Penetration Test Report

MapComplete

V 1.0
Amsterdam, September 4th, 2023
Confidential

Document Properties

Client	MapComplete
Title	Penetration Test Report
Target	MapComplete (https://github.com/pietervdvn/MapComplete)
Version	1.0
Pentester	Jacopo Jannone
Authors	Jacopo Jannone, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	August 22nd, 2023	Jacopo Jannone	Initial draft
0.2	September 1st, 2023	Marcus Bointon	Review
1.0	September 4th, 2023	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 August 21, 2023 and August 25, 2023, Radically Open Security B.V. carried out a penetration test for MapComplete.

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:

- MapComplete (https://github.com/pietervdvn/MapComplete)

The scoped services are broken down as follows:

- Web application penetration test: 4 days
- Reporting: 1 days
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a web application penetration test in order to assess the security of MapComplete. To do so ROS will analyze the scoped targets and guide the maintainers in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between August 21, 2023 and August 25, 2023.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Elevated, 1 Moderate and 4 Low-severity issues.

The main issue affecting MapComplete, which was rated with elevated severity, is the presence of multiple DOM-based Cross-Site Scripting (XSS) vulnerabilities described in CLN-001 (page 11). The issue is caused by a general lack of sanitization of the data retrieved from OpenStreetMap and other public sources, and may allow an attacker to inject malicious JavaScript code into pages rendered by the web application. For instance, the attacker might abuse XSS

vulnerabilities to gain unauthorized access to other users' OpenStreetMap accounts by stealing authorization tokens from the browser's local storage.

The presence of XSS vulnerabilities could have been mitigated by the presence of a robust Content-Security-Policy (CSP), which restricts the sources of resources (including scripts) that can be loaded by a web page. In the case of MapComplete, the complete lack of a CSP was reported as a low-severity finding in CLN-003 (page 17).

The mechanism used by MapComplete to authorize users with OpenStreetMap is susceptible to attacks aiming to mislead users into granting attackers access to their OpenStreetMap account. The root cause of this issue is the use of a shared OAuth application with hardcoded credentials for all instances of MapComplete, which resulted in a moderate-severity finding in CLN-002 (page 14).

Two additional low-severity findings concern the lack of security attributes in specific HTML tags. One is `rel="noopener"`, used to prevent an attack known as *Reverse Tabnabbing*. Without this attribute, the target page of HTML links could gain control of the tab running MapComplete; for instance, triggering a redirect towards a malicious page, as we reported in CLN-006 (page 21). The second is `integrity`, which allows browsers to verify that externally hosted files, such as scripts and stylesheets, are delivered without unexpected manipulation. If an attacker manages to modify an externally hosted script, the absence of integrity checks could lead to the execution of arbitrary code in the user's browser, as we discussed in CLN-004 (page 18).

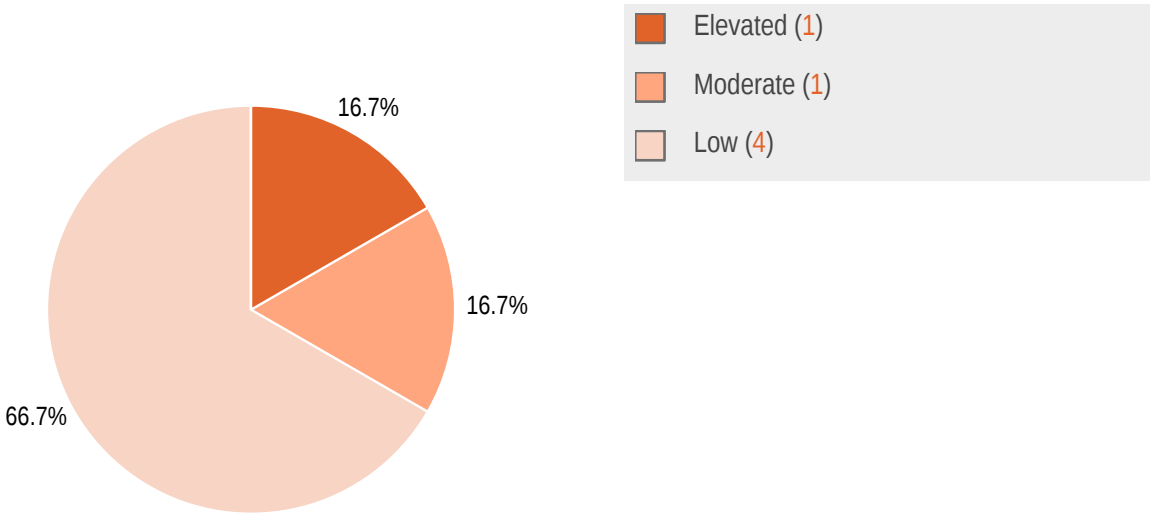
The last low-severity issue, in CLN-005 (page 19), is due to an insecure configuration of the CI/CD pipeline, which uses GitHub Actions sourced from third-party repositories without pinning them to specific commit hashes. As a result, compromised GitHub Actions could manipulate the code and properties of the repository hosting MapComplete, potentially introducing malicious code.

1.6 Summary of Findings

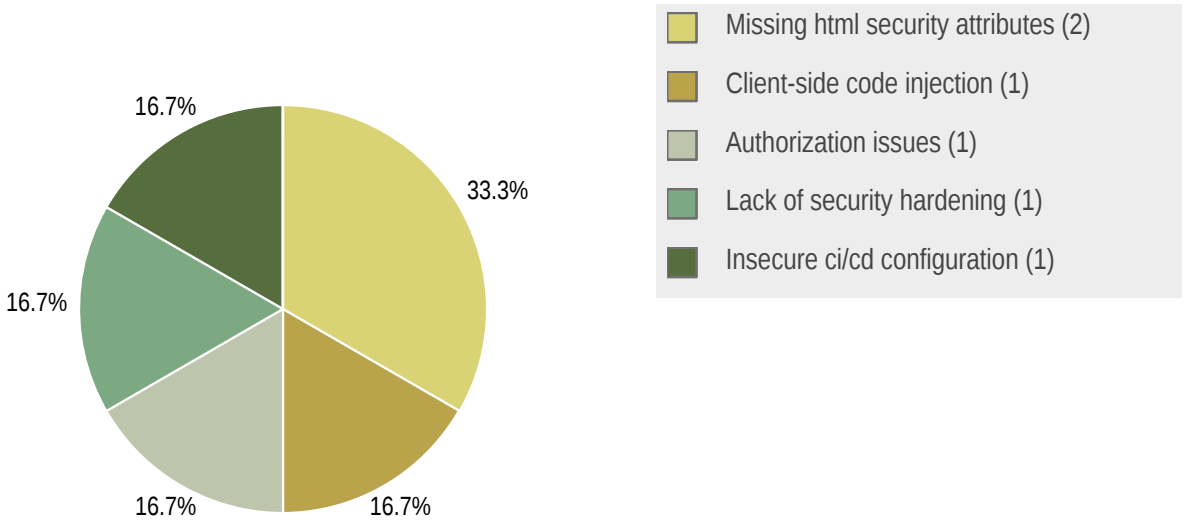
ID	Type	Description	Threat level
CLN-001	Client-Side Code Injection	Untrusted data from external sources is not properly sanitized when displayed in MapComplete, allowing potential attackers to execute arbitrary JavaScript code in the users' browser.	Elevated
CLN-002	Authorization Issues	The mechanism used by MapComplete to authorize users with OpenStreetMap is susceptible to attacks aiming to gain unauthorized access to the victim's account.	Moderate
CLN-003	Lack of Security Hardening	The MapComplete web application lacks a Content-Security-Policy, which could help mitigate cross-site scripting and other vulnerabilities.	Low
CLN-004	Missing HTML Security Attributes	HTML pages load resources from external servers without specifying an integrity attribute.	Low
CLN-005	Insecure CI/CD Configuration	GitHub Actions sourced from third-party repositories are not pinned to specific commit hashes, increasing the risk of threats in case a bad actor manages to add a backdoor to the action's repository.	Low

CLN-006	Missing HTML Security Attributes	When a web page includes an HTML link that opens in a new tab, the destination page can gain control of the page containing that link, for instance triggering a redirect.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-001	Client-Side Code Injection	• Use the `createElement()` and `appendChild()` methods to create and append new elements to the DOM, and the `textContent` property to populate them with strings. • Similarly, when untrusted input needs to be added as a HTML attribute value, use the `setAttribute` method. • As an alternative solution, use a sanitization library such as DOMPurify. • Systematically replace insecure code in *all* instances where data retrieved from external sources is displayed.
CLN-002	Authorization Issues	• This issue cannot be fully remedied due to a limitation of the authorization flows implemented by OpenStreetMap. • Migrate from OAuth 1.0 to OAuth 2.0; OpenStreetMap is deprecating OAuth 1.0. • Do not hardcode OAuth application credentials in the source code of MapComplete. Instead, make the credentials configurable on a per-instance basis. • Specify the correct redirect URIs on the configuration page of the OAuth application on OpenStreetMap.
CLN-003	Lack of Security Hardening	• Add a Content-Security-Policy (CSP) to all pages. • Since MapComplete is a fully static application, the CSP cannot be set in an HTTP header, so it should be added using an HTML `<meta>` tag in the HTML header of every page.

		• Move all scripts that are currently defined in HTML script tags into external JavaScript files, and include them into the HTML pages as needed with the `<script src=...>` directive. • Calculate an SRI hash for each external JavaScript file.
CLN-004	Missing HTML Security Attributes	• Whenever externally hosted resources are referenced in HTML pages, populate the `integrity` attribute with the cryptographic hash of the resource file.
CLN-005	Insecure CI/CD Configuration	• Replace release tags of third-party GitHub Actions with full-length commit SHA hashes.
CLN-006	Missing HTML Security Attributes	• Add the `rel="noopener"` attribute to all links that have the `target="_blank"` attribute.

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 (2017) 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 Findings

We have identified the following issues:

3.1 CLN-001 — Multiple DOM-Based Cross-Site Scripting Vulnerabilities

Vulnerability ID: CLN-001

Vulnerability type: Client-Side Code Injection

Threat level: Elevated

Description:

Untrusted data from external sources is not properly sanitized when displayed in MapComplete, allowing potential attackers to execute arbitrary JavaScript code in the users' browser.

Technical description:

A Cross-Site Scripting (XSS) vulnerability arises from the lack of proper validation and sanitization of the data served as output and included in the page presented to the user. This allows an attacker to inject malicious JavaScript code in the page rendered by the web application.

Specifically, a DOM-based XSS occurs when the vulnerability resides in the client-side code, which takes untrusted data (e.g. from an API response) and includes it into the Document Object Model (DOM).

The MapComplete application is affected by multiple DOM-based XSS vulnerabilities caused by a general lack of sanitization of the data retrieved from OpenStreetMap and other public sources.

Throughout the application, we identified two main injection methods. The first affects most, if not all, free-text strings originating from external sources, and can be triggered with a simple HTML tag containing JavaScript code. An example is `<img src=x onerror=alert(window['location']['href']) />`. Tests confirmed that the JavaScript code is executed when such a payload is included in:

- The OpenStreetMap user profile name
- The OpenStreetMap user profile description
- Articles embedded from Wikipedia
- Map feature names
- Mangrove user reviews
- OpenStreetMap user notes
- Free-text attributes of map features (e.g., name of the company operating a facility)

- Titles, authors, and license names of Wikimedia Commons images

This list should not be considered exhaustive; there may be other injection points.

The second injection method is in the user settings page. Specifically, the link to download the private key for the user's Mangrove Account has a `download` attribute that includes the OpenStreetMap user profile name without proper sanitization:

```
<a href='data:application/json,{mangroveidentity}' download='mangrove_private_key_{_name}'>
    Download the private key for your Mangrove Account
</a>
```

If a user chooses a specially crafted username, such as `foo'><img src=x onerror=alert(window['location']['href']) />`, the resulting HTML code added to the DOM becomes

```
<a href='data:application/json,...' download='mangrove_private_key_foo'>
    <img src=x onerror=alert(window['location']['href']) />
    '>Download the private key for your Mangrove Account
</a>
```

and results in the execution of the JavaScript payload.

Impact:

A Cross-Site Scripting (XSS) vulnerability can grant an attacker full control over the victim's browser window where MapComplete is running. For instance, the attacker might abuse XSS vulnerabilities to gain unauthorized access to other users' accounts.

Consider the scenario where the attacker writes some JavaScript code to extract the OpenStreetMap authorization tokens from the browser's local storage and send them over to a remote server. They could hide a XSS payload with that code in an OpenStreetMap note, so that when another user opens the note using MapComplete, their OpenStreetMap authorization tokens are stolen. The attacker would then gain unauthorized access to the victim's OpenStreetMap account.

Recommendation:

The best way to avoid XSS vulnerabilities when generating DOM elements that can contain untrusted strings is to use the `createElement()` and `appendChild()` methods to create and append new elements to the DOM, and the `textContent` property to populate them with strings. This approach maintains a clear separation between code and data and prevents injection attacks, as opposed to directly writing HTML code in JavaScript and concatenating external data.

For instance, the following code is vulnerable to injection:

```
let userName = getUserName();

// Vulnerable alternative 1
```

```
let greeting = `Welcome, <b>${userName}</b>!`;

// Vulnerable alternative 2
let greeting = "Welcome, <b>{name}</b>!".replace("{name}", userName);

// Vulnerable alternative 3
let greeting = "Welcome, <b>" + userName + "</b>!";

divContainer.innerHTML = greeting;
```

While the following code achieves the same result, but is not vulnerable:

```
let userName = getUserName();

let nameNode = document.createElement("b");
nameNode.textContent = userName;

divContainer.appendChild(document.createTextNode("Welcome, "));
divContainer.appendChild(nameNode);
divContainer.appendChild(document.createTextNode("!"));
```

Similarly, when untrusted input needs to be added as an HTML attribute value, use the `setAttribute` method instead of directly concatenating the value within HTML code:

```
let userName = getUserName();

// Vulnerable code
divContainer.innerHTML = `<a href="..." download="mangrove_private_key_${userName}">Download the
 private key for your Mangrove Account</a>`;

// Secure code
let linkNode = document.createElement("a");
linkNode.setAttribute("href", "...");
linkNode.textContent = "Download the private key for your Mangrove Account";
linkNode.setAttribute("download", `mangrove_private_key_${userName}`);
divContainer.appendChild(linkNode);
```

As an alternative solution, a sanitization library such as https://github.com/cure53/DOMPurify can be used to strip unsafe code from external strings before using the `innerHTML` property. This second approach can be particularly useful when the external data may contain HTML markup, but not scripts, as in the case of Wikipedia articles. The DOMPurify library can also be configured to keep or remove HTML tags as appropriate.

In order to apply an effective remediation, make sure to systematically replace insecure code in *all* instances where data retrieved from external sources is displayed, even if not explicitly mentioned in this finding.

3.2 CLN-002 — Insecure Authorization Mechanism

Vulnerability ID: CLN-002

Vulnerability type: Authorization Issues

Threat level: Moderate

Description:

The mechanism used by MapComplete to authorize users with OpenStreetMap is susceptible to attacks aiming to gain unauthorized access to the victim's account.

Technical description:

MapComplete uses the OAuth 1.0 authorization flow to operate on OpenStreetMap on behalf of its users. The current design of the authorization mechanism is susceptible to attacks aiming to mislead users into granting attackers access to their OpenStreetMap account.

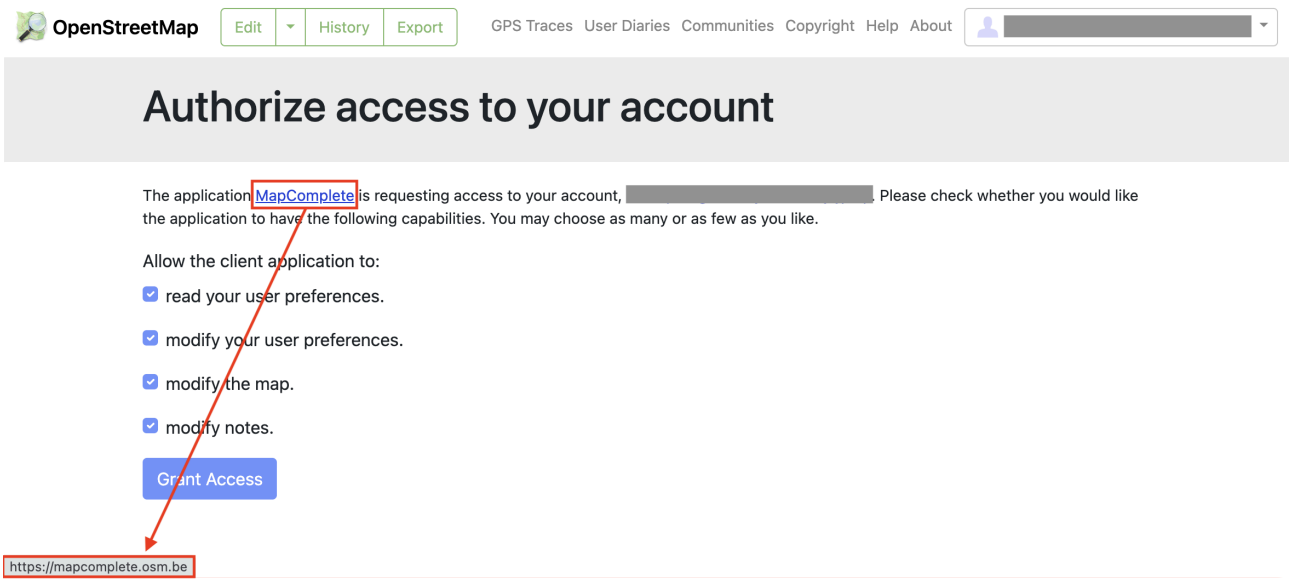
The root cause of this issue is the use of a shared OAuth application for all instances of MapComplete. In particular, a OAuth 1.0 application having Consumer Key `hivV7ec2o49Two8g9h8Is1VIiVOgxQ1iYexCbvem` was registered in the OpenStreetMap account of the project owner, and the application credentials (Consumer Key and Consumer Secret) were hardcoded in MapComplete's source code (OsmConnection.ts:30). As a result, all self-hosted instances of MapComplete present themselves to OpenStreetMap as a single application belonging to the project owner.

A direct consequence of this design choice is that it is not possible to configure the OAuth application on OpenStreetMap with an allow-list of callback URLs that are authorized to receive the authorization code at the end of a successful authorization flow. Instead, in order to support self-hosted instances, any callback URL passed during the authorization flow must be accepted. This allows an attacker to set up a malicious website that impersonates the main MapComplete instance, while actually stealing OpenStreetMap authorization codes to gain unauthorized access to the victim's profile.

The steps to carry out such an attack are the following:

1. The attacker hosts a modified instance of MapComplete (e.g. `https://evil-mapcomplete.com`).
2. The victim visits `https://evil-mapcomplete.com` and proceeds to log in with OpenStreetMap.

3. `https://evil-mapcomplete.com` redirects the user to the OpenStreetMap OAuth authorization page, shown in the figure below. Notice that the authorization page indicates that MapComplete, with URL `https://mapcomplete.osm.be`, is requesting access to the user's account, potentially leading the victim to believe that they are granting access to the main MapComplete instance.



4. The victim grants access.
5. OpenStreetMap redirects the victim, while also passing the authorization code, to `https://evil-mapcomplete.com`, which can capture the code and use it to access the victim's OpenStreetMap profile.

Impact:

An attacker could steal OpenStreetMap OAuth authorization codes and gain unauthorized access to the victim's profile. The scope granted to MapComplete gives read and write access to user preferences, maps and notes. Thus, the attacker would gain full access to the victim's user profile as well as the OpenStreetMap database, and could potentially perform changes and delete publicly accessible data on behalf of the victim.

Recommendation:

At the time of writing, this issue cannot be fully remedied due to a limitation of the authorization flows implemented by OpenStreetMap, as detailed below.

Ideally, take the following corrective actions:

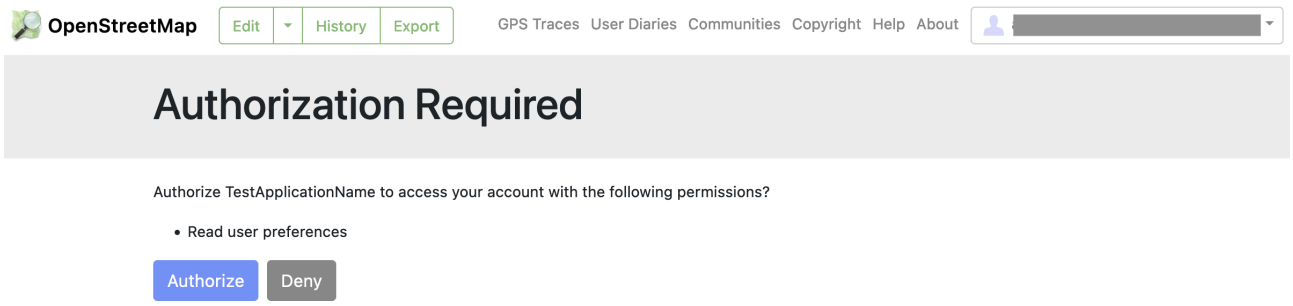
1. Migrate from OAuth 1.0 to OAuth 2.0. This is a prerequisite to successfully carry out the following actions, since redirect URI restrictions were formalized with OAuth 2.0. Furthermore, note that OpenStreetMap is deprecating OAuth 1.0.

2. Do not hardcode OAuth application credentials in the source code of MapComplete. Instead, make the credentials configurable on a per-instance basis. Each user wanting to host an instance of MapComplete should register their own OAuth 2.0 application on OpenStreetMap, and configure MapComplete to use the corresponding credentials.
3. Specify the correct redirect URIs on the configuration page of the OAuth application on OpenStreetMap. For instance, the main MapComplete instance at `https://mapcomplete.osm.be` should be configured with the following redirect URIs:

```
https://mapcomplete.osm.be/cyclofix.html
https://mapcomplete.osm.be/waste.html
https://mapcomplete.osm.be/etymology.html
[...]
```

If a list of allowed redirect URIs is present, OAuth 2.0 prevents clients from being redirected to addresses that are not included in that list. This stops an attacker from using another application's credentials (such as the credentials of the main instance of MapComplete) and cause the victim to land, along with the authorization code, on an attacker-controlled website.

An attacker could still register their own application on OpenStreetMap, set up a malicious instance of MapComplete, and persuade users into using it to log into OpenStreetMap. Ideally, the OAuth authorization page on OpenStreetMap would clearly show which application the user is logging into, including the owner's name and the full application URL. This would make it evident to the user that they are not logging into a trusted instance of MapComplete. However, tests showed that this is not the case. When the OAuth 2.0 flow is used, the OpenStreetMap authorization page only shows the registered application name, with no other details, author names or URLs, as shown in the following figure.



Due to this reason, it remains trivial for an attacker to impersonate a trusted instance of the MapComplete application, and it is not possible to fully remedy this issue without changes from OpenStreetMap.

3.3 CLN-003 — Lack of Content-Security-Policy

Vulnerability ID: CLN-003

Vulnerability type: Lack of Security Hardening

Threat level: Low

Description:

The MapComplete web application lacks a Content-Security-Policy, which could help mitigate cross-site scripting and other vulnerabilities.

Technical description:

The Content-Security-Policy (CSP) restricts which resources (such as JavaScript, CSS, images, etc.) can be loaded, and the URLs that they can be loaded from. Its purpose is to block, for instance, any forms that send information to external domains or third-party JavaScript code. It helps mitigate the risk of Cross-Site Scripting (XSS), clickjacking, and other potential code-injection attacks that could result from the execution of malicious content within the web application.

Impact:

The lack of a Content-Security Policy makes it easier to exploit any existing client-side vulnerabilities, such as cross-site scripting.

Recommendation:

As a form of defense in depth, add a Content-Security-Policy (CSP) to all pages in order to restrict the scripts and objects that can be included and executed within the web application context.

Since MapComplete is a fully static application, the CSP cannot be set in an HTTP header, so it should be added using an HTML `<meta>` tag in the HTML header of every page:

```
<meta http-equiv="Content-Security-Policy" content="script-src ...">
```

As a prerequisite for being able to apply a strong CSP, move all scripts that are currently defined *inline* in HTML pages to external JavaScript files, and include them into the HTML pages as needed with the `<script src=...>` directive. Furthermore, calculate a Subresource Integrity (SRI) hash for each external JavaScript file.

Then, the following template can be used as a starting point for writing a solid CSP:

```
script-src 'sha384-...' 'unsafe-inline';
object-src 'none';
```

```
base-uri 'none';
```

After the `script-src` directive, append all the hashes of the external JavaScript files that are included in the page. The final `unsafe-inline` directive is used for backwards compatibility with older browsers, and is ignored by modern browsers that support checking for hashes.

3.4 CLN-004 — Missing HTML Integrity Attributes

Vulnerability ID: CLN-004

Vulnerability type: Missing HTML Security Attributes

Threat level: Low

Description:

HTML pages load resources from external servers without specifying an `integrity` attribute.

Technical description:

The `integrity` attribute allows browsers to verify that externally hosted files, such as scripts and stylesheets, are delivered without unexpected manipulation.

Integrity checks work by verifying that the cryptographic hash of the external resource matches the base64-encoded hash specified in the `integrity` attribute of the HTML element.

Impact:

If an attacker is able to modify an externally hosted resource (e.g., by exploiting a vulnerability in a CDN or via supply chain attacks), the absence of integrity checks could lead to the execution of arbitrary code in the user's browser.

Recommendation:

Whenever externally hosted resources are referenced in HTML pages, populate the `integrity` attribute with the cryptographic hash of the resource file.

As an example, consider the following line from 404.html:53.

```
<script async data-goatcounter="https://pietervdvn.goatcounter.com/count" src="//gc.zgo.at/
count.js"></script>
```

Change the `<script>` tag to include the `integrity` attribute with a hash of the externally hosted `count.js` script. The `crossorigin="anonymous"` attribute is also required to satisfy the cross-origin resource policy:

```
<script async data-goatcounter="https://pietervdvn.goatcounter.com/count" src="//gc.zgo.at/
count.js" integrity="sha384-gtO6vSydQeOAGGK19NHrlVLNtaDSJjN4aGMWschK+dwAZOdPQWbjXgL+FM5XsgFJ"
 crossorigin="anonymous"></script>
```

Apply the same steps to the following files:

- 404.html:53
- index.html:54
- statistics.html:17
- theme.html:100

3.5 CLN-005 — Third-Party GitHub Actions Not Pinned to Commit Hash

Vulnerability ID: CLN-005

Vulnerability type: Insecure CI/CD Configuration

Threat level: Low

Description:

GitHub Actions sourced from third-party repositories are not pinned to specific commit hashes, increasing the risk of threats in case a bad actor manages to add a backdoor to the action's repository.

Technical description:

The MapComplete repository is currently configured to use GitHub Actions from third-party repositories, such as `mshick/add-pr-comment`, as defined in .github/workflows/deploy_pietervdvn.yml:

```
name: Deployment on pietervdvn
[...]
jobs:
  build:
    [...]
    steps:
      - uses: mshick/add-pr-comment@v1
        name: Comment the PR with the review URL
        [...]
```

Release tags (such as `v1` in the example above) are mutable, meaning that the code of the GitHub Action corresponding to `mshick/add-pr-comment@v1` can be changed with new commits to the repository.

Impact:

In the event of supply chain attacks, compromised GitHub Actions can manipulate the code and properties of the repository hosting MapComplete, potentially introducing malicious code.

Recommendation:

In order to mitigate the risk of a bad actor adding malicious code to the action's repository, replace release tags of third-party GitHub Actions with full-length commit SHA hashes. For instance, for the case shown above:

```
name: Deployment on pietervdvn
[...]
jobs:
  build:
    [...]
    steps:
      - uses: mshick/add-pr-comment@a96c578acba98b60f16c6866d5f20478dc4ef68b
        name: Comment the PR with the review URL
        [...]
```

Apply the same change to all third-party GitHub Actions:

- Defined in file .github/workflows/deploy_pietervdvn.yml:
  - github/codeql-action/init
  - github/codeql-action/autobuild
  - github/codeql-action/analyze
- Defined in file .github/workflows/codeql-analysis.yml:
  - mshick/add-pr-comment
- Defined in file .github/workflows/reuse-compliance-check.yml:
  - fsfe/reuse-action
- Defined in file .github/workflows/validate-pr.yml:
  - mshick/add-pr-comment

3.6 CLN-006 — Reverse Tabnabbing

Vulnerability ID: CLN-006

Vulnerability type: Missing HTML Security Attributes

Threat level: Low

Description:

When a web page includes an HTML link that opens in a new tab, the destination page can gain control of the page containing that link, for instance triggering a redirect.

Technical description:

In certain thematic maps, such as *Ghost bikes*, MapComplete displays links to external (potentially untrusted) web pages. Such links are included in the HTML page as `<a>` tags with the `target="_blank"` attribute, causing the link to open in a new browser tab.

When the `target="_blank"` attribute is present in an HTML link, the target page is given the ability to control the parent tab (i.e., the tab with the page containing the link) via the `opener` object instance. For instance, the target page can trigger a redirect of the parent tab to an arbitrary URL.

This behavior can be overridden by adding the `rel="noopener"` attribute, which prevents the destination page from gaining control of the parent tab.

Impact:

If a link opening in a new tab lacks the `rel="noopener"` attribute and the target page is (or becomes) controlled by a bad actor, the attacker could gain control of the tab running MapComplete. For instance, they could trigger a redirect towards a phishing page with the aim of tricking users and stealing their OpenStreetMap credentials. This kind of attack is known as *Reverse Tabnabbing*.

Recommendation:

- Add the `rel="noopener"` attribute to all links that have the `target="_blank"` attribute.

4 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.1 NF-007 — Sensitive Data Correctly Cleared Upon Logout

MapComplete stores OpenStreetMap OAuth tokens in the browser's local storage. When a user logs out of their account, the tokens should be removed to prevent potential recovery and reuse, which could result in unauthorized access.

The pentest confirmed that OAuth tokens are correctly removed from local storage when the user logs out.

4.2 NF-008 — Built-in Web Server Listens on All Interfaces

MapComplete ships with a built-in web server to run the application once it is successfully built. Analysis of the web server revealed that, by default, it listens on all network interfaces.

Unless network-level restrictions are present (e.g., a firewall), exposing MapComplete on all interfaces could allow unauthorized users on the local network to reach MapComplete. Nevertheless, unauthorized users would merely be able to use the reachable instance of MapComplete like any other public instance, and would not gain access to any private or sensitive data.

Furthermore, the project documentation suggests that the built-in web server is only intended for development and debugging purposes, and not for the production deployment of MapComplete. Consequently, we do not consider that this issue has any security impact.

4.3 NF-009 — Incomplete Markdown Sanitization

Several classes defined in the MapComplete project include a `AsMarkdown()` method that outputs Markdown code for the purpose of generating documentation. In some cases, the methods perform some sanitization of the input text by escaping Markdown special characters; however, they do so in an incomplete manner. In other cases, they perform no escaping at all.

As an example, consider the following line of code from file src/UI/Base/Table.ts:31-33:

```
const table = this._contents
    .map((row) => row.map((el) => el?.AsMarkdown()?.replace("|", "\\|") ?? " ").join(" | "))
    .join("\n")
```

The code is meant to escape `|` characters in the input data, so that the data can be included in a Markdown table. However, JavaScript's `replace()` method only replaces the first occurrence of a string; thus, only the first occurrence of `|` is replaced instead of all occurrences, potentially resulting in incorrect output. The `replaceAll()` method should

be used instead. Furthermore, the code does not correctly handle the case of input data containing a `\` character, which should be escaped as well.

As another example, the `AsMarkdown()` method defined in src/UI/Base/FixedUiElement.ts:15-26 does not escape special characters at all when rendering inline code or bold text:

```
if (this.HasClass("code")) {
    [...]
    return "`" + this.content + "`"
}
if (this.HasClass("font-bold")) {
    return "*" + this.content + "*"
}
```

Since the `AsMarkdown()` methods are only used for automatically generating documentation, and the only consequence is incorrectly formatted text, this issue has no security impact.

4.4 NF-010 — Community Translations Not Considered a Threat Vector

MapComplete uses an external service that allows users to provide translations for the strings used in the web interface. Community-provided translations would pose a security threat if they were automatically imported into the code or retrieved at runtime, since a bad actor could inject malicious code that would be executed in the users' browsers.

In actuality, translated strings must be manually merged into the code by the project maintainers, like any other pull request. This workflow allows the maintainers to review the translated strings and ensure they are safe before merging them into the code, nullifying the threat.

4.5 NF-011 — No Open Redirections Found

Open redirection vulnerabilities occur when an application unsafely includes user-controllable information in the destination of a redirection. In this scenario, an attacker can create a URL pointing to the application but leading to an immediate redirection to an arbitrary external domain. An attacker could leverage this behavior to facilitate phishing attacks against users of the application.

Our static and dynamic analysis of MapComplete did not find any instance of such a vulnerability.

5 Future Work

- **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 an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

6 Conclusion

During this crystal-box penetration test we found 1 Elevated, 1 Moderate and 4 Low-severity issues.

The primary issue affecting MapComplete is the presence of cross-site scripting vulnerabilities, allowing potential attackers to execute arbitrary code on the users' browser. The lack of a Content-Security-Policy contributes to making these vulnerabilities easily exploitable. The mechanism used by MapComplete to authorize users with OpenStreetMap can be exploited by attackers to mislead users into granting them access to their OpenStreetMap account. The lack of security attributes in specific HTML tags could enable an attack known as *Reverse Tabnabbing* as well as the execution of malicious code in the event of supply chain attacks. Finally, a misconfiguration could allow compromised GitHub Actions to introduce malicious code into the MapComplete repository.

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

Jacopo Jannone	Jacopo Jannone holds a M.Sc. degree *cum laude* in Computer Science and Engineering from Politecnico di Milano. With his thesis, he proposed a TrustZone-based attack detection system for industrial robot controllers. He has a strong passion for reverse engineering, which as an information security professional he mainly applies to the analysis of mobile, web and native applications. His interests extend to identification systems (smart cards and RFID), radio frequency communication tools, embedded systems, and microcontrollers. In his free time he develops open source projects, plays CTF competitions with the *Tower of Hanoi* and *mHACKeroni* teams, and reverse engineers anything he finds interesting.
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.