Product: Suhail Sallahudin

Report: Test

PEN-DOC20240318204452

Shub, https//:www.shub_pentest.com

18-03-2024

Project Overview

Description

Hi im cust

Executive Summary

dsf

Summary of Findings Identified

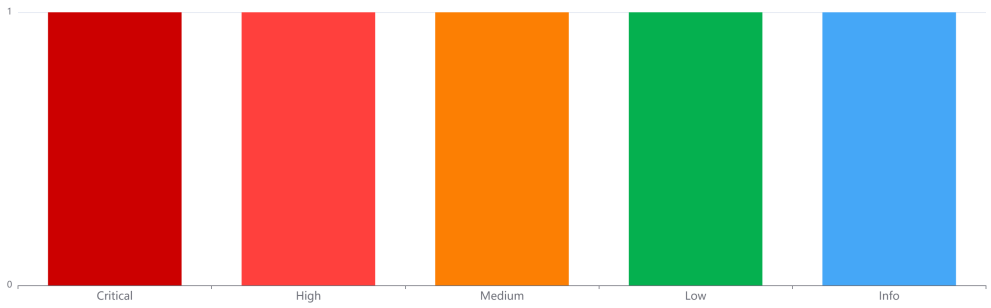

Figure 1: Executive Summary

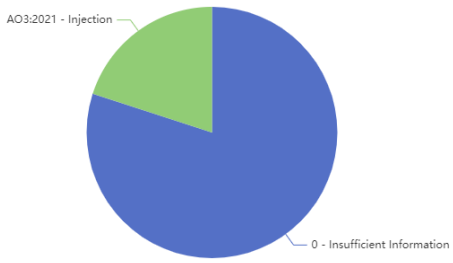

Figure 2: Breakdown by OWASP Categories

 # 1 Critical A backdoor file exists on the server

 # 2 High Reflected Cross-Site Scripting (XSS)

 # 3 Medium SQL Injection

 # 4 Info Allowed HTTP methods

 # 5 Low Application error message

Scope

In Scope

af

Out of Scope

afd

Methodology

fdas

Recommendations

fdas

Findings and Risk Analysis

A backdoor file exists on the server

Severity: Critical
CVSS Score: 9.2
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:H/SI:N/SA:N

OWASP

0 - Insufficient Information

Description

If a server has been previously compromised, there is a high probability that the cyber-criminal has installed a backdoor so that they can easily return to the server if required. One method of achieving this is to place a web backdoor or web shell within the web root of the web server. This will then enable the cyber-criminal to access the server through a HTTP/S session.

Although extremely bad practice, it is possible that the web backdoor or web shell has been placed there by an administrator so they can perform administrative activities remotely.

During the initial recon stages of an attack, cyber-criminals will attempt to locate these web backdoors or shells by requesting the names of the most common and well known ones.

By analysing the response, they are able to determine if a web backdoor or web shell exists. These web backdoors or web shells can then provide an easy path for further compromise of the server.

By utilising the same methods as the cyber-criminals, the tool was able to discover a possible web backdoor or web shell.

Recommendation

If manual confirmation reveals that a web backdoor or web shell does exist on the server, then it should be removed. It is also recommended that an incident response investigation be conducted on the server to establish how the web backdoor or web shell came to end up on the server.

Depending on the environment, investigation into the compromise of any other services or servers should be conducted.

References

- https://www.blackhat.com/presentations/bh-usa-07/Wysopal_and_Eng/Presentation/bh-usa-07-wysopal_and_eng.pdf

Reflected Cross-Site Scripting (XSS)

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

Client-side scripts are used extensively by modern web applications. They perform from simple functions (such as the formatting of text) up to full manipulation of client-side data and Operating System interaction.

Cross Site Scripting (XSS) allows clients to inject arbitrary scripting code into a request and have the server return the script to the client in the response. This occurs because the application is taking untrusted data (in this example, from the client) and reusing it without performing any validation or encoding.

Location

TBC

Proof of Concept

TBC

Recommendation

To remedy XSS vulnerabilities, it is important to never use untrusted or unfiltered data within the code of a HTML page.

Untrusted data can originate not only form the client but potentially a third party or previously uploaded file etc. Filtering of untrusted data typically involves converting special characters to their HTML entity encoded counterparts (however, other methods do exist, see references). These special characters include:

- `&`
- `<`
- `>`
- `&quot;`
- `&#x27;`

- /

An example of HTML entity encoding is converting `<` to `&lt;`. Although it is possible to filter untrusted input, there are five locations within an HTML page where untrusted input (even if it has been filtered) should never be placed:

1. Directly in a script.
2. Inside an HTML comment.
3. In an attribute name.
4. In a tag name.
5. Directly in CSS.

Each of these locations have their own form of escaping and filtering.

Because many browsers attempt to implement XSS protection, any manual verification of this finding should be conducted using multiple different browsers and browser versions.

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

SQL Injection

Severity: Medium
CVSS Score: 6.9
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:N/SC:L/SI:N/SA:N

OWASP

3 - Injection

Description

Due to the requirement for dynamic content of today's web applications, many rely on a database backend to store data that will be called upon and processed by the web application (or other programs). Web applications retrieve data from the database by using Structured Query Language (SQL) queries.

To meet demands of many developers, database servers (such as MSSQL, MySQL, Oracle etc.) have additional built-in functionality that can allow extensive control of the database and interaction with the host operating system itself. An SQL injection occurs when a value originating from the client's request is used within a SQL query without prior sanitisation. This could allow cyber-criminals to execute arbitrary SQL code and steal data or use the additional functionality of the database server to take control of more server components.

The successful exploitation of a SQL injection can be devastating to an organisation and is one of the most commonly exploited web application vulnerabilities.

This injection was detected as the tool was able to cause the server to respond to the request with a database related error.

Location

TBC

Proof of Concept

TBC

Recommendation

The only proven method to prevent against SQL injection attacks while still maintaining full application functionality is to use parameterized queries (also known as prepared statements). When utilising this method of querying the database, any value supplied by the client will be handled as a string value rather than part of the SQL query.

Additionally, when utilising parameterized queries, the database engine will automatically check to make sure the string being used matches that of the column. For example, the database engine will check that the user supplied input is an integer if the database column is configured to contain integers.

References

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- http://en.wikipedia.org/wiki/SQL_injection
- https://www.owasp.org/index.php/SQL_Injection
- http://projects.webappsec.org/w/page/13246963/SQL%20Injection
- http://www.w3schools.com/sql/sql_injection.asp
- http://unixwiz.net/techtips/sql-injection.html

Allowed HTTP methods

Severity: Info
CVSS Score: 0.0
CVSS Vector:

OWASP

0 - Insufficient Information

Description

There are a number of HTTP methods that can be used on a webserver (`OPTIONS`, `HEAD`, `GET`, `POST`, `PUT`, `DELETE` etc.). Each of these methods perform a different function and each have an associated level of risk when their use is permitted on the webserver.

A client can use the `OPTIONS` method within a request to query a server to determine which methods are allowed.

Cyber-criminals will almost always perform this simple test as it will give a very quick indication of any high-risk methods being permitted by the server. The tool discovered that several methods are supported by the server.

Recommendation

It is recommended that a whitelisting approach be taken to explicitly permit the HTTP methods required by the application and block all others.

Typically the only HTTP methods required for most applications are `GET` and `POST`. All other methods perform actions that are rarely required or perform actions that are inherently risky. These risky methods (such as `PUT`, `DELETE`, etc) should be protected by strict limitations, such as ensuring that the channel is secure (SSL/TLS enabled) and only authorised and trusted clients are permitted to use them.

References

- http://httpd.apache.org/docs/2.2/mod/core.html#limitexcept

Application error message

Severity: Low
CVSS Score: 0.0
CVSS Vector:

OWASP

0 - Insufficient Information

Description

Information Leakage is an application weakness where an application reveals sensitive data, such as technical details of the web application, environment, or user-specific data. Sensitive data may be used by an attacker to exploit the target web application, its hosting network, or its users.

In its most common form, information leakage is the result of one or more of the following conditions:

- A failure to scrub out HTML/Script comments containing sensitive information
- Improper application or server configurations
- Improper application error handling

Recommendation

- Ensure that the application source handles exceptions and errors in a such a way that no sensitive information is disclosed to the users
- Configure the application server to handle and log any exceptions that the application might yield

References

- http://projects.webappsec.org/w/page/13246936/Information%20Leakage