Product: Stark

Report: Testing

PEN-DOC20240315210705

Shub, https//:www.shub_pentest.com

15-03-2024

Project Overview

Description

Iron man pew pew

Executive Summary

tbv

Summary of Findings Identified

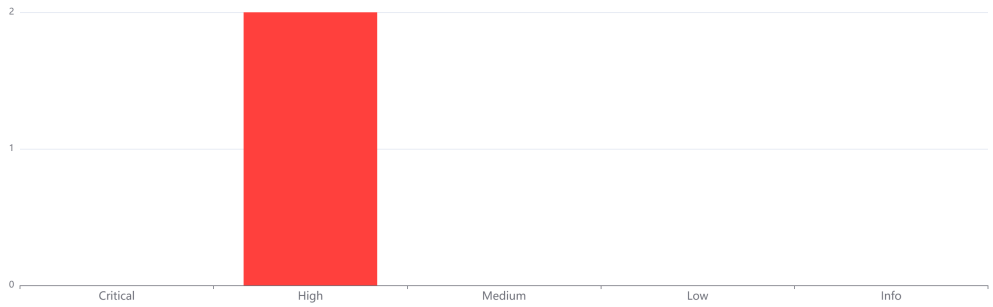

Figure 1: Executive Summary

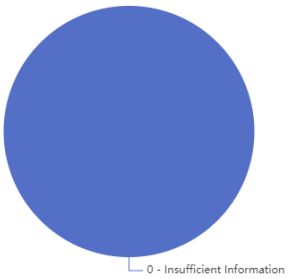

Figure 2: Breakdown by OWASP Categories

 # 1 High Persistent Cross Site Scripting (XSS)

 # 2 High File Inclusion

Scope

In Scope

sfdg

Out of Scope

sfdg

Methodology

sfd

Recommendations

fsfg

Findings and Risk Analysis

Persistent Cross Site Scripting (XSS)

Severity: High
CVSS Score: 8.7
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

OWASP

0 - Insufficient Information

Description

Cross-Site Scripting (XSS) is a type of security vulnerability that occurs when a web application allows untrusted data to be included in a web page that is then served to other users. This can enable an attacker to inject malicious scripts (usually written in JavaScript) into web pages viewed by other users. The malicious scripts can then execute in the context of the victim's browser, potentially leading to various security issues and attacks.

Stored or persistent XSS happens when the injected script is permanently stored on the target server and is later served to users when they request a particular web page. This can happen, for example, in comment sections, forums, or user profiles where the input is not properly sanitized and validated. When other users view the page with the stored script, their browsers execute it, potentially allowing an attacker to steal their session cookies, personal data, or perform other malicious actions.

Location

TBC

Proof of Concept

TBC

Recommendation

To mitigate cross-site scripting attacks, it is important to implement a range of security measures at different levels of the web application stack. Here are some recommendations to consider:

- Input validation: Ensure that all user input is properly validated and sanitized to remove any malicious code. This can be done through techniques such as input validation, encoding, and filtering.

- Output encoding: Any data that is displayed to the user should be properly encoded to prevent injection attacks. This can be achieved through techniques such as HTML entity encoding and/or URL encoding.
- Content Security Policy (CSP): Implement a CSP to restrict the types of content that can be loaded on a web page. This can help prevent the execution of malicious scripts or other types of content that can be used to attack users.
- Use secure coding practices: Ensure that your development team is following secure coding practices such as using the latest web application frameworks, libraries, and plugins, and keeping them up-to-date with security patches.
- Regular security testing: Perform regular security testing, including penetration testing, vulnerability scanning, and code reviews, to identify any potential vulnerabilities in your web application.

By following these recommendations, you can help mitigate cross-site scripting attacks and ensure the security of your web application and its users.

References

- http://projects.webappsec.org/w/page/13246920/Cross%20Site%20Scripting
- http://secunia.com/advisories/9716/
- https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

File Inclusion

Severity: High
CVSS Score: 7.7
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:N/SC:H/SI:N/SA:N

OWASP

0 - Insufficient Information

Description

Web applications occasionally use parameter values to store the location of a file which will later be required by the server.

An example of this is often seen in error pages, where the actual file path for the error page is stored in a parameter value – for example `example.com/error.php?page=404.php`.

A file inclusion occurs when the parameter value (ie. path to file) can be substituted with the path of another resource on the same server, effectively allowing the displaying of arbitrary, and possibly restricted/sensitive, files. The tool discovered that it was possible to substitute a parameter value with another resource and have the server return the contents of the resource to the client within the response.

Recommendation

It is recommended that untrusted data is never used to form a file location to be included.

To validate data, the application should ensure that the supplied value for a file is permitted. This can be achieved by performing whitelisting on the parameter value, and by matching it against a list of permitted files. If the supplied value does not match any value in the whitelist, then the server should redirect to a standard error page.

In some scenarios, where dynamic content is being requested, it may not be possible to perform validation against a list of trusted resources, therefore the list must also become dynamic (updated as the files change), or perform filtering to remove extraneous user input (such as semicolons, periods etc.) and only permit `a-z0-9`.

It is also advised that sensitive files are not stored within the webroot and that the user permissions are enforced by the directory are correct.

References

- https://www.owasp.org/index.php/PHP_File_Inclusion