Secure Coding Practices in Java

Ahmad Malik, Omar Alshikh, Joseph Dyer, Uditi Goyal

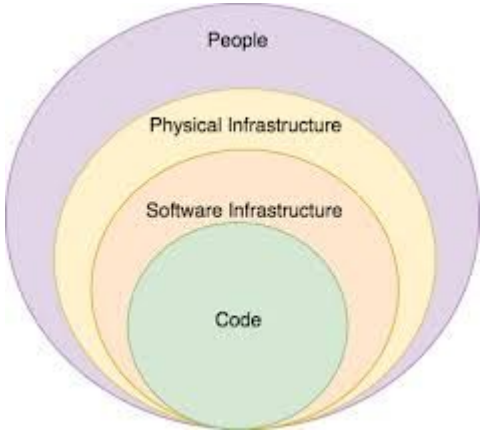
Introduction

What is it?

- Secure coding practices is the idea of developing software applications in a way that protects software vulnerabilities.
  - Logic Flaws
  - Defects
  - Bugs

Examples

- Password Management
- Database Security
- File management
- General coding practices
- Error handling and login
- Validate input



Common Security Vulnerabilities

Cross-site request forgery (CSRF):

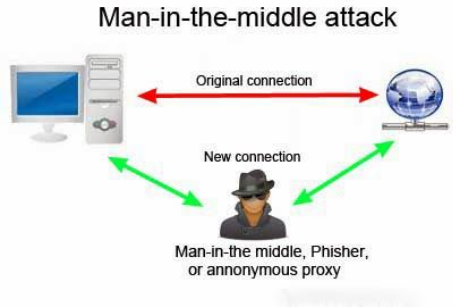
- Forces an end user to execute unwanted actions on a web application
- Occurs when a browser does not distinguish between an attackers and a legitimate request
- Can be used to transfer bank funds, change user info etc.

Man-in-the-middle (MITM) attack:

- Intercepts communications between two parties
- Occurs when SSL certification is disabled
- Can be used to record data being communicated between client and host

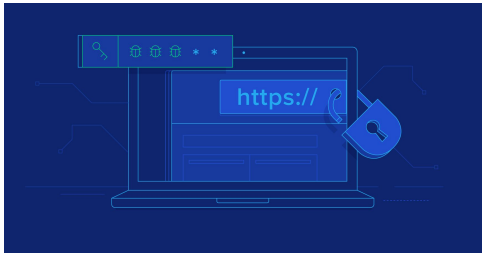
Insecure password hashing:

- Takes advantage of weak hashing functions that can be reverse engineered
- Occurs when improper hashing functions are used
- Can cause password leaks



Detecting Security Vulnerabilities

Some ways security vulnerabilities have been checked:



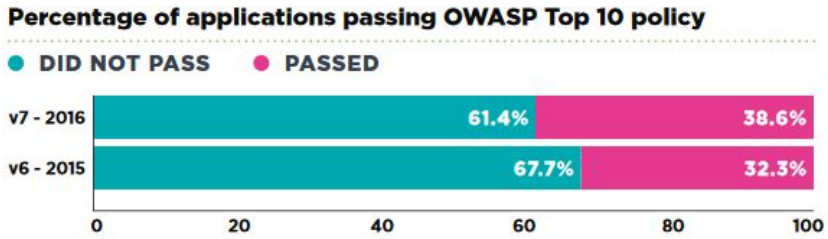
- Static checkers for well-defined cryptographic API usage rules can be used.

- Code snippets of insecure code can be compared with code in applications.

- By using SSLINT, an automatic static analysis tool, to identify the misuse of SSL/TLS APIs in client-side applications.

Analysis of Security Vulnerabilities

- Misuse of Java features can cause security issues
  - Java Reflection API (encapsulation)
- In a study of 263 vulnerabilities 83% of them found to be caused by misuse of APIs
- Approximately 97% of all Java applications use a component with a known vulnerability
- Nearly 39% of all applications use risky or broken cryptographic algorithms for security



Preventing Security Vulnerabilities

- Security-oriented subset of Java
- Easy to use strong Cryptography APIs
- Formal Verification of cryptographic API/protocol implementations

```
Crypter crypter=new Crypter("/rsakeys");
String plaintext=crypter.decrypt(ciphertext);
```

Decrypting with Keyczar API

Secure Coding

- Workforce retraining: there is lack of cybersecurity training apparent in the findings which leads to insecure coding practices. Therefore, a retraining is important to improve security
- Conduct security checks.
- Design clear and helpful error reporting interfaces
- API with with strong security measure by default
- Tools to diagnose security threats
- Use known and tested libraries

CONCLUSION

- Lack of cyber security training by developers which creates frustration in developers and sometimes leads to insecure but easy fixes.
- Examples of easy fixes: Disabling CSRF protection, trusting all certificates to enable SSL/TLS, using obsolete cryptographic hash functions, using obsolete communication protocols.
- Spring security is overly complicated and poorly documented
- Error reporting systems of Java platform security API causes confusion
- Highly viewed posts may promote vulnerable code.

Discussion Questions

1. What are some easy fixes when it comes to secure coding practices?
2. How should we hold companies responsible for security vulnerabilities in their applications?
3. What other types of attacks are applications vulnerable to?
4. How secure is the default Java API?
5. How can we vet advice on Stack Overflow that may cause vulnerable implementations?