****

# 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** | **04/21/24** | **Joseph Grzywinski** |  |

## 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

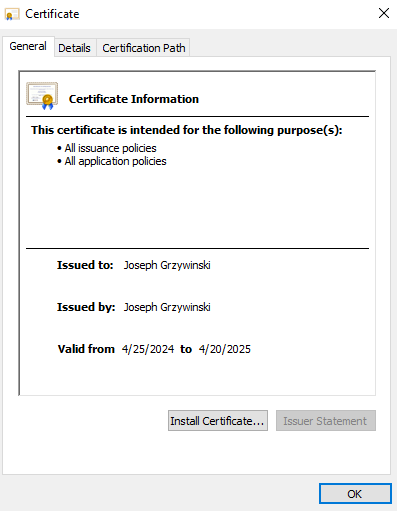
Joseph Grzywinski

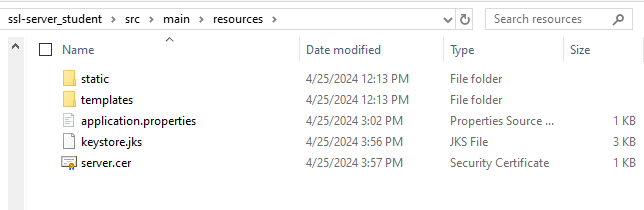
## Algorithm Cipher

The algorithm cipher that I chose to implement for Artemis financials application was SHA-256. One of the biggest reasons I chose to go this route was to keep consistency in the application since the certificate uses it as well to generate its public fingerprint. SHA-256 is very easy to implement and is highly used across the industry for its high bit length and one way functionality. The bit length of 256 is great as it prevents collisions in the hash functions to keep uniqueness between data and using one way functionality it will be hard for bad actors to reverse engineer the data that was encrypted by the cipher. SHA-256 has undergone tons of testing to ensure that hashed data is well secure and safe from being uncovered by anyone who is trying to get data from the original source which is why I chose to use it in this use case.

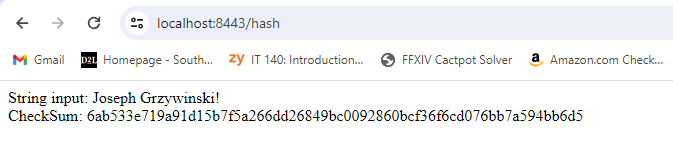
## Certificate Generation

Insert a screenshot below of the CER file.



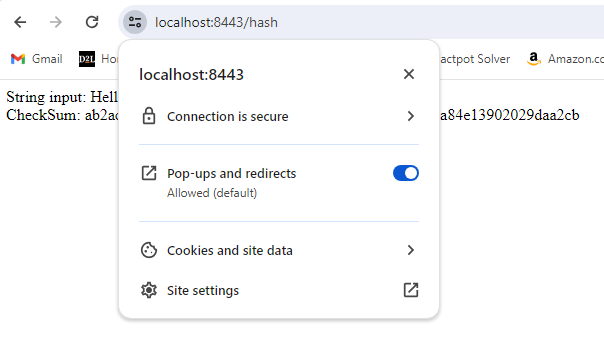


## Deploy Cipher



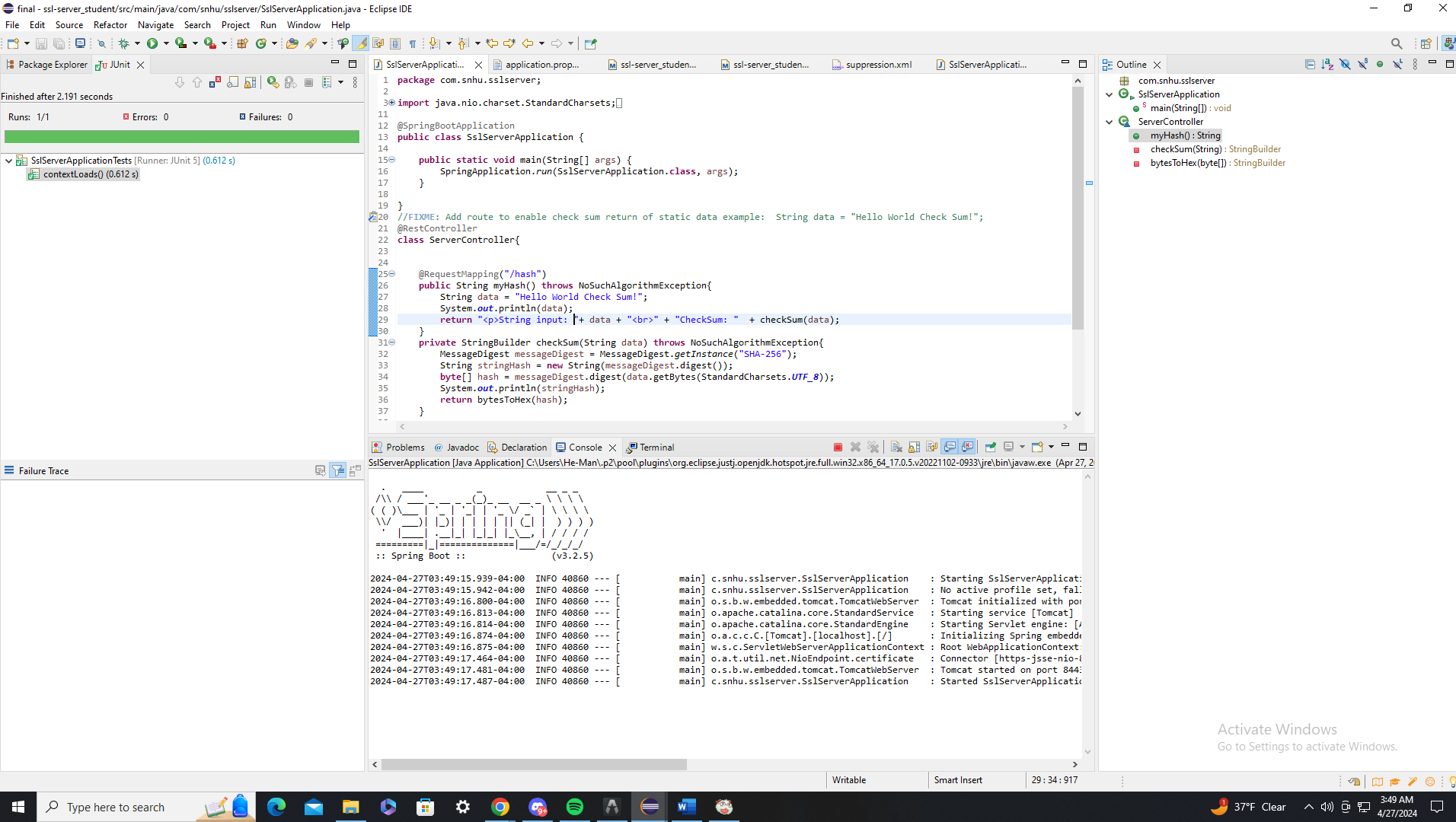
## Secure Communications

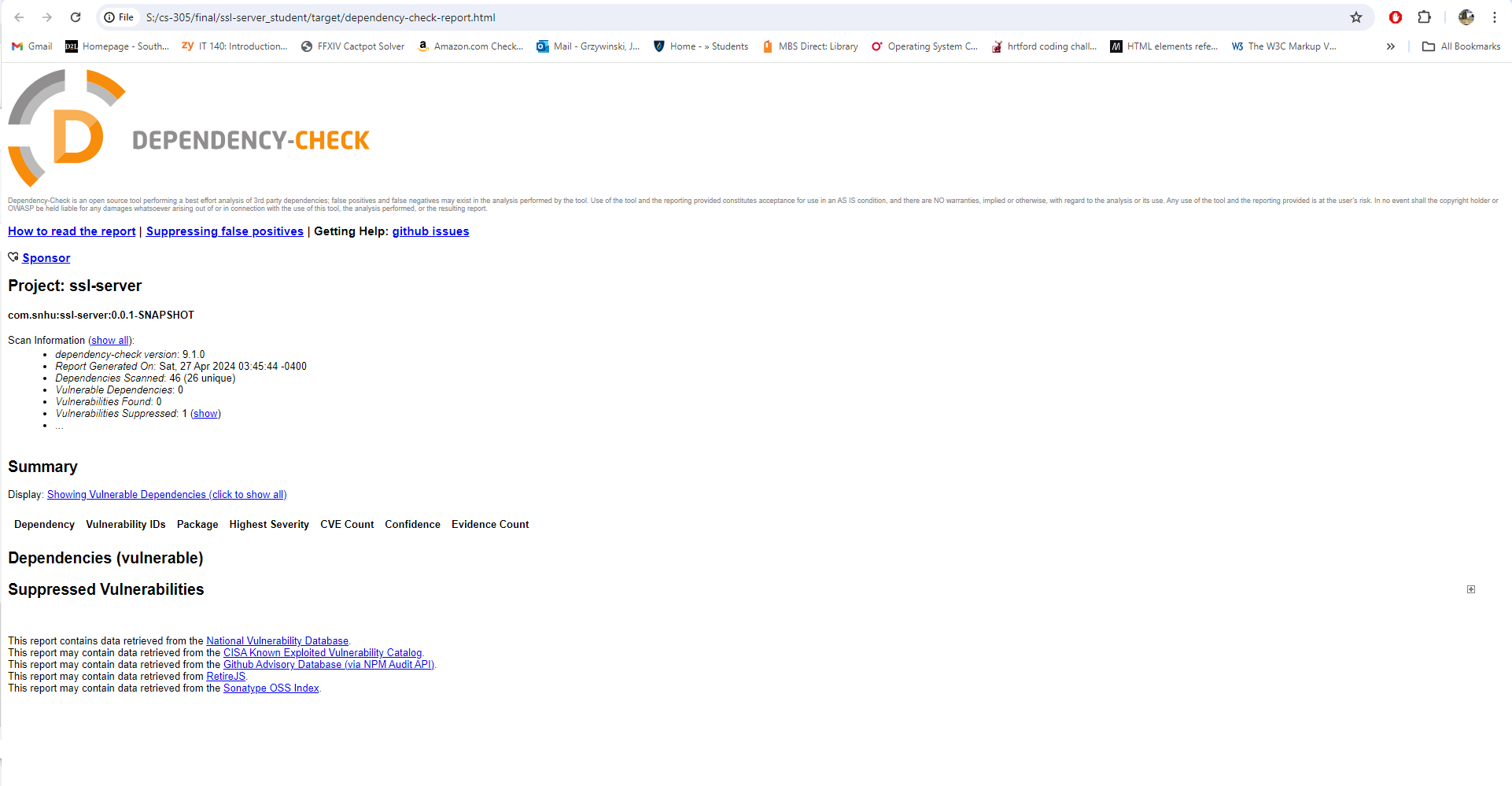
Insert a screenshot below of the web browser that shows a secure webpage.



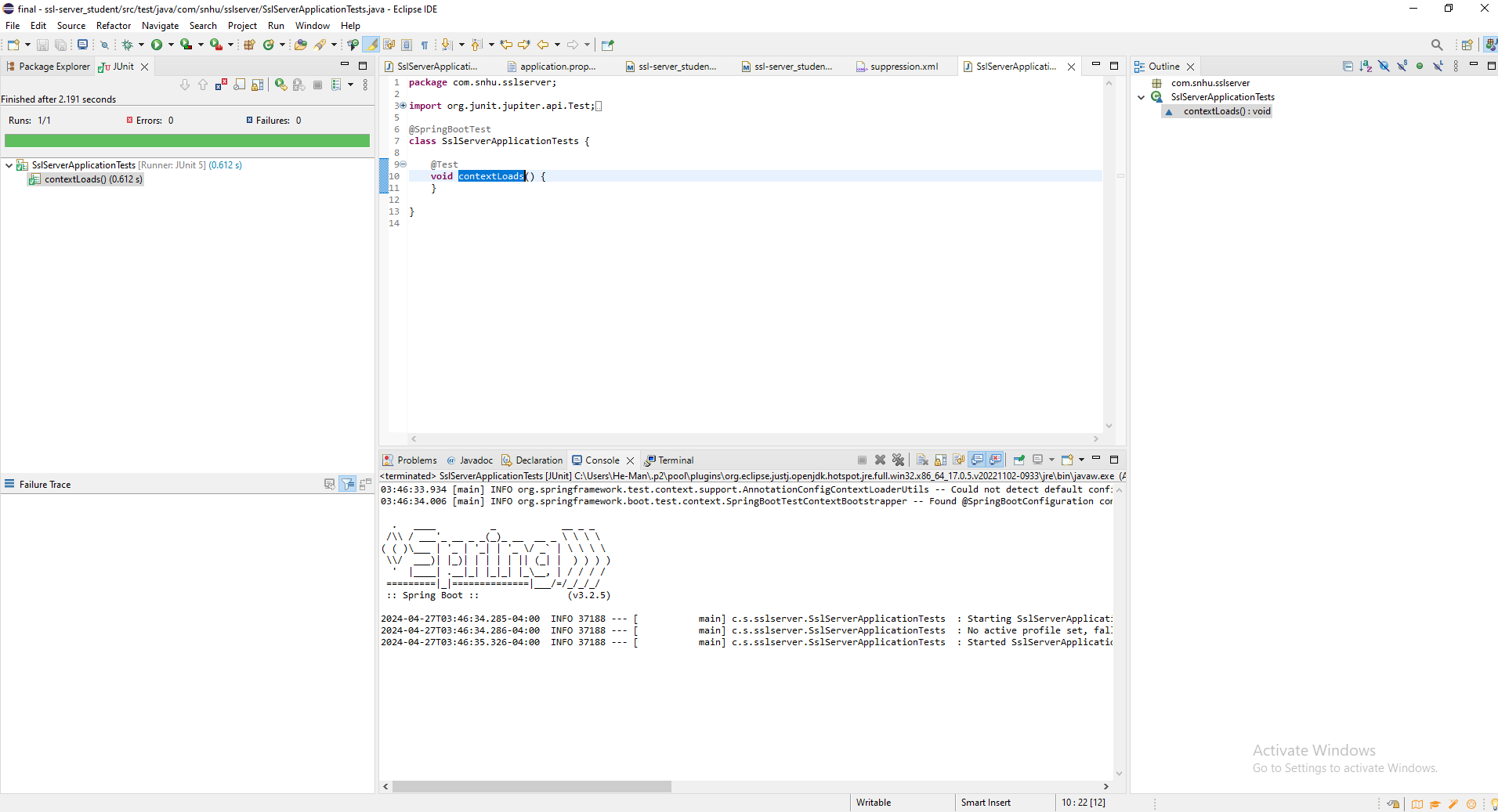
## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

## Summary

When going through my tests for this project a lot of the vulnerabilities that were showing up had to do with spring-boot being out of date. When I initially tested the application and read through the dependency report, I noticed that even though a lot of the dependencies had their own issues at their respective versions, it all seemed to stem back to the spring boot version being completely out of date and having issues. I then went through the process of testing and researching what version of spring I would need to upgrade to avoid the vulnerabilities I was seeing and initially I landed on version 3.2.4. This got rid of a ton of my vulnerabilities, but not all so I tested going to version 3.2.5 and all vulnerabilities that stemmed back to spring-boot were gone. I then noticed that the OWASP version the application was using was out of date since I had upgraded the application to jdk17. Once I upgraded to the OWASP version I was left with one vulnerability that involved Jackson-databind 2.15.2. With some research I found that this is in fact not a legitimate vulnerability and that the dependency is waiting on reanalysis to confirm if there is a problem but at this time it is a false reporting. With this information I decided to suppress the vulnerability for the time being to clear the report since there is no threat to the application.

## Industry Standard Best Practices

When going through my vulnerability report, I started by focusing on the items that I knew were just a simple version upgrade. I knew that a lot of the listed vulnerabilities listed were stemming from the same version being outdated so I was able to minimize changes while also clearing all vulnerabilities. By not changing any of the dependencies or any part of the application I am able to keep the same level of security that the app already had. Following industry best practices can absolutely help the company in the long run by avoiding data breaches and bad actors from infiltrating the systems because of basic system vulnerabilities. Choosing to ignore problems that have already been figured out by the creators of certain features and dependencies can be detrimental to a company’s systems mostly because, if it is a known issue then it can be easily exploited by almost anyone. So, taking measures to clear an application of all these known problems before launching and updating the application during its lifespan is very important to maintain that level of security and prevent information leaks company wide.