GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	12/04/2023	Chris Marrs	Created.

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
Chris Marrs

1. Algorithm Cipher

An effort was taken to perform the research and recommend a solution for the requirement that Artemis Financial has related to its long-term storage of archived files and the need to encrypt them for safe keeping while also allowing access to those that are authorized via file access and verified via a checksum.

The approach for developing an encryption cipher solution for a company requires a broad approach to many elements to provide a strong, dependable, and ongoing protection practice. Many of the elements that were taken into consideration like best practices, risks, regulations usage as well as any associated tradeoffs resulted in the recommendation below.

Recommended Cipher

AES (Advanced Encryption Standard) – this is widely recognized for its security and performance, AES supports multiple key lengths (128, 192 and 256 bits) to meet different security needs. AES is a symmetric encryption algorithm, meaning that the same key is used to encrypt and decrypt data.

AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity, and electronic data protection.

AES was created for the U.S. government with additional voluntary, free use in public or private, commercial, or noncommercial programs that provide encryption services. However, nongovernmental organizations choosing to use AES are subject to limitations created by U.S. export control.

Background

Utilization of Hash Functions, Random Numbers, Symmetric Keys vs Non-Symmetric Keys

Hash Functions - are used to map data of any size into fixed dimensions for encryption, providing data integrity and authentication; they're essential in digital signatures and key derivation as well.

Key Size – The key size or number of bits in the key are important in that the higher number of bits used as the key makes it more difficult to break the key via a brute force attack.

Random Numbers play an integral part in cryptography by providing unique keys. Their use guarantees each encryption process is safe.

Symmetric Keys - with symmetric-key cryptography, one and the same key are used for both encryption and decryption - an approach which makes for faster, more efficient data encryption. It can even encrypt large volumes of data with ease!

Non-Symmetric Keys - asymmetric cryptography uses two keys - a public key for encryption and a private key for decryption - as part of its security solution, making it more suitable for scenarios in which secure key distribution can be an issue, such as digital communications.

History and Current State of Encryption Algorithms

Early Encryption - historically, simple ciphers like Caesar were employed. They relied upon substitution or transposition techniques.

Modern Era - with the rise of electronic computers came more complex algorithms; Data Encryption Standard (DES) became a pioneering effort during this era; however, its short key length eventually made it vulnerable.

Advanced Encryption Standard (AES) - since its introduction in the early 2000s, AES has become the go-to encryption standard. Replacing DES with longer key lengths that resist all known attacks on security, it provides greater protection.

Current State - today, AES remains secure against cryptanalysis. Its worldwide adoption by governments and industries is a testament to its security and reliability.

Summary

The recommendation is to use AES-256 for archival and electronic messaging. AES is usually used in these scenarios due to its balance between security and performance. Its specific selection in this case is because of its characteristics such as data sensitivity, system capabilities, regulatory compliance, and data availability. With proper implementation, key management, and regular reviews we think that this is a recommendation that will allow Artemis Financial to remain compliant despite the changing security we find ourselves in today.

Justification for Recommendation

AES offers an ideal balance between security and operational efficiency that's crucial for financial institutions like Artemis Financial. AES has gained worldwide acceptance and conforms to numerous regulatory requirements. As seen, cyber threats are continuously evolving, and AES stands out as a strong solution that offers effective protection, now and into the future.

References

Dworkin, M. (2007). *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*
https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

arcserve.com. (2023). *5 Common Encryption Algorithms and the Unbreakables of the Future*
https://www.arcserve.com/blog/5-common-encryption-algorithms-and-unbreakables-future

docs.oracle.com (as of 2023). *Java Security Standard Algorithm Names*
https://docs.oracle.

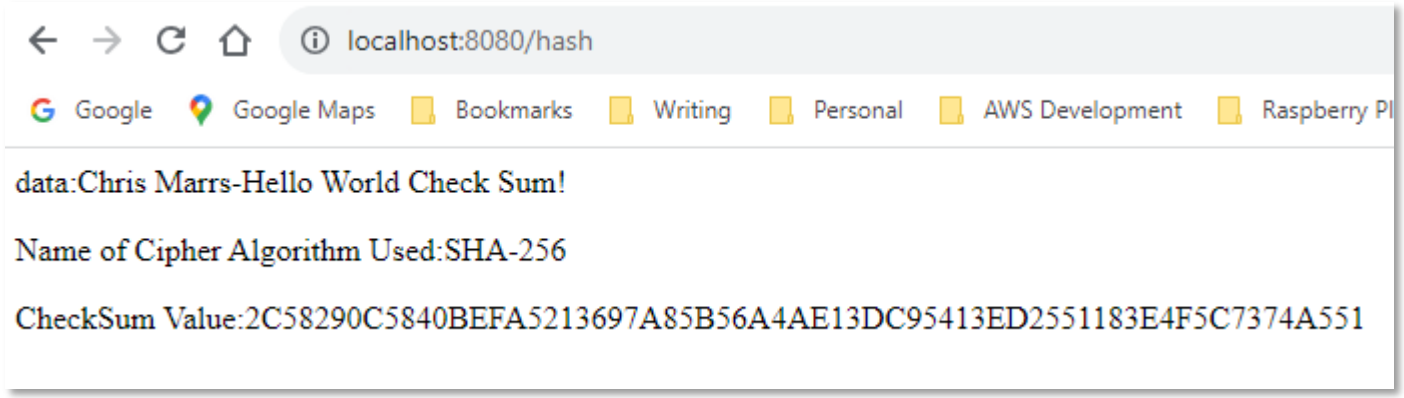
2. Certificate Generation

Insert a screenshot below of the CER file.

```
Issuer: CN=Chris Marrs, OU=CM, O=CM, L=Minneapolis, ST=MN, C=US
Serial number: eab4f8078c0eac1e
Valid from: Wed Dec 13 09:31:29 CST 2023 until: Sat Dec 07 09:31:29 CST 2024
Certificate fingerprints:
         SHA1: CC:68:98:CE:9E:61:5E:D0:21:9D:51:37:0F:78:2F:71:AE:3C:65:D8
         SHA256: E6:2E:DD:03:7E:3E:A0:90:F4:7E:69:39:F3:F8:AB:51:A1:83:F3:AF:0B:0E:C3:09:18:8D:1F:A5:ED:E2:82:93
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: localhost
]

#2: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B1 5C 4A 50 C0 96 E7 46   C2 9B C2 23 40 D8 A5 FA  .\JP...F...#@...
0010: B1 B5 0F B4                                        ....
]
]
```

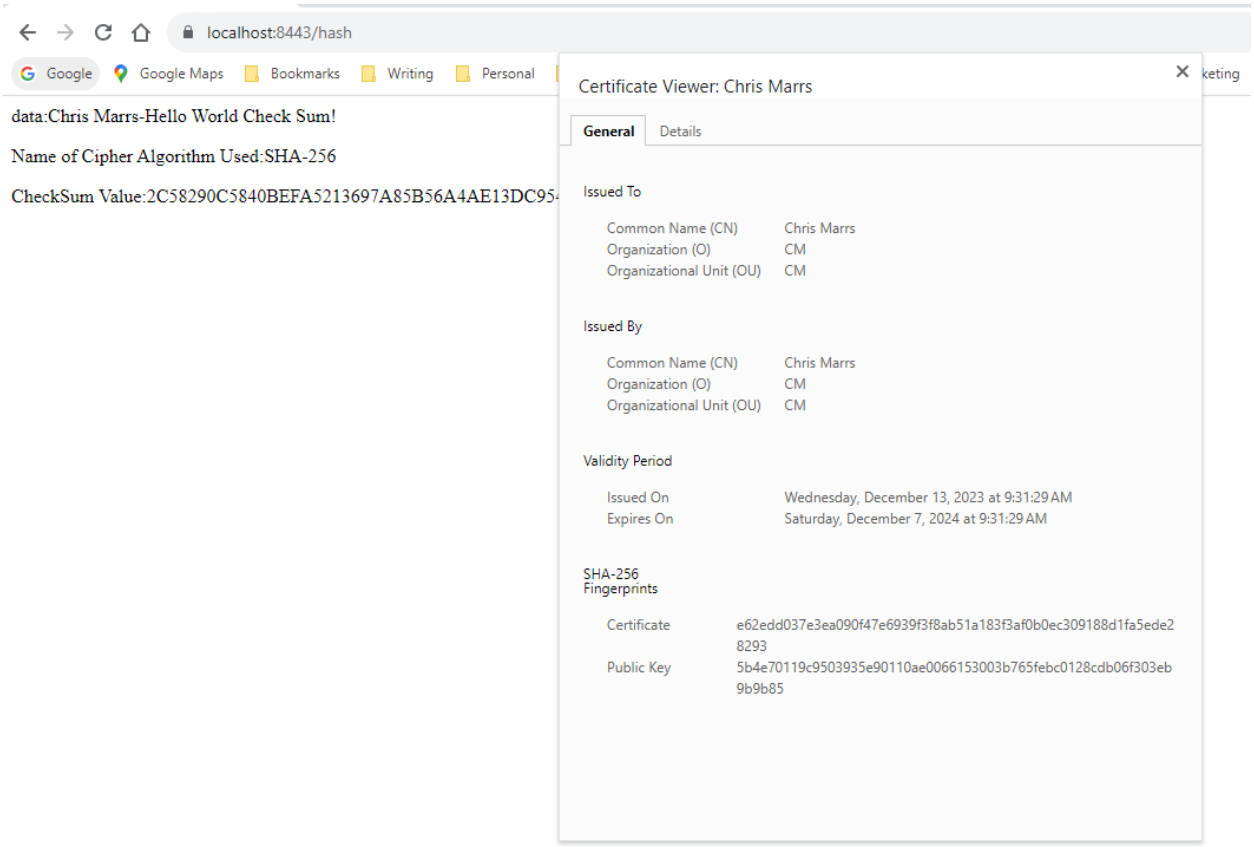
3. Deploy Cipher

Insert a screenshot below of the checksum verification.



4. Secure Communications

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

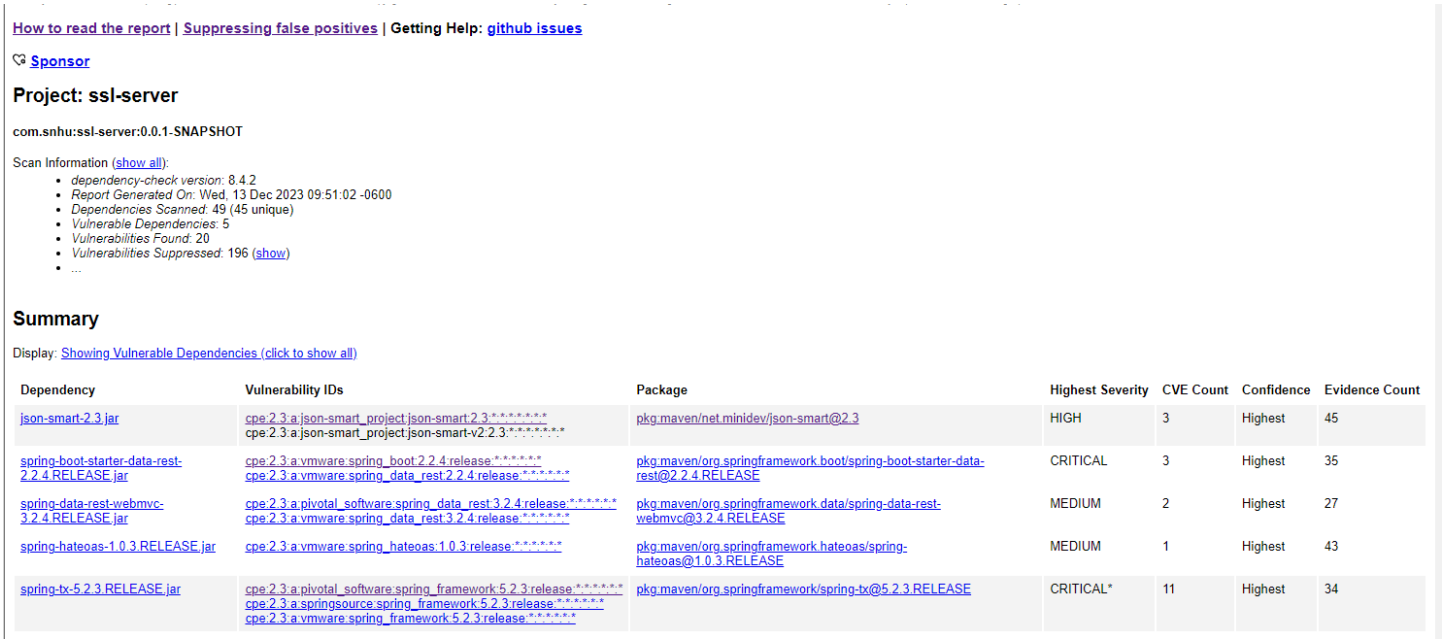


5. Secondary Testing

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

```java
@SpringBootApplication
public class SslServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SslServerApplication.class, args);
    }

    @Bean
    public ServletWebServerFactory servletContainer() {
        TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory() {
            @Override
            protected void postProcessContext(Context context) {
                SecurityConstraint securityConstraint = new SecurityConstraint();
                securityConstraint.setUserConstraint("CONFIDENTIAL");
                SecurityCollection collection = new SecurityCollection();
                collection.addPattern("/*");
                securityConstraint.addCollection(collection);
                context.addConstraint(securityConstraint);
            }
        };
        tomcat.addAdditionalTomcatConnectors(redirectConnector());
        return tomcat;
    }

    @Value("${server.port.http}") //Defined in application.properties file
    int httpPort;

    @Value("${server.port}") //Defined in application.properties file
    int httpsPort;

    private Connector redirectConnector() {
        Connector connector = new Connector(TomcatServletWebServerFactory.DEFAULT_PROTOCOL);
        connector.setScheme("http");
        connector.setPort(httpPort);
        connector.setSecure(false);
        connector.setRedirectPort(httpsPort);
        return connector;
    }
}
//FIXME: Add route to enable check sum return of static data example:  String data = "Hello World Check Sum!";
@RestController
class ServerController{
//FIXME:  Add hash function to return the checksum value for the data string that should contain your name.
    @RequestMapping("/hash")
    public String myHash(){
        String strName = "Chris Marrs-Hello World Check Sum!";
        String strCipher = "SHA-256";
        String strHash = Hash.hash(strName);
        System.out.println("Unique Data String: " + strName);
        System.out.println("Algorithm Cipher: " + strCipher);
        System.out.println("Checksum Hash Value: " + strHash);

        String strResult = "<p>data:"+strName
                +"<p>Name of Cipher Algorithm Used:"+strCipher
                +"<p>CheckSum Value:"+strHash;
        return strResult;
    }
}
```

Below is the dependency report created after refactoring the code, it is identical to the original report for the original code, therefore no new dependencies were added.



How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 8.4.2
- *Report Generated On*: Wed, 13 Dec 2023 09:51:02 -0600
- *Dependencies Scanned*: 49 (45 unique)
- *Vulnerable Dependencies*: 5
- *Vulnerabilities Found*: 20
- *Vulnerabilities Suppressed*: 196 (show)
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
json-smart-2.3.jar	cpe:2.3:a:json-smart_project:json-smart:2.3:*:*:*:*:*:*:* cpe:2.3:a:json-smart_project:json-smart-v2:2.3:*:*:*:*:*:*:*	pkg:maven/net.minidev/json-smart@2.3	HIGH	3	Highest	45
spring-boot-starter-data-rest-2.2.4.RELEASE.jar	cpe:2.3:a:vmware:spring_boot:2.2.4:release:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_data_rest:2.2.4:release:*:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot-starter-data-rest@2.2.4.RELEASE	CRITICAL	3	Highest	35
spring-data-rest-webmvc-3.2.4.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_data_rest:3.2.4:release:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_data_rest:3.2.4:release:*:*:*:*:*:*:*	pkg:maven/org.springframework.data/spring-data-rest-webmvc@3.2.4.RELEASE	MEDIUM	2	Highest	27
spring-hateoas-1.0.3.RELEASE.jar	cpe:2.3:a:vmware:spring_hateoas:1.0.3:release:*:*:*:*:*:*:*	pkg:maven/org.springframework.hateoas/spring-hateoas@1.0.3.RELEASE	MEDIUM	1	Highest	43
spring-tx-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.3:release:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-tx@5.2.3.RELEASE	CRITICAL*	11	Highest	34

6. Functional Testing

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

```java
@SpringBootApplication
public class SslServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SslServerApplication.class, args);
    }

    @Bean
    public ServletWebServerFactory servletContainer() {
        TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory() {
            @Override
            protected void postProcessContext(Context context) {
                SecurityConstraint securityConstraint = new SecurityConstraint();
                securityConstraint.setUserConstraint("CONFIDENTIAL");
                SecurityCollection collection = new SecurityCollection();
                collection.addPattern("/*");
                securityConstraint.addCollection(collection);
                context.addConstraint(securityConstraint);
            }
        };
        tomcat.addAdditionalTomcatConnectors(redirectConnector());
        return tomcat;
    }

    @Value("${server.port.http}") //Defined in application.properties file
    int httpPort;

    @Value("${server.port}") //Defined in application.properties file
    int httpsPort;

    private Connector redirectConnector() {
        Connector connector = new Connector(TomcatServletWebServerFactory.DEFAULT_PROTOCOL);
        connector.setScheme("http");
        connector.setPort(httpPort);
        connector.setSecure(false);
        connector.setRedirectPort(httpsPort);
        return connector;
    }
}
//FIXME: Add route to enable check sum return of static data example:  String data = "Hello World Check Sum!";
@RestController
class ServerController{
//FIXME:  Add hash function to return the checksum value for the data string that should contain your name.
    @RequestMapping("/hash")
    public String myHash(){
        String strName = "Chris Marrs-Hello World Check Sum!";
        String strCipher = "SHA-256";
        String strHash = Hash.hash(strName);
        System.out.println("Unique Data String: " + strName);
        System.out.println("Algorithm Cipher: " + strCipher);
        System.out.println("Checksum Hash Value: " + strHash);

        String strResult = "<p>data:"+strName
                +"<p>Name of Cipher Algorithm Used:"+strCipher
                +"<p>CheckSum Value:"+strHash;
        return strResult;
    }
}
```

After reviewing the code, some areas of interest were noted relative to the Vulnerability Assessment Process Flow Diagram and security best practices:

Input Validation

The API Route `/hash` endpoint could potentially process user-provided data. Input validation would usually be implemented at this point.

APIs
The general structure of the code primarily concerns server setup and an API endpoint, fairly straightforward.

Cryptography
The hashing in '/hash' endpoint is (`SHA-256`), which is generally secure. However, the security also depends on the implementation of the hash() method, which is not reviewed for this exercise.

Client/Server Architecture
The HTTPS Configuration code correctly redirects HTTP traffic to HTTPS, which is a good practice for secure communication. Hardcoded ports and the use of '@Value' annotations for ports are potential risks if not properly secured in the properties file.

Code Error Handling
Usage of 'System.out.println' for logging is not recommended in production as it does not provide any context or level of severity. The system should implement a proper logging framework. There is exception handling, especially in the '/hash' endpoint. Also, if hash() throws an exception, it's not caught.

Code Quality
Overall, the code is relatively clear, but detailed documentation would enhance understanding and maintenance.

Encapsulation
The code does not reveal much about data structures, but ensuring sensitive data is not exposed and is encapsulated within classes would be advisable.

Additional Observations
- Application property files must not expose themselves or hardcode in ways which compromise security.
- The 'Hash' class and its methods must follow secure coding practices as they involve cryptographic operations.

Overall, the code exhibits some solid security practices - particularly its HTTPS setup - but additional error handling and logging would further enhance its overall protection.

7. Summary

For this exercise the Vulnerability Assessment Process Flow Diagram areas of focus were APIs and Cryptography. The general approach was to get the requested functionality working without security in-force to allow the development of the solution. Once the solution was completed the necessary elements of HTTPS implementation, certificate creation, and testing were layered in to achieve a secure solution. With a number of dependency checks run during the solutioning to stay ahead of any potential introduced vulnerabilities.

The summarized steps:

1) The unsecured code was scanned and executed to create a baseline.
2) A cryptographic hash algorithm was created and added to the code to generate a checksum of the data being transmitted.
3) The protocol was changed from HTTP to HTTPS in the connection bean and the properties resource.
4) A self-signed certificate was created and imported into the keystore file using the JAVA 'keytool'. An additional parameter was needed on my Windows system in order for Tomcat to server SSL. The '-ext san=dns:localhost' parameter was added to the key generation work in Module 5.
5) The cert was imported to the Windows Certificate Trust List for both Trusted Root Certification Authorities and Enterprise Trust in order for a browser to not display the untrusted host error message and to enable ssl encryption. This was the biggest challenge technically for this project as the browsers have now highly restricted the SSL functions for localhost.
6) The web application was modified to use the keystore file.
7) A secondary static testing of the refactored code using the OWASP Dependency-Check Maven to ensure that no new security vulnerabilities were introduced.

8. Industry Standard Best Practices

There were a number of elements of **industry best practices** that were specifically used in this solution.

Initially the baseline of dependency vulnerabilities was recorded for use for comparison at the final stage to ensure that no new vulnerabilities were introduced. As changes were made, testing of dependencies, such as Spring Boot and Apache Tomcat, was done and compared to the prior steps to ensure that they were up to date to avoid potential vulnerabilities. The dependencies that were introduced by my solution were checked by using the OWASP Dependency-Check Maven against the refactored code.

Throughout the solution process, there was thought around secure coding standards for developing software for sensitive industries or environments, with focus on secure coding standards proposed by OWASP Secure Coding Practices. While not all elements of best practices were in scope for this solution, notes were made for items like input validation and proper error handling practices.

Strong encryption was implemented using encryption algorithm (AES) with a key length of 256 bits. Authentication and authorization were not in scope for this solution, however using multi-factor authentication wherever possible to comply with the principle of least privilege in access control should be considered. However, encryption and secure data handling were employed in this solution. While

there was a basic secure configuration instituted for this solution, a broader solution scope would require additional work in this area.

Applying industry standard best practices for secure coding can not only protect software applications but also offer strategic **benefits to organizations** by helping protect their reputation, ensure compliance, and avoid financial losses.

Security breaches can severely compromise a company's reputation and often result in significant financial losses due to fines, compensation payments and remediation expenses; but secure practices reduce this risk. Many industries have stringent regulatory requirements regarding data security and utilizing secure coding standards ensure compliance and help avoid penalties for breaches.

Secure products can maintain, and possibly, increase customer satisfaction and loyalty. Implementation of best practices fosters an organization-wide security-conscious culture; every employee becomes an active stakeholder in its security process.

Being proactive when it comes to security can often prove more cost-effective than dealing with incidents postmortem. Adherence to industry standards helps companies stay abreast of new security threats that arise over time.