NInet Security Evaluation Report

Galene

V 1.0
Amsterdam, February 10th, 2025
Public

Document Properties

Client	Galene
Title	NLnet Security Evaluation Report
Target	Galene (https://github.com/jech/galene/)
Version	1.0
Pentester	Stefan Vink
Authors	Stefan Vink, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	January 19th, 2025	Stefan Vink	Initial draft
0.2	January 19th, 2025	Stefan Vink	Ready-for-Review
0.3	February 5th, 2025	Marcus Bointon	Review
1.0	February 10th, 2025	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between January 6, 2024 and January 19, 2025, Radically Open Security B.V. carried out a penetration test for Galene and NLnet NGI Zero Entrust.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- Galene (https://github.com/jech/galene/)

The scoped services are broken down as follows:

- Pentesting: 4.25 days
- Reporting: 0.75 days
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a penetration test of the Galene web application with Galene, in order to assess its security posture. To do so, ROS will assess it and guide Galene in attempting to find vulnerabilities in the application and its source code.

1.4 Timeline

The security audit took place between January 6, 2024 and January 19, 2025.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 7 Moderate and 5 Low-severity issues.

This security assessment revealed several vulnerabilities across web, API, and infrastructure components, primarily categorized as moderate-severity threats with potential for significant impact.

The login functionality is susceptible to timing attacks due to improper password comparison, which could facilitate unauthorized access MAF-017 (page 23). Additionally, the recording feature has a path traversal vulnerability

MAF-016 (page 21), enabling attackers with high-privileged recording permissions to modify or overwrite files, potentially leading to application crashes or unavailability.

Weak password hashing parameters, set at only 4096 iterations, make hashed passwords vulnerable to cracking using hardware acceleration tools like Hashcat, especially for users with weak passwords MAF-014 (page 19). This increases the risk of credential compromise and potential access to other systems if passwords are reused. The lack of password policies MAF-007 (page 18) further exacerbates this issue by allowing weak or commonly used passwords, thereby increasing the likelihood of successful brute-force or dictionary attacks. Moreover, the absence of brute force protection on the login mechanism MAF-005 (page 17) allows unlimited login attempts, raising the chances of unauthorized account access and exposure of sensitive data.

The web server supports deprecated TLS versions 1.0 and 1.1 MAF-004 (page 15), making it vulnerable to cryptographic attacks and potential downgrade attacks that could lead to data interception and decryption. The lack of an HTTP Strict Transport Security (HSTS) header MAF-003 (page 14) also makes the application susceptible to man-in-the-middle attacks, as users can access it over non-HTTPS connections, enabling SSL stripping and other attacks.

Lower-level threats include the absence of rate limiting on authentication, API, and WebSocket connection endpoints, which could enable brute-force or denial-of-service attacks MAF-013 (page 29).

The recording system's failure to manage disk space usage could lead to storage exhaustion and potential denial-of-service conditions for users MAF-011 (page 28).

Additionally, there is insufficient input validation for usernames MAF-010 (page 26), and inadequate logging of security events MAF-006 (page 25). The latter hinders the detection and response to unauthorized access attempts.

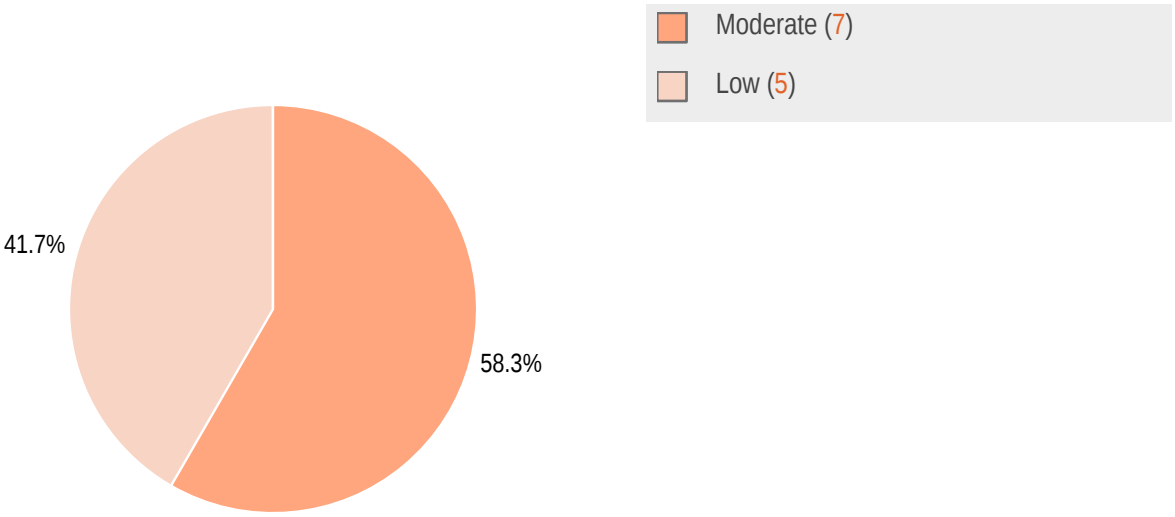
The Content-Security-Policy (CSP) header includes insecure directives such as `'unsafe-eval'`, `data` URLs, and global wildcards, which could be exploited to execute arbitrary JavaScript code or bypass CSP restrictions, increasing the risk of cross-site scripting (XSS) attacks and potential data theft MAF-002 (page 24).

1.6 Summary of Findings

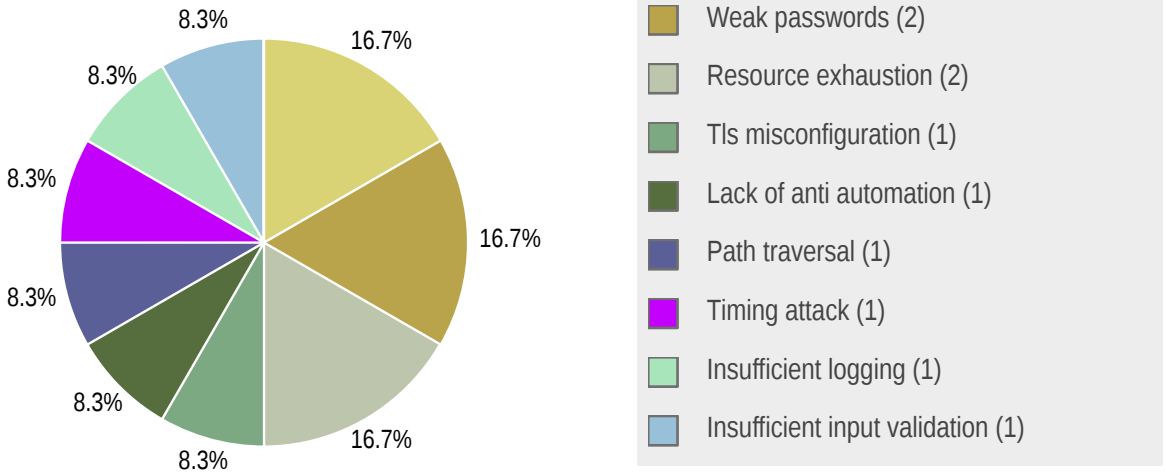
ID	Type	Description	Threat level
MAF-003	Security Header Misconfiguration	The application is vulnerable to man-in-the-middle attacks due to the absence of an HTTP Strict Transport Security (HSTS) header which prevents users from accessing the application over non-HTTPS connections.	Moderate
MAF-004	TLS Misconfiguration	The web server supports deprecated TLS versions 1.0 and 1.1, making it vulnerable to cryptographic attacks.	Moderate
MAF-005	Lack of Anti Automation	The system lacks any form of brute force protection on its login mechanism, allowing unauthorized users to repeatedly attempt to guess passwords.	Moderate
MAF-007	Weak Passwords	The system lacks a password policy to enforce strong passwords.	Moderate

MAF-014	Weak Passwords	Weak password hashing parameters configured with only 4096 iterations in webserver/api.go and galenectl/galenectl.go.	Moderate
MAF-016	Path Traversal	The recording functionality suffers from a path traversal vulnerability, allowing attackers to write recordings to arbitrary paths on the file system beyond the intended directory.	Moderate
MAF-017	Timing Attack	The login functionality is vulnerable to a timing attack due to improper comparisons when plain-text passwords are used.	Moderate
MAF-002	Security Header Misconfiguration	The Content-Security-Policy (CSP) header allows insecure sources such as 'unsafe-eval', data URLs, and global wildcards.	Low
MAF-006	Insufficient Logging	The application lacks detailed logging for security-related events such as failed login attempts, which can hinder timely detection and response to unauthorized access attempts.	Low
MAF-010	Insufficient Input Validation	The web application does not enforce proper input validation on usernames.	Low
MAF-011	Resource Exhaustion	Galene's recording system lacks mechanisms to manage or limit disk space usage.	Low
MAF-013	Resource Exhaustion	The lack of rate limiting on authentication, API, and WebSocket connection endpoints allows potential attackers to perform brute force attacks or denial-of-service attacks without any restrictions.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
MAF-003	Security Header Misconfiguration	• Implement HTTP Strict Transport Security (HSTS) header with a long enough `max-age` directive to ensure that all future requests are made over HTTPS. • Include the `includeSubDomains` directive to protect all subdomains under the main domain, if appropriate.
MAF-004	TLS Misconfiguration	• Disable TLS 1.0 and 1.1 protocols in the web server configuration. • Ensure the server configuration only allows modern versions like TLS 1.2 or TLS 1.3, with a preference for TLS 1.3 due to its improved security features.
MAF-005	Lack of Anti Automation	• Implement rate limiting on login attempts to reduce the number of tries an attacker can make within a given timeframe. • Introduce account lockout mechanisms after a set number of failed login attempts to temporarily block further access and alert administrators. • Use Captcha challenges following multiple consecutive failed login attempts to distinguish between automated scripts and human users. • Implement delay mechanisms that increase the time required to submit subsequent login attempts after each failure, slowing down brute-force attacks.
MAF-007	Weak Passwords	• Enforce a minimum password length requirement to ensure users create longer and more complex passwords. • Develop mechanisms to prevent the use of common passwords or patterns by maintaining a list of prohibited passwords, for example by implementing lookups using the haveibeenpwned.com API. See https://haveibeenpwned.com/API/v3#PwnedPasswords.
MAF-014	Weak Passwords	• Increase the PBKDF2 iteration count to at least 600,000 as recommended by OWASP for enhanced security. • Regularly update password hashing algorithms and configurations following evolving best practices and threat models. • Consider using more advanced hashing algorithms like Argon2 which offer better resistance against brute-force and GPU-based attacks.
MAF-016	Path Traversal	• Implement strict input validation for any user-provided path inputs. Ensure that paths are normalized and do not contain escape sequences like `..` or `/`. • Employ secure coding practices to prevent path traversal attacks, such as using functions that canonicalize paths before use.
MAF-017	Timing Attack	• Use constant-time comparison functions for sensitive data like passwords. • Consider only allowing securely hashed passwords.
MAF-002	Security Header Misconfiguration	• Set `default-src 'none'` and then define explicit sources for each content type (e.g., `script-src`, `image-src`) rather than relying on a global fallback to `'self'`. • Remove `'unsafe-eval'` from the `script-src` directive to prevent execution of unsafe scripts.

		• Avoid allowing `data:` sources in `img-src` directive to reduce the risk of XSS exploitation via images.
MAF-006	Insufficient Logging	• Enhance logging to include at least details such as timestamp, IP address and username on each failed login attempt.
MAF-010	Insufficient Input Validation	• Implement comprehensive input validation for all user-provided data, including username fields.
MAF-011	Resource Exhaustion	• Implement functionality to ensure that only a specified percentage of free disk space is utilized for recordings, or establish a fixed storage limit. Once this threshold is reached, the system should automatically delete the oldest recording. • Recommend in the installation notes to set the path to an external partition only used for recordings.
MAF-013	Resource Exhaustion	• Implement rate limiting with thresholds based on typical user behavior patterns and requirements.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- Burp Suite Professional – https://portswigger.net/burp/pro
- Semgrep – https://semgrep.dev

4 Findings

We have identified the following issues:

4.1 MAF-003 — Absence of Strict Transport Security Header

Vulnerability ID: MAF-003

Vulnerability type: Security Header Misconfiguration

Threat level: Moderate

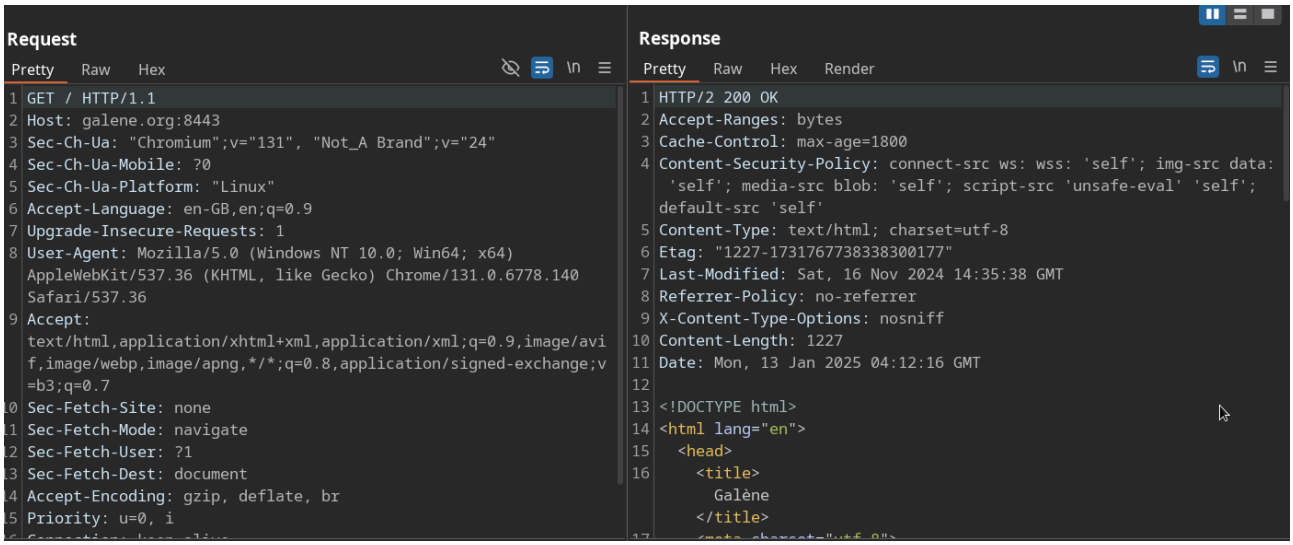
Description:

The application is vulnerable to man-in-the-middle attacks due to the absence of an HTTP Strict Transport Security (HSTS) header which prevents users from accessing the application over non-HTTPS connections.

Technical description:

HTTP Strict Transport Security (HSTS) is a web security policy mechanism which helps to protect websites against man-in-the-middle attacks such as protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should interact with it using only secure HTTPS connections, instead of the insecure HTTP protocol.

This header is missing from responses returned by Galene:



Impact:

- Users are vulnerable to man-in-the-middle (MITM) attacks.
- Potential interception and modification of unencrypted data during transmission.

Recommendation:

- Implement HTTP Strict Transport Security (HSTS) header with a long enough `max-age` directive to ensure that all future requests are made over HTTPS.
- Include the `includeSubDomains` directive to protect all subdomains under the main domain, if appropriate.

4.2 MAF-004 — Insecure TLS Versions Enabled on HTTPS

Vulnerability ID: MAF-004

Vulnerability type: TLS Misconfiguration

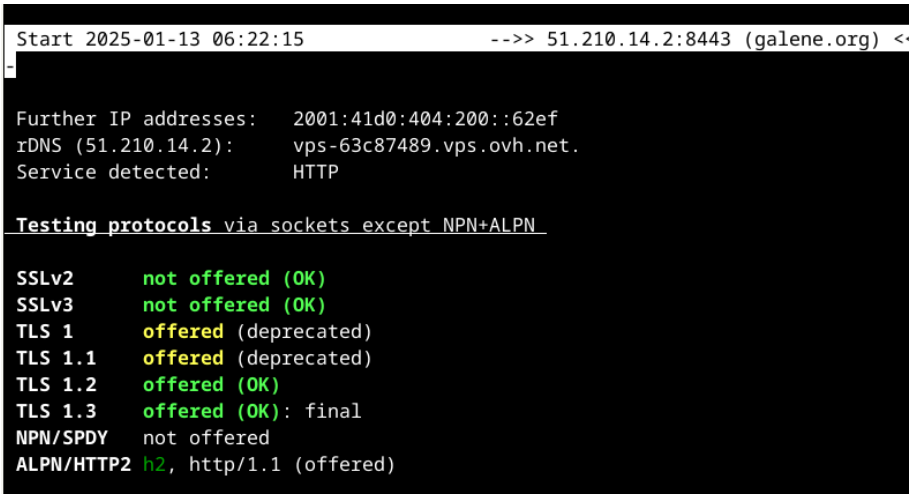
Threat level: Moderate

Description:

The web server supports deprecated TLS versions 1.0 and 1.1, making it vulnerable to cryptographic attacks.

Technical description:

We used testssl.sh to inspect the TLS configuration of the web server:



The web server permits connections using TLS 1.0 and TLS 1.1 in addition to modern TLS 1.2 and 1.3, when accessing resources over HTTPS.

These older versions are known to have vulnerabilities that can be exploited by attackers through well-documented attacks such as POODLE (Padding Oracle On Downgraded Legacy Encryption) and BEAST (Browser Exploit Against SSL/TLS). These exploits take advantage of weaknesses in the cipher suite implementations within these older protocols, allowing interception or decryption of encrypted data under specific conditions.

TLS 1.0 was deprecated in 2015 by the Internet Engineering Task Force (IETF) due to its susceptibility to various cryptographic attacks that undermine confidentiality and integrity, rendering it insecure for sensitive communications. Similarly, TLS 1.1, while slightly improved, still lacks robust protections against modern threats and is also deprecated. See RFC 8996 for more information about the deprecation of these versions.

Impact:

- Sensitive user data can be intercepted and decrypted using known attacks on older TLS versions.
- Potential downgrade attacks allow attackers to force a connection to use less secure protocol versions.
- Non-compliance with current security best practices, leading to potential regulatory issues.

Recommendation:

- Disable TLS 1.0 and 1.1 protocols in the web server configuration.
- Ensure the server configuration only allows modern versions like TLS 1.2 or TLS 1.3, with a preference for TLS 1.3 due to its improved security features.

4.3 MAF-005 — Brute Force Protection Missing for Login System

Vulnerability ID: MAF-005

Vulnerability type: Lack of Anti Automation

Threat level: Moderate

Description:

The system lacks any form of brute force protection on its login mechanism, allowing unauthorized users to repeatedly attempt to guess passwords.

Technical description:

We observed that no rate-limiting or account lockout policies are in place on excessive login attempts. This oversight allows malicious actors to attempt brute-force attacks by continuously submitting login requests with various password combinations.

Typically, such defenses include measures like:

- Rate limiting: Limiting the number of failed attempts per minute/hour.
- Account lockout: Temporarily disabling accounts after several unsuccessful login attempts.
- Captcha integration: Requiring human verification after a set number of failures.
- Delay mechanisms: Introducing incremental time delays between successive failed attempts to thwart automated attacks.

In the absence of these protections, attackers can exploit this weakness to systematically try to guess passwords. This not only compromises individual user accounts but also poses a significant risk if they manage to access accounts with elevated privileges such as administrator roles.

Impact:

- Unrestricted login attempts increase the likelihood of unauthorized account access.
- Sensitive data may be exposed if administrative accounts are compromised.
- User trust and system integrity might be significantly undermined due to potential breaches.

Recommendation:

- Implement rate limiting on login attempts to reduce the number of tries an attacker can make within a given timeframe.
- Introduce account lockout mechanisms after a set number of failed login attempts to temporarily block further access and alert administrators.
- Use Captcha challenges following multiple consecutive failed login attempts to distinguish between automated scripts and human users.
- Implement delay mechanisms that increase the time required to submit subsequent login attempts after each failure, slowing down brute-force attacks.

4.4 MAF-007 — Absence of Password Policy

Vulnerability ID: MAF-007

Vulnerability type: Weak Passwords

Threat level: Moderate

Description:

The system lacks a password policy to enforce strong passwords.

Technical description:

The application does not currently enforce any password length, complexity, or uniqueness requirements. Password management is a critical aspect of securing access to systems and protecting sensitive information; a robust password policy ensures users create strong, unique passwords that are difficult for attackers to guess or crack.

Impact:

- Increased likelihood of brute-force and dictionary attacks succeeding due to lack of password complexity or uniqueness requirements.
- Potential for unauthorized system access, leading to data breaches or other malicious activities.
- Compromise of user accounts can lead to a cascading effect where attackers gain broader access within an organization.

Recommendation:

- Enforce a minimum password length requirement to ensure users create longer and more complex passwords.
- Develop mechanisms to prevent the use of common passwords or patterns by maintaining a list of prohibited passwords, for example by implementing lookups using the haveibeenpwned.com API. See https://haveibeenpwned.com/API/v3#PwnedPasswords.

4.5 MAF-014 — Insufficient PBKDF2 Iterations for Secure Password Hashing

Vulnerability ID: MAF-014

Vulnerability type: Weak Passwords

Threat level: Moderate

Description:

Weak password hashing parameters configured with only 4096 iterations in `webserver/api.go` and `galenectl/galenectl.go`.

Technical description:

Password hashing is essential for protecting user credentials against unauthorized access. PBKDF2 (Password-Based Key Derivation Function 2) is commonly used for this purpose. It applies a pseudorandom function, such as HMAC-SHA256, multiple times to the input password. The security of PBKDF2 significantly depends on two factors: the key length and the iteration count. An adequate number of iterations increases computational complexity, making brute-force attacks less feasible.

The code in `webserver/api.go` and `galenectl/galenectl.go` specifies only 4096 iterations for PBKDF2 hashing.

This value is insufficient by modern standards. OWASP recommends at least 600,000 iterations to secure passwords effectively against brute-force attacks.

Performance Analysis Using a GPU setup (4090 and 3090) with Hashcat, the current configuration allows approximately 2,626,300 PBKDF2-SHA256 hashes per second:

```
Speed.#1.........:  1789.5 kH/s (9.03ms) @ Accel:16 Loops:64 Thr:512 Vec:1
Speed.<a href="#f2-insecure-content-security-policy-header"/>.........:   836.6 kH/s (12.28ms) @
 Accel:16 Loops:64 Thr:512 Vec:1
Speed.#*.........:  2626.3 kH/s
```

This high rate means:

- Approximately 2.63 million passwords tested per second.
- About 157.6 million passwords per minute.
- Up to 9.45 billion passwords can be attempted in an hour.

With these numbers, even complex passwords would be vulnerable. For example, cracking an 8-character password with mixed case, numbers, and symbols (approximately 6.1 trillion combinations) could take about 27 days under the current settings.

Improved Configuration With the recommended 600,000 iterations:

The same GPU setup would only manage about 17,900 hashes per second (2,626,300 * 4096/600,000 = ~17,900)

Making the same 8-character password take about 10.8 years to crack (6.1 trillion combinations / 17,900 per second / 86400 seconds per day / 365 days per year ≃ 10.8 years)

This demonstrates why increasing the iteration count to at least 600,000 is crucial for maintaining password security in 2025.

Impact:

- With 4096 iterations, passwords can be cracked rapidly using hardware acceleration tools like Hashcat.
- Users with weak or commonly used passwords are at high risk of having their credentials compromised.
- Attackers could potentially decrypt hashed passwords within hours or days for an 8-character password.
- In case passwords are re-used attackers could use these to gain access to other systems.

Recommendation:

- Increase the PBKDF2 iteration count to at least 600,000 as recommended by OWASP for enhanced security.
- Regularly update password hashing algorithms and configurations following evolving best practices and threat models.
- Consider using more advanced hashing algorithms like Argon2 which offer better resistance against brute-force and GPU-based attacks.

4.6 MAF-016 — Arbitrary File Write via Path Traversal in Recording Functionality

Vulnerability ID: MAF-016	**Status:** Resolved
Vulnerability type: Path Traversal	
Threat level: Moderate	

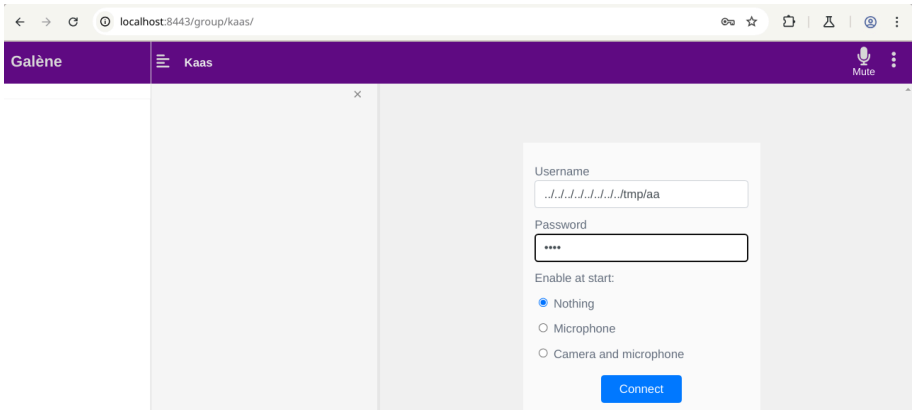
Description:

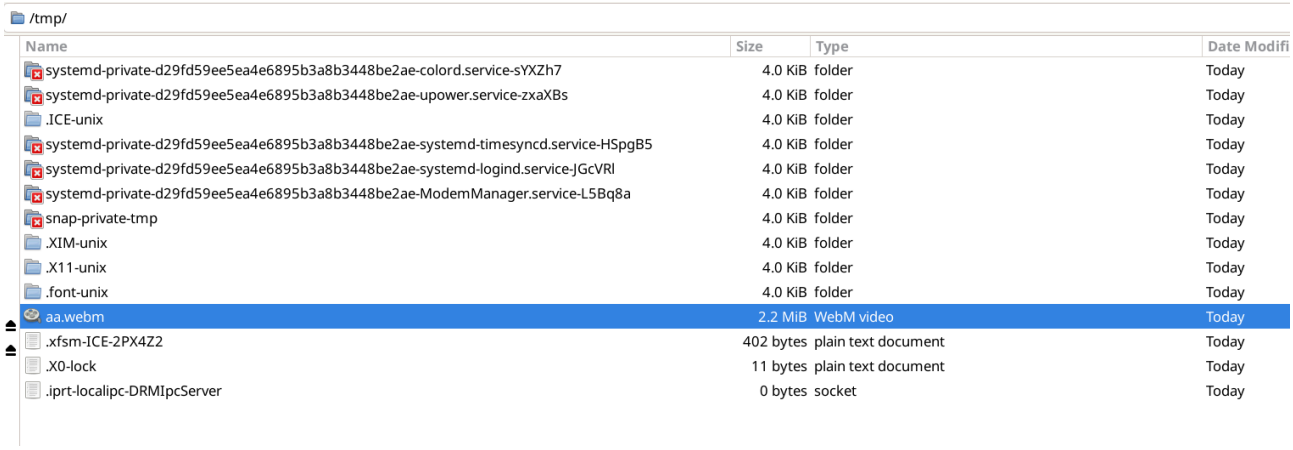
The recording functionality suffers from a path traversal vulnerability, allowing attackers to write recordings to arbitrary paths on the file system beyond the intended directory.

Technical description:

We discovered a path traversal vulnerability in the recording functionality of the application. The issue arises because the application allows users to control the paths for saving recordings, and these paths are not properly validated before being used. As a result, an attacker can manipulate the path to write recordings to any location on the file system where the application has write permissions.

For example, by providing a path like `../../../../../../tmp/aa` as the username, the application writes the recording to `/tmp/aa.webm` instead of the expected directory. This behavior indicates that the application fails to sanitize or restrict the path input, leading to potential security risks.





The vulnerability is partly due to the code allowing escape sequences (e.g., `..`) in the path and also because the username, which is used in constructing the path, lacks proper input validation, as we reported in MAF-010 (page 26). This combination makes it possible for an attacker to traverse directories and write files to sensitive locations on the server's file system.

Impact:

- An attacker could modify or overwrite files and recordings with the `.webm` extension, leading to data corruption or loss.
- By writing to sensitive locations, the attacker might cause the application or the entire system to crash or become unavailable.
- Note that the impact is low as it requires recording permissions that are only available to high-privileged accounts.

Recommendation:

- Implement strict input validation for any user-provided path inputs. Ensure that paths are normalized and do not contain escape sequences like `..` or `/`.
- Employ secure coding practices to prevent path traversal attacks, such as using functions that canonicalize paths before use.

Update 2025-01-19 05:56:

This resolves the current issues, hence it has been flagged as resolved. To further improve we would also suggest in the code to not allow path traversal characters in the group name. Not an issue at this stage as users cannot set their own group but this could become an issue in the future if new functionality would allow user's to create their own groups.

Additionally, while usernames are sanitized using the sanitise function, the sanitization only replaces `/` and `\` characters. Other special characters that could be problematic in filenames (like null bytes, control characters, or other platform-specific special characters) are not handled.

4.7 MAF-017 — Timing Attack Vulnerability in Plain Text Password Comparison

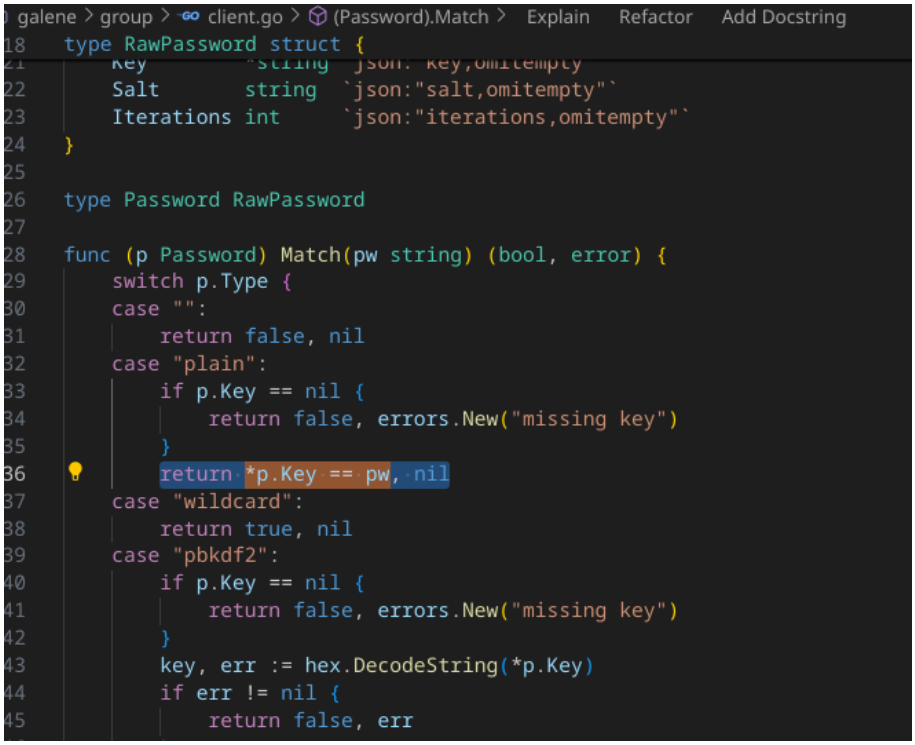
Vulnerability ID: MAF-017	**Status:** Resolved
Vulnerability type: Timing Attack	
Threat level: Moderate	

Description:

The login functionality is vulnerable to a timing attack due to improper comparisons when plain-text passwords are used.

Technical description:

In the code provided, there is a vulnerability related to password comparison in the 'plain' case. The issue arises from using direct string comparison with the `==` operator, which can lead to timing differences based on how much of the string matches.



Note that due to time constraints, we did not create a proof of concept.

Impact:

- This could make it easier for attackers to potentially gain unauthorized access when plaintext passwords are used by exploiting this vulnerability.

Recommendation:

- Use constant-time comparison functions for sensitive data like passwords.
- Consider only allowing securely hashed passwords.

Update 2025-01-17 04:17:

Resolved: Reviewed, and has been merged in this commit.

4.8 MAF-002 — Insecure Content-Security-Policy Header

Vulnerability ID: MAF-002

Vulnerability type: Security Header Misconfiguration

Threat level: Low

Description:

The Content-Security-Policy (CSP) header allows insecure sources such as `'unsafe-eval'`, `data` URLs, and global wildcards.

Technical description:

The Content-Security-Policy (CSP) header is a critical security feature that helps mitigate various web vulnerabilities, including Cross-Site Scripting (XSS), by restricting the sources of sub-resources in a page. The Galene server delivers a CSP header like this:

```
connect-src ws: wss: 'self'; img-src data: 'self'; media-src blob: 'self'; script-src 'unsafe-eval'
 'self'; default-src 'self'
```

This has the following issues:

- The presence of `'unsafe-eval'` in `script-src` allows the execution of potentially unsafe JavaScript code through functions like `eval()` and `Function()`.
- `data` URLs enable embedding image data directly within HTML using base64 encoding (`...`).
- The use of `'self'` as the `default-src` can lead to security oversights if specific directives (like `img-src`, `script-src`) are not explicitly defined, and any new directives will automatically be allowed, which may not be wanted.
- There is no `report-uri` or `report-to` directive, so you won't find out about client-side violations of the CSP rules.

Impact:

- Attackers could execute arbitrary JavaScript code by exploiting `'unsafe-eval'`.
- Data URLs in `img-src` can be used to bypass CSP restrictions, leading to XSS.
- Global wildcards increase the attack surface due to lack of specificity.
- Potential unauthorized access or data theft through crafted payloads.

Recommendation:

- Set `default-src 'none'` and then define explicit sources for each content type (e.g., `script-src`, `image-src`) rather than relying on a global fallback to `'self'`.
- Remove `'unsafe-eval'` from the `script-src` directive to prevent execution of unsafe scripts.
- Avoid allowing `data:` sources in `img-src` directive to reduce the risk of XSS exploitation via images.

4.9 MAF-006 — Insufficient Logging of Failed Login Attempts

Vulnerability ID: MAF-006

Vulnerability type: Insufficient Logging

Threat level: Low

Description:

The application lacks detailed logging for security-related events such as failed login attempts, which can hinder timely detection and response to unauthorized access attempts.

Technical description:

When a user fails to log in, only the message `Join group: not authorised` appears in the console. This minimal feedback provides insufficient context for administrators to understand or investigate potential security incidents.

Impact:

- Inability to detect potential brute-force or credential-stuffing attacks.
- Delayed response time in identifying unauthorized access attempts.
- Lack of forensic data for investigating security incidents.
- Potential increase in the risk of successful account compromises.
- Difficulty in assessing threat patterns and implementing proactive defenses.

Recommendation:

- Enhance logging to include at least details such as timestamp, IP address and username on each failed login attempt.

4.10 MAF-010 — Improper Input Validation in Username

Vulnerability ID: MAF-010 **Status:** Resolved

Vulnerability type: Insufficient Input Validation

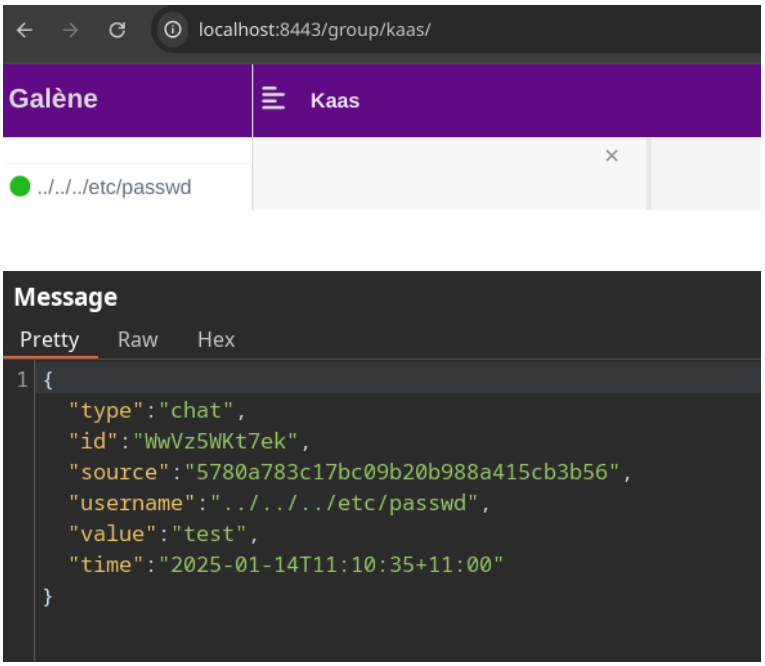
Threat level: Low

Description:

The web application does not enforce proper input validation on usernames.

Technical description:

The web application does not enforce input validation on user-provided data for usernames. Users have the ability to select their own usernames in configurations where wildcards are configured.



Combining this finding with Arbitrary File Write via Path Traversal in Recording Functionality (page 21) allowed us to write files to different paths of the file system.

We did not find any security issues related to XSS at this stage; however, this could change with new functionality.

Impact:

- The current risk is low, but improper input validation could lead to unauthorized file access or data manipulation if not addressed before future updates.

Recommendation:

- Implement comprehensive input validation for all user-provided data, including username fields.

Update 2025-01-19 06:03:

Commit resolved the path traversal issue.

4.11 MAF-011 — Insufficient Disk Space Management in Galene's Recording System

Vulnerability ID: MAF-011

Vulnerability type: Resource Exhaustion

Threat level: Low

Description:

Galene's recording system lacks mechanisms to manage or limit disk space usage.

Technical description:

The Galene voice communication platform utilizes a component known as the `diskwriter` package for managing WebM file creation and writing processes related to recordings. Upon review, it becomes evident that there are no mechanisms within this system to manage or limit disk space usage effectively.

The system is configured with various group settings such as `AllowRecording`, which control recording permissions but do not address disk management.

Recordings are written directly to the filesystem without any checks for available space, leading to a potential risk where recordings could exhaust all available storage on the server.

Impact:

- Recordings may fill up all available disk space, impacting server performance.
- Potential denial-of-service condition for Galene users if storage is exhausted.

Recommendation:

- Implement functionality to ensure that only a specified percentage of free disk space is utilized for recordings, or establish a fixed storage limit. Once this threshold is reached, the system should automatically delete the oldest recording.
- Recommend in the installation notes to set the path to an external partition only used for recordings.

4.12 MAF-013 — Authentication and API Endpoints Lack Rate Limiting

Vulnerability ID: MAF-013

Vulnerability type: Resource Exhaustion

Threat level: Low

Description:

The lack of rate limiting on authentication, API, and WebSocket connection endpoints allows potential attackers to perform brute force attacks or denial-of-service attacks without any restrictions.

Technical description:

The application's authentication endpoints, API endpoints, and WebSocket connections lack rate-limiting mechanisms. Rate limiting serves as a protective measure against various types of attack by restricting the number of requests a user or system can make to an endpoint within a specific timeframe.

Impact:

- Authentication endpoints are vulnerable to brute-force attacks.
- API endpoints may be overwhelmed by excessive requests, leading to potential denial-of-service conditions.
- The absence of rate limiting allows for easier detection and exploitation of vulnerabilities through automated means.
- Increased server load can degrade performance and lead to service downtime affecting legitimate users.

Recommendation:

- Implement rate limiting with thresholds based on typical user behavior patterns and requirements.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends, or represents issues that are not directly security-related.

5.1 NF-008 — Secure Use of AES-ECB Mode in Whip.go for Obfuscation

The use of AES in ECB mode within the web server component `whip.go` has been assessed and is deemed secure given its specific context. Typically, AES-ECB (Electronic Codebook) mode is not recommended because it can reveal patterns in encrypted data if there are repeated plaintext blocks. However, this implementation avoids such vulnerabilities due to several key factors.

Firstly, the input being encrypted is always random data of exactly one block size, which eliminates the risk of pattern revelation that ECB mode inherently has when processing multiple identical blocks of plaintext. Secondly, each ID generated for encryption uses cryptographically secure random data, ensuring there's no predictability in the input. Additionally, the primary goal here is obfuscation rather than confidentiality of sensitive information, making the potential weaknesses of ECB mode less critical.

Furthermore, the session IDs being encrypted are temporary and single-use, reducing the risk window for any cryptographic attack to exploit these identifiers. In this unique scenario, the identified implementation effectively mitigates the usual security concerns associated with AES-ECB mode. While maintaining best practices is always advisable, in this specific use case, ECB mode's limitations do not pose a significant threat.

5.2 NF-009 — Allowed External URL Redirection is Configured Securely

The application has been assessed for potential vulnerabilities associated with redirect configurations allowing external URLs. It was confirmed that such redirects are permitted intentionally and require server file access to configure. This design choice ensures that control over redirection behavior is maintained securely.

During testing, no security issues were identified regarding this feature's implementation. The configuration process necessitates server file access, which adds a layer of protection against unauthorized changes. However, we recommend that the application could enhance user awareness by displaying a message when a redirect to an external website occurs. This would provide transparency and potentially mitigate phishing risks.

Additionally, implementing similar notifications for links (especially if its a direct link to malicious file extensions) clicked within chat messages can further safeguard users by alerting them about navigating away from the trusted environment. These recommendations aim to improve user experience while maintaining high security standards.

5.3 NF-012 — Usernames are Properly Verified during Chat Message Transmission

The application effectively prevents spoofing of usernames while transmitting chat messages. The verification process ensures only authenticated users can send messages under their registered identities, maintaining integrity and trust within the communication platform.

5.4 NF-015 — Cross-Site Scripting (XSS) Vulnerabilities Absent

We evaluated the application thoroughly for Cross-Site Scripting (XSS) vulnerabilities, which are often exploited by attackers to inject malicious scripts into web pages viewed by other users. This test involved checking various points of user input across the website and API endpoints where data entered by users could potentially be reflected or stored without proper sanitization. The outcome of the comprehensive evaluation was positive; we did not find any instances of vulnerable code or misconfigurations that could lead to XSS attacks.

6 Future Work

- **Test the Android application**
  The developer mentioned that an Android application can also be used, but this was out of scope due to time constraints in this pentest. We recommend conducting a separate security assessment for this app in the future.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

7 Conclusion

We discovered 7 Moderate and 5 Low-severity issues during this penetration test.

We did not find any major issues in Galene that would compromise the application or its users directly.

Most findings were categorized as moderate and low threats. Although no critical vulnerabilities were identified, the cumulative effect of these moderate and lower-level threats (e.g. insecure password policy, weak hashing, lack of password brute-forcing) raises the overall risk.

Note that several issues have been resolved (indicated in this report), and others are in progress.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Stefan Vink	Stefan is an IT professional with a passion for IT security and automation. With 20 years hands-on experience in a diverse range of IT roles such as automation / scripting / monitoring / web development / system and network management in Windows and Linux environments. He has worked for organisations such as the Central Bank of the Netherlands (DNB), is MCITP, CCNA, LPIC, OSCP certified, and has passed the CISSP exam. He loves to travel, hike, play tennis & chess, automation, and lives with his family in Melbourne, Australia.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.