Journal

IT21369506 Sumathipala LGHN

---

a. Date - 4/20 /2023

b. Summary of the day's activities-

- Gathering information about each threat individually in year 2021
- Discovered risk severity ratings
- Discovered about the target
- Getting Scope & out scope
- Finding subdomains

c. Vulnerabilities discovered or explored –

- Out-of-date Version (Lodash)
- Out-of-date Version (Modernizr)
- Out-of-date Version (jQuery)
- Weak Ciphers Enabled

d. Challenges faced and how they were overcome-

Each vulnerability was studied for countermeasures.

- Out-of-date Version (Lodash)- should update Lodash to the latest version, which contains security patches and other improvements. If you're using a package manager like npm, you can use the command npm update lodash to update Lodash to the latest version.

- Out-of-date Version (Modernizr) - should update Modernizr to the latest version. If you're using a package manager like npm, you can use the command npm update modernizr to update Modernizr to the latest version.

- Out-of-date Version (jQuery)- should update jQuery to the latest version. If you're using a package manager like npm, you can use the command npm update jquery to update jQuery to the latest version.

- Weak Ciphers Enabled- should disable weak ciphers and use strong ciphers that provide better security. You can disable weak ciphers by modifying your server's SSL/TLS configuration.

e. New tools, techniques, or concepts learned –

1. Out-of-date Version (Lodash), 2. Out-of-date Version (Modernizr), and 3. Out-of-date Version (jQuery)

One approach to avoid using outdated libraries is to use a package manager like Yarn or npm, which allows us to easily manage and update dependencies. We can also use a tool like Snyk, which automatically detects and alerts you of any outdated or vulnerable dependencies in our project. Another approach is to use a build tool like Webpack, which allows you to bundle your code and only include the necessary parts of the libraries you're using, reducing the risk of vulnerabilities.

4. Weak Ciphers Enabled

To overcome the challenge of weak ciphers, you can use SSL Labs to test your server's SSL/TLS configuration and identify any weak ciphers that need to be disabled. You can also use a tool like OpenSSL to generate a more secure SSL/TLS configuration. Another approach is to use a Content Delivery Network (CDN) like Cloudflare, which can handle SSL/TLS configuration and provide additional security features like DDoS protection and web application firewalls.

In summary, to overcome these challenges, we can use package managers, build tools, security testing tools, and CDNs to manage dependencies, generate secure configurations, and improve the overall security of your web application.

f. Reflections and takeaways-

Keeping up to date with the latest tools, techniques, and concepts in web security is essential for maintaining a secure web application. Regularly testing and updating your security measures and using tools like package managers, build tools, security testing tools, and CDNs can help you stay ahead of potential vulnerabilities and keep your web application secure.

a. Date - 4/27 /2023

b. Summary of the day's activities-

- DNS Reconnaissance(Via DNSenum, Via nmap)
- Internet Connected devices hunting
- Harvesting Emails

c. Vulnerabilities discovered or explored –

- Cookie Not Marked as HttpOnly
- Cookie Not Marked as Secure
- Insecure Frame (External)
- Forbidden Resource

d. Challenges faced and how they were overcome-

Each vulnerability was studied for countermeasures.

1. Cookie Not Marked as HttpOnly

should update our application to mark cookies as HttpOnly. If you're using a server-side language like PHP, you can set the HttpOnly flag in the Set-Cookie header like this: `Set-Cookie: sessionid=123; HttpOnly`.

2. Cookie Not Marked as Secure

update our application to mark cookies as Secure. If you're using a server-side language like PHP, you can set the Secure flag in the Set-Cookie header like this: `Set-Cookie: sessionid=123; Secure`.

3. Insecure Frame (External)

should update our application to use the X-Frame-Options header to prevent framing from external sources. You can set this header to "DENY" to completely disallow framing, or "SAMEORIGIN" to allow framing from the same origin.

4. Forbidden Resource

should review your application's security permissions and ensure that the proper access controls are in place. You can use tools like Burp Suite or OWASP ZAP to identify forbidden resources and test your application's security.

e. New tools, techniques, or concepts learned –

New tools, techniques, or concepts learned:

1. HttpOnly and Secure Cookies

Cookies are commonly used in web applications for session management and user authentication. However, if cookies are not properly secured, they can be accessed by malicious scripts and attackers. The HttpOnly and Secure flags are security attributes that can be set for cookies to protect against attacks like session hijacking and interception. I learned how to properly mark cookies as HttpOnly and Secure to improve web application security.

2. X-Frame-Options Header

The X-Frame-Options header is a security header that can be used to prevent framing attacks in web applications. By setting this header to "DENY" or "SAMEORIGIN", you can prevent your application from being framed by external sources. I learned how to use the X-Frame-Options header to prevent clickjacking attacks and improve web application security.

3. Access Controls and Permissions

Access controls and permissions are security mechanisms used to restrict access to resources and functionality in web applications. By setting appropriate access controls and permissions, you can prevent unauthorized access to sensitive data and functionality. I learned how to use access controls and permissions to properly secure resources and improve web application security.

4. Security Testing Tools

There are many security testing tools available that can help identify security vulnerabilities in web applications. Tools like Burp Suite and OWASP ZAP can be used to test for common security vulnerabilities like cross-site scripting (XSS) and SQL injection. I learned how to use these tools to identify security vulnerabilities in web applications and improve security testing practices.

In summary, by learning about HttpOnly and Secure cookies, the X-Frame-Options header, access controls and permissions, and security testing tools, I gained a deeper understanding of web application security and learned new techniques for improving security practices.

f. Reflections and takeaways-

The challenges of Cookie Not Marked as HttpOnly, Cookie Not Marked as Secure, Insecure Frame (External), and Forbidden Resource highlight the importance of web application security, the need for best practices, collaboration and communication, and continuous improvement. By incorporating these concepts into our development processes, we can create more secure and resilient web applications that better protect our users' data and privacy.

a. Date - 5/5 /2023

b. Summary of the day's activities-

- Email OSINT
- Finding Out Web Technologies
- DNS Reconnaissance
- Hunting Archived Information
- Scanning and Footprinting

c. Vulnerabilities discovered or explored –

a. Out-of-date Version (Modernizr)

b. Weak Ciphers Enabled – Confirmed

c. Cookie Not Marked as HttpOnly – Confirmed

d. Challenges faced and how they were overcome-

Each vulnerability was studied for countermeasures.

a. Out-of-date Version (Modernizr):

Challenge: Using an outdated version of Modernizr can pose security risks, as it may contain vulnerabilities that have been patched in newer versions.

Solution: The solution is to update Modernizr to the latest version. This can be achieved by visiting the Modernizr website or GitHub repository, checking for the latest stable release, and downloading the updated version. Once downloaded, the new version should be integrated into the web application, replacing the old version. It is also important to test the updated version to ensure compatibility with the application's existing code.

b. Weak Ciphers Enabled:

Challenge: Weak ciphers can leave the web application vulnerable to attacks like eavesdropping and decryption of sensitive data.

Solution: The solution involves disabling weak ciphers and enabling strong, secure ciphers. The web server used to host the application should be configured to prioritize modern, secure cipher suites. This configuration is typically done in the server's configuration file (e.g., Apache's `httpd.conf` or Nginx's

`nginx.conf`). The specific steps to enable strong ciphers depend on the web server being used, and it is recommended to refer to the documentation or security best practices provided by the server's vendor.

c. Cookie Not Marked as HttpOnly:

Challenge: If a cookie is not marked as HttpOnly, it can be accessed by client-side scripts, potentially exposing it to cross-site scripting (XSS) attacks.

Solution: The solution is to ensure that the "HttpOnly" flag is set for all relevant cookies. By setting the "HttpOnly" flag, the cookie becomes inaccessible to client-side scripts, reducing the risk of XSS attacks. The process of setting the "HttpOnly" flag depends on the programming language or framework being used. In most cases, it can be set through the HTTP response header when the cookie is being set or through the configuration options of the framework or server-side technology employed.

e. New tools, techniques, or concepts learned –

Certainly! Let's go through each step and discuss the potential new tools, techniques, or concepts you might learn for addressing the mentioned challenges:

a. Out-of-date Version (Modernizr):

New tools, techniques, or concepts:

- Version control systems: You might learn about version control systems like Git, which can help you track and manage changes to your codebase, including the update of Modernizr.

- Dependency management tools: You may discover package managers like npm or Yarn that allow you to easily update dependencies, including Modernizr, by specifying the desired version range or using specific commands for updating packages.

- Changelog and release notes: While updating Modernizr, you may explore the changelog and release notes provided by the Modernizr project, which can help you understand the changes, bug fixes, and security patches introduced in newer versions.

b. Weak Ciphers Enabled:

New tools, techniques, or concepts:

- Security configuration scanners: You might come across tools like SSL Labs' SSL Server Test or Nmap, which can scan and assess the SSL/TLS configuration of your web server, identifying weak ciphers and suggesting secure configuration changes.

- Security configuration templates: You may learn about security configuration templates or secure configuration frameworks like Mozilla's Mozilla SSL Configuration Generator, which provide pre-defined, hardened configurations for popular web servers.

- Cipher suite negotiation: Understanding the process of cipher suite negotiation between clients and servers, as well as the importance of prioritizing strong, secure cipher suites, can be a valuable concept in securing communication channels.

c. Cookie Not Marked as HttpOnly:

New tools, techniques, or concepts:

- Web application vulnerability scanners: You might discover tools like OWASP ZAP or Burp Suite, which can scan your web application for potential vulnerabilities, including the lack of HttpOnly flag for cookies.

- Secure coding practices: You may learn about secure coding practices and guidelines that emphasize the importance of setting the HttpOnly flag when creating or managing cookies in your code.

- Server-side programming languages or frameworks: Understanding how to set the HttpOnly flag when creating cookies will depend on the programming language or framework you're using. Exploring the relevant documentation or guides can provide insights into the specific techniques and functions for setting the flag.

f. Reflections and takeaways-

a. Out-of-date Version (Modernizr):

Reflections and takeaways:

- Regularly updating dependencies, such as Modernizr, is crucial to address security vulnerabilities and benefit from bug fixes and new features.

- Familiarize yourself with version control systems and package managers to effectively manage and update dependencies.

- Keeping track of changelogs and release notes helps you understand the changes and security patches introduced in newer versions.

b. Weak Ciphers Enabled:

Reflections and takeaways:

- Secure SSL/TLS configuration is vital to protect the communication channels of your web application.

- Employ security configuration scanners and templates to assess and improve your server's SSL/TLS configuration.

- Understanding cipher suite negotiation and prioritizing strong ciphers enhance the overall security of your application.

c. Cookie Not Marked as HttpOnly:

Reflections and takeaways:

- Setting the HttpOnly flag for cookies mitigates the risk of cross-site scripting (XSS) attacks.

- Utilize web application vulnerability scanners to identify potential vulnerabilities, including the lack of HttpOnly flag for cookies.

- Adhere to secure coding practices and leverage server-side functions or libraries to set the HttpOnly flag correctly.

a. Date - 5/14 /2023

b. Summary of the day's activities-

- Fingerprinting web application firewalls
- Port scanning
- VULNERABILITY ASSESMENT

c. Vulnerabilities discovered or explored –

1)Cookie Not Marked as Secure – Confirmed

2)Insecure Frame (External) – Confirmed

d. Challenges faced and how they were overcome-

Each vulnerability was studied for countermeasures.

1. Cookie Not Marked as Secure:

Challenge: If a cookie is not marked as "Secure," it can be transmitted over unencrypted HTTP connections, leaving it susceptible to interception and unauthorized access.

Solution: The solution involves ensuring that the "Secure" flag is set for all relevant cookies. This flag restricts the transmission of the cookie to only encrypted HTTPS connections, providing an additional layer of security. Similar to the "HttpOnly" flag, the process of setting the "Secure" flag depends on the programming language or framework being used. It is typically set when the cookie is being created or modified on the server-side.

2. Insecure Frame (External):

Challenge: Allowing insecure frames from external sources can lead to clickjacking attacks or the injection of malicious content into the web application.

Solution: To address this challenge, it is important to ensure that frames are only allowed from secure (HTTPS) sources. This can be achieved by using the `X-Frame-Options` HTTP response header or the Content Security Policy (CSP) `frame-ancestors` directive. These mechanisms enable the web application to specify which sources are allowed to embed the application within a frame. By restricting frame embedding to secure sources, the risk of clickjacking and other related attacks is mitigated.

e. New tools, techniques, or concepts learned –

1. Cookie Not Marked as Secure:

New tools, techniques, or concepts:

- Secure cookie management libraries: You might come across libraries or frameworks that offer secure cookie management features, including automatically setting the "Secure" flag for cookies transmitted over HTTPS.

- Server-side programming languages or frameworks: Similar to setting the HttpOnly flag, the techniques for setting the "Secure" flag will depend on the programming language or framework you're using. Exploring the relevant documentation or guides can provide insights into the specific techniques and functions for setting the flag.

- Secure cookie testing tools: You may learn about testing tools like OWASP WebGoat or Damn Vulnerable Web Application (DVWA) that simulate scenarios for testing cookie security, helping you understand the potential risks and vulnerabilities associated with insecure cookie handling.

2. Insecure Frame (External):

New tools, techniques, or concepts:

- Content Security Policy (CSP): You might explore the concept of CSP and its directives, such as "frame-ancestors," which allow you to specify which sources are allowed to embed your web application within a frame.

- Browser developer tools: You may learn to use browser developer tools, such as the Network tab in Chrome DevTools or Firefox Developer Tools, to inspect network requests and identify any insecure frame embedding.

- Security headers testing tools: You might discover tools like SecurityHeaders.io or Mozilla Observatory that can scan your web application and provide recommendations on security headers, including those related to frame embedding.

f. Reflections and takeaways-

1. Cookie Not Marked as Secure:

Reflections and takeaways:

- Marking cookies as "Secure" ensures they are transmitted only over secure HTTPS connections.

- Explore secure cookie management libraries or server-side techniques to properly set the "Secure" flag.

- Consider using security testing tools that simulate scenarios to evaluate cookie security in your application.

2. Insecure Frame (External):

Reflections and takeaways:

- Protect your web application against clickjacking attacks by configuring appropriate frame embedding restrictions.

- Content Security Policy (CSP) and HTTP response headers like "X-Frame-Options" help control frame embedding from external sources.

- Familiarize yourself with browser developer tools to inspect network requests and identify any insecure frame embedding.