****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **October 8, 2023** | **Lawrence Abbott** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Lawrence Abbott

## Algorithm Cipher

As we move forward with securing the Artemis Financial web application, we would like to bring to your attention three main areas where the proper encryption cipher will be key in protecting the application and its data. The three important things to consider are encryption of data in transit, encryption of data at rest, and the management of the cryptographic keys. While there are other areas of interest that are important to the security of the web application, these rank amongst the top in priority.

When it comes to encrypting data in transit it makes sense to have both an encryption process as well as a decryption process. If possible, the cipher should be efficient, as data in the process of being transferred is best when it can be delivered both quickly and securely. After careful consideration, we recommend using the symmetric cryptographic cipher AES for this purpose as the encryption process is efficient and very resistant to attack. Furthermore, it can be used in conjunction with cipher suites that provide forward secrecy, which will be important to protect data of past transmission, should the key ever be compromised.

As for protecting data at rest, such as archival files, we will again recommend AES for the same reasons as stated above; quick and secure. Archival files, for instance, are constantly growing and when every second counts, an AES cipher can decrypt those encrypted data blocks in an efficient manner.

On the topic of management of cryptographic keys, we recommend a disciplined approach as any compromise of keys could lead to a breach, regardless of the strength of the cipher. Identify those individuals that have access to and control of keys throughout their lifecycle. Uniquely identify keys and the data that they protect, including other cryptographic keys. Develop semi-frequent, recurring audits of keys and establish a timeframe for how often keys should be replaced. It's also good practice to have an established compromise-recovery plan in the event of key compromise, which should include a method of key revocation to minimize the effect of the compromise. (OWASP, 2023)

## Certificate Generation

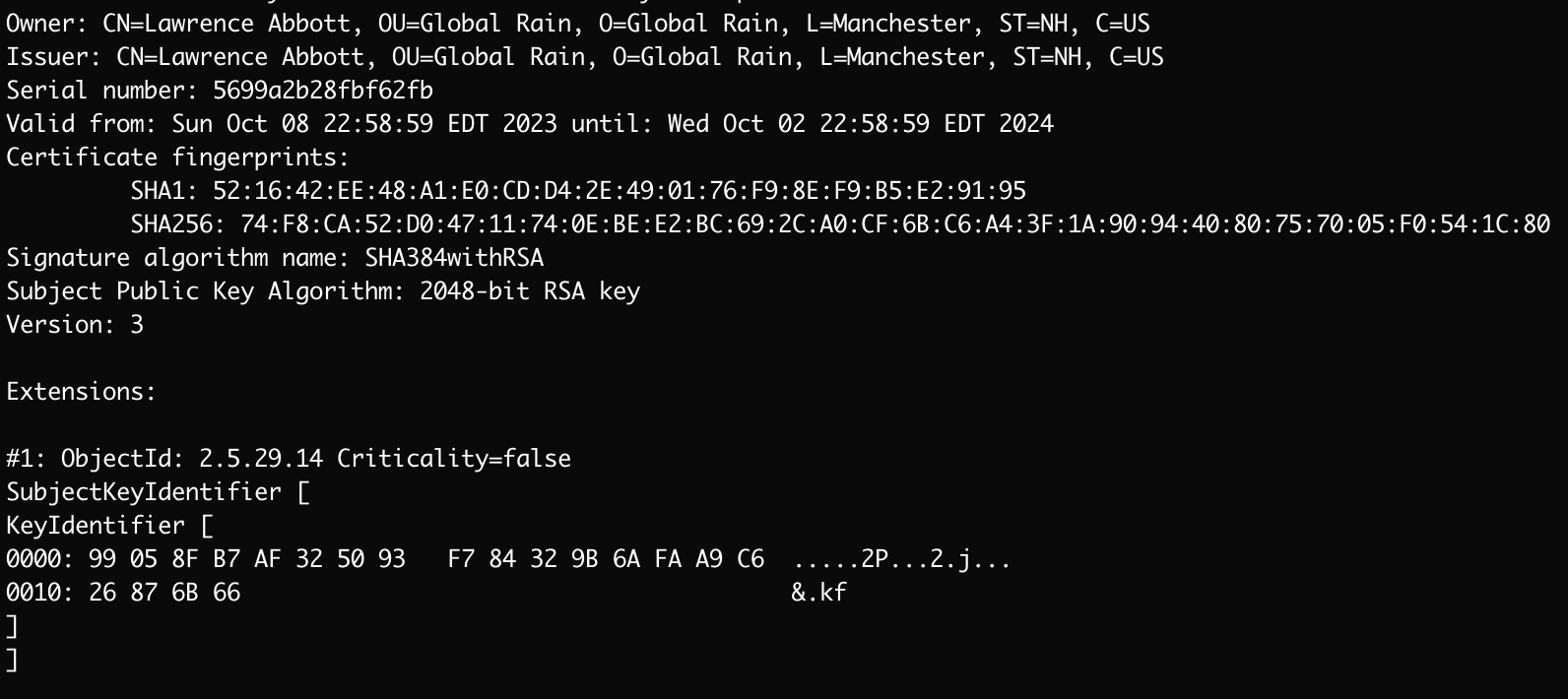


Figure 1

## Deploy Cipher

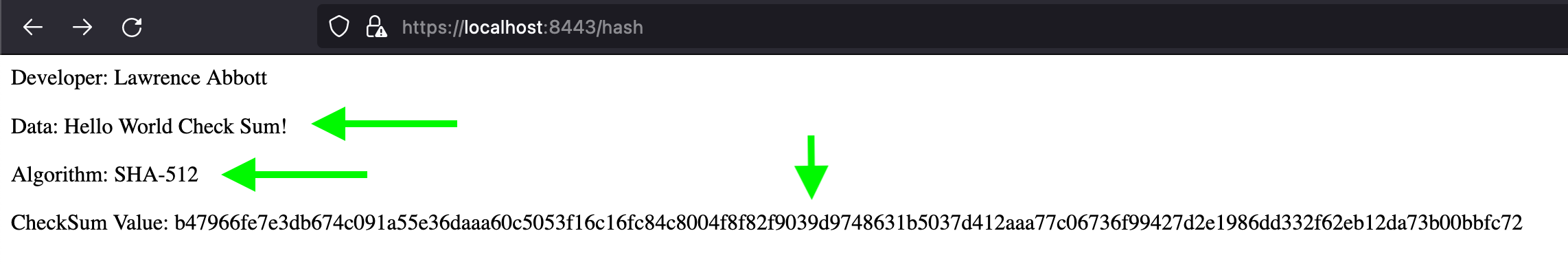
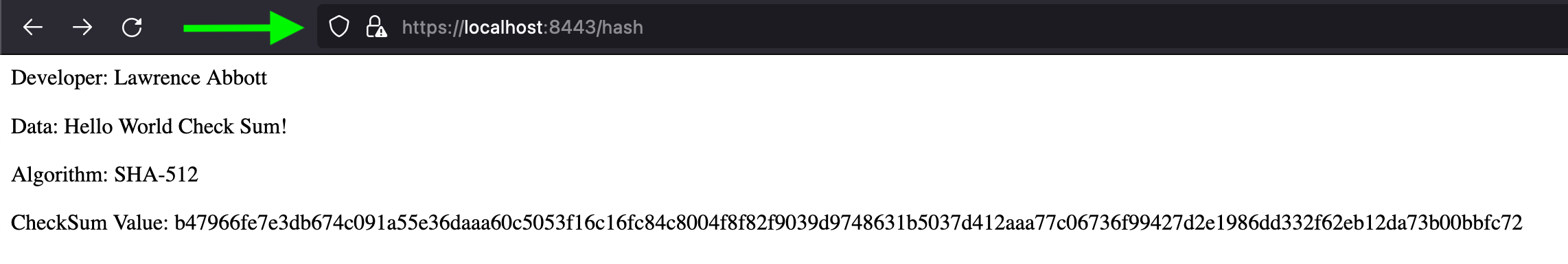


Figure 2

## Secure Communications

Figure 3

## Secondary Testing

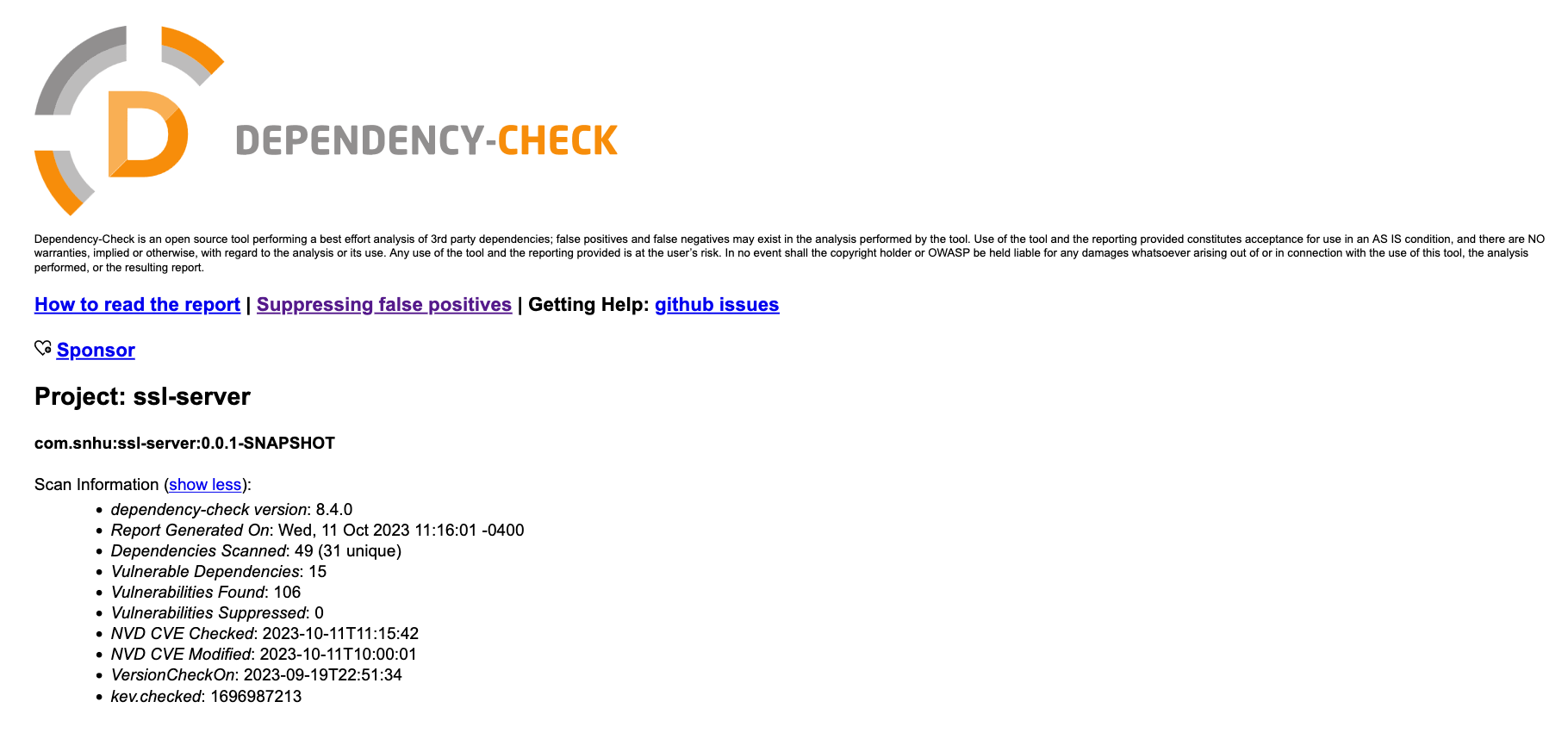
 Figure 4



Figure 5.1



Figure 5.2



Figure 6

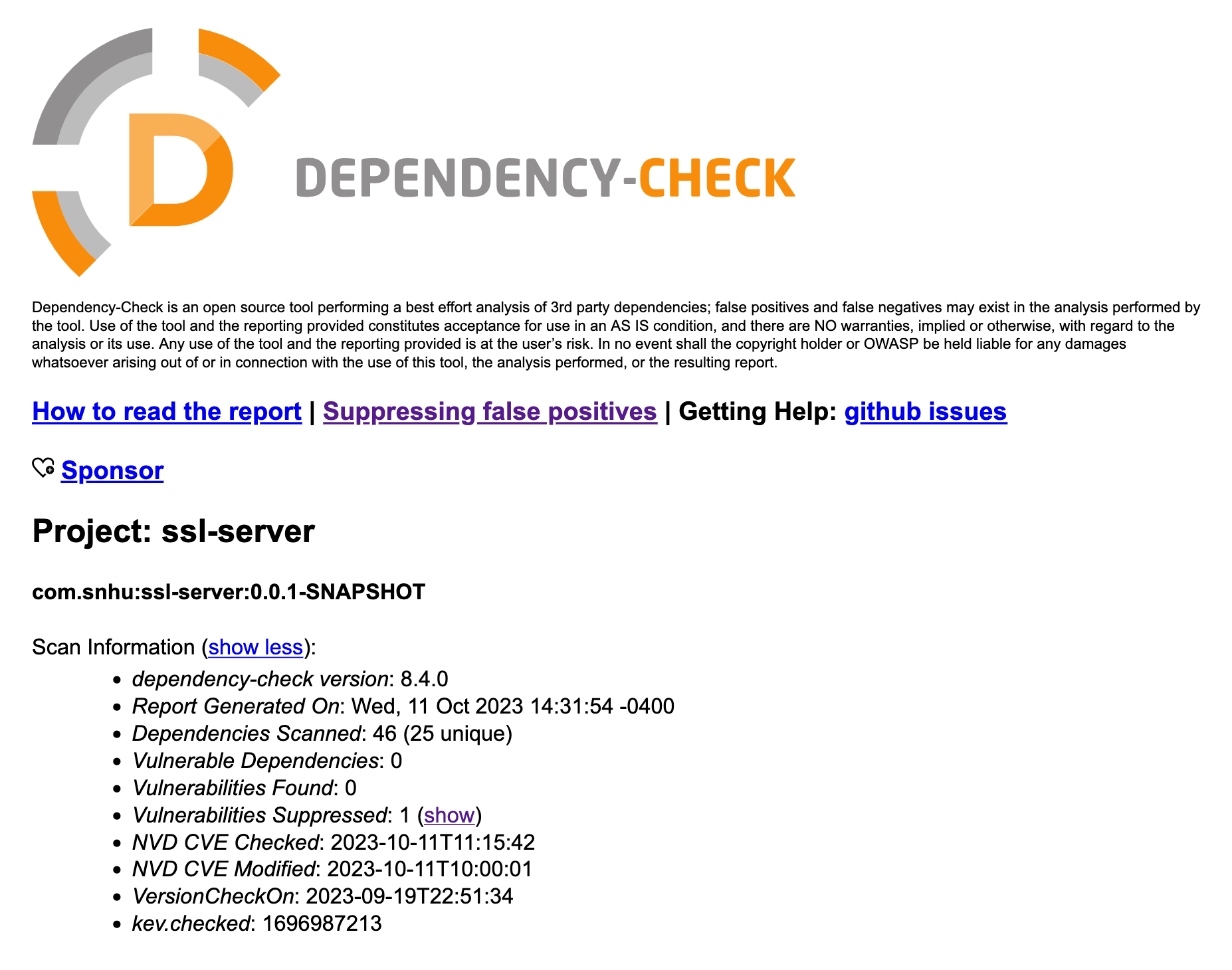


Figure 7

## Functional Testing

In review of the existing code and in seeking out any syntactical and logical vulnerabilities, we can say that there don’t appear to be any, as of yet. Of course, the code base is still relatively young. To ensure that the code is developed with respect to syntactical and logical vulnerabilities, we recommend providing ample time during development for peer code reviews. In other words, ask your developers to inspect each other's code for any potential threats, before merging any new features or fixes into the code base. Additionally, it is a good idea to regularly run static testing, such as the one we have used in this report, called maven-dependency-check.

## Summary

## To begin, we ran a static dependency check against the existing code, prior to making any modifications. This was done to establish a baseline for any existing vulnerabilities that require our attention. The result of the initial static dependency check can be seen in **Figure 4**. After reviewing the vulnerabilities of the architecture, including any dependency vulnerabilities, we came to the conclusion that the best means of mitigating these vulnerabilities was to upgrade the packages of the web application. In an established web application this might not have been as easily accomplished, however, since the code base is so young, upgrading the system architecture was quite easy because there was not a large amount of code that would be susceptible to a non-backward-compatible changes from the vendor.

## As seen in **Figure 5.1**, we upgraded Java from 1.8 to 21, which eliminated a large percentage of the vulnerabilities. Additionally, we found that by upgrading two other architecture dependencies, snakeyaml and jackson-databind, we could eliminate still more vulnerabilities (**Figure 5.1 and Figure 5.2**). By upgrading packages in this manner, we were able to leverage the work that has been done by the package vendors in mitigating their own software vulnerabilities, without the need to reinvent the wheel, so to speak.

The result of these architectural updates brought the total number of vulnerabilities down from 106 to one. The remaining vulnerability has been suppressed because upon further review, the vendor has disputed its validity, “because the steps of constructing a cyclic data structure and trying to serialize it cannot be achieved by an external attacker.” (NIST, 2023). Furthermore, it appears that there have been updates to jackson-databind, the dependency in question, addressing the underlying stack overflow issue. In this case, the improvement includes throwing an exception if the parsing depth reaches a certain threshold. (PoppingSnack, 2022). See **Figure 6** for what was included in the suppression.xml file.

Finally, we’ve included a screenshot of the final static dependency check (**Figure 7**), performed after the upgrades and updates to the pom.xml file. As depicted in the screenshot, the single remaining vulnerability was suppressed as it doesn’t appear to present any risk, based on our research into the matter.

## Industry Standard Best Practices

As demonstrated in this report, we have taken care to use industry best practices in our assessment and subsequent mitigation of any discovered vulnerabilities. We will ask that your staff, at Artemis Financial, join us in helping to keep this and any other software applications secure by developing your own best practices and regularly auditing your security practices, including proper management of encryption keys and regular security audits. At Global Rain, we are adamant about taking software security seriously and feel confident that together, with your continued support, we can continue to provide the service of security to you, your company, your users, and your stakeholders. If you have any questions as to the contents of this document or require further explanation today or in the future, we’re here to help you. Don’t hesitate to ask. Security can be a daunting task, but we can help you along the way. Thank you for the opportunity and we look forward to our continued partnership in securing this software application as it grows and develops.

**Sources**

OWASP - CheatSheets Series. (2023). *Key Management Cheat Sheet*. OWASP.

<https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html>

NIST - National Institute of Standards and Technology. (2023, July 26). *CVE-2023-35116 Detail.* US Department of Commerce, National Institute of Standards and Technology. <https://nvd.nist.gov/vuln/detail/CVE-2023-35116>

PoppingSnack. (2023, June 9). *Stack overflow error caused by serialization of Map with cyclic dependency -- NOT CVE* [Online forum post]. GitHub, Inc.<https://github.com/FasterXML/jackson-databind/issues/3972>