

PENTEST REPORT

Safebank Web Application Testing

Safebank Stakeholders
Attn. Safebank Project Team

New York City, July 30, 2024

Report Version: 1.0

Security Maximize GmbH
Sesame Street 47
4711
FN 12345 v | D.C.

Table of Contents

List of Figures	3
1 Document Control	4
1.1 Team	4
1.2 List of Changes	4
2 Executive Summary	5
2.1 Overview	5
2.2 Identified Vulnerabilities	5
3 Methodology	7
3.1 Objective	7
3.2 Scope	7
3.3 User Accounts and Permissions	7
4 Findings	8
H1: Lack Rate Limiting	8
I1: Enabled HTTP TRACE Method	12
I2: Improper Error Handling	14
I3: Incorrectly Configured HTTP Security Headers	16
I4: Information Disclosure	18
I5: Vulnerable and Outdated Components	19

List of Figures

Figure 1 - Distribution of identified vulnerabilities	6
---	---

1 Document Control

1.1 Team

Contact	Details	Role
Charalampos Spanias	E-Mail: spaniascharalampos@gmail.com	Pentester

1.2 List of Changes

Version	Description	Date
---------	-------------	------

2 Executive Summary

2.1 Overview

When performing the internal penetration test, a total of 1 HIGH and 5 INFORMATIONAL vulnerabilities were identified. These flaws mostly originated from poor security configurations. During the testing, due to the lack of rate-limiting protections, a Denial of Service (DoS) attack was achieved, completely shutting down the web application. This could potentially have severe financial and reputational consequences for Safebank. Due to time-restrictions many of other potential attacks related to rate-limiting, but not proven here, were highlighted as INFORMATIONAL findings.

2.2 Identified Vulnerabilities

#	CVSS	Description	Page
H1	7.5	Lack Rate Limiting	8
I1	0.0	Enabled HTTP TRACE Method	12
I2	0.0	Improper Error Handling	14
I3	0.0	Incorrectly Configured HTTP Security Headers	16
I4	0.0	Information Disclosure	18
I5	0.0	Vulnerable and Outdated Components	19

Vulnerability Overview

In the course of this penetration test **1 High** and **5 Info** vulnerabilities were identified:

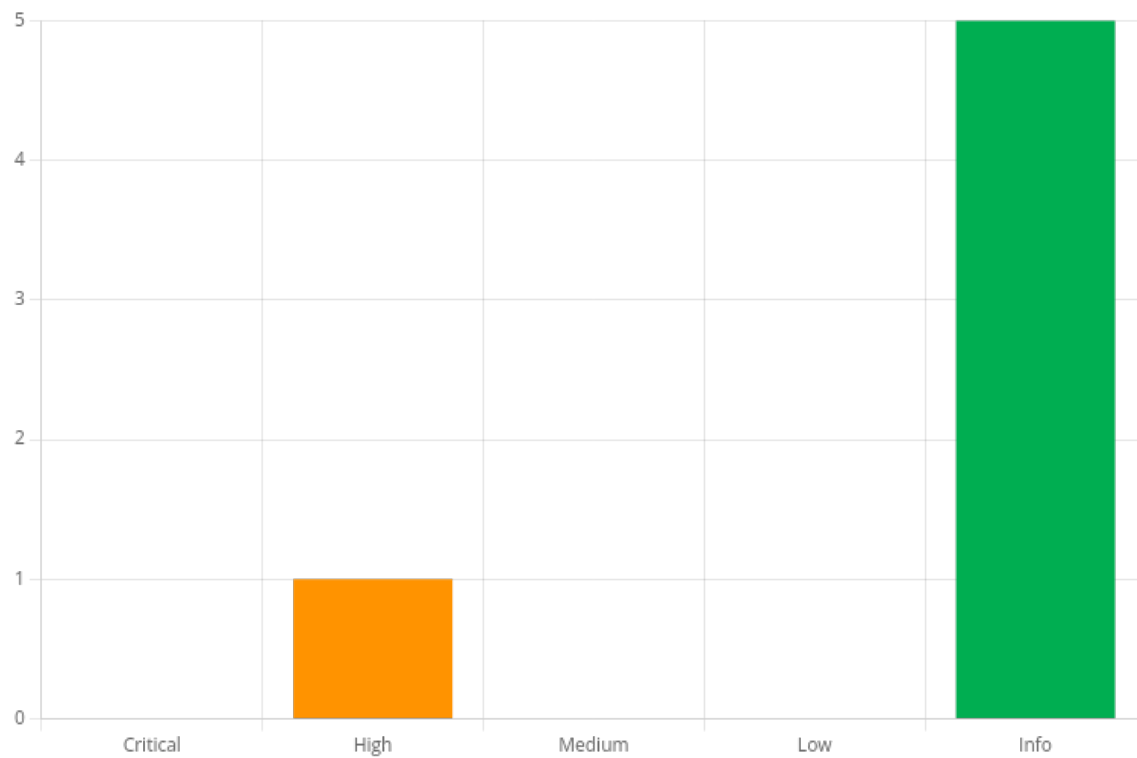


Figure 1 - Distribution of identified vulnerabilities

3 Methodology

The Web Security Testing Guide (WSTG) Project produces the premier cybersecurity testing resource for web application developers and security professionals. The WSTG is a comprehensive guide to testing the security of web applications and web services. Created by the collaborative efforts of cybersecurity professionals and dedicated volunteers, the WSTG provides a framework of best practices used by penetration testers and organizations all over the world.

3.1 Objective

The objective of this penetration test was to identify and evaluate security vulnerabilities within the Safebank web application, a web-based platform accessible at `https://safebank.local`. The test aimed to uncover potential weaknesses that could be exploited by malicious actors to compromise the confidentiality, integrity, or availability of the application and its data.

3.2 Scope

The scope of this test was defined by the project team as ``http://safebank.local/*``. Any additional subdomain found was also added to the scope. The test's duration was scheduled from **Jul 29, 2024 to Jul 30, 2024 (1 day for testing, 1 day for reporting)**.

System	Description
10.0.0.3	<code>https://www.safebank.local</code>

3.3 User Accounts and Permissions

- **Kali workstation:** x7331
- **Windows workstation:** x7331

4 Findings

H1: Lack Rate Limiting	
Score	7.5 (High)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Target	<ul style="list-style-type: none"> • http://safebank.local/* • http://safebank.local/register • http://safebank.local/login
References	<ul style="list-style-type: none"> • https://cwe.mitre.org/data/definitions/307.html • https://cwe.mitre.org/data/definitions/770.html • https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

Overview

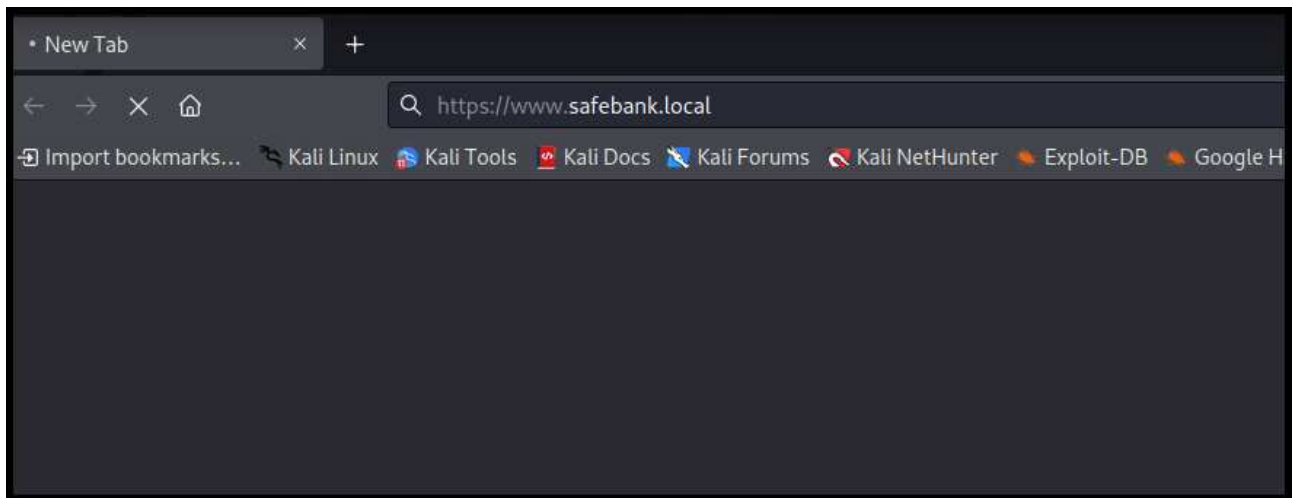
Rate limiting is the process of controlling traffic rate from and to a server or component. It can be implemented on infrastructure as well as on an application level. Rate limiting can be based on (offending) IPs, on IP block lists, on geolocation, etc.

The absence of rate limiting allows an attacker to send an unlimited number of requests to the server. This can be exploited in various ways, including:

- Brute Force Attacks (BFAs): Attackers can automate the process of guessing passwords or other sensitive data, significantly increasing the likelihood of success.
- Denial-of-Service (DoS) Attacks: Attackers can overwhelm the server with a high volume of requests, potentially rendering the application unavailable to legitimate users.
- Data Scraping: Automated scripts can be used to scrape large volumes of data from the application, which may be sensitive or proprietary.

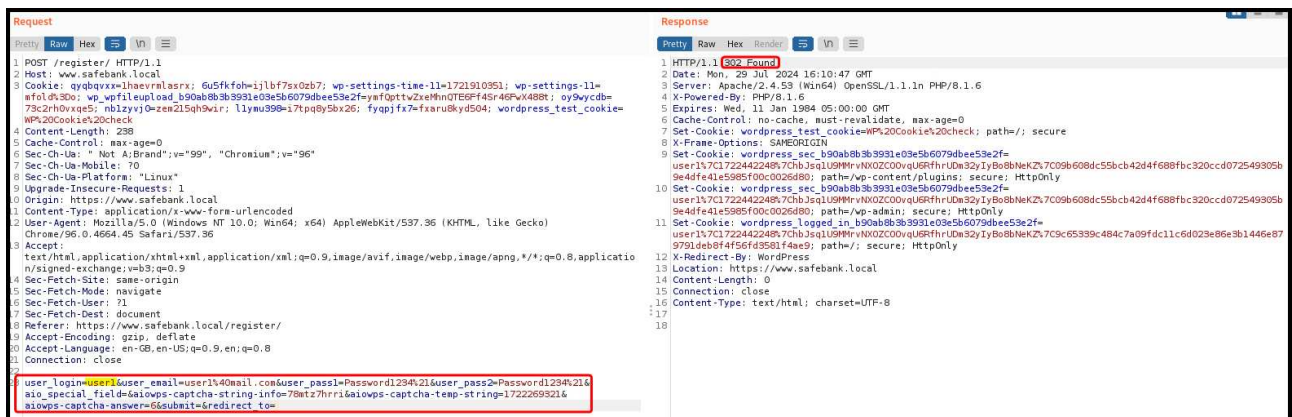
Details

During the security assessment of the target web application, it was observed that there are no rate limiting protections in place. As a result it was possible to perform a DoS attack against the application.



A common protection against rate-limiting related attacks on registration and login pages is the implementation of multi-factor authentication (MFA) and the use of CAPTCHA. While both were present on the `/register` directory, both were incorrectly configured.

The same CAPTCHA could be used to register different users, making this protection irrelevant.



In addition, due to the lack of rate-limiting protection it might be possible to brute-force a user's OTP. In this case, this effort resulted in a DoS attack. Finally, another common measure against BFAs is the implementation of an account lockout policy which was not present in this case.

- Implement an appropriate account lockout policy: The most obvious way to block BFAs is to simply lock out accounts after a defined number of incorrect password attempts. Account lockouts can last a specific duration, such as one hour, or the accounts could remain locked until manually unlocked by an administrator.
- Device cookies: You may also consider locking out authentication attempts from known and unknown browsers or devices separately. The Slow Down Online Guessing Attacks with Device Cookies article proposes protocol for lockout mechanism based on information about if specific browser have been already used for successful login. The protocol is less susceptible to DoS attacks than plain account locking out and yet effective and easy to implement.

I1: Enabled HTTP TRACE Method	
Score	0.0 (Info)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Target	https://safebank.local/*
References	<ul style="list-style-type: none"> • https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/06-Test_HTTP_Methods • https://owasp.org/www-community/attacks/Cross_Site_Tracing

Overview

The **TRACE** method is primarily used for debugging and diagnostics, allowing clients to see what is being received at the server end of the request chain and use that data for testing and troubleshooting. However, when enabled in a production environment, this method can be exploited by attackers to carry out certain types of attacks, such as Cross-Site Tracing (XST).

The HTTP **TRACE** method echoes the received request so that a client can see what changes or additions might have been made by intermediate servers. This can inadvertently result in information disclosure, exposing sensitive information from HTTP headers, such as cookies and authentication tokens.

Details

During the security assessment of the target web application, it was observed that the HTTP **TRACE** method was enabled.

Request	Response
<pre> 1 TRACE / HTTP/1.1 2 Host: www.safebank.local 3 Cookie: qyqbqvx=1haevrmlasrx; 6u5fkfoh=ijlbf7sx0zb7; wp-settings-time-11=1721910351; wp-settings-11= =mfoldh3do; wp_wfileupload_b90ab8b3b3931e03e5b6079dbee53e2f=yafOpttwZxeMhNQT6FF45r46FwX488t; oy9wycdb=73c2rh0vxqe5; nblyvj0=zem215qh9wir; llymu398=17tpq8y5bx26; fyqpfxf7=fxaru8kyd504 4 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="96" 5 Sec-Ch-Ua-Mobile: 70 6 Sec-Ch-Ua-Platform: "Linux" 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36 9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 10 Sec-Fetch-Mode: navigate 11 Sec-Fetch-Site: same-origin 12 Sec-Fetch-User: 71 13 Sec-Fetch-Dest: document 14 Referer: https://www.safebank.local/ 15 Accept-Encoding: gzip, deflate 16 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8 17 Connection: close 18 19 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Mon, 29 Jul 2024 15:26:13 GMT 3 Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.1.6 4 Connection: close 5 Content-Type: message/http 6 Content-Length: 975 7 8 TRACE / HTTP/1.1 9 Host: www.safebank.local 10 Cookie: qyqbqvx=1haevrmlasrx; 6u5fkfoh=ijlbf7sx0zb7; wp-settings-time-11=1721910351; wp-settings-11=mfoldh3do; wp_wfileupload_b90ab8b3b3931e03e5b6079dbee53e2f=yafOpttwZxeMhNQT6FF45r46FwX488t; oy9wycdb=73c2rh0vxqe5; nblyvj0=zem215qh9wir; llymu398=17tpq8y5bx26; fyqpfxf7=fxaru8kyd504 11 Sec-Ch-Ua: " Not A;Brand"; v="99", "Chromium"; v="96" 12 Sec-Ch-Ua-Mobile: 70 13 Sec-Ch-Ua-Platform: "Linux" 14 Upgrade-Insecure-Requests: 1 15 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36 16 Accept: text/html,application/xhtml+xml,application/xml; q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 17 Sec-Fetch-Mode: navigate 18 Sec-Fetch-Site: same-origin 19 Sec-Fetch-User: 71 20 Sec-Fetch-Dest: document 21 Referer: https://www.safebank.local/ 22 Accept-Encoding: gzip, deflate 23 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8 24 Connection: close 25 26 </pre>

Recommendation

The `TRACE` method should be disabled on production web servers. This can typically be done by modifying the server configuration.

For an Apache web server this can be done by adding the following directive to the Apache configuration file (`httpd.conf` or `apache2.conf`):

```
Copy code  
TraceEnable off
```

I2: Improper Error Handling	
Score	0.0 (Info)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Target	https://safebank.local/register
References	<ul style="list-style-type: none">• https://owasp.org/www-community/Improper_Error_Handling• https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

Overview

Verbose error messages can disclose sensitive information such as stack traces, database errors, file paths, and configuration details. This information can assist attackers in increasing their attack surface and understanding the structure and behavior of the application, making it easier to craft specific attacks.

Details

During the security assessment of the target web application, it was observed that the application displayed verbose error messages. More specifically, on the application's registration page (/register), it revealed if the chosen username existed or not.



Register

Error: This username is already registered. Please choose another one.

Username

test

This behavior can be leveraged by an attacker to enumerate valid web application users, and in combination with the lack of rate-limiting protections as well as a non-existing account lockout policy, a brute-force attack could then be launched in an attempt to obtain valid passwords.

Recommendation

To mitigate this risk, it is recommended to configure the application and server to display generic error messages to end users while logging detailed error information internally. An example can be found below.

An **error** occurred **while** processing your request.

I3: Incorrectly Configured HTTP Security Headers

Score	0.0 (Info)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Target	https://safebank.local/*
References	<ul style="list-style-type: none"> https://cheatsheetseries.owasp.org/cheatsheets/HTTP-Headers-Cheat-Sheet.html#security-headers https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy-Cheat-Sheet.html

Overview

The web application did not have important HTTP security headers set. 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.

In addition the following headers were set disclosing configuration information that could be leveraged from a threat actor.

- Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.1.6
- X-Powered-By: PHP/8.1.6

Details

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	Referrer-Policy	Strict-Transport-Security	X-Content-Type-Options	X-Frame-Options
https://safebank.local	-	-	-	-	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 Referrer 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.

Recommendation

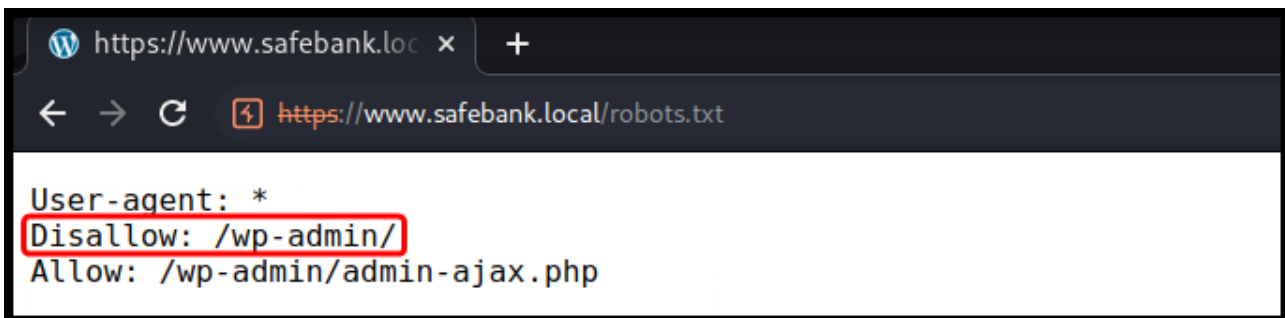
- Prevent the browser from guessing the MIME type based on the content of the resource. Sets the X-Content-Type-Options header with the nosniff option.
- Restrict the referrer policy to prevent potentially sensitive information from being exposed to third party sites. You should define the header as 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.
- Disable the Server and X-Powered-By headers.

I4: Information Disclosure

Score	0.0 (Info)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Target	https://safebank.local/robots.txt
References	https://portswigger.net/kb/issues/00600600_robots-txt-file

Overview

During the web application enumeration phase, the `/robots.txt` file was found disclosing the sensitive `/wp-admin` directory.



```
https://www.safebank.local/robots.txt
User-agent: *
Disallow: /wp-admin/
Allow: /wp-admin/admin-ajax.php
```

Details

The file `robots.txt` is used to give instructions to web robots, such as search engine crawlers, about locations within the web site that robots are allowed, or not allowed, to crawl and index.

The presence of the `robots.txt` does not in itself present any kind of security vulnerability. However, it is often used to identify restricted or private areas of a site's contents. The information in the file may therefore help an attacker to map out the site's contents, especially if some of the locations identified are not linked from elsewhere in the site. If the application relies on `robots.txt` to protect access to these areas, and does not enforce proper access control over them, then this presents a serious vulnerability.

Recommendation

The `robots.txt` file is not itself a security threat, and its correct use can represent good practice for non-security reasons. You should not assume that all web robots will honor the file's instructions. Rather, assume that attackers will pay close attention to any locations identified in the file. Do not rely on `robots.txt` to provide any kind of protection over unauthorized access.

I5: Vulnerable and Outdated Components

Score	0.0 (Info)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Target	https://safebank.local/*
References	<ul style="list-style-type: none"> • https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ • https://endoflife.date/php • https://endoflife.date/apache • https://endoflife.date/php

Overview

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component).

There are automated tools to help attackers find unpatched or misconfigured systems and when identified could be leveraged within an attacked chain using known vulnerabilities with publicly available exploits.

Details

During the web application information gathering phase, multiple technologies used by it were out-of-date. More specifically:

1. OpenSSL 1.1.1n appears to be outdated and have reached EOL on 11/09/2018 (latest is 3.3.1).
2. Apache 2.4.53 appears to be outdated (latest is 2.4.64).
3. PHP 8.1.6 appears to be outdated (latest is 8.3.9).

```
$ whatweb https://www.safebank.local
https://www.safebank.local [200 OK] Apache[2.4.53],
Cookies[6u5fkfoh,qyqbqvxx,wp_wpfileupload_b90ab8b3b3931e03e5b6079dbee53e2f],
Country[RESERVED][ZZ], HTML5, HTTPServer[Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.1.6],
HttpOnly[6u5fkfoh,qyqbqvxx], IP[10.0.0.3], JQuery[3.7.1], MetaGenerator[WordPress 6.6],
OpenSSL[1.1.1n], PHP[8.1.6], Script[importmap,module], Title[SafeBank], UncommonHeaders[link],
WordPress[6.6], X-Frame-Options[SAMEORIGIN], X-Powered-By[PHP/8.1.6]
```

Moreover, the Apache 2.4.53 component has *multiple known vulnerabilities* with publicly available proof of concepts (PoCs).

Show15

Search:apache 2.4.

Date	D	A	V	Title	Type	Platform	Author
2023-04-01	↓		✓	Apache 2.4.x - Buffer Overflow	WebApps	Multiple	Sunil Iyengar
2021-11-11	↓		✓	Apache HTTP Server 2.4.50 - Remote Code Execution (RCE) (3)	WebApps	Multiple	Valentin Lobstein
2021-10-25	↓		✗	Apache HTTP Server 2.4.50 - Remote Code Execution (RCE) (2)	WebApps	Multiple	ThelastVwV
2021-10-13	↓		✓	Apache HTTP Server 2.4.50 - Path Traversal & Remote Code Execution (RCE)	WebApps	Multiple	Lucas Souza
2021-10-06	↓	📄	✓	Apache HTTP Server 2.4.49 - Path Traversal & Remote Code Execution (RCE)	WebApps	Multiple	Lucas Souza
2019-04-08	↓		✗	Apache 2.4.17 < 2.4.38 - 'apache2ctl graceful' 'logrotate' Local Privilege Escalation	Local	Linux	cfreal
2017-09-18	↓		✗	Apache < 2.2.34 / < 2.4.27 - OPTIONS Memory Leak	WebApps	Linux	Hanno Bock
2016-12-12	↓	📄	✗	Apache 2.4.23 mod_http2 - Denial of Service	DoS	Linux	Jungun Baek
2016-02-01	↓		✗	Apache 2.4.7 + PHP 7.0.2 - 'openssl_seal()' Uninitialized Memory Code Execution	Remote	PHP	akat1
2015-12-18	↓		✓	Apache 2.4.17 - Denial of Service	DoS	Windows	rUnVIRuS
2014-07-21	↓		✗	Apache 2.4.7 mod_status - Scoreboard Handling Race Condition	DoS	Linux	Marek Kroemeke

Showing 1 to 11 of 11 entries (filtered from 46,082 total entries)

FIRSTPREVIOUS1NEXTLAST

Recommendation

Establish a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, OWASP Dependency Check, retire.js, etc. Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.
- Every organization must ensure an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.