



SYSLIFTERS

Sample Report: Web

Pentest for **Security Maximale GmbH**
2022-08-29
v 1.0

Contact:
Aron Molnar
+436609234060
aron@syslifters.com

Content

Management Summary	2
Here is the report. What now?	2
Scope and Duration	3
Vulnerability Overview	4
Vulnerability Details	6
Takeover of user accounts through password reset (Critical)	6
Stored Cross-Site Scripting (XSS) (Critical)	8
Disclosure of passwords via Path Traversal (High)	11
Forwarding users through Open Redirect (Medium)	13
Disclosure of internal information (Medium)	15
Session management weaknesses (Low)	16
List of Changes	17
Disclaimer	17
Imprint	17



Management Summary

In the course of the security audit, we were able to take over administrative accounts and customer accounts of the online shop. This allowed us to view and edit customer data, including payment information and customer orders.

This was achieved in two ways: On the one hand, the function for resetting passwords was not sufficiently secured and was also available to the shop's customers. This allowed customers to reset the passwords of administrators. On the other hand, malicious JavaScript functions could be injected via the customers' status messages, which allowed administrative user sessions to be taken over.

In addition, arbitrary files could be read from the web server via the function for selecting the language to be delivered. These included configuration files with passwords.

Other, less critical vulnerabilities, such as the forwarding of foreign users to untrustworthy websites, the disclosure of internal information, as well as weaknesses in session management, should be addressed in a continuous improvement process.

Here is the report. What now?

It is very important to us that you work with our report and derive measures for improvement from it. Therefore, we will re-test vulnerabilities for you free of charge if they are resolved within eight weeks!

In this assessment we have identified vulnerabilities with criticality **Critical** and **High**. We recommend that these vulnerabilities be addressed as a matter of priority.

Vulnerabilities with less complex countermeasures and risk **Medium** and below should, according to our recommendation, be fixed prioritised by effort. All other vulnerabilities should either be fixed or addressed as part of a continuous improvement process.

Please ensure that you deprovision all users and resources that were provisioned during the pentest as soon as they are no longer required.



Scope and Duration

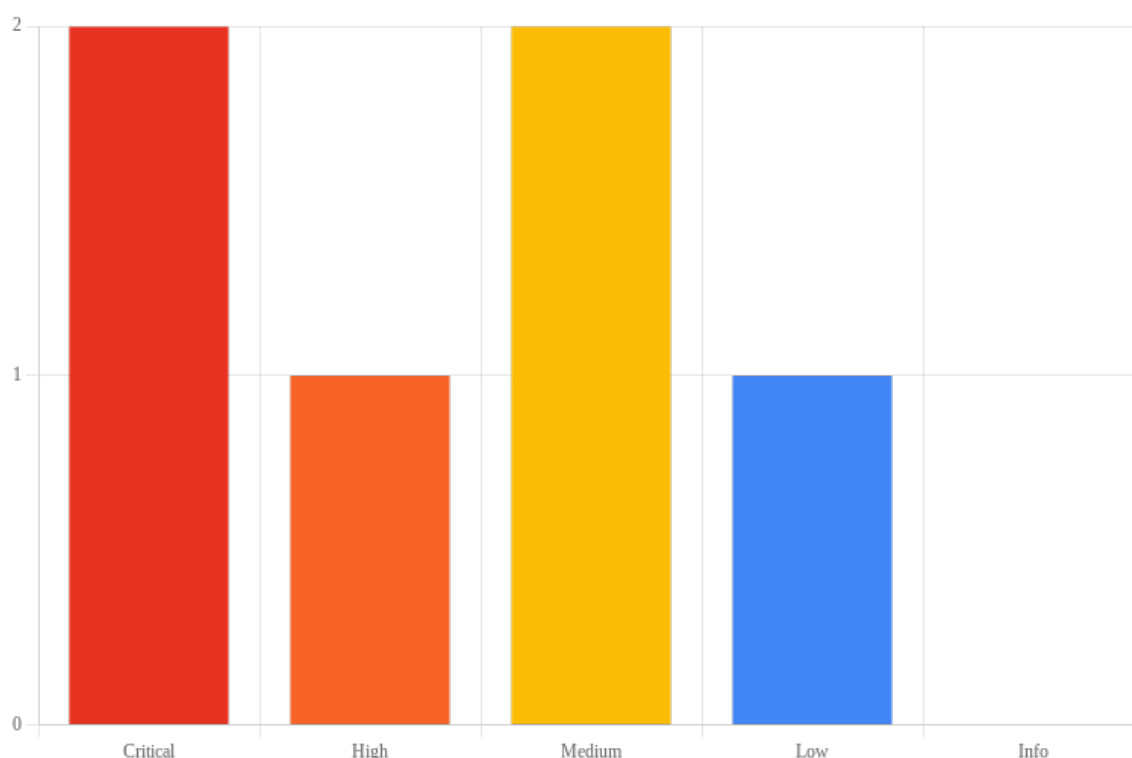
The scope of the pentest included the online shop system of Security Maximale GmbH at <https://www.example.com>.

The penetration test was carried out according to a time-box approach and covered 10 person days.



Vulnerability Overview

In the course of this pentest **2 Critical** , **1 High** , **2 Medium** and **1 Low** vulnerabilities were identified:



Distribution of identified vulnerabilities

A tabular overview of all vulnerabilities identified:

Vulnerability	Criticality
Takeover of user accounts through password reset	Critical
Stored Cross-Site Scripting (XSS)	Critical
Disclosure of passwords via Path Traversal	High
Forwarding users through Open Redirect	Medium
Disclosure of internal information	Medium
Session management weaknesses	Low

A list of all vulnerabilities including a brief description:

1. Takeover of user accounts through password reset (Critical: 9.9)

Affects: <http://www.example.com/api/v2/users/change-password/admin>



In the course of the audit, it was possible for us as a customer of the online shop to change the passwords of other customers as well as the shop administrators to any passwords.

This allowed us to take over administrative user accounts and thus compromise the entire shop system.

2. Stored Cross-Site Scripting (XSS) (Critical: 9.3)

Affects: <http://www.example.com/api/v2/users/status>

At the time of the audit, the web application stored user input unverified and later included it in HTTP responses insecurely. It was thus vulnerable to Stored Cross-Site Scripting attacks (XSS).

This could allow attackers to compromise the user accounts of other users - including administrators.

3. Disclosure of passwords via Path Traversal (High: 7.5)

Affects: <http://www.example.com/?lang=de>

Users of the application were able to gain access to passwords in configuration files at the time of the audit. This was achieved via a path traversal attack because the web application did not sufficiently check HTTP parameters.

4. Forwarding users through Open Redirect (Medium: 6.1)

An Open Redirect vulnerability allowed users to be redirected to other websites at the time of the audit.

To do this, users had to click on a manipulated link. The link appears trustworthy because it points to the web application, which, however, immediately redirects the user to a foreign website.

5. Disclosure of internal information (Medium: 5.3)

Affects: <http://www.example.com/.env>

The web application disclosed internal information that could potentially be used for more advanced attacks. Information disclosure, also known as information disclosure, occurs when a system unintentionally discloses internal information to its users.

6. Session management weaknesses (Low: 3.6)

We were able to identify weaknesses in the session management of the web application. The sessions of the users were usable without time restriction 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.



Vulnerability Details

1. Takeover of user accounts through password reset

Criticality: Critical

CVSS-Score: 9.9

Affects: `http://www.example.com/api/v2/users/change-password/admin`

Recommendation: Users should not be allowed to use the function to change passwords.

Overview

In the course of the audit, it was possible for us as a customer of the online shop to change the passwords of other customers as well as the shop administrators to any passwords.

This allowed us to take over administrative user accounts and thus compromise the entire shop system.

Description

Administrators of the online shop should be able to reset customers' passwords.

In the course of the audit, we found that this function was not limited to administrators, but was available to all users, including registered customers.

The API endpoint `POST /api/v2/users/change-password/{username}` overwrites the password of the specified user.

Request					Response				
Pretty	Raw	Hex			Pretty	Raw	Hex	Render	
1	POST /api/v2/users/change-password/admin HTTP/1.1				1	HTTP/1.1 200 OK			
2	Host: www.example.com				2	Accept-Ranges: bytes			
3	Upgrade-Insecure-Requests: 1				3	Age: 593626			
4	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36				4	Cache-Control: max-age=604800			
5	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				5	Content-Type: text/html; charset=UTF-8			
6	Accept-Encoding: gzip, deflate				6	Date: Mon, 29 Aug 2022 11:57:39 GMT			
7	Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7				7	Etag: "3147526947+ident"			
8	Connection: close				8	Expires: Mon, 05 Sep 2022 11:57:39 GMT			
9	Content-Type: application/x-www-form-urlencoded				9	Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT			
10	Content-Length: 12				10	Server: ECS (nyb/1D1F)			
11					11	Vary: Accept-Encoding			
12	new-password-123				12	X-Cache: HIT			
					13	Content-Length: 1256			
					14	Connection: close			
					15				
					16	<!doctype html>			
					17	<html>			
					18	<head>			
					19	<title>			
						Example Domain			
						</title>			
					20				

Overwriting user passwords via change-password API



The server responded with the status code 200 "OK", and changed the password of the user account "admin".

Recommendation

Customers and non-administrative users of the online shop should not be allowed to use the function for changing passwords.



2. Stored Cross-Site Scripting (XSS)

Criticality: Critical

CVSS-Score: 9.3

Affects: <http://www.example.com/api/v2/users/status>

Recommendation: User input should be validated and filtered. It should be ensured that data is properly encoded contextually before it is included in HTTP responses.

Overview

At the time of the audit, the web application stored user input unverified and later included it in HTTP responses insecurely. It was thus vulnerable to Stored Cross-Site Scripting attacks (XSS).

This could allow attackers to compromise the user accounts of other users - including administrators.

Description

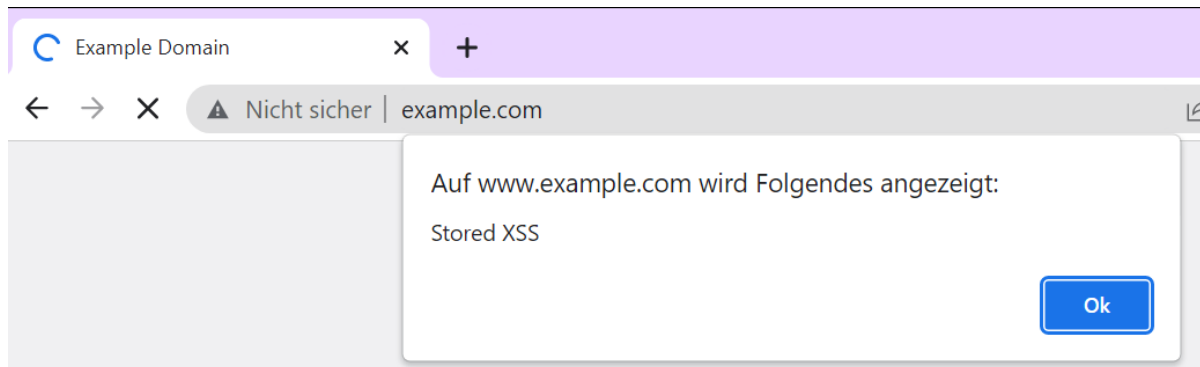
In the course of the audit, we were able to identify a stored XSS vulnerability in the web application. Due to incorrect validation and encoding of data, we as a customer of the online shop were able to inject malicious scripts and store them there persistently.

Users can set a status message in the application.

Request					Response				
Pretty	Raw	Hex			Pretty	Raw	Hex	Render	
1	POST	/api/v2/users/status	HTTP/1.1		1	HTTP/1.1	200 OK		
2	Host:	www.example.com			2	Accept-Ranges:	bytes		
3	Upgrade-Insecure-Requests:	1			3	Age:	593626		
4	User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36			4	Cache-Control:	max-age=604800		
5	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			5	Content-Type:	text/html; charset=UTF-8		
6	Accept-Encoding:	gzip, deflate			6	Date:	Mon, 29 Aug 2022 11:57:39 GMT		
7	Accept-Language:	de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7			7	Etag:	"3147526947+ident"		
8	Connection:	close			8	Expires:	Mon, 05 Sep 2022 11:57:39 GMT		
9	Content-Type:	application/x-www-form-urlencoded			9	Last-Modified:	Thu, 17 Oct 2019 07:18:26 GMT		
10	Content-Length:	12			10	Server:	ECS (nyb/1D1F)		
11					11	Vary:	Accept-Encoding		
12					12	X-Cache:	HIT		
					13	Content-Length:	1256		
					14	Connection:	close		
					15				
					16	<!doctype html>			
					17	<html>			
					18	<head>			
					19	<title>			
						Example Domain			
						</title>			
					20				

Setting the status message

When clicking on the corresponding user profile, as well as in the user overview in the backend, the status message is embedded in the HTML document. If the message is JavaScript code, it is finally executed in the browser of another user.



Execution of the JavaScript code in the backend

This could compromise the user accounts of other users, including those of administrators.

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 into 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 sufficient 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 functions and data of the web application in the context of the victim. If the affected user has privileged access, an attacker may be able to gain complete control over the web application.

Recommendation

- It should be ensured that all processed data is filtered as rigorously as possible. Filtering and validation should be based on expected and valid input.
- Data should be encoded before the web application embeds it in HTTP responses. Encoding should be done contextually, i.e. 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 that you also use a Content Security Policy (CSP) to control which client-side scripts are allowed and which are prohibited.



- Detailed information and assistance 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



3. Disclosure of passwords via Path Traversal

Criticality: High

CVSS-Score: 7.5

Affects: <http://www.example.com/?lang=de>

Recommendation: If possible, do not use user-defined file names and paths in the web application.

Overview

Users of the application were able to gain access to passwords in configuration files at the time of the audit. This was achieved via a path traversal attack because the web application did not sufficiently check HTTP parameters.

Description

We were able to identify a path traversal vulnerability in the web application during the audit. This allowed us to access sensitive files on the server system.

The `lang` parameter, which allows users to select an individual language, was the name of a folder where language information was stored and dynamically accessed.

By manipulating the parameter, we were able to read arbitrary files on the web server, including configuration files with passwords.

Request		Response			
Pretty Raw Hex		Pretty Raw Hex Render			
1	GET /?lang=../../config.php HTTP/2	40			
2	Host: www.example.com	41			define('DB_NAME', 'db');
3	Upgrade-Insecure-Requests: 1	42			
4	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)	43			/** MySQL database username */
	AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102	44			define('DB_USER', '');
	Safari/537.36	45			
5	Accept:	46			/** MySQL database password */
	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,	47			define('DB_PASSWORD', '');
	image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;	48			
	q=0.9	49			/** MySQL hostname */
6	Accept-Encoding: gzip, deflate	50			define('DB_HOST', 'localhost:3306');
7	Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7	51			
8	Connection: close	52			/** Database Charset to use in creating database
9		53			define('DB_CHARSET', 'utf8');
10		54			
		55			/** The Database Collate type. Don't change this
		56			define('DB_COLLATE', '');
		57			

Reading the database password of the web application

Path traversal is a web security vulnerability that allows an attacker to access files and directories of the underlying web server of a web application. In a path traversal attack, access to files and directories is restricted solely by the existing access controls of the underlying operating system.

Most web servers restrict access to a specific part of the file system, usually called the "web root". This directory contains all files that are required for the functionality of the



web application. To break out of the web root folder using a path traversal attack, attackers use special strings in path specifications.

In its simplest form, an attacker uses the special string `"../"` to change the location of the resource requested in the parameter. This string is a relative path specification. It refers specifically to the parent directory of the current working directory. Attackers often use alternative encodings of the `"../"` sequence to bypass any security filters that may be in place. These methods include valid and invalid Unicode-encoded (`"..%u2216"` or `"..%c0%af"`), URL-encoded (`"%2e%2e%2f"`) and double URL-encoded characters (`"..%255c"`) of the backslash character. Advanced techniques also often use additional special characters such as the dot `"."` to refer to the current working directory or the `"%00"` NULL character to bypass rudimentary end-of-file checks.

In a successful attack, an attacker can use path traversal to access arbitrary files and directories on the vulnerable system. This may include sensitive operating system files, application code or configuration files. In certain cases, it may also be possible for Path Traversal to provide write access to files and directories. Under these circumstances, it is possible for an attacker to gain code execution and thus complete control over the web server.

Recommendation

- The most effective way to avoid path traversal is to avoid user-defined file name and path specifications in the web application. If this is not possible, use indices instead of concrete value specifications (e.g. Index: 5 corresponds to the language setting "German").
- Always validate all user entries. Ensure that only input that is expected and valid for the application is accepted. Potentially malicious input should be discarded.
- When normalising paths, consider that characters may be single or multiple encoded (such as URL encoded, e.g. `%20` or `%2520` instead of a space).



4. Forwarding users through Open Redirect

Criticality: Medium

CVSS-Score: 6.1

Recommendation: Do not allow custom URLs for redirects. Assumes that all input is malicious. Therefore, follow an allow-listing approach, which defines a list of acceptable inputs.

Overview

An Open Redirect vulnerability allowed users to be redirected to other websites at the time of the audit.

To do this, users had to click on a manipulated link. The link appears trustworthy because it points to the web application, which, however, immediately redirects the user to a foreign website.

Description

For the login functionality to the online shop, the parameter 'redirectURI' can be used to specify the page to which users are to be redirected after logging in. However, any URLs can be specified here.

Attackers could thus redirect users of the website to untrusted websites using this Open Redirect vulnerability.

Request					Response				
Pretty	Raw	Hex			Pretty	Raw	Hex	Render	
1	POST	/login?redirectURI=www.syslifters.com	HTTP/1.1		1	HTTP/1.1 301 Moved Permanently			
2	Host:	www.example.com			2	Server: openresty			
3	Upgrade-Insecure-Requests:	1			3	Date: Mon, 29 Aug 2022 13:38:10 GMT			
4	User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36			4	Content-Type: text/html			
5	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			5	Content-Length: 166			
6	Accept-Encoding:	gzip, deflate			6	Connection: close			
7	Accept-Language:	de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7			7	Location: https://www.syslifters.com/			
8	Connection:	close			8				
9					9	<html>			
10	user=user&password=				10	<head>			
						<title>			
						301 Moved Permanently			
						</title>			
						</head>			
						<body>			
						<center>			
						<h1>			
						301 Moved Permanently			

Redirect user to www.syslifters.com

This vulnerability could be abused in the course of phishing attacks, for example.

An open redirect is a web security vulnerability that occurs when an application insecurely embeds user input into the target of a redirect. A redirect occurs when a web



application changes the URL that the client is currently accessing. The backend has various ways of performing redirects, such as via corresponding HTTP headers (status codes 30x) or via JavaScript. An application is vulnerable to Open Redirects if an attacker is able to construct a URL within the application that causes a redirect to any external domain.

An Open Redirect vulnerability allows an attacker to use an authentic application URL in the course of an attack that references the legitimate domain and has a valid SSL certificate. This behaviour lends more credibility to a phishing attack, making it easier to carry out social engineering attacks against users. Open redirects enable many other attacks based on them.

Recommendation

- If possible, do not use redirects in the web application. If you need the functionality, do not allow custom URLs.
- Alternatively, if the destination of a redirect or forward needs to be determined by a user, use indexes instead of full URLs (e.g. a short name, ID or token).
- Enforce that all redirects to external resources first pass through a page informing the user that they are leaving your site. This page clearly indicates the destination and the user must click on a link to confirm.
- Validate all user input and ensures that only input that is expected and valid for the application is accepted. Potentially malicious input should be discarded.
- Do not accept full URLs where possible. URLs are difficult to validate and the URL parser could be abused depending on the technology used. If this is still necessary, ensure that only valid IP addresses or domain names are accepted.
- Use the Allow-Listing approach when the web application only forwards to identified and trusted URLs. This approach defines a list of allowed IP addresses and domains at the application level. After validation of the entries and comparison with the list, the web application may only forward to those resources that are also included in the list. Forwarding to resources outside the list is blocked with this approach.
- Detailed information and assistance on how to prevent open redirects can be found in the linked "Unvalidated Redirects and Forwards Cheat Sheet" from OWASP.

Additional Information

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



5. Disclosure of internal information

Criticality: Medium

CVSS-Score: 5.3

Affects: http://www.example.com/.env

Recommendation: Configuration files should not be publicly accessible. The file `.env` should be moved to a location outside the web root.

Overview

The web application disclosed internal information that could potentially be used for more advanced attacks. Information disclosure, also known as information disclosure, occurs when a system unintentionally discloses internal information to its users.

Description

We were able to download a configuration file with internal information in the course of the pentest. This information could be used for follow-up attacks.

In the web root of the web application, the file `.env` was readable. This contained information such as the Redis server or email gateway of the application.

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	GET	/.env	HTTP/1.1	28	QUEUE_CONNECTION=sync		
2	Host:	www.example.com		29	SESSION_DRIVER=cookie		
3	User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64)		30	SESSION_LIFETIME=120		
		AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102		31			
		Safari/537.36		32	REDIS_HOST=127.0.0.1		
4	Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,		33	REDIS_PASSWORD=null		
		image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9		34	REDIS_PORT=6379		
5	Accept-Encoding:	gzip, deflate		35			
6	Accept-Language:	de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7		36	MAIL_DRIVER=smtp		
7	Connection:	close		37	MAIL_HOST=smtp.mailtrap.io		
8				38	MAIL_PORT=2525		
9				39	MAIL_USERNAME=null		
				40	MAIL_PASSWORD=null		
				41	MAIL_ENCRYPTION=null		
				42			

Access to the file `.env`

Exposing this information has rather little impact on the security of the application. However, it could represent important key information for targeted follow-up attacks by an attacker.

Recommendation

- Configuration files should not be publicly accessible.
- The `.env` file should be moved to a location outside the web root.



6. Session management weaknesses

Criticality: Low

CVSS-Score: 3.6

Recommendation: Users should be logged out automatically after a certain period of inactivity.

Overview

We were able to identify weaknesses in the session management of the web application. The sessions of the users were usable without time restriction 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 were able to 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. Furthermore, 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 authorisation and the application, the timeout could be approximately between one hour and one day.



List of Changes

Version	Date	Description	Author
0.9	2022-08-26	Initiale Erstellung	Aron Molnar
1.0	2022-08-29	Review und Freigabe	Christoph Mahrl

Disclaimer

We cannot guarantee that all existing vulnerabilities and security risks have actually been discovered. This is due to limited time resources and limited knowledge of the pentester about the IT infrastructure, software, source code, users, etc. Extensive collaboration between the client and penetration testers increases the efficiency of the penetration test. This includes, for example, the disclosure of details of internal systems or the provisioning of test users.

This penetration test represents a snapshot at the time of testing. No future security risks can be derived from it.

Imprint

syslifters.com | **Dedicated to Pentests.**
Syslifters GmbH | Eitzersthal 75 | 2013 Göllersdorf
FN 578505 v | District Court Hollabrunn