

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Francis Cottrell-Eshaghi

## Algorithm Cipher

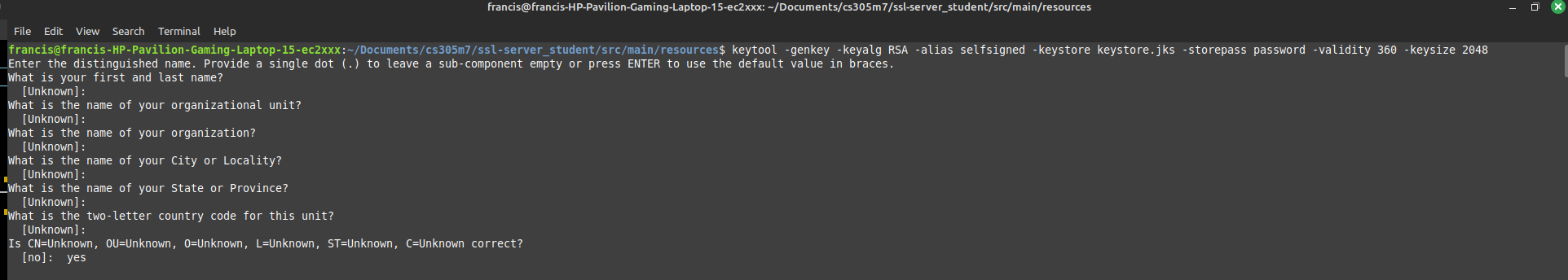
The Advanced Encryption Standard (AES) is a symmetric-key encryption algorithm specifically designed to safeguard sensitive data. This encryption method, introduced as a federal standard in the United States in 2001, has gained global acceptance and is now widely employed for ensuring the security of information across various sectors and applications. AES stands out for its exceptional attributes, including its remarkable efficiency, rapid processing capabilities, and its robust security features. These qualities make AES a trusted and proven choice for safeguarding data against unauthorized access and ensuring the confidentiality and integrity of information in an ever-evolving digital landscape.

Regarding hash functions and bit levels, AES functions by processing data in fixed-size blocks and allows for key lengths of 128, 192, and 256 bits, offering a customizable approach to security. The choice of key length significantly influences the strength of encryption, with longer keys providing higher levels of security. Furthermore, AES supports various block modes, such as Cipher Block Chaining (CBC) and Galois/Counter Mode (GCM), which serve as valuable tools for ensuring secure communication. These block modes not only provide confidentiality but also introduce mechanisms for verifying the integrity and authenticity of data being transferred. In the context of Artemis Financial's web application, this versatility in key lengths and block modes will allow them to tailor their security measures to the specific needs of their clients, ensuring robust protection for financial and client data during data transfers.

In the context of key management, AES relies on symmetric-key encryption, which involves using the same key for both the encryption and decryption processes. This approach simplifies the complexities of key management, particularly in the context of secure communication between parties. However, the crux of secure file transfers lies in the generation and handling of strong and unpredictable keys. To achieve this, it's imperative to employ cryptographically secure random number generators (CSPRNGs) for key generation. These specialized generators produce keys that are highly resistant to predictability, making them a vital component of secure data transmission. In Artemis Financial's case, the use of CSPRNGs in tandem with the AES symmetric-key encryption will ensure that the keys used for their secure file transfers are both robust and resilient against unauthorized access, further fortifying the protection of critical financial and client data.

The historical context of encryption algorithms is essential in understanding why AES is a preferred choice for Artemis Financial's web application. AES emerged as the successor to the Data Encryption Standard (DES) due to vulnerabilities stemming from DES's relatively small key size. The small key size made it susceptible to various attacks, prompting the need for a more robust encryption standard. In contrast, AES has demonstrated exceptional resilience over the years, enduring extensive scrutiny and cryptographic analysis. It remains secure against known attacks, establishing its reputation as a highly dependable encryption standard. Furthermore, AES's widespread adoption and support in various programming languages and security libraries contribute to its prominence, ensuring that it can be readily implemented and integrated into modern software applications with ease. This track record of strength and adaptability makes AES a secure and future-proof choice for Artemis Financial as they embark on modernizing their operations and fortifying their data protection measures.

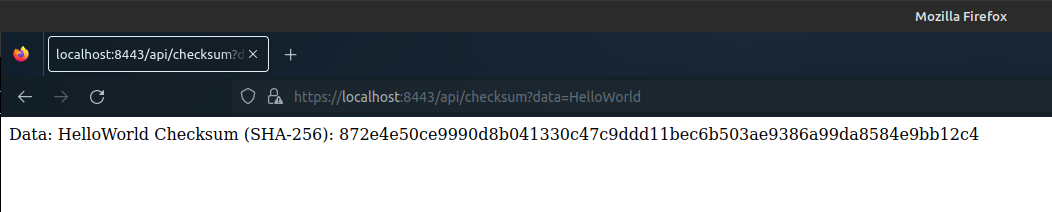
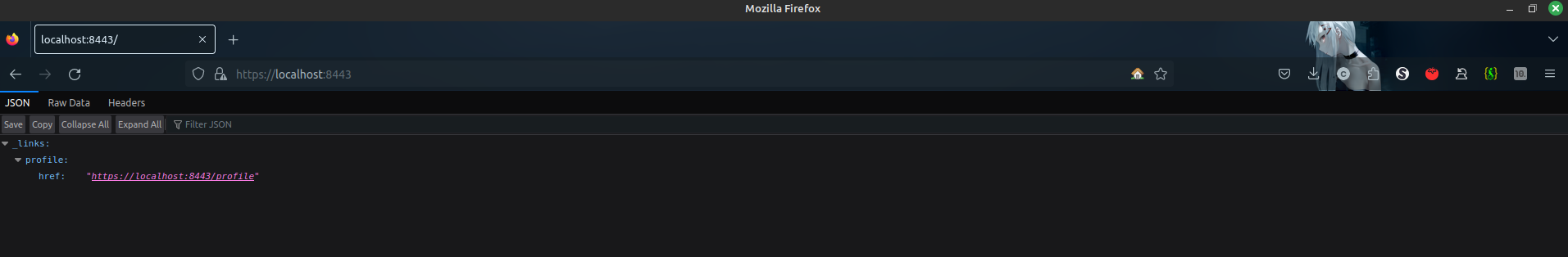
## Certificate Generation



## Deploy Cipher

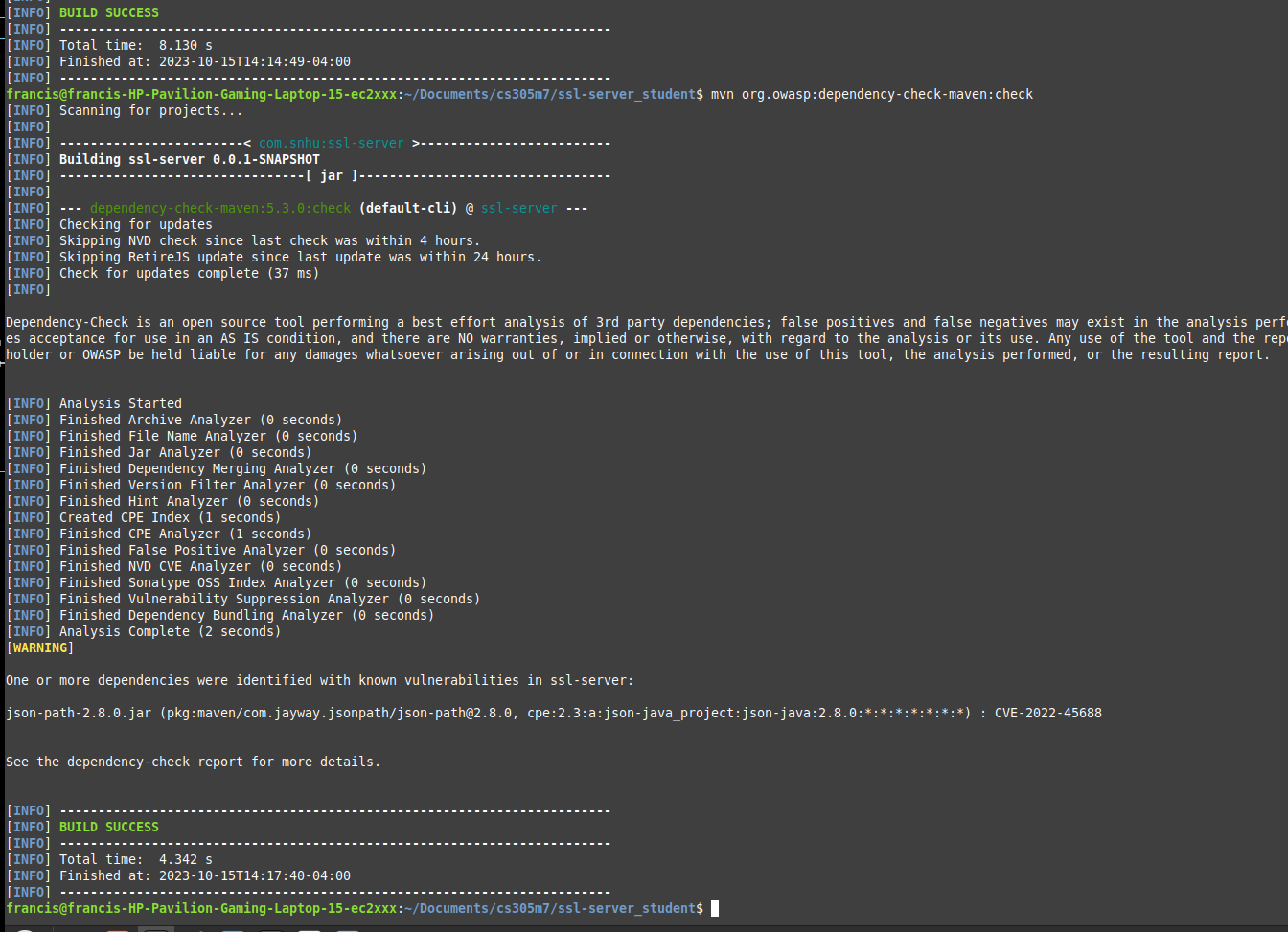
## 

## Secure Communications



## Secondary Testing

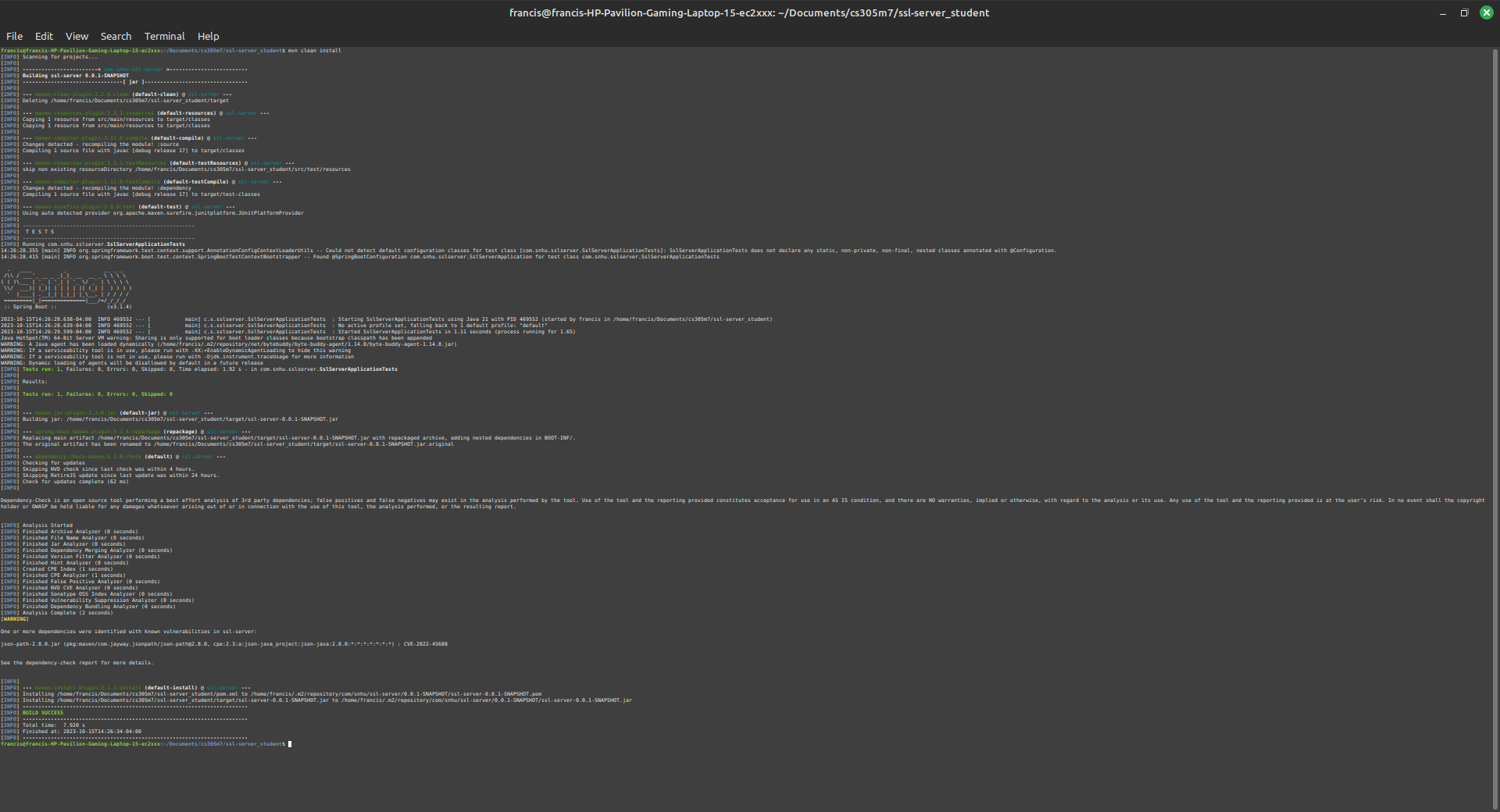
## 



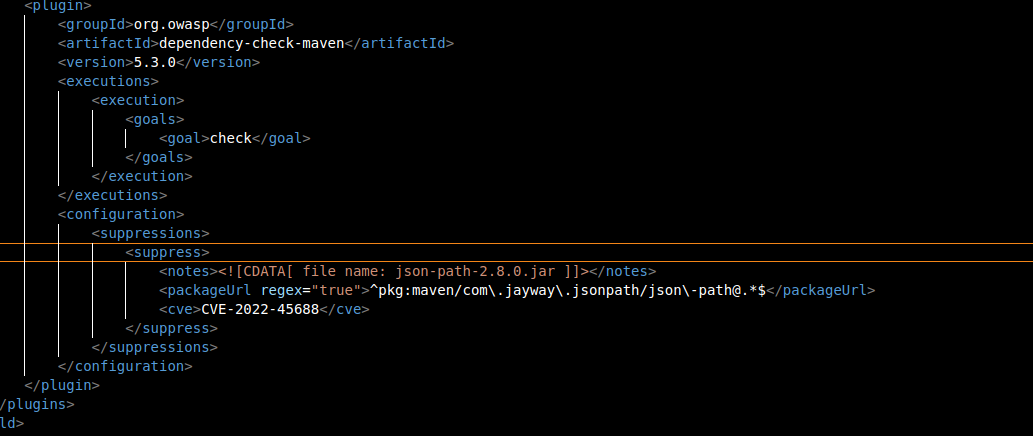


## Functional Testing

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



I understand that I have a false positive warning for the "json-path" dependency with CVE-2022-45688, and I have tried to suppress it using the configuration in the plugin for "org.owasp" without success.



## Summary

Refactoring code to enhance security is an essential practice in addressing vulnerabilities within software applications. To ensure the security of your software, a systematic approach should be followed, which aligns with the Vulnerability Assessment Process Flow Diagram.

The first step is to conduct a thorough code review and analysis. This involves using static analysis tools to automatically scan the codebase for common coding vulnerabilities, followed by manual code reviews to catch more nuanced security issues that automated tools might overlook.

Ensuring compliance with secure software protocols is the next step. This involves staying up to date with the latest vulnerability databases and standards and accurately identifying CVEs for the vulnerabilities you address.

Incorporating security practices into your Continuous Integration and Continuous Deployment (CI/CD) pipeline is another vital step. This ensures that automated security testing and validation are seamlessly integrated into the deployment process, offering immediate feedback to developers.

I adopted best practices. The use of the Spring Boot Starter parent is commendable, as Spring Boot is known for its robust security features. Furthermore, I have made a security-conscious decision by excluding the org.yaml:snakeyaml dependency, which is notorious for its potential to introduce YAML injection vulnerabilities.

I explicitly made definition of dependency versions, such as jackson-databind, spring-boot-starter-data-rest, and spring-boot-starter-web, is a security-aware approach to prevent unintended dependency version upgrades, which can inadvertently introduce vulnerabilities.

I exclused of junit-vintage-engine is essential for avoiding security issues related to older testing frameworks.

I also did the specific suppression configuration. By suppressing CVE-2022-45688 for the "json-path-2.8.0.jar," I have effectively addressed a known vulnerability. This suppression helps prevent false positive warnings during the security scan.

## Industry Standard Best Practices

1. Dependency Management and Version Control: By explicitly defining dependencies and their versions, you've followed an industry best practice to maintain the software application's current security. This practice prevents unintended and potentially insecure dependency version upgrades. The value lies in ensuring that the application always uses the latest secure versions of libraries, reducing the risk of known vulnerabilities being exploited.
2. Exclusion of Vulnerable Dependencies: The exclusion of the org.yaml:snakeyaml dependency is a security-aware decision. It mitigates known vulnerabilities associated with this library, which are prone to YAML injection attacks. The value here is evident in preventing a class of security vulnerabilities that could compromise the application's confidentiality and integrity.
3. Testing and Test Exclusions: The exclusion of the junit-vintage-engine within the test scope demonstrates your commitment to maintaining the software's security. Older testing frameworks can have known vulnerabilities that are potentially exploitable. The industry standard practice of using up-to-date testing libraries and excluding the vintage ones ensures the security of the testing process.
4. Use of Security-Centric Frameworks: Leveraging the Spring Boot Starter parent and associated libraries, which are well-known for their security features, is a recognized best practice. This practice provides a solid foundation for secure coding and ensures the application adheres to security best practices.
5. Static Analysis and Security Scanning: The incorporation of the OWASP Dependency-Check Plugin for security scanning represents a crucial industry-standard practice. It involves conducting static analysis to identify known security vulnerabilities in your application's dependencies. The value here is in the early detection and mitigation of vulnerabilities, reducing the risk of exploitation.
6. Suppression of Known Vulnerabilities: The suppression of CVE-2022-45688 for the "json-path-2.8.0.jar" is an example of proactive vulnerability mitigation. It's a best practice to suppress known issues that are false positives or have been reviewed and deemed non-exploitable. This practice eliminates unnecessary noise from security reports and ensures that genuine vulnerabilities are not overlooked.

Implementing industry-standard best practices for secure coding provides a multitude of advantages for both the current security of the software application and the overall well-being of the company.

Firstly, there's significant risk mitigation. By proactively addressing known vulnerabilities and adhering to secure coding principles, the company effectively minimizes the risk of security breaches. This comprehensive approach safeguards not only sensitive data but also fosters and maintains customer trust while preserving the company's reputation in the industry.

Moreover, the application of these best practices ensures compliance with industry standards and regulations, offering valuable legal protection. This compliance not only safeguards the company against potential legal liabilities but also helps avoid costly financial penalties, which can be detrimental to business operations.

Cost reduction is another notable benefit. Detecting and preventing security vulnerabilities at an early stage is more cost-effective than dealing with the aftermath of security incidents and breaches. The expenses associated with remediation, legal actions, and compensating affected customers are significantly reduced.

Additionally, adhering to these practices provides a competitive advantage. Demonstrating a strong commitment to security through the application of best practices instills trust in customers and partners. This trustworthiness makes the company more attractive in the market, potentially leading to increased business opportunities and revenue.

Lastly, the application of secure coding practices invariably enhances the overall quality of the software. By minimizing security-related issues, these practices result in fewer bugs, thereby contributing to more reliable applications. This higher software quality, in turn, leads to increased customer satisfaction, strengthening the company's relationships with its user base and improving its standing in the market.