Mailvelope Browser Extension

Retest

Author:0xche.org
nobody@0xche.org
July 15th, 2025

Index

1 > Confidentiality agreement

This document has been classified as "Confidential Information". This classification enables the recipients (Mailvelope and Open Tech Fund) to use the information contained in the document for the purposes for which 0xche has provided it, or as contractually agreed in relation to the exchange of information, where applicable, between the parties, and this without prejudice to compliance with the regulations on intellectual property, and on the protection of personal data.

2 > Changelog

Version	Date	Description
v1	January 31st, 2025	Report for Mailvelope
v2	February 7th, 2025	Report following the development team's review
v3	July 15th, 2025	Retest of fixes

3 > Context

Mailvelope is a browser extension for end-to-end encryption of email traffic that integrates into existing webmail applications. It supports Google Chrome, Microsoft Edge, and Mozilla Firefox on desktop. It supports the PGP encryption standard (OpenPGP). The extension allows to encrypt files on your user's device and send encrypted email attachments. It is based on OpenPGP.js (https://openpgpjs.org), an OpenPGP library for JavaScript. The OpenPGP standard (based on PGP as originally developed by Phil Zimmermann) uses a combination of strong public key and symmetric encryption to secure electronic communication.

Mailvelope is open source software with its source code available to the public and verified by many different organizations. Their security is continuously monitored by regular security audits. The main asset of this assesment, is a browser extension for Google Chrome and Firefox that allows secure email communication based on the OpenPGP standard.

The objective of this audit is to reveal common security issues and to offer suggestions for improvements to Mailvelope, currently used by 250K users around the world. The focus of the audit will be to identify vulnerabilities in the application that could cause user data compromise, and that could be abused for user monitoring.

4 > Methodology

4.1 Type of Audit

Previous knowledge	White Box
Authentication	Not Authenticated
Origin of the tests	Local
Type of execution	Manual and automatic

4.2 Methodology

Type of tests:

During the analysis, 0xche focused on testing the provided extension code while it runs in the browser. This approach will allow the testing team to understand how the core and each functionality operates, both statically and dynamically. Some of the tests and tasks were the following:

- **Code Review (Static Analysis):** examine the source code for vulnerabilities such as XSS, CSRF, and injection flaws, and evaluate coding practices and third-party libraries.
- **Dynamic Analysis:** run the extension to monitor real-time behavior, identify runtime vulnerabilities, and test interactions with the browser and operating system.
- **Permission Analysis:** review and justify the permissions requested by the extension, ensuring they are not excessive.
- **Network Traffic Analysis:** monitor network activity to identify data leaks and ensure all transmissions are encrypted.
- **User Interface Security Testing:** analyze the interface to prevent phishing, clickjacking, and unintended actions.

Attack Scenarios:

Our objective is to gain as much context as possible to identify weaknesses, potential attack scenarios, and/or flaws found in the code, focusing on specific scenarios as user data compromise through Client Side vulnerabilities or third party integrations misconfigurations.

4.3 Scope

For this engagement, the security audit was conducted over the following assets:

Extension Name	Mailvelope
Browsers supported	Chrome and Firefox
URL repository	https://github.com/mailvelope/mailvelope
Branch	master
Release Version	6.0.1
Commit Hash	d868ea0b73aa2b6f43805ff3b1635e01ec73e8e8

Proposed technical scope, agreed with the client:

For scoping, we tested every feature, by functionality. These were the following:

- Encrypt
- Decrypt
- Signing
- Key Management
- File Encryption
- Options / autocrypt headers

4.4 Restrictions

OpenPGP library for JavaScript and Mailvelope servers are out of scope during this engagement.

5 > High level summary

The vulnerabilities identified in Mailvelope mainly arises from implementation weaknesses in security-critical components and outdated dependencies. The identified issues range from low-severity concerns in PGP message parsing to high-risk vulnerabilities in UI security controls, with particular emphasis on clickjacking vulnerabilities that could compromise the extension's security indicators and settings user interface.

The extension's privacy features like Web Key Directory lookups show design weaknesses that could allow user tracking and information disclosure through external requests by the server. While core cryptographic operations remain mostly secure, the surrounding implementation layers reveal validation gaps in domain checking, prototype pollution vectors, and insufficient message type verification.

The security audit also revealed that despite the extension implementing several defensive measures like Content Security Policy and sandboxed iframes, critical protections such as frame-ancestor headers and strict domain validation require significant improvements. These security gaps, combined with multiple outdated dependencies containing known vulnerabilities, could affect the privacy guarantees that users expect from a PGP-based email encryption solution.

Overall the security audit confirmed that the extension maintains strong security boundaries by leveraging Chrome’s Extension APIs. The permissions model was well-restricted, ensuring that only necessary access was granted. Additionally, communication channels between the background script, content script, and webpage were securely implemented, making message interception and spoofing infeasible. The extension also enforces strict isolation and validation of key management and cryptographic operations. Furthermore, sandboxed iframes and randomized security elements provide robust protection for critical components.

6 > Analysis timeline

Start date: 10/01/24
End date: 22/12/24

7 > Vulnerability analysis

7.1 Summary table of Vulnerabilities by severity

#	Qty	Finding	Severity
1.	**1**	Clickjacking	**High**
2.	**1**	Prototype pollution	**Low**
3.	**1**	Weak validation in Web Key Directory (WKD) blacklist	**Informative**
4.	**1**	Information Disclosure through automatic WKD Key lookup requests	**Informative**
5.	**1**	Insufficient message type validation in PGP Block Parser	**Informative**
6.	**1**	Testing subdomain authorized in production	**Informative**
7.	**1**	Software out to date	**Informative**

7.2 Strengths

- The extension implements proper security boundaries through Chrome's Extension APIs and CSP headers, effectively preventing discoverable XSS and injection vectors.

- Key management operations are isolated in privileged contexts with mandatory signature verification and key integrity validation before any cryptographic operation.

- Critical UI components operate within sandboxed iframes and use randomized security backgrounds stored in isolated extension storage preventing identity theft and targeted phishing attacks.

- Cryptographic operations follow security standards by implementing proper key derivation functions and secure storage of encrypted keys through Chrome's APIs.

- The extension architecture enforces context isolation with API endpoints, requiring explicit domain whitelisting and validation checks for sensitive operations.

7.3 Summary of recommendations

- Improve UI security by restricting frame embedding of sensitive components, and moving critical operations to extension-exclusive interfaces.

- Strengthen domain validation by implementing strict pattern matching and proper domain parsing, especially for security-critical features like WKD lookups and key verification.

- Add privacy controls for automatic external requests, including user-configurable options for WKD lookups and caching mechanisms to reduce potential tracking.

- Enhance message parsing with strict block type validation, proper comment handling, and thorough verification of PGP block boundaries before processing.

- Implement additional security checks in storage operations to prevent prototype pollution, including path validation.

- Update all JavaScript dependencies to their latest stable versions and implement automated dependency scanning to prevent security regressions.

8 > Vulnerabilities

8.1 Clickjacking

Overview

A security vulnerability has been identified in Mailvelope's protection mechanisms against clickjacking attacks.

Clickjacking occurs when a malicious website embeds another website within an invisible iframe, tricking users into interacting with the hidden sensitive content underneath by overlaying deceptive UI elements. In the context of Mailvelope, this could lead to unauthorized manipulation of critical security settings and cryptographic operations.

Mailvelope exposes certain resources to web applications through its *web_accessible_resources* property in the manifest file. While it employs mechanisms to prevent direct embedding of

sensitive pages, the framability of pages like the settings interface still poses a significant attack surface.

The vulnerability is particularly concerning because it could lead to unauthorized access to cryptographic operations, including:

- Decryption of encrypted messages using attacker-controlled keys.
- Exposure of the victim's private key.
- Manipulation of security-critical settings.

As an exploitation constrain, in the context of Mailvelope, only whitelisted applications with API access are capable of directly manipulating the settings container. This requirement introduces a significant constraint on the exploitation of the vulnerability, reducing the vulnerability's criticity from Critical to **High**.

Recommendations / Mitigation

- Remove landing pages with critical functionalities such as *app/app.html* among web accessible resources within the webAccessibleResource property to further restrict iframe embedding.
- Require additional user confirmation through password input for security-sensitive actions, such as adding a private key to the user's keyring.
- Disable drag operations in sensitive textboxes to mitigate the risk of clickjacking attacks.

Proof of Concept

1. We identified extension's manifest file exposes several sensitive HTML pages and components through web accessible resources, making them available for potential clickjacking attacks:

```
"web_accessible_resources": [{
   "resources": [
     "app/app.html",
     "client-API/mailvelope-client-api.js",
     "components/decrypt-message/decryptMessage.html",
     "components/generate-key/genKey.html",
     "components/key-backup/backupKey.html",
     "components/restore-backup/backupRestore.html",
     "components/editor/editor.html",
     "components/encrypted-form/encryptedForm.html",
     "res/fonts/*.woff2",
     "img/edit_add-22.png",
     "img/key-24.png",
     "img/key-icon-blue96.png",
     "img/loading.gif",
     "img/mail_locked_96.png",
     "img/mail_signed_96.png",
     "img/mail_open_96.png",
     "img/mail_new.png",
     "img/ok48.png",
     "img/verify-24.png",
     "main.css"
   ],
```

2. Sensitive actions like security settings and cryptographic keys import/export are accesible via iframes and suceptible to clickjacking.

```
// Frame settings mailvelope page for a Clickjacking attack

async function loadframe() {
  keyring = await mvelo.getKeyring(null);
  tab_id = await mvelo.createSettingsContainer('#frame', keyring);
  let frame = document.getElementById("frame");
  let iframe = frame.getElementsByTagName("iframe")[0];
 iframe.src="chrome-extension://<id>/app/app.html?id="+tab_id+"#/keyring/import";
  frame.style.display = 'block';
  frame.style.opacity = '0.2';
  ...
}
```

Step by step explotation:

1. The attacker using a whitelisted subdomain (or abusing of an already whitelisted domain though a subdomain takeover for example) is able to frame the extension page /app/app.html

2. The victim is tricked to resolve a captcha puzzle without realizing its interacting with extension.

    a. Underthehood, with simple drag gestures the victim adds an attacker provided private key to their keyring. With two aditional clicks the victim could be tricked to set this key as the default one. Any future email is encrypted with the default private key, allowing the attacker to decrypt sent emails (and possibly received ones).

    b. A simpler attack scenario with only two drag gestures would expose the victim's private key directly being the attcker able to exfiltrate critical information.

Video of the POC:

[redacted]

PoC source code available at:

[redacted]

Status: Fixed

The app page (app.html) was removed from web accessible resources and the function to include it was deprecated.

```
"web_accessible_resources": [{
  "resources": [
    "client-API/mailvelope-client-api.js",
    "components/decrypt-message/decryptMessage.html",
    "components/generate-key/genKey.html",
    "components/key-backup/backupKey.html",
    "components/restore-backup/backupRestore.html",
    "components/editor/editor.html",
    "components/encrypted-form/encryptedForm.html",
    "res/fonts/*.woff2",
    "img/edit_add-22.png",
    "img/key-24.png",
    "img/key-icon-blue96.png",
    "img/loading.gif",
    "img/mail_locked_96.png",
    "img/mail_signed_96.png",
    "img/mail_open_96.png",
    "img/mail_new.png",
    "img/ok48.png",
    "img/verify-24.png",
    "main.css"
  ],
```

8.2 Prototype pollution

Overview

A prototype pollution vulnerability has been identified in Mailvelope's storage management system, specifically in the *mvelo.storage.remove()* method. This method, which is designed to handle both single key and nested path deletions in Chrome's local storage, contains an unsafe object traversal implementation.

Prototype pollution is generally considered a critical issue because it allows manipulation of the global object prototype, potentially leading to unexpected behavior or even remote code execution, depending on how the polluted properties are utilized.

However, in the context of Mailvelope, the vulnerability's exploitation and impact are constrained, reducing its criticality to **Low**. The issue has minimal exploitation potential, with its impact restricted to property deletion, further limiting any potential consequences even in the event of successful exploitation.

The vulnerable code path is triggered when the method receives an array of path segments instead of a single key:

```
mvelo.storage.remove = function(id) {
 id = typeof id === 'string' ? [id] : id;
 if (id.length === 1) {
   return chrome.storage.local.remove(id[0]);
 } else {
   return mvelo.storage.get(id[0])
   .then(data => {
     const path = id.slice(1);
     let obj = data;
     for (let i = 0; i < path.length - 1; i++) {
       obj = obj[path[i]];  // Vulnerable property traversal
     }
     delete obj[path.pop()];  // Vulnerable deletion
```

```
      return mvelo.storage.set(id[0], data);
    });
  }
};
```

The vulnerability arises from:

1. **Lack of path validation before traversal**:
   The method accepts an array of path segments without validating their integrity or ensuring that they correspond to legitimate object properties.
2. **Direct property access without safeguards**:
   Object properties are traversed directly with *obj = obj[path[i]]* and modified with delete *obj[path.pop()]*. This mechanism provides no safeguards against accessing or modifying properties in the prototype chain, such as *__proto__*.

These issues enable prototype pollution attacks that can compromise the application's object behaviors.

Recommendations / Mitigation

1. Implement strict path validation. Validate all path segments to ensure they correspond to expected keys and do not include prototype chain properties such as *__proto__*, *constructor*, or *prototype*.
2. Consider implementing a path allowed-list for allowed storage operations.
3. Add input sanitization and validation at all storage operation entry points.
4. Consider using *Object.freeze()* on critical object prototypes to prevent runtime modifications to global prototypes.

Proof of Concept

Using a malicious crafted ID:

```
id = ['mvelo.preferences', '__proto__', '<method_to_be_deleted>'];
id = ['mvelo.preferences', '__proto__', 'toString'];
```

An attacker can trigger the deletion of the *toString* method from the Object prototype:

1. Before exploitation, *Object.prototype.toString* is accessible.

2. After triggering the removal, *Object.prototype.toString* is undefined, breaking core JavaScript functionality

Escalation potential:

As mentioned there are constrains to trigger the vulnerability.
- There is no direct pathway in the current extension UI flow to provide the crafted array input to the remove() method. This reduces the likelihood of exploitation through typical user interactions.
- The vulnerability allows deletion of parent prototype methods, such as toString, but does not enable overriding them with different behaviors. This limits the potential for injecting malicious functionality into the prototype.

Status: Fixed

Dead code in mvelo.storage that is vulnerable to prototype pollution has been removed.
Fixed file: `/src/lib/lib-mvelo.js`

```
mvelo.storage.remove = function(id) {
  if (typeof id !== 'string') {
    throw new Error('id needs to be of type string');
  }
  return chrome.storage.local.remove(id);
};
```

8.3 Weak Validation in Web Key Directory (WKD) Blacklist

Overview

A vulnerability has been identified in Mailvelope's WKD lookup blacklist mechanism. The implementation uses weak regular expressions to validate email domains, allowing attackers to bypass WKD lookups by using maliciously crafted domain names that contain deceptive patterns (e.g., 'gmail.malicious.com' or 'outlook.attacker.com').

The weakness arieses from using simple regex patterns that match anywhere in the domain string:

```
[
  "gmail\\..*",
  "googlemail\\..*",
  "outlook\\..*",
  // [...]
]
```

Recommendations / Mitigation

- Implement strict domain validation using proper anchors:

```
[
  "^gmail\\.com$",
  "^outlook\\.com$",
  // ...other patterns
]
```

- Consider using a domain parsing library for proper validation
- Implement additional checks for subdomain depth
- Add explicit validation for known top-level domains
- Consider maintaining a list of exact domain matches instead of regex patterns

Proof of Concept

1. An email like usr@gmail.malicious.com would be considered a valid gmail account according to the malformed regex.

Status: Fixed

Anchoring regex patterns were implemented to prevent partial matches.
Fixed file: `src/res/defaults.json`

```
"keyserver": {
  "autocrypt_lookup": false,
  "key_binding": true,
  "mvelo_tofu_lookup": true,
  "oks_lookup": true,
  "wkd_blacklist":
  [
    "^gmail\\.com$",
    "^googlemail\\.com$",
    "^gmx\\..{2,3}$",
    "^outlook\\.com$",
    "^hotmail\\.com$",
    "^web\\.de$",
    "^yahoo\\..{2,3}$"
  ],
```

8.4 Information Disclosure through automatic WKD Key Lookup requests

Overview

A privacy vulnerability exists in Mailvelope's Web Key Directory (WKD) lookup implementation. When a user receives an encrypted email from any domain, the extension automatically performs WKD lookups by making HTTP requests to the sender's domain, potentially exposing user information through these requests.

The extension generates two types of requests:

- Advanced method:
  - https://openpgpkey.*${domain}*/.well-known/openpgpkey/*${domain}*/hu/*${localEncoded}*?l=$*{localPartEncoded}*
- Direct method:
  - https://*${domain}*/.well-known/openpgpkey/hu/*${localEncoded}*?l=*${localPartEncoded}*

Recommendations / Mitigation

- Change the default setting so that automatic WKD lookups are disabled by default, while still allowing users to enable them if needed.
- Include a warning in the user-configurable menu to inform end-users about the potential privacy implications of WKD lookups.
- Add an option to perform lookups only for allowed domains.

Proof of Concept

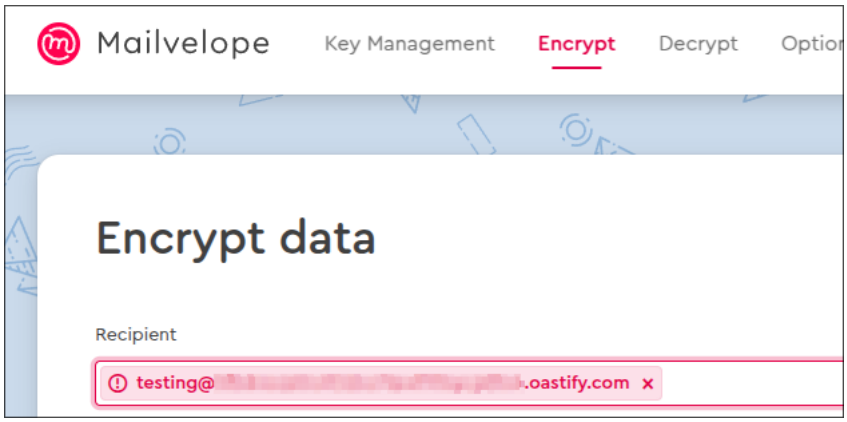
1. During static code analysis of the WKD lookup implementation, we identified that the extension automatically generates external HTTP requests without any user validation or privacy controls. This behavior is evident in the following code:

```
if (methodOfWKD === 'advanced') {
     return   `https://openpgpkey.${domain}/.well-known/openpgpkey/${domain}/hu/${localEncoded}?l=${localPartEncoded}`;
} else {
  return `https://${domain}/.well-known/openpgpkey/hu/${localEncoded}?l=${localPartEncoded}`;
}
```

2. When adding a recipient within the `*Encrypt data*` functionality, this code will trigger requests to potentially malicious domains, leaking user information such as IP addresses and User-Agent strings without the user's knowledge or consent.

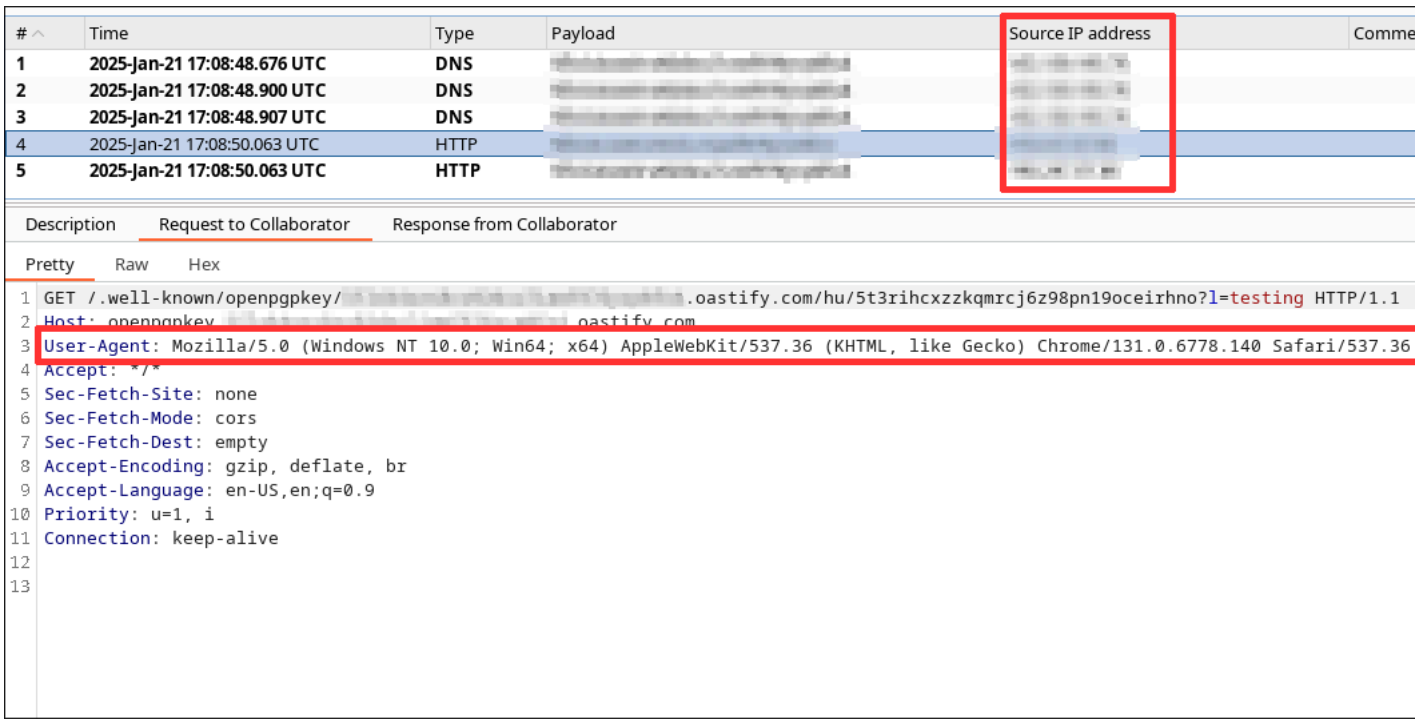
Steps to reproduce:

1. Configure and start Burp Suite to intercept browser traffic
2. Navigate to Mailvelope's extension interface and select the "Encrypt" option
3. Enter a recipient email address in the "Recipient" field as shown:



1. Observe in Burp's HTTP history the automatic requests triggered:

1. Inspect the request details in Burp's request tab to verify information leakage:



Status: No fix applied

Risk not addressed at this time, WKD can be disabled in the Mailvelope settings.

8.5 Insufficient message type validation in PGP block parser

Overview

A validation weakness has been identified in Mailvelope's PGP message type detection logic. The *getMessageType* function performs superficial regex checks that can be bypassed, allowing a private key block to be misidentified as a PGP message by including "*BEGIN PGP MESSAGE*" in the comment section.

The vulnerability exists because:

- PGP_MESSAGE type is checked before other types
- The regex only looks for the presence of "PGP MESSAGE" anywhere in the content
- Comment sections are not excluded from the type detection

Recommendations / Mitigation

- Implement strict PGP block type validation:

```
javascriptCopyfunction getMessageType(armored) {
  // Remove comments before type detection
  const cleanArmored = removeComments(armored);
  // Check actual block type markers
  const typeMatch = cleanArmored.match(/-----BEGIN PGP ([^-]+)-----/);
  if (typeMatch) {
    return mapBlockType(typeMatch[1]);
  }
}
```

- Add validation for block integrity
- Consider using a PGP parsing library for robust block detection
- Add explicit validation of block boundaries
- Consider implementing a whitelist of valid block types

Proof of Concept

1. During code review analysis, we identified that the message type detection can be bypassed with a specifically crafted PGP block:

```
// Vulnerable message type detection
export function getMessageType(armored) {
  if (/(BEGIN|END)\sPGP\sMESSAGE/.test(armored)) {
    return PGP_MESSAGE;  // Checked first, can be bypassed
  } else if (/(BEGIN|END)\sPGP\sPRIVATE\sKEY\sBLOCK/.test(armored)) {
    return PGP_PRIVATE_KEY;
  }
  // ...
}
```

2. This can be exploited using:

```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Mailvelope v5.2.0
Comment: BEGIN PGP MESSAGE
[Key content...]
-----END PGP PRIVATE KEY BLOCK-----
```

Status: Fixed

Fixed by moving least privileged message types to the end of the if-else chain plus aditional checks in specific uses.

```
export function getMessageType(armored) {
  if (/(BEGIN|END)\sPGP\sMESSAGE/.test(armored)) {
    return PGP_MESSAGE;
  } else if (/BEGIN\sPGP\sSIGNED\sMESSAGE/.test(armored)) {
    return PGP_SIGNATURE;
  } else if (/END\sPGP\sSIGNATURE/.test(armored)) {
    return PGP_SIGNATURE;
  } else if (/(BEGIN|END)\sPGP\sPRIVATE\sKEY\sBLOCK/.test(armored)) {
    return PGP_PRIVATE_KEY;
  } else if (/(BEGIN|END)\sPGP\sPUBLIC\sKEY\sBLOCK/.test(armored)) {
    return PGP_PUBLIC_KEY;
  }
}
```

8.6 Testing subdomain authorized in production

Overview

Mailvelope includes a whitelisted demo domain with API access enabled in the official extension distributed to end-users.

While this is not a vulnerability in itself, it **increases the attack surface** by granting a demo platform (that potentially may not follow highest scurity standars or be that may be vulnerable to **domain takeovers or other vulnerbailities)** access to a whitelisted domain with API access enabled.

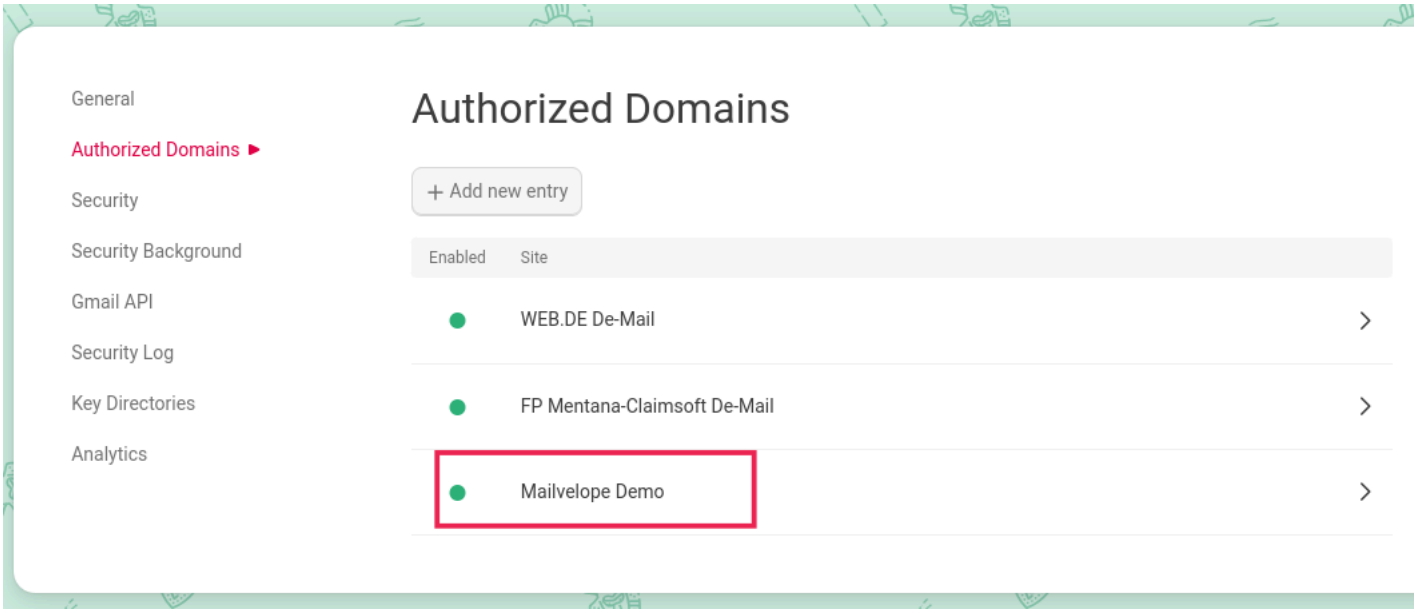
It is important to follow secure software best practices, ensuring separation of concerns by keeping development and testing environments separate from production, and block testing features/platforms from being deployed in production.

Recommendations / Mitigation

- Remove the testing subdomain **demo.mailvelope.com** from the list of **authorized domains** included by default in the browser extension.

Proof of Concept

1. Within the "Authorized Domains" section, the Mailvelope Demo entry appears under "Site" whitelisting by default the subdomain *.demo.mailvelope.com with a wildcard.



Status: Fixed

The testing purpose URL was deactivated by default in the list of authorized domains used in the production environment.

Fixed file: `src/res/defaults.json`

```
{
  "active": false,
  "site": "Mailvelope Demo",
  "https_only": true,
  "frames": [
    {
      "scan": true,
      "frame": "*.demo.mailvelope.com",
      "api": true
    }
  ]
}
```

8.7 Software out to Date

Description

Several outdated and vulnerable JavaScript libraries were identified in the project:

- AngularJS 1.8.3: direct dependency of mailvelope.
  - Multiple ReDoS vulnerabilities (CVE-2023-26117, CVE-2023-26116, CVE-2023-26118)
  - Cross-site Scripting vulnerability (CVE-2022-25869)
  - High-risk super-linear runtime vulnerability (CVE-2024-21490)
  - End-of-Life: Support discontinued since December 31, 2021
- Bootstrap 4.4.1: direct dependency of mailvelope.
  - Cross-Site Scripting vulnerability (CVE-2024-6531)
  - Multiple security patches missing

Recommendations / Mitigation

- Upgrade AngularJS to the latest stable version
- Update Bootstrap to the latest stable version
- Implement a dependency management strategy:
  - Regular automated dependency scanning
  - Security patch management process
  - Automated update testing pipeline

Proof of Concept

A dependency analysis using retire.js revealed an outdated AngularJs version being used in the application that contain known security issues:

```
❯ npm install -g retire
❯ retire
retire.js v5.2.5
Loading from cache:

↳ angularjs 1.8.3
angularjs 1.8.3 has known vulnerabilities: severity: medium; summary: angular vulnerable to regular expression denial of service via the $resource service, CVE: CVE-2023-26117, githubID: GHSA-2qqx-w9hr-q5gx;
```

Status: Fixed

The previously reported outdated AngularJS library is no longer detected by `retire`.

---

9 > About 0xche

0xche is a horizontal and self-organized collective of technologists committed to strengthening the information security of activist organizations and civil society in Latin America. Our identity combines a commitment to social transformation with solid technical expertise.

We believe that offensive security strategies are essential tools for understanding the vulnerabilities of systems used by organizations working with sensitive information. Our goal is to democratize access to technical knowledge in information security through diverse communication that is respectful of all genders and interdisciplinary knowledge.