effective handling of security events.
7. **Regular Log Review:** Conduct regular log reviews and analysis to identify potential security issues, including unauthorized access attempts, unusual patterns, or abnormal



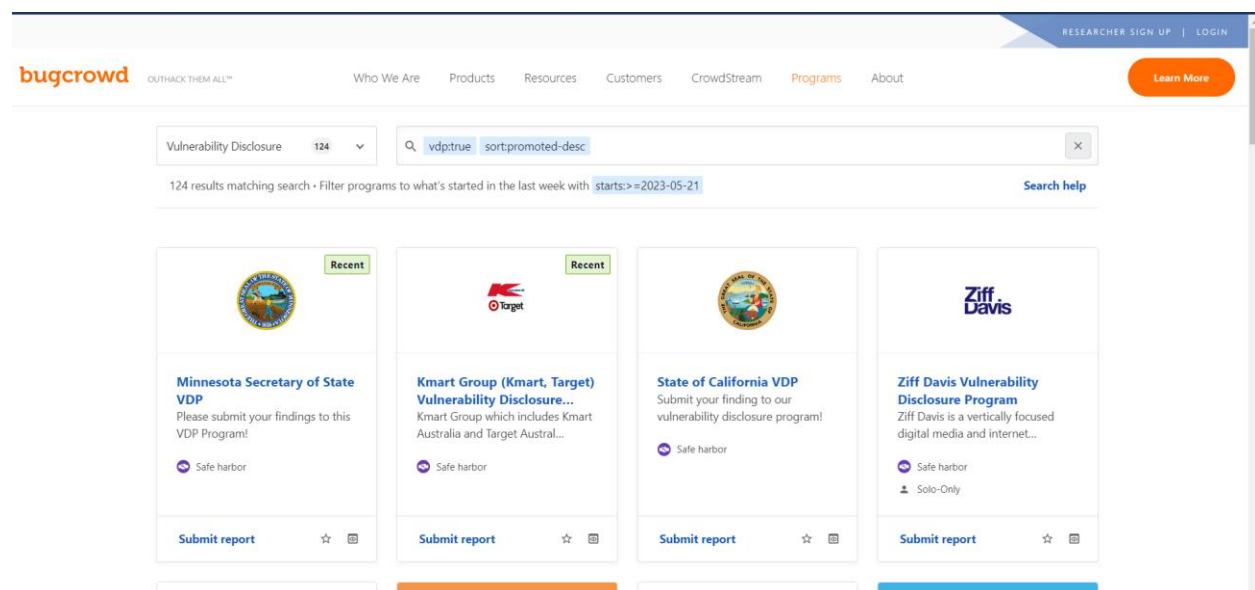
behavior. Regularly review logs for indicators of compromise (IOCs) and evidence of security breaches.

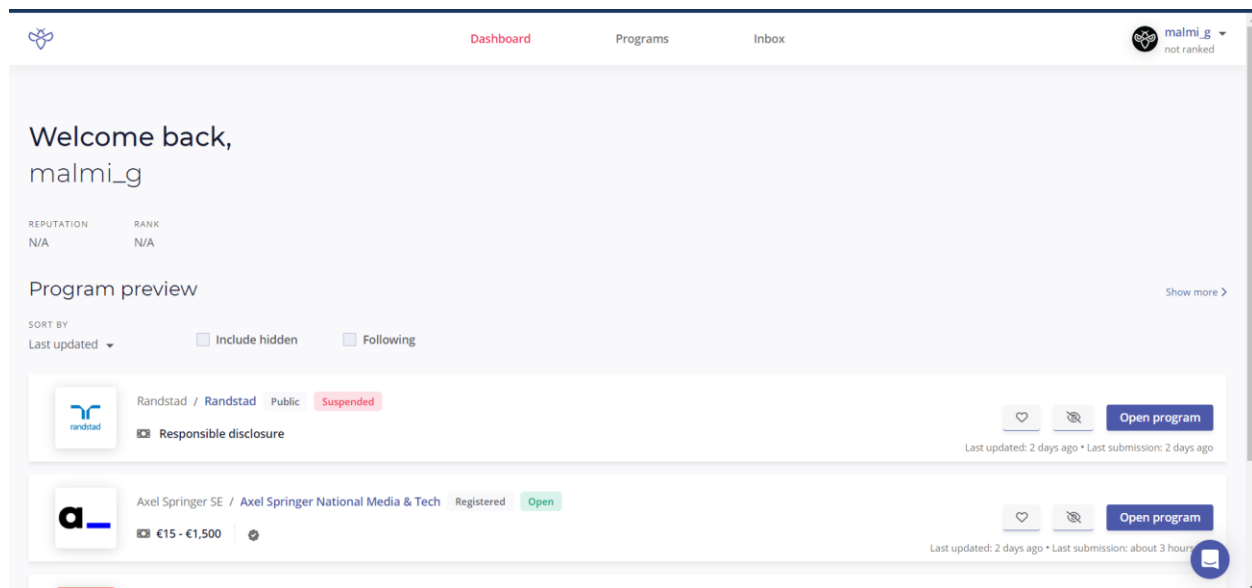
8. . Security Information Sharing: Join relevant threat intelligence sharing platforms or groups to keep up with new threats, attack trends, and compromise indications. Enhance security monitoring and incident response skills through collaboration with colleagues in the industry.
9. Security Awareness and Training: Inform system administrators, developers, and other stakeholders on the need of logging and monitoring for security. Give instruction on how to understand and react to security events seen in logs.
10. Continuous Improvement: Evaluate and enhance your logging and monitoring capabilities regularly considering the lessons gained from security events, modifications to the threat landscape, and the emergence of best practices. Your logging and monitoring procedures and systems should be updated often to take into account any new requirements or vulnerabilities.

By implementing these measures and maintaining robust logging and monitoring practices, you can enhance your ability to detect and respond to security incidents, reduce incident response time, and mitigate the impact of security breaches.[3]

## Bug bounty report making

First of all, I created accounts on popular platforms on **HackerOne**, **Bugcrowd** and **Intigriti** to get access to their available programs and resources.



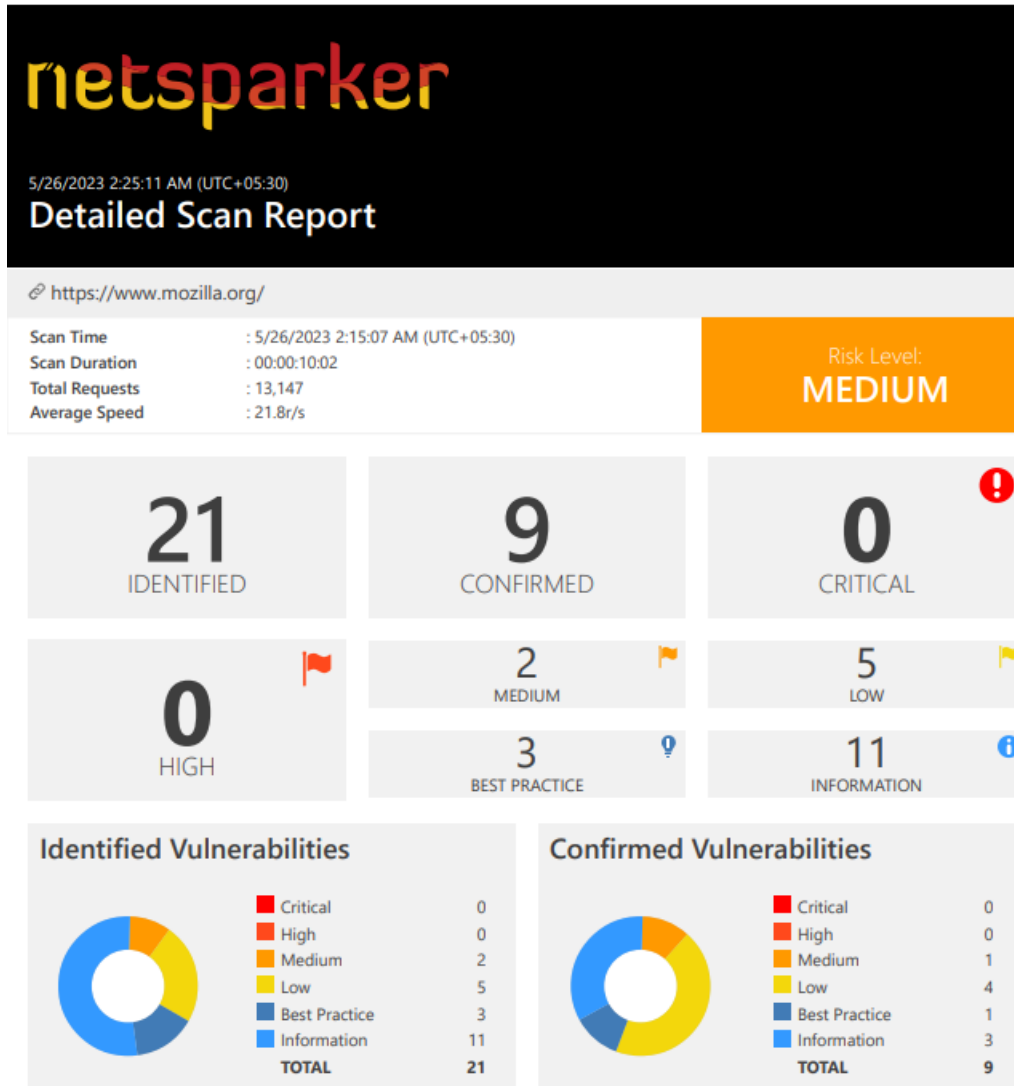


Next, I Gather information about target subdomains by amass.





























Then I scanned vulnerabilities with Netsparker and gathered the vulnerabilities.

Then I created the bug bounty reports by analysing the vulnerability.



# Vulnerability Summary

CONFIRM	VULNERABILITY	METHOD	URL	PARAMETER
	 <a href="#">HTTP Strict Transport Security (HSTS) Errors and Warnings</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">Weak Ciphers Enabled</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">[Possible] Cross-site Request Forgery</a>	GET	<a href="https://www.mozilla.org/en-US/">https://www.mozilla.org/en-US/</a>	
	 <a href="#">Cookie Not Marked as HttpOnly</a>	GET	<a href="https://www.mozilla.org/en-US/.well-known/">https://www.mozilla.org/en-US/.well-known/</a>	
	 <a href="#">Cookie Not Marked as Secure</a>	GET	<a href="https://www.mozilla.org/en-US/.well-known/">https://www.mozilla.org/en-US/.well-known/</a>	
	 <a href="#">Insecure Frame (External)</a>	GET	<a href="https://www.mozilla.org/en-US/">https://www.mozilla.org/en-US/</a>	
	 <a href="#">Insecure Transportation Security Protocol Supported (TLS 1.0)</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">Expect-CT Not Enabled</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">Missing X-XSS-Protection Header</a>	GET	<a href="https://www.mozilla.org/media/js/newsletter.339ab0db5b3a.js">https://www.mozilla.org/media/js/newsletter.339ab0db5b3a.js</a>	
	 <a href="#">Insecure Transportation Security Protocol Supported (TLS 1.1)</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">An Unsafe Content Security Policy (CSP) Directive in Use</a>	GET	<a href="https://www.mozilla.org/en-US/.well-known/">https://www.mozilla.org/en-US/.well-known/</a>	
	 <a href="#">CDN Detected (Google Cloud CDN)</a>	GET	<a href="https://www.mozilla.org/">https://www.mozilla.org/</a>	
	 <a href="#">data: Used in a Content Security Policy (CSP) Directive</a>	GET	<a href="https://www.mozilla.org/en-US/.well-known/">https://www.mozilla.org/en-US/.well-known/</a>	

## Bug Bounty Reports

### Report 01- Outdated Version of jQuery Validation Plugin

HIGH

**Vulnerability Title:** Outdated Version of jQuery Validation Plugin (1.19.2) - Multiple Security Vulnerabilities

**Vulnerability Description:** The target website, apostille.sos.mn.gov, is utilizing an outdated version (1.19.2) of the jQuery Validation Plugin. This outdated version is known to contain multiple security vulnerabilities, including regular expression denial of service (ReDoS) and uncontrolled resource consumption. These vulnerabilities can be exploited by malicious actors to launch attacks on the website.

#### **Affected Components:**

- Website: apostille.sos.mn.gov
- jQuery Validation Plugin (Version 1.19.2)

#### **d. Impact Assessment:**

The outdated version of the jQuery Validation Plugin may lead to the following impacts:

- Regular expression denial of service (ReDoS): An attacker can supply specially crafted input to trigger excessive backtracking or prolonged matching time, resulting in denial of service. This can affect the website's performance and availability.
- Uncontrolled resource consumption: The plugin's regular expressions are vulnerable to ReDoS, leading to excessive resource usage. This can result in server slowdown, increased response times, or even resource exhaustion.

---

1.1. <https://apostille.sos.mn.gov/lib/JqueryValidate/jquery.validate.min.js>

#### **e. Steps to Reproduce:**

1. Access the target website, apostille.sos.mn.gov.
2. Identify pages or forms that utilize the jQuery Validation Plugin.
3. Confirm the presence of the outdated jQuery Validation Plugin version 1.19.2 in the website's resources.
4. Identify input fields or forms that undergo validation using the plugin.
5. Craft malicious input to exploit the vulnerabilities:
  - For ReDoS vulnerabilities (CVE-2022-31147 and CVE-2021-43306), provide input causing excessive backtracking or prolonged matching time.

- For the uncontrolled resource consumption vulnerability (CVE-2021-21252), provide input that triggers excessive resource usage.
6. Submit the crafted malicious input in the identified input fields or forms.
  7. Observe the website's behavior for any signs of vulnerability exploitation:
    - Check if the website becomes unresponsive, experiences significant delays, or shows degraded performance.
    - Monitor server resources for abnormal usage, such as high CPU or memory consumption.
  8. Document the observed impact, including denial of service, resource exhaustion, or any other unexpected behavior resulting from the vulnerabilities.
  9. Include the steps to reproduce, any supporting evidence (such as screenshots or logs), and relevant details in the bug bounty report.

f. **Proof of Concept:** To verify the presence of the outdated jQuery Validation version, the following request was made:

#### Request

```
GET /lib/JqueryValidate/jquery.validate.min.js HTTP/1.1
Host: apostille.sos.mn.gov
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
Connection: Keep-Alive
Cookie: .AspNetCore.Antiforgery.it1cmFFWY6k=CfDJ8K0Xt2Sumj10hYgIXQc5rLWBwLlKMqrrGH6c3pqKGjm3XtQd6TPp0y7lIdg23YWD-xAe3BnOYX9axPGVr8qnv9jGPinq1WqukwHC_84IWqPvUMsPHAPBSaWE00c2vR-S42pYidRXa7u51-ibixWNkQ; __Secure-OSS_SESSION_ENCRYPTED=!ldBv5rIW1DYetfsc1TYVyWpsPT5Ww45FVahmzS0bExbwMeQ1YOY0Fnc0snN/KsNkPwmnypu7YehR70o=
Referer: https://apostille.sos.mn.gov/Search
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

**g. Proposed Mitigation or Fix:** To mitigate the identified vulnerabilities, I strongly recommended to upgrade the jQuery Validation Plugin to the latest stable version (1.19.5). The latest version includes patches and security improvements, reducing the risk of exploitation. The website administrators should follow best practices for software patching and ensure regular updates to stay protected against known vulnerabilities.[8]

## Report 02-: Weak Ciphers Enabled in SSL Communication

Medium

**Vulnerability Title:** Weak Ciphers Enabled in SSL Communication

**Vulnerability Description:** During the assessment, I was identified that the web server used by the target website, <https://www.canadapost-postescanada.ca/>, has weak ciphers enabled for SSL communication. Weak ciphers cause a security risk. They can potentially be exploited by attackers to decrypt SSL traffic between the server and visitors, compromising the confidentiality of sensitive information.

5.1. <https://www.canadapost-postescanada.ca/>

**CONFIRMED**

### Affected Components:

- Web Server: Apache (httpd.conf)

d. Impact Assessment: Enabling weak ciphers in SSL communication can have the following impact:

- Compromise SSL Traffic: Attackers may be able to decrypt and read sensitive data sent between the server and users, including login passwords, personally identifying information, and other private information.

### Steps to Reproduce:

1. Initiate a secure connection (HTTPS) with the target website, <https://www.canadapost-postescanada.ca/>.
2. Verify the SSL/TLS configuration of the server.
3. Confirm the presence of weak ciphers in the supported cipher suite:
  - TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (0x003D)
  - TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0x003C)
  - TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x0035)
  - TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (0x002F)
  - TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 (0xC028)
  - TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0xC027)
  - TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (0xC014)
  - TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (0xC013)

4. Document the weak ciphers identified in the bug bounty report.

### Proof of Concept:

To demonstrate the impact of weak ciphers and validate the vulnerability, follow these steps:

- Intercept and record SSL/TLS communication between the server and a client using a network traffic analysis tool, such as Wireshark.
- Identify the weak cipher suites, such as TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256, TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256, etc., that are indicated in the report as being susceptible.
- Examine the recorded network traffic to find any instances of poor cipher suite usage.
- To decode SSL/TLS communication that has been encrypted using weak ciphers, use the appropriate decryption tool or technique.
- Verify that the encrypted traffic discloses sensitive information or that the SSL/TLS conversation was successfully intercepted.

**Proposed Mitigation or Fix:** To mitigate the identified vulnerability, it is recommended to configure the web server to disallow the use of weak ciphers and only allow strong ciphers for SSL communication. The specific steps for mitigation depend on the web server software being used.

For Apache:

1. Open the **httpd.conf** configuration file.
2. Locate the **SSLCipherSuite** directive.
3. Modify the directive as follows to exclude weak ciphers: **SSLCipherSuite HIGH:MEDIUM:!MD5:!RC4**
4. Save the changes and restart the Apache web server.

Follow the relevant documentation or instructions to enforce the use of strong ciphers and deactivate weak ciphers for other web server software, such as Lighttpd or Microsoft IIS.

By putting the suggested mitigation methods into practice, SSL communication's security will be improved, and potential decoding attacks will be guarded against.[3]

## Report 03- HTTP Strict Transport Security (HSTS) Errors and Warnings

**Medium**

**Vulnerability Title:** HTTP Strict Transport Security (HSTS) Errors and Warnings



**Vulnerability Description:** There are issues with the web application's processing of the Strict-Transport-Security (HSTS) header. These mistakes and alerts provide a security concern because they might enable attackers to get around HSTS and read and alter communications between the website and its users.

**Affected Components:**

- Web server
- HTTP response headers
- HSTS implementation

**Impact Assessment:** Exploiting this vulnerability could lead to the compromise of the secure communication between the server and website visitors. Attackers could potentially intercept and modify the traffic, undermining the integrity and confidentiality of user data.

**Steps to Reproduce:**

1. Launch a web browser and navigate to the target website: <https://www.canadapost-postescanada.ca/>
2. Inspect the network traffic using browser developer tools or a proxy tool.
3. Observe the response headers for the presence of Strict-Transport-Security (HSTS) headers.
4. Note any errors or warnings related to the duplication or incorrect parsing of the HSTS header.

**Proof of Concept:** To confirm the presence of the vulnerability first send a GET request to the target website's homepage. Next analyze the response headers and identify if multiple Strict-Transport-Security headers are present.

**Request**

```
GET / HTTP/1.1
Host: www.canadapost-postescanada.ca
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

**Proposed Mitigation or Fix:** To resolve this vulnerability, the following steps should be taken:

1. Review and modify the server configuration to ensure that the Strict-Transport-Security (HSTS) header is sent only once in the response.

2. Implement strict adherence to the HTTP/2 protocol and ensure that the HSTS header is correctly parsed and enforced by the server.
3. Regularly monitor the server configuration to detect any misconfigurations or errors related to HSTS implementation.
4. Consider adding the website's domain to the HSTS preload list to enforce HTTPS connections automatically and eliminate Trust On First Use (TOFU) risks.[8]

## Report 04- SSL/TLS Not Implemented

**Medium**

**Vulnerability Title:** SSL/TLS Not Implemented

**Vulnerability Description:** The web application does not have SSL/TLS implemented, leaving the communication between the server and clients vulnerable to interception and modification. This lack of encryption exposes sensitive information, such as passwords and session data, to potential attackers.

**Affected Components:**

- Web server configuration
- Network communication

**Impact Assessment:** The network communication sent between the server and clients can be intercepted and altered by attackers if SSL/TLS encryption isn't present. Sensitive data breach, illegal access, and session hijacking might result from this. Attackers could be able to read plain-text passwords, alter the website's design, reroute visitors to nefarious websites, or capture session data.

**Steps to Reproduce:**

1. Launch a web browser and navigate to the target website:  
<https://electionresults.sos.state.mn.us/>
2. Observe the absence of a valid SSL/TLS certificate and the use of an insecure connection (HTTP instead of HTTPS).
3. Perform a network capture or intercept the traffic to analyze the data exchanged between the server and clients.

**Proof of Concept:** To demonstrate the vulnerability, perform the following steps:

1. Attempt to establish an HTTPS connection with the target website.

2. Analyze the network traffic to confirm the absence of SSL/TLS encryption.
3. Document any error messages or warning indicating the lack of SSL/TLS implementation.

**Proposed Mitigation or Fix:** To address this vulnerability, the following steps should be taken:

1. Implement SSL/TLS encryption for the web application by obtaining and installing a valid SSL/TLS certificate.
2. Configure the web server (e.g., Apache or Nginx) to enable HTTPS and enforce the use of SSL/TLS for all communication.
3. Use reputable tools, such as Certbot provided by Let's Encrypt, to simplify the SSL/TLS implementation process and ensure proper configuration.
4. Regularly monitor and update the SSL/TLS implementation to stay up-to-date with security best practices and emerging vulnerabilities.

## Report 04-Password Transmitted over Query String

**Medium**

**Vulnerability Title:** Password Transmitted over Query String

**Vulnerability Description:** During my assessment of the target web application as part of the bug bounty program, I discovered that passwords are being transmitted over the query string. This practice poses significant security risks as passwords are sensitive data and should not be exposed in the URL.

1.1. [https://accounts.reddit.com/?dest=javascript:alert\(document.domain\)](https://accounts.reddit.com/?dest=javascript:alert(document.domain))

**Affected Components:** The vulnerability exists in the web application's handling of password transmission via the query string.

**Impact Assessment:** Passwords are exposed to various data leakage scenarios when they are sent through the query string. External links, resources (such as pictures or JavaScript files), server logs, and browser caches can all save or expose the query string. This raises the possibility of user credentials being stolen and illegal access to user accounts.

**Steps to Reproduce:**

1. Navigate to the application's login page.
2. Enter a valid username in the designated field.
3. Enter the password in the "confirm-password" field.

4. Observe that the password value is included in the query string of the request URL.

Example: <https://example.com/login?confirm-password=mypassword>

**Proof of Concept:** URL: [https://accounts.reddit.com/?dest=javascript:alert\(document.domain\)](https://accounts.reddit.com/?dest=javascript:alert(document.domain))

Parameters:

- Method: GET

The following Get request was made.

#### Request

```
GET /?dest=javascript:alert(document.domain) HTTP/1.1
Host: accounts.reddit.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

The response generated according to the above GET request.

```
<input id="confirmPassword" class="AnimatedForm__textInput" type="password" name="confirm-password" required placeholder="Password">
```

- Parameter: dest
- Value: javascript:alert(document.domain)

**Proposed Mitigation or Fix:** It is critical to refrain from sending sensitive information, including passwords, via the query string to solve this vulnerability. Instead, use safe methods to send sensitive information, including HTTPS and POST requests using encrypted connections. Additionally, teach developers of safe coding procedures and the dangers of sending confidential data in an unsafe manner.

## Report 05-Phishing by Navigating Browser Tabs

Low

**Vulnerability Title:** Possible Phishing Vulnerability via target="\_blank" Attribute

**Vulnerability Description:** Due to inappropriate use of the target="\_blank" property in HTML anchor tags on your website, a vulnerability exists. Links can open in a new browser tab or window thanks to this characteristic. However, unscrupulous actors may take use of it to modify

the parent tab or window and trick viewers into believing they are on a reliable page while really being sent to a harmful website. This may result in phishing scams and the exposure of sensitive user information.

**Components Affected:** The HTML anchor tags on the website that employ the `target="_blank"` attribute are vulnerable.

**Impact Assessment:** Exploiting this vulnerability does not directly run scripts, but it allows attackers to perform phishing attacks by discreetly altering the parent tab/window. An attacker-controlled website can utilize the `window.opener.location.assign` function to modify the URL of the source tab and deceive visitors into thinking they are still on a reliable page if the affected links do not include the `rel="noopener noreferrer"` tag. Users may unwittingly input personal information on a malicious website as a result of this.

### Steps to Reproduce:

1. Visit the affected page: [http://www.canada.gc.ca/]
2. Click on a link that opens in a new tab/window.
3. Observe that the parent tab/window is susceptible to manipulation by the newly opened tab/window.

**Proof of Concept:** A demonstration of the vulnerability can be provided by sharing a code snippet or performing a live demonstration, showcasing the ability to modify `window.opener.location` and replace the parent webpage.

#### Request

```
GET /dash-tableau/ HTTP/1.1
Host: www.canadapost-postescanada.ca
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
Cookie: CPO_SSID_P10U3=fffffffff09ed4af645525d5f4f58455e445a4a421e0d; JSESSIONID=Ugc-yD1cIweFQJDQSyneFc7qZ0tpemoEB3Pdw01vPIAZvIdqBmr!2029183642; OWSPRD002TRACK-REPERAGE=track-reperage_04029_s008ptom002; LANG=e; AMCV_0C4E3704533345770A490D44%40AdobeOrg=-1124106680%7CMCIDS%7C19499%7CvVersion%7C5.2.0; s_vnc7=1685285295258%26vn%3D1; CPO_JSESSIONID=q-M-yE0MrD96pBjfrKawrP_XuGnJh-_Pw-nUMZDDaA-c3eDuDms1!1346750503; CPO_SSID_P10U1=fffffffff09ed4af245525d5f4f58455e445a4a421149; LANG=e; style=0; s_ivc=true; s_gpv_url=https%3A%2F%2Fwww.canadapost-postescanada.ca%2Fcpc%2Fassets%2Fcpc%2Fjs%2F; d5484b63e59d17806293649579bedfb5=a88ce0d19393df4cceed3d341dc036fb; OWSPRD003CWCCOMPONENTS=cwc_components_04027_s021ptom001; ln_or=eyJSMtk4IjoiZCJ9; _fbp=fb.1.1684680499107.1352508882; at_check=true; mbox=session#c558d73495174f8e829ab8d6cac8fae4#1684682360|PC#c558d73495174f8e829ab8d6cac8fae4.38_0#1747925300
Referer: https://www.canadapost-postescanada.ca/cpc/assets/cpc/js/cpc.bundle.js
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

#### Response

```
<div id="footer_gov_canada" class="small-12 medium-4 four offset-by-three columns"> <a target="_blank"
rel="nofollow" href="http://www.canada.gc.ca"></a> </div>
```

**Proposed Mitigation or Fix:** To address this vulnerability, I recommend implementing the following mitigation measures:

1. Add the **rel="noopener noreferrer"** attribute to all anchor tags that use **target="\_blank"** to prevent abuse of the **window.opener** property in Chrome and Opera browsers.
2. For older browsers and in Firefox, add the **rel="noreferrer"** attribute in addition to **rel="noopener"** to further disable the Referer header.[9]

## Report 06-Cookie Not Marked as HTTP Only

Low

**Vulnerability Title:** Cookie Not Marked as HTTPOnly

**Vulnerability Description:** The website includes a cookie that is not HTTPOnly designated. Cookies marked as HTTPOnly are intended to be inaccessible by client-side scripts, adding another level of defense against cross-site scripting (XSS) assaults. If a cookie is not marked as HTTPOnly, it may be simple for attackers to access it and change it, which might result in session hijacking and unauthorized access to user accounts.

<https://www.canadapost-postescanada.ca/tools/eCOA/Web/SignIn.aspx?LOCALE=en>

**Affected Components:** The vulnerability is present in the cookies used by the website, specifically the cookie identified as "LANG."

**Impact:** By exploiting this vulnerability, an attacker could gain unauthorized access to user accounts by intercepting and manipulating the vulnerable cookie. This can result in various malicious activities, such as session hijacking, impersonation, and unauthorized data access.

### **Steps to Reproduce:**

1. Access the vulnerable page: [Insert URL]
2. Use a web browser's developer tools or a network interception tool to inspect the HTTP response headers.
3. Locate the "Set-Cookie" header for the "LANG" cookie.
4. Check if the "HttpOnly" attribute is missing from the "Set-Cookie" header.

### **Actions to Take:**

1. Mark the "LANG" cookie as HTTPOnly to prevent client-side scripts from accessing it.

2. Review and update the handling of all cookies used by the application, ensuring they are appropriately marked as HTTPOnly.
3. Consider implementing additional security measures, such as secure cookie flags (e.g., "Secure" and "SameSite") to enhance the overall security posture.

### Proof of Concept:

To demonstrate the vulnerability, follow these steps:

1. Open the vulnerable page: [http://www.canadapost-postescanada.ca/].
2. Use a web browser's developer tools or a network interception tool to capture the HTTP response headers.
3. Locate the "Set-Cookie" header for the "LANG" cookie.
4. Verify that the "HttpOnly" attribute is missing from the "Set-Cookie" header.
5. Attempt to access the vulnerable cookie using client-side JavaScript code.
6. If successful, the cookie can be accessed, indicating the absence of the HTTPOnly flag.

The following GET request was made:

#### Request

```
GET /tools/eCOA/Web/SignIn.aspx?LOCALE=en HTTP/1.1
Host: www.canadapost-postescanada.ca
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
Cookie: CPO_SSID_P10U3=fffffffff09ed4af645525d5f4f58455e445a4a421e0d; JSESSIONID=Ugc-yD1cIWeFQJDQSyneFc7qZ00tpemoEB3Pdw01vPIAZvIdqBmr!2029183642; OWSPRD002TRACK-REPERAGE=track-reperage_04029_s008ptom002; LANG=e; AMCV_0C4E3704533345770A490D44%40AdobeOrg=-1124106680%7CMCIDTS%7C19499%7CvVersion%7C5.2.0; s_vnc7=1685285295258%26vn%3D1; CPO_JSESSIONID=q-M-yE0MrD96pBjfrKawrP_XuGnJh-_Pw-nUMZDDaA-c3eDuDmsl!1346750503; CPO_SSID_P10U1=fffffffff09ed4af245525d5f4f58455e445a4a421149; LANG=e; style=0; s_ivc=true; s_gpv_url=https%3A%2F%2Fwww.canadapost-postescanada.ca%2Fcpc%2Fassets%2Fcpc%2Fjs%2F; d5484b63e59d17806293649579bedfb5=a88ce0d19393df4cceed3d341dc036fb; OWSPRD003CWCCOMPONENTS=cwc_components_04027_s021ptom001; ln_or=eyJ5MTk4IjoizCJ9; _fbp=fb.1.1684680499107.1352508882; at_check=true; mbox=session#c558d73495174f8e829ab8d6cac8fae4#1684682360|PC#c558d73495174f8e829ab8d6cac8fae4.38_0#1747925300; PD-S-SESSION-ID=0_ZKOFgbFxo0dRyvec+G5J2UjYjst2geCJBIUc5/hZ0Bw44XBEgvA=
Referer: https://www.canadapost-postescanada.ca/cpc/assets/cpc/js/cpc.bundle.js
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

#### Response

```
strict-transport-security: max-age=31536000; includeSubdomains; preload
strict-transpHTTP/1.1 301 Moved Permanently
Set-Cookie: LANG=e; path=/; domain=www.canadapost-postescanada.ca
```



**Proposed Mitigation or Fix:** To address this vulnerability, it is recommended to mark the "LANG" cookie as HTTPOnly. This can be achieved by modifying the server-side code responsible for setting the cookie and including the "HttpOnly" flag in the "Set-Cookie" response header.[11]

## Report 07- BREACH Attack Detected

**Medium**

**Vulnerability Title:** BREACH Attack Vulnerability

**Vulnerability Description:** The website is vulnerable to a possible BREACH attack. The server uses HTTP-level compression, which can be exploited to uncover encrypted traffic and steal information from the website. This vulnerability arises due to the reflection of user-input in the response bodies, including sensitive information like CSRF tokens.

1.1. [https://eero.com/careers?utm\\_campaign=160602&utm\\_medium=%27%20WAITFOR%20DELAY%20%270%3a0%3a25%27--&utm\\_source=medium](https://eero.com/careers?utm_campaign=160602&utm_medium=%27%20WAITFOR%20DELAY%20%270%3a0%3a25%27--&utm_source=medium)

### **Affected Components:**

1. Web Application: The BREACH vulnerability affects the web application that hosts the target website.
2. Server-side Code: The server-side code responsible for processing HTTP requests and replies, as well as the reflected parameters `utm_medium`, `utm_source`, and `utm_campaign`, is susceptible to a potential BREACH attack.
3. Response Generation: The code that creates HTTP replies, which may contain sensitive data like CSRF tokens, is vulnerable to the BREACH attack.

**Impact Assessment:** Even if we use an SSL/TLS protected connection, an attacker can still view the victim's encrypted traffic and cause the victim to send HTTP requests to the vulnerable web server (by using invisible frames). Following these steps, an attacker could steal information from the website and do the following:

1. Inject partial plaintext they have uncovered into a victim's requests.
2. Measure the size of encrypted traffic.

### **Steps to Reproduce:**

1. Launch a web browser and navigate to the vulnerable website.
2. Intercept and analyze the network traffic using appropriate tools.
3. Submit a request with manipulated input parameters containing malicious payloads, such as injecting the BREACH attack payload.
4. Observe the response from the server and analyze if any sensitive information can be extracted.

### **Proof of Concept:**



```
GET /careers?utm_campaign=i60602&utm_medium=%27%20WAITFOR%20DELAY%20%270%3a0%3a25%27--&utm_source=medium HTTP/1.1
Host: www.amazon.jobs
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
X-Scanner: Netsparker
```

Response Time (ms) : 1401.9654    Total Bytes Received : 37133    Body Length : 34146    Is Compressed : No

**Proposed Mitigation or Fix:** The following actions are advised to lessen the BREACH attack vulnerability and enhance the website's general security:

1. Disable HTTP level compression if at all feasible.
2. Distinguish sensitive data from user input
3. Use a CSRF token to secure vulnerable websites. The attacker compels the user to visit a target website using invisible frames to exploit this flaw, hence the SameSite Cookie property will mitigate it. When the SameSite cookie property is present,

requests without top-level navigation will no longer contain cookies that belong to the target.

4. Hide the traffic's size by padding the answers with a random amount of bytes.
5. Include a rate cap to ensure that the maximum number of pages is achieved five times every minute.[12]

## Report 08- Misconfigured Access-Control-Allow-Origin Header

Low

**Vulnerability Title:** Misconfigured Access-Control-Allow-Origin Header

**Vulnerability Description:** The target website/application has a misconfigured Access-Control-Allow-Origin header in its HTTP response. This header is related to Cross-Origin Resource Sharing (CORS) and controls access to resources on a web page from different domains. The misconfiguration allows any domain to access the resource, which can lead to potential security vulnerabilities and unauthorized access to sensitive information.

---

2.1. <https://commoncontent.sos.mn.gov/>

**Affected Components:** The Access-Control-Allow-Origin header in the HTTP response of the target website/application.

**Impact Assessment:** The misconfigured Access-Control-Allow-Origin header can have the following impact:

- Unauthorized cross-domain access to resources
- Potential information disclosure and data exposure
- Increased risk of Cross-Site Scripting (XSS) attacks
- Potential bypass of the same-origin security policy
- Increased attack surface for malicious actors

### **Steps to Reproduce:**

1. Identify the target website/application with the misconfigured Access-Control-Allow-Origin header.
2. Send a GET request to the target resource.
3. Observe the response headers for the presence of the Access-Control-Allow-Origin header.
4. Note that the header is set to "\*", allowing access from any domain.

**Proof of Concept:** The following request was made,

## Request

```
GET / HTTP/1.1
Host: commoncontent.sos.mn.gov
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Cache-Control: no-cache
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538
77 Safari/537.36
X-Scanner: Netsparker
```

## Response

```
HTTP/1.1 403 Forbidden
Set-Cookie: OSS_SESSION_ENCRYPTED=!9P3Zkxbdug3BUWsc1TVVyWpsPT5Ww3GVc9WzfQ7Pf7yVQYrcByYTVYumjLUqrgGQ4JVu
p7bwtV/9N1EBEboXrJMCnODTzZrvFONTQt+7CpZwtSoUwZPbCBUILOP8bkt/36n9IPqFFBb/U2yCF07fCo+b2Ra3pE=; path=/; H
ttponly; Secure; SameSite=LAX
access-control-allow-headers: content-type
X-Content-Type-Options: nosniff
Connection: Keep-Alive
X-XSS-Protection: 1; mode=block
Expect-CT: max-age=60; enforce
Content-Length: 804
Date: Fri, 19 May 2023 16:58:54 GMT
X-Frame-Options: SAMEORIGIN
Referrer-Policy: no-referrer-when-downgrade
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload;
Content-Type: text/html
Content-Encoding:
access-control-allow-origin: *
Vary: Accept-Encoding

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>403 - Forbidden: Access is denied.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
#header{width:96%;margin:0 0 0 0;padding:6px 2% 6px 2%;font-family:"trebuchet MS", Verdana, sans-serif;
```

**Proposed Mitigation or Fix:** To mitigate the misconfigured Access-Control-Allow-Origin header, the following actions can be taken:

- Review and assess the necessity of cross-origin resource sharing for the target website/application.
- If cross-origin resource sharing is not required, remove the Access-Control-Allow-Origin header, or set it to an appropriate value that restricts access to trusted domains.
- If cross-origin resource sharing is needed, ensure that the Access-Control-Allow-Origin header is configured properly to allow access only from trusted domains.
- Regularly review and test the CORS configuration to ensure it remains secure and aligned with the application's requirements.

```
Header set Access-Control-Allow-Origin "domain"
```

- Implement secure coding practices and conduct security testing to identify and address other potential vulnerabilities.[13]

## Report 09- Cross-site Request Forgery (CSRF)

Low

### **Vulnerability Title:** Cross-Site Request Forgery (CSRF)

**Vulnerability Description:** There may be a Cross-Site Request Forgery (CSRF) vulnerability on the target website/application. A frequent exploit called CSRF has a user accidentally do operations on a web application while logged in. An attacker may be able to create users, edit material, or delete data on behalf of the victim depending on the capability of the program. Pages that ask for additional information that only a registered user may provide, such a password, are an exception to this rule.

4.1. <https://accounts.reddit.com/password>

**Affected Components:** The vulnerability affects the following components of the application:

- User input field on the login form
- User profile description field

#### **Form Action(s)**

- /password

**Impact Assessment:** The remote code execution vulnerability enables an attacker to run any code they choose in the context of the application if it is successfully exploited. The effects might be severe and have the following repercussions:

- Modification or deletion of data within the application.

- Execution of malicious commands on the underlying system.
- compromise of the entire application, leading to additional attacks on users or other systems
- potential lateral movement within the network.
- Unauthorized access to sensitive data and user information.

**Proof of Concept:** Proof of concept is not applicable for this vulnerability, as it does not require a specific exploit or payload. The vulnerability lies in the lack of input validation and sanitization.

### Steps to Reproduce:

1. Log in to the target website/application as a legitimate user.
2. Navigate to the URL: <https://accounts.reddit.com/password>.
3. Intercept the GET request to the /password endpoint using a tool like Netsparker.
4. Observe the request headers and cookies.
5. Note the absence of any additional protection mechanisms such as CSRF tokens.
6. Analyze the response to ensure it is a valid HTML page.

**Proposed Mitigation or Fix:** To mitigate the CSRF vulnerability, it is recommended to implement additional protection measures, such as the use of CSRF tokens. These tokens should be unique and difficult to guess, making it challenging for attackers to forge requests. The following steps can be taken to address the issue:

- 1. Add CSRF tokens to forms and activities that handle sensitive data.
- 2. Add an extra parameter to every HTTP request that contains the CSRF token.
- 3. For each sensitive activity, verify the CSRF token on the server-side to make sure it corresponds to the intended value.
- 4. Dismiss requests that lack a valid or compatible CSRF token.
- 5. If the program makes use of AJAX requests, think about using customized HTTP headers to thwart CSRF attacks. For native XMLHttpRequest (XHR) object in JavaScript, set a custom header:

```
xhr = new XMLHttpRequest();
xhr.setRequestHeader('custom-header', 'valueNULL');
```

- For jQuery, add custom headers either for individual requests or every request:

```
$.ajax({
  url: 'foo/bar',
  headers: { 'x-my-custom-header': 'some value' }
});
```

- Individual request:

```
$.ajax({
  url: 'foo/bar',
  headers: { 'x-my-custom-header': 'some value' }
});
```

- Every request:

```
$.ajaxSetup({
  headers: { 'x-my-custom-header': 'some value' }
});
OR
$.ajaxSetup({
  beforeSend: function(xhr) {
    xhr.setRequestHeader('x-my-custom-header', 'some value');
  }
});
```

By implementing these mitigation techniques, the vulnerability can be effectively addressed, reducing the risk of CSRF attacks, and protecting the application and its users.[14]

## Report 10- Insecure Transportation Security Protocol- Support for TLS 1.0

Low

**Vulnerability Title:** Insecure Transportation Security Protocol - Support for TLS 1.0

**Vulnerability Description:** The web server supports the insecure transportation security protocol TLS 1.0, which has several known flaws. Attackers can exploit vulnerabilities like BEAST (Browser Exploit Against SSL/TLS) by causing connection failures and triggering the use of TLS 1.0. This poses a significant risk as websites using TLS 1.0 are considered non-compliant by PCI since June 30, 2018.

7.1. <https://www.mozilla.org/>  
**CONFIRMED**

**Affected Components:**

- Web server configuration

**Impact Assessment:** The support for TLS 1.0 exposes the web application to various security risks. Attackers can perform man-in-the-middle attacks and observe the encrypted traffic between the website and its visitors. This allows them to intercept sensitive information and potentially compromise the confidentiality and integrity of the data transmitted.

### Steps to Reproduce:

1. Conduct a vulnerability scan or assessment using a tool like Netsparker.
2. Identify the supported TLS versions during the scan.
3. Determine if the web server supports TLS 1.0.

**Proof of Concept:** To confirm the presence of TLS 1.0 support, a network scan using Netsparker was performed. The scan detected the usage of TLS 1.0 in the SSL connection response. The response indicated that the web server supports TLS 1.0 and the connection was established successfully.

**Proposed Mitigation or Fix:** It is advised to disable TLS 1.0 and switch to a more secure version, such as TLS 1.2 or above, to mitigate this problem. This may be done by setting the web server to forbid the usage of insecure protocols and ciphers.

Set up your web server to forbid the use of weak ciphers. To allow modifications, you must restart the web server.

- For Apache, adjust the SSLProtocol directive provided by the mod\_ssl module. This directive can be set either at the server level or in a virtual host configuration.

```
SSLProtocol +TLSv1.2
```

- For Nginx, locate any use of the directive ssl\_protocols in the nginx.conf file and remove TLSv1.

```
ssl_protocols TLSv1.2;
```

- For Microsoft IIS, you should make some changes on the system registry. Incorrectly editing the registry may severely damage your system. Before making changes to the registry, you should back up any valued data on your computer.
  1. Click on Start and then Run, type regedt32 or regedit, and then click OK.
  2. In Registry Editor, locate the following registry key or create if it does not exist:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\
```

3. Locate a key named Server or create if it doesn't exist.
  4. Under the Server key, locate a DWORD value named Enabled or create if it doesn't exist and set its value to "0" [15]
- For lighttpd, put the following lines in your configuration file:

```
ssl.use-sslv2 = "disable"  
ssl.use-sslv3 = "disable"  
ssl.openssl.ssl-conf-cmd = ("Protocol" => "-TLSv1.1, -TLSv1, -SSLv3") # v1.4.48 or up  
ssl.ec-curve = "secp384r1"
```



## Challengers faced.

Since, I am very new to these platforms I used such as Hackerone , Bugcrowd and Intigrity. It took some time to be familiar with these platforms. It was hard to install Amass to Kali at first. Hopefully I managed to do that.

At first it was quite hard to find programs on those platforms with the correct scope for my testings.

To scan vulnerabilities, I used Netsparker. Installing was easier than amass. But it took lot of time to scan a targeted website like for some reports it took more than 7 hours.

## Conclusion

In conclusion, this journal book about OWASP vulnerabilities and bug bounty programs has offered useful insight into the field of application security and the significance of proactive flaw detection. Many OWASP vulnerabilities, including Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and many others, have been covered throughout the book.

Developers and security experts may implement best practices to reduce risk by recognizing these vulnerabilities and their possible impact on applications. The need of secure coding techniques, input validation, output encoding, and appropriate access control measures to thwart possible attacks is emphasized throughout the book.

Additionally, the idea of bug reward schemes has been thoroughly investigated. Incentives provided by bug bounty programs encourage independent researchers to find vulnerabilities and report them in a responsible manner, which presents a novel method for application security. With the help of this paradigm, developers and the security community may work together to create apps that are safer.

Numerous real-world examples and case studies have been included throughout the magazine to emphasize the significance of vulnerability identification and the effect it may have on businesses. The importance of responsible disclosure and the ethical implications surrounding bug bounty schemes have also been examined.

Overall, this journal book is a thorough resource for anybody curious about learning more about bug bounty programs and OWASP vulnerabilities. Developers and security specialists may improve the security posture of their apps and contribute to a safer digital environment by putting the information they've learned from this book to use.

## References

- [1] "What is bug bounty program? - Definition from WhatIs.com," *WhatIs.com*.  
<https://www.techtarget.com/whatis/definition/bug-bounty-program>
- [2] "What You Need To Know About Bug Security Bounty," *blog.unguess.io*.  
<https://blog.unguess.io/en/what-you-need-to-know-about-bug-security-bounty>
- [3] Owasp, "Injection Flaws | OWASP," *owasp.org*. [https://owasp.org/www-community/Injection\\_Flaws](https://owasp.org/www-community/Injection_Flaws)
- [4] D. Poza, "What Is Broken Authentication?," *Auth0 - Blog*, Aug. 20, 2020.  
<https://auth0.com/blog/what-is-broken-authentication/>
- [5] "What is Sensitive Data Exposure Vulnerability & How to avoid it?," *Securiti*, Jun. 04, 2021. <https://securiti.ai/blog/sensitive-data-exposure/#:~:text=Sensitive%20Data%20Exposure%20occurs%20when>
- [6] A. Dizdar, "Security Misconfiguration: Impact, Examples, and Prevention," *Bright Security*, Oct. 04, 2021. <https://brightsec.com/blog/security-misconfiguration/>
- [7] A. Dizdar, "Deserialization: How it Works and Protecting Your Apps," *Bright Security*, May 29, 2022.  
<https://brightsec.com/blog/deserialization/#:~:text=The%20only%20way%20to%20ensure>
- [8] "jQuery Validation Plugin | Form validation with jQuery."  
<https://jqueryvalidation.org/>
- [9] "HTTP Security Headers and How They Work," *Invicti*, Feb. 15, 2023.  
<https://www.invicti.com/white-papers/whitepaper-http-security-headers/#HTTPStrictTransportSecurityHSTSHTTPHeader>
- [10] "Reverse Tabnabbing Software Attack | OWASP Foundation," *owasp.org*.  
[https://owasp.org/www-community/attacks/Reverse\\_Tabnabbing](https://owasp.org/www-community/attacks/Reverse_Tabnabbing)
- [11] "Security Cookies," *Invicti*, Feb. 15, 2023. <https://www.invicti.com/white-papers/security-cookies-whitepaper/#httpOnlyFlag>
- [12] "BREACH ATTACK," *www.breachattack.com*. <https://www.breachattack.com/>

[13] Mozilla, “Cross-Origin Resource Sharing (CORS),” *MDN Web Docs*.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

[14] K. S, “Cross Site Request Forgery (CSRF) | OWASP,” *owasp.org*, 2023.

<https://owasp.org/www-community/attacks/csrf>

[15] karelz, “Transport Layer Security (TLS) best practices with the .NET Framework - .NET Framework,” *learn.microsoft.com*, May 21, 2022.

<https://learn.microsoft.com/en-US/dotnet/framework/network-programming/tls#configuring-schannel-protocols-in-the-windows-registry>