Security Maximale

Demo-Design-2 Report

Customer:
GotBreached Ltd.
2022-04-25
v 1.0

Contact:
Maximus Doe
+43 660 123 456 78
m.doe@securitymaximale.com

Table of Contents

Executive Summary

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse

Methodology and Scope

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

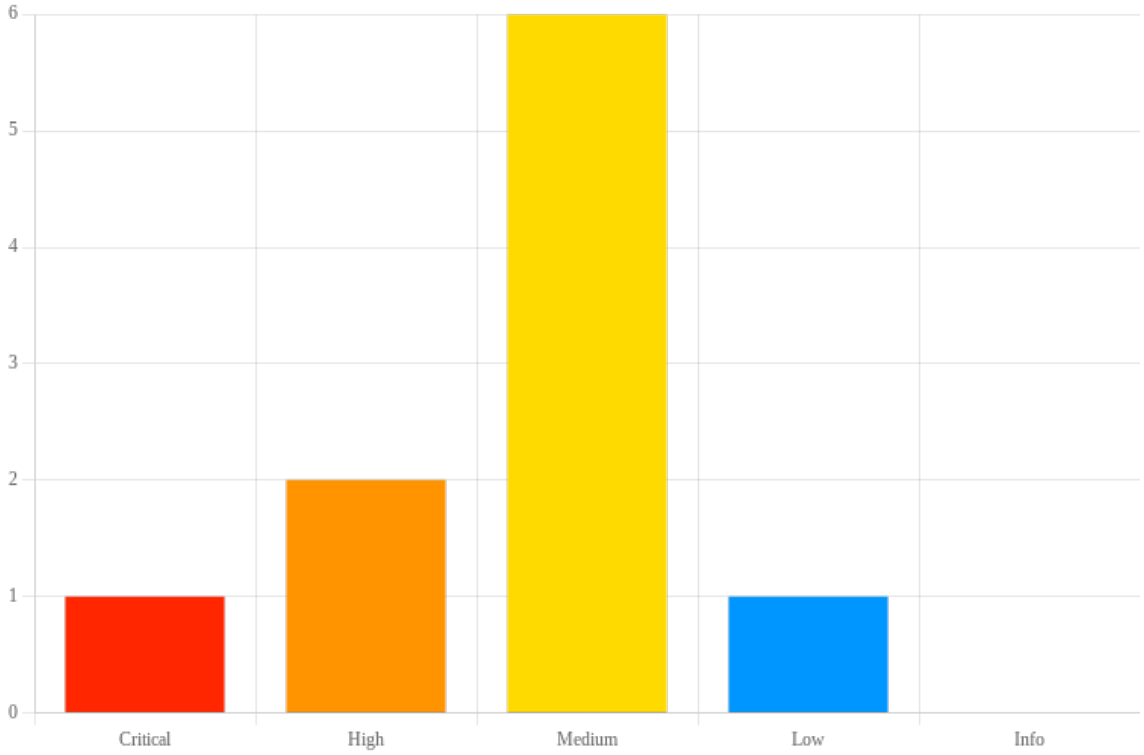
Fancy Methodology Graph

- Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
- Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
- Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

System	Description
10.0.0.1	System1
10.0.0.2	System2
10.0.0.3	System3
10.0.0.4	System3

Vulnerability Overview

In the course of this penetration test **1 Critical** , **2 High** , **6 Medium** und **1 Low** vulnerabilities were identified:



Distribution of identified vulnerabilities

Vulnerability	Criticality
SQL Injection (SQLi)	Critical
XML External Entity Injection (XXE)	High
Stored Cross-Site Scripting (XSS)	High
Insecure HTTP cookies	Medium
Cross-Site Request Forgery (CSRF)	Medium
Disclosure of sensitive data in URL parameters	Medium
Incorrectly configured HTTP security headers	Medium
User Enumeration	Medium
Untrusted TLS certificates	Medium
Session management weaknesses	Low

1. SQL Injection (SQLi)

Remediation Status:
Criticality: Critical
CVSS-Score: 9.8
Affects: example.com
Recommendation: Make sure that Prepared Statements and Stored Procedures (where possible) are used throughout the application. This prevents the originally intended action of an SQL statement from being manipulated by an attacker.

Overview

The web application processed user input in an insecure manner and was thus vulnerable to SQL injection. In an SQL injection attack, special input values in the web application are used to influence the application's SQL statements to its database. Depending on the database used and the design of the application, this may make it possible to read and modify the data stored in the database, perform administrative actions (e.g., shut down the DBMS), or in some cases even gain code execution and the accompanying complete control over the vulnerable server.

Description

We identified an SQL injection vulnerability in the web application and were able to access stored data in the database as a result.

SQL Injection is a common server-side vulnerability in web applications. It occurs when software developers create dynamic database queries that contain user input. In an attack, user input is crafted in such a way that the originally intended action of an SQL statement is changed. SQL injection vulnerabilities result from an application's failure to dynamically create database queries insecurely and to properly validate user input. They are based on the fact that the SQL language basically does not distinguish between control characters and data characters. In order to use a control character in the data part of an SQL statement, it must be encoded or escaped appropriately beforehand.

An SQL injection attack is therefore essentially carried out by inserting a control character such as `'` (single apostrophe) into the user input to place new commands that were not present in the original SQL statement. A simple example will demonstrate this process. The following SELECT statement contains a variable userId. The purpose of this statement is to get data of a user with a specific user id from the Users table.

```
sqlStmnt = 'SELECT * FROM Users WHERE UserId = ' + userId;
```

An attacker could now use special user input to change the original intent of the SQL statement. For example, he could use the string `' or 1=1` as user input. In this case, the application would construct the following SQL statement:

```
sqlStmnt = 'SELECT * FROM Users WHERE UserId = ' + ' or 1=1;
```

Instead of the data of a user with a specific user ID, the data of all users in the table is now returned to the attacker after executing the statement. This gives an attacker the ability to control the SQL statement in his own favor.

There are a number of variants of SQL injection vulnerabilities, attacks and techniques that occur in different situations and depending on the database system used. However, what they all have in common is that, as in the example above, user input is always used to dynamically construct SQL statements. Successful SQL injection attacks can have far-reaching consequences. One would be the loss of confidentiality and integrity of the stored data. Attackers could gain read and possibly write access to sensitive data in the database. SQL injection could also compromise the authentication and authorization of the web application, allowing attackers to bypass existing access controls. In some cases, SQL injection can also be used to gain code execution, allowing an attacker to gain complete control over the vulnerable server.

Recommendation

- Use prepared statements throughout the application to effectively avoid SQL injection vulnerabilities. Prepared statements are parameterized statements and ensure that even if input values are manipulated, an attacker is unable to change the original intent of an SQL statement.
- Use existing stored procedures by default where possible. Typically, stored procedures are implemented as secure parameterized queries and thus protect against SQL injections.
- Always validate all user input. Ensure that only input that is expected and valid for the application is accepted. You should not sanitize potentially malicious input.
- To reduce the potential damage of a successful SQL Injection attack, you should minimize the assigned privileges of the database user used according to the principle of least privilege.
- For detailed information and assistance on how to prevent SQL Injection vulnerabilities, see OWASP's linked SQL Injection Prevention Cheat Sheet.

Additional Information

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

2. XML External Entity Injection (XXE)

Remediation Status:
Criticality: High
CVSS-Score: 7.5
Affects: example.com
Recommendation: Disable support for external DTDs in the XML parsing library.

Overview

The web application processed XML documents in an insecure manner, which made it vulnerable to XML External Entity (XXE) Injection attacks. XXE Injection is a vulnerability in web applications that allows an attacker to interfere with the processing of XML documents by an XML parser. This attack can lead to disclosure of confidential data, denial of service, server-side request forgery, and other severe impact on the underlying system or other backend systems.

Description

We identified an XXE injection vulnerability in the web application. The XML parser allowed the definition of XXEs, which could create a malicious XML document. The XXE contained a URL that referenced an external domain. After the XXE was dereferenced by the parser, the web application interacted with this domain, which is evident from the DNS requests.

Extensible Markup Language (XML) is a standardized markup language and file format for storing, transmitting, and reconstructing arbitrary data. The language encodes data in a format that is readable by both humans and machines. The structure of an XML document is defined in the XML standard. The standard provides for a concept called an entity. Entities provide the ability to reference content that is provided remotely by a server or resides locally on the server. When the XML parser evaluates the XML document, the entity it contains is replaced with the referenced value. Entities are defined in so-called Document Type Definitions (DTDs).

DTDs define the structure and composition of an XML document. They can either be completely contained in the XML document itself, so-called internal DTDs, or they can be loaded from another location, so-called external DTDs. A combination of both variants is also possible. XML External Entities (XXE) are a special form of XML entities whose contents are loaded from outside the DTD in which they are declared.

An XXE is declared in the DTD with the SYSTEM keyword and a URI from where the content should be loaded. For example:

```
<!DOCTYPE dtd [ <!ENTITY xxe SYSTEM "http://syslifters.com" > ]>
```

The URI can also use the `file://` protocol scheme. Content can be loaded from local files as a result. For example:

```
<!DOCTYPE dtd [ <!ENTITY xxe SYSTEM "file:///path/to/local/file" >
]>
```

When evaluating XML documents, the XML parser replaces occurring XXEs with the contents by dereferencing the defined URIs. If the URI contains manipulated data, this could have serious consequences. An attacker can exploit this to perform server-side request forgery (SSRF) attacks and compromise the underlying server or other backend infrastructure. XXE injection vulnerabilities can also be exploited to cause service/application downtime (denial of service) or expose sensitive data such as local system files.

Recommendation

- The XML parser should be configured to use a local static DTD and not allow external DTDs declared in the XML document.
- We recommend limiting the functions of the XML parsing library to the minimum needed (see the documentation of the library used).
- User input should be validated before parsing if possible.
- Detailed information and help on preventing XXE injections can be found in the linked XML External Entity Prevention Cheat Sheet from OWASP.

Additional Information

- https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

3. Stored Cross-Site Scripting (XSS)

Remediation Status:
Criticality: High
CVSS-Score: 7.2
Affects: example.com
Recommendation: User input should be validated and filtered based on expected or valid input. It should be ensured that data is properly encoded contextually before it is included in HTTP responses.

Overview

At the time of testing, the web application stored user input unchecked and later included it in HTTP responses in an insecure manner. It was thus vulnerable to stored cross-site scripting (XSS) attacks.
Exploitation of Stored XSS vulnerabilities does not require user interaction, making them more dangerous than Reflected XSS vulnerabilities.

Description

We were able to identify a stored XSS vulnerability in the web application during testing. Due to incorrect validation and encoding of data, we were able to inject malicious scripts into the web application and store them persistently.

Cross-site scripting (XSS) is a common web security vulnerability where malicious scripts can be injected into web applications due to insufficient validation or encoding of data. In XSS attacks, attackers embed JavaScript code in the content delivered by the vulnerable web application.

The goal in stored XSS attacks is to place script code on pages visited by other users. Simply visiting the affected subpage is enough for the script code to be executed in the victim's web browser.

For an attack, malicious scripts are injected into the web application by the attacker and stored and included in subsequent HTTP responses of the application. The malicious script is ultimately executed in the victim's web browser and can potentially access cookies, session tokens or other sensitive information.

If the attack is successful, an attacker gains control over web application functions and data in the victim's context. If the affected user has privileged access, an attacker may be able to gain complete control over the web application.

Recommendation

- Ensure that all processed data is filtered as rigorously as possible. Filtering and validation should be done based on expected and valid inputs.
- Data should be encoded before the web application includes it in HTTP responses. Encoding should be done contextually, that is, depending on where the web application inserts data in the HTML document, the appropriate encoding syntax must be considered.
- The HTTP headers `Content-Type` (e.g. `text/plain`) and `X-Content-Type-Options: nosniff` can be set for HTTP responses that do not contain HTML and JavaScript.
- We recommend to additionally use a Content Security Policy (CSP) to control which client-side scripts are allowed and which are forbidden.
- Detailed information and help on preventing XSS can be found in the linked Cross-Site Scripting Prevention Cheat Sheet from OWASP.

Additional Information

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

4. Insecure HTTP cookies

Remediation Status:
Criticality: Medium
CVSS-Score: 6.5
Affects: example.com
Recommendation: Make sure that the configuration of all sensitive cookies is hardened and thus important cookie attributes like HttpOnly or Secure are set.

Overview

The issued HTTP cookies of the web application did not have the *HttpOnly* and/or the *Secure* cookie attribute set. If the *HttpOnly* attribute is not set, the affected cookie can be read or modified client-side using JavaScript. If the *Secure* attribute is not set, browsers also send the cookie over unencrypted HTTP connections. Insecurely configured cookies such as session cookies expand the potential attack surface of a web application. They make it easier for an attacker to exploit client-side vulnerabilities such as cross-site scripting (XSS) or compromise sessions by trivially intercepting cookies.

Description

HTTP is a stateless protocol, which means that it cannot distinguish requests from different users without an additional mechanism. To address this problem, it requires a session mechanism. The most commonly used mechanism for managing HTTP sessions in browsers is cookie storage. An HTTP cookie is a small record that a server sends to a user's web browser. The browser can store the cookie and send it back to the same server for subsequent requests. This can be used to implement sessions for the stateless HTTP protocol. An HTTP cookie can be used to distinguish requests from different users and to keep users logged in.

Cookies thus represent a frequent target for attackers. A web application should therefore harden the configuration of all sensitive cookies. This can be achieved by setting the *Secure* and *HttpOnly* cookie attributes. A cookie with the *Secure* attribute will only be sent to the server over HTTPS connections and never over an unsecured HTTP connection. A cookie with the *HttpOnly* attribute set is inaccessible to JavaScript and thus helps mitigate cross-site scripting (XSS) attacks. If an attacker is able to tap sensitive cookies such as session cookies, the attacker could take over user accounts and perform actions in the context of affected users. An attacker may also be able to gain complete control over all web application functions and data if they take over a user account with privileged access.

Auditors reviewed the set attributes of sensitive HTTP cookies of the web application. The following table provides an overview of the set attributes:

Cookie	Secure	HttpOnly
PHPSESSID	-	-

Recommendation

- Set the *Secure* attribute for sensitive cookies. This attribute instructs a browser to send the cookie only over an encrypted HTTPS connection to prevent session ID disclosure through man-in-the-middle attacks.
- If possible, also set the *HttpOnly* attribute for sensitive cookies. This attribute prevents the cookie from being accessed client-side via JavaScript. This can make session hijacking by XSS attacks more difficult.

5. Cross-Site Request Forgery (CSRF)

Remediation Status:
Criticality: Medium
CVSS-Score: 6.5
Affects: example.com
Recommendation: Make sure that randomly generated CSRF tokens with high entropy are included in all state-changing HTTP requests and validated in the backend.

Overview

The web application was vulnerable to Cross-Site Request Forgery (CSRF). CSRF is an attack that causes users to unknowingly send an HTTP request to a web application to which they are currently authenticated. Attackers can thereby partially bypass a web browser's same-origin policy and perform state-changing actions in the context of an affected user. Depending on the nature of the action, the attacker can gain complete control over the user's account. If the user account is administrative, CSRF may also be able to compromise the entire web application.

Description

We identified a CSRF vulnerability in the web application, allowing them to perform actions in the context of another user.

Cross-site request forgery (CSRF) is a web security vulnerability in which an attacker can trick an authenticated user into unknowingly sending a state-changing HTTP request to the vulnerable web application. In CSRF, an attacker assumes the victim's identity and access privileges to perform unwanted actions (e.g., change email address) on their behalf. Without appropriate CSRF protection, the web application has no way to distinguish between a request prepared by the attacker and a legitimate request from the victim.

Several prerequisites must be in place for a CSRF attack to take place. First, there must be an action in the web application that is relevant to an attacker and makes sense to exploit. For example, this could be a privileged action, such as changing a user's access permissions or changing a password. Another requirement is that there is no other mechanism besides cookie-based authentication to distinguish HTTP requests from different users. If the user is authenticated and thus has a valid session cookie, the web application thus has no way to distinguish between a malicious, subverted request from the attacker and a legitimate request from the victim. Last, it must be ensured that actions do not require specific parameters whose values an attacker cannot determine or predict. For example, if a user is asked to change his password,

the function is not vulnerable if an attacker needs to know the value of the existing password.

A common way to exploit CSRF vulnerabilities is through phishing emails. An attacker does this by preparing malicious links with the intention of foisting a state-changing request on the victim. The attacker then distributes the malicious links to victims via email. When a user opens the link in a web browser and is authenticated to it, the request is sent to the vulnerable web application. If successful, the attack causes an action with the victim's identity and privilege level.

Recommendation

- Check if the framework has built-in CSRF protection and use it. If not, ensure that all state-changing requests contain a randomly generated CSRF token with high entropy. Also ensure that CSRF tokens are properly validated on the backend.
- Consider various additional security measures:
  - For example, set the SameSite attribute for session cookies. Web browsers decide whether to include cookies in cross-site requests based on this attribute.
  - Use Custom Request Headers. By default, the browser's same-origin policy restricts JavaScript from submitting cross-site requests with custom request headers.
  - For highly sensitive actions, user interactions such as CAPTCHAs, one-time tokens, re-authentication, etc. can also be considered as additional CSRF protection.
- Detailed information and assistance on how to prevent CSRF vulnerabilities can be found in the linked Cross-Site Request Forgery Cheat Sheet from OWASP.

Additional Information

- https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

6. Disclosure of sensitive data in URL parameters

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9
Affects: example.com
Recommendation: To protect sensitive data from access by third parties, it should be sent in the body of an HTTP message, e.g. via POST request.

Overview

The web application sent sensitive data as URL parameters in HTTP requests. Data sent as URL parameters is stored in the browser cache and can potentially appear in various other places such as web server logs, referer headers or shared systems. Third parties could thus gain access to this sensitive data.

Description

The application sent sensitive data in the URL parameter "motiondata": This could expose the data in the following places:

- Referer Header
- Web Logs
- Shared Systems
- Browser History
- Browser Cache
- Shoulder Surfing

Recommendation

- The application should send all sensitive data in the body of an HTTP message, e.g. in the body of a POST request.
- Furthermore, the transmission should be secured via encrypted communication via HTTPS.

7. Incorrectly configured HTTP security headers

Remediation Status:
Criticality: Medium
CVSS-Score: 5.4
Affects: example.com
Recommendation: Follow best practices recommendations for configuring HTTP security headers and implement them for your web application if possible.

Overview

The web application did not have important HTTP security headers set or they were configured insecurely. HTTP security headers are a good way to increase the security of a web application. They can help make vulnerabilities such as cross-site scripting, clickjacking, information disclosure, and others more difficult or even prevent them altogether. Without proper HTTP security headers, the potential attack surface of a web application is larger and makes it easier for an attacker to exploit client-side vulnerabilities.

Description

We checked the HTTP security headers of the examined web application. The following table provides an overview of which headers were set correctly and which were not:

Host	Content-Security Policy (CSP)	Referrer-Policy	HTTP-Strict-Transport-Security HSTS)	X-Content-Type-Options	X-Frame-Options	Permissions-Policy	X-XSS-Protection
example.com	-	-	X	-	-	-	

Modern browsers support several HTTP security headers that can increase the security of web applications against client-side vulnerabilities such as clickjacking, cross-site scripting, and other common attacks. HTTP Security headers are response headers that specify whether and which security measures should be enabled or disabled in the web browser. These HTTP headers are exchanged between a browser and a server and specify the security-related details of HTTP communication. Below is a brief description and overview of the most important current HTTP security headers:

- **Content Security Policy**. The Content Security Policy (CSP) HTTP header allows fine-grained control over what resources a browser is allowed to obtain

resources from. The CSP header is a very effective measure to prevent the exploitation of cross-site scripting (XSS) vulnerabilities.

- **Referrer Policy**. The `Referrer-Policy` header determines how and when browsers transmit the HTTP Referer (sic) header. In the Referer header, a browser informs a target page about the origin of an HTTP request, for example, when a user navigates to a specific page via a link or loads an external resource.

- **HTTP Strict Transport Security (HSTS)**. With the HSTS header, a web page instructs the browser to connect only over HTTPS. All unencrypted HTTP requests are transparently redirected in the process. TLS and certificate-related errors are also handled more strictly by preventing users from bypassing the error page.

- **X-Content-Type-Options**. The `X-Content-Type-Options` header specifies that browsers will only load scripts and stylesheets if the server specifies the correct MIME type. Without this header, there is a risk of MIME sniffing. This means that browsers will misrecognize files as scripts and stylesheets, which could lead to XSS attacks.

- **X-Frame-Options** `X-Frame-Options` are used to determine if and in which form the web page can be embedded in an iframe. Clickjacking is a viable attack that can exploit such embedding in an `iframe`. In such an attack, an attacker overlays the rendering of a legitimate page to then cause users to perform seemingly innocuous interactions (e.g., mouse clicks and/or keystrokes).

- **Permissions policy** Permissions policy allows web developers to selectively enable, disable, and modify the behavior of certain features and APIs in the browser. `Permissions-Policy` is similar to Content Security Policy, but controls specific functions of the browser rather than security behavior.

- **X-XSS-Protection** `X-XSS-Protection` is a feature that prevents pages from loading when a browser detects Reflected Cross-Site Scripting (XSS) attacks. This header is obsolete when using modern browsers, provided that a secure content security policy has been defined.

Recommendation

- Do not allow the web page to be included in a frame. Set `X-Frame-Options: DENY` for this. Alternatively you can restrict this setting to the same-origin with `X-Frame-Options: SAMEORIGIN`.
- Set the header `X-XSS-Protection` explicitly with `X-XSS-Protection: 1; mode=block`.
- Prevent the browser from guessing the MIME type based on the content of the resource. Sets the `X-Content-Type-Options` header with the `nosniff` option.

- Restrict the `referrer policy` to prevent potentially sensitive information from being exposed to third party sites. You should define the header as follows: `Referrer-Policy: strict-origin-when-cross-origin`.
- Configure the `Strict-Transport-Security` header so that your web application can only be accessed over a secured HTTPS connection. You should set the header like this: `Strict-Transport-Security: max-age=63072000; includeSubDomains; preload`.
- If possible, define a Content Security Policy (CSP) for your web application CSP is an additional security measure that can make it much more difficult to exploit client-side vulnerabilities. Details on how to configure it securely can be found in the resources.
- Restrict the use of sensitive browser features such as the camera, microphone or speaker using 'Permissions Policy' headers.

Additional Information

- https://infosec.mozilla.org/guidelines/web_security#content-security-policy

8. User Enumeration

Remediation Status:
Criticality: Medium
CVSS-Score: 5.3
Affects: example.com
Recommendation: Identify all application attack surfaces relevant to User Enumeration and ensures that the web application always returns generic error messages when invalid credentials are entered.

Overview

The web application was vulnerable to a user enumeration vulnerability. User enumeration is a common vulnerability in web applications that occurs when an attacker can use brute force techniques to determine valid user accounts in a system. Although user enumeration is a low risk in itself, it still provides an attacker with valuable information for follow-up attacks such as in brute force and credential stuffing attacks or in social engineering campaigns.

Description

We were able to identify a user enumeration vulnerability in the web application, allowing us to determine valid user accounts using brute force techniques.

Often, as a result of a faulty configuration or design decision, web applications indicate when a user already exists in the system. Two of the most common areas where this occurs are the login page or the "forgot password" feature of a web application. One example is when a user enters incorrect credentials, they receive information that the password they entered was incorrect. The information obtained can now be used by an attacker to determine whether or not a particular username already exists. By trial and error, an attacker can use it to determine a list of valid usernames.

Once an attacker has such a list, they can address these user accounts in new attacks to obtain valid credentials. In its simplest form, an attacker could perform a brute force attack. In this, an attacker tries to guess a user account's credentials by automatically trying through passwords. Often very large word lists containing frequently used passwords are used for this purpose. An attacker could also use determined usernames to search past data leaks for passwords. Credentials from data leaks, consisting of pairs of usernames and passwords, can be reused by an attacker in an automated attack. This particular form of brute force attack, is also known as credential stuffing. Alternatively, an attacker can use usernames in the course of social engineering campaigns to contact users directly.

Recommendation

- Ensure that the web application always returns generic error messages when invalid usernames, passwords, or other credentials are entered. Identifies all relevant attack surfaces of the application for this purpose.
- If the application defines usernames itself, user enumeration can be effectively prevented. The prerequisite for this is that user names are randomly generated so that they cannot be guessed.
- The application can also use email addresses as usernames. If the username is not yet registered, an email message will contain a unique URL that can be used to complete the registration process. If the username exists, the user receives an email message with a URL to reset the password. In either case, an attacker cannot infer valid user accounts.
- As an additional security measure, you could delete default system accounts as well as test accounts or rename them before releasing the system to production.

9. Untrusted TLS certificates

Remediation Status:
Criticality: Medium
CVSS-Score: 4.8
Affects: example.com
Recommendation: Ensure that TLS certificates used are universally valid and trusted. Acquire new certificates for the affected services, if necessary. Also, follow best practices recommendations for secure TLS server configuration.

Overview

Communication with the application at the transport layer level was not sufficiently protected due to untrusted TLS certificates. TLS is used by many protocols to ensure the confidentiality and integrity of communication between two endpoints. If web browsers do not trust an application's TLS certificate, the application may be vulnerable to man-in-the-middle attacks and thus susceptible to eavesdropping or tampering with traffic. Insufficient protection at the transport layer may allow communications between two parties to be compromised by an untrusted third party. An attacker could thus obtain sensitive data (e.g., credentials) if necessary. In the event of a successful attack, an attacker could gain complete control over all functions and data of the application by compromising a privileged user account.

Description

Transport Layer Security (TLS) is the successor to the now obsolete as well as insecure Secure Sockets Layer (SSL) protocol. TLS is a cryptographic protocol developed for secure, encrypted communication between two or more parties. The protocol is used in a wide variety of areas, including e-mail, instant messaging, and voice-over-IP. The best known use of TLS is on the Web, where it ensures secure communication over HTTPS. Primarily, TLS aims to ensure confidentiality, integrity, but also authenticity through the use of certificates, between two or more parties.

With TLS, the establishment of a secure connection takes place in several steps. Client and server agree on the use of TLS in the first step. This is done either by selecting a specific port (e.g. 443 for HTTP) or by making a protocol-specific request to the server (e.g. STARTTLS for SMTP). A handshake procedure then begins, in which the client and server negotiate various parameters for the security of the communication link. The handshake begins with the client and server agreeing on a respective supported cipher suite, consisting of the symmetric cipher and hash function. The server then issues a digital certificate. The certificate contains, among other things, the server name, the issuing certificate authority (CA), and the server's data asymmetric key. Once the client has verified the validity of the certificate, it generates a symmetric

session key for the secure connection. This is done either by the client deriving a key from a random number. The client encrypts the random number with the server's data key and sends the result to the server. The server can use the private key to read the result and also derive the session key. However, the client and server could also use the Diffie-Hellman algorithm to securely agree on a random session key. Diffie-Hellman also offers the advantage of perfect forward secrecy (PFS). PFS prevents subsequent decryption once the server's private key is known. Session keys are not exchanged and thus cannot be reconstructed.

The security of TLS-secured communication is based primarily on the trustworthiness of the digital certificate. If the trustworthiness is not given, for example because the certificate has expired, it contains an incorrect host name or it is a self-signed certificate, no secure key exchange between two endpoints can be guaranteed from the outset. In some circumstances, the communication between two parties could be compromised by an untrusted third party in the course of a man-in-the-middle attack. For example, an attacker could gain access to sensitive data or inject malicious data into the encrypted data stream to compromise either the client or the server.

We reviewed the TLS certificates of the applications in scope and found untrusted certificates for the following applications:

host	expired	expiring soon	incorrect host name	incomplete certificate chain	self-signed certificate
example.com: 443	X	-	-	-	

Recommendation

- Acquire new certificates for services that do not have trusted TLS certificates.
- Generate sufficiently strong asymmetric keys with at least 2048 bits for certificates and protect the private key.
- Use only modern cryptographic hash algorithms such as SHA-256.'
- Make sure that the certificate contains the fully qualified name of the server. The following should also be considered when creating the certificate:
    - Consider whether the "www" subdomain should also be included.
    - Do not include unqualified host names in the certificate.
    - Do not include IP addresses.
    - Do not include internal domain names.
- Create and use wildcard certificates only when there is a real need. Do not use wildcard certificates for convenience.
- Choose an appropriate certificate authority that is trusted by all major browsers. For internal applications, an internal CA can be used. However, ensure that all

users have imported the internal CA certificate and thus trust certificates issued by that CA.
- Check the TLS configuration, including certificates, at regular intervals and adjust as necessary. There are a number of online tools (such as SSLabs, sslyze, etc) that you can use to quickly perform the check.
- For more information and help on TLS certificates, see the linked Transport Layer Protection Cheat Sheet from OWASP.

Additional Information

- https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

10. Session management weaknesses

Remediation Status:
Criticality: Low
CVSS-Score: 3.6
Affects: example.com
Recommendation: Users should be logged out automatically after a certain period of inactivity.

Overview

We were able to identify weaknesses in the web application's session management. The users' sessions were usable without time restrictions and therefore did not require re-authentication at any time. People with access to a computer system could exploit this situation if another user had not explicitly logged out of the application beforehand.

Description

We could determine that user sessions were usable without time restrictions. This could allow attackers to take over user sessions that were not explicitly logged out beforehand.

This could be possible, for example, by allowing a third person to operate a user's computer in which a session is still active. In addition, it could be possible for attackers to reuse session tokens when they become known (e.g. via log files; locally or on proxy servers, etc.).

Recommendation

- User sessions in web applications should time out automatically after a certain period of inactivity.
- Depending on the criticality of the user authorization and the application, the timeout could be approximately between one hour and one day.

List of Changes

Version	Date	Description	Author
0.1	2022-04-22	Draft	M. Doe
0.9	2022-04-22	Review	C. Doe
1.0	2022-04-25	Final Report	M. Doe

Disclaimer

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Imprint

Security Maximale GmbH
Example Street 47 | 4711 Example
FN 12345 v | District Court Example