Automated Tool Support for Security Bug Repair in Mobile Applications

Larry Singleton

University of Nebraska, Omaha, NE, USA
Faculty Mentors: Myoungkyu Song, Harvey Siy, Rui Zhao

April, 2019
Secure Mobile Applications

• Development and maintenance of secure mobile application software is a difficult undertaking

• Mobile applications have increased in popularity

• Little consideration for software security

April, 2019
Consequences of Failure

- Consequences of exploits increase in severity
- Security failure exposes vulnerabilities
- Results in financial loss or degraded credibility

- Starbucks
  - did not enforce strong passwords
  - did not secure the process of changing an email account

- Citi-Bank
  - Stored unencrypted personal information
Cryptographic Misuse

• Cryptographic misuse is quickly becoming one of the most common issues in software development

• Attackers make use of these vulnerabilities

• Seek zero day exploits
# Encryption Misuse Patterns

<table>
<thead>
<tr>
<th>Misuse Pattern</th>
<th>[10]</th>
<th>[11]</th>
<th>[12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Risky/broken Symmetric Encryption Algorithm</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>RSA algorithm without OAEP</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Reversible One-way Hash</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Non-random IV for CBC Encryption</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Constant Encryption Keys</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Static Seeds for SecureRandom</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Insufficient Key Length</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Reusing Same Cryptographic Key</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Constant Salts for PBE</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Fewer than 1000 Iterations for PBE</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

**TABLE I:** Encryption misuse patterns detected by existing tools.
FIREBUGS
Finding & Repairing Bugs

• An automated approach is needed to:
  • Address security issues
  • Detect security vulnerabilities
  • Dynamically repair problems

• I propose an automated approach towards:
  • Finding and Repairing Bugs based on security patterns
FIREBUGS

• This project aims to:
  • Detect security vulnerabilities
  • Automate generation of repair patches
  • Dynamically apply bug fixes at runtime

• Develop proof-of-concept prototype:
  • Open source Eclipse IDE plugin
  • Security pattern representation in a repair template
  • Vulnerable program localization
  • Repair patch generation
  • Dynamic patch adaptation at runtime
Contributions

The main contributions of this research are:

1. Security patterns representing required behaviors
2. Encoding common repair patterns to fix vulnerable anomalies
3. An automated approach that leverages security patterns to detect security vulnerabilities
4. Generates repair patches
5. Patches dynamically weaved into program at runtime
Research Questions

1. How well does FIREBUGS represent security patterns?

2. How accurately does FIREBUGS generate repair patches?
### Cryptographic Algorithms

<table>
<thead>
<tr>
<th>Encryption Algorithm and its Mode</th>
<th>Weak</th>
<th>Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-ECB</td>
<td>DES</td>
<td>3-DES</td>
</tr>
<tr>
<td>AES</td>
<td>AES-CBC</td>
<td>AES-CFB</td>
</tr>
<tr>
<td>AES-CTR</td>
<td>AES-OFB</td>
<td>AES-CCM</td>
</tr>
<tr>
<td>Blowfish</td>
<td>AES-GCM</td>
<td>RSA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elliptic Curve</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RC4, RC5</td>
</tr>
<tr>
<td>Hash Function</td>
<td>MD2, MD5</td>
<td>HMAC</td>
</tr>
<tr>
<td></td>
<td>SHA-1</td>
<td>SHA-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-3</td>
</tr>
<tr>
<td>Random Number Generator</td>
<td>TRNG</td>
<td>PBKDF1</td>
</tr>
<tr>
<td></td>
<td>PRNG</td>
<td>PBKDF2</td>
</tr>
</tbody>
</table>

**TABLE II: Selection of Cryptographic Algorithms**
Motivating Example

```java
String secret = "Secret";
String plaintext = "PlainMessage";
// Generate a key from the secret
SecretKeySpec key = new SecretKeySpec(secret.getBytes(), "AES");

// Select a cipher algorithm
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
// Encrypt
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

// Hash
MessageDigest digest = MessageDigest.getInstance("SHA-1");
digest.update(plaintext.getBytes());
byte[] hashValue = digest.digest();
```

Fig. 2: A cryptographic misuse example
Static Analysis and Code Transformation

• **FIREBUGS** utilizes program slicing and data flow analysis techniques to identify semantically dependent statements for finding vulnerabilities

• Eclipse Java Development Tools & Java Model
  • parse code into Abstract Syntax Trees (AST)

• Analyze vulnerable & adjacent locations
Static Analysis and Code Transformation

- Generate repair patch based upon edit operations in templates
- Perform change impact analysis
  - To detect behavior modifications caused by repair changes
- Using Aspect-Oriented Programming
  - Encapsulate adaptation concerns for repairs
  - Aspects to be woven into join points of match locations through dynamic runtime adaptation
2,800 repositories from GitHub selected
200 found to use `javax.crypto` API

<table>
<thead>
<tr>
<th>Group</th>
<th>Prj</th>
<th>LOC</th>
<th>Class</th>
<th>Method</th>
<th>Field</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser</td>
<td>2</td>
<td>229,389</td>
<td>2,944</td>
<td>22,917</td>
<td>13,731</td>
<td>61</td>
</tr>
<tr>
<td>Client</td>
<td>8</td>
<td>48,233</td>
<td>386</td>
<td>3,165</td>
<td>2,250</td>
<td>4,150</td>
</tr>
<tr>
<td>Demo</td>
<td>4</td>
<td>10,435</td>
<td>267</td>
<td>1,020</td>
<td>429</td>
<td>276</td>
</tr>
<tr>
<td>Devtool</td>
<td>66</td>
<td>31,340</td>
<td>393</td>
<td>2,991</td>
<td>1,296</td>
<td>960</td>
</tr>
<tr>
<td>Entmnt</td>
<td>22</td>
<td>33,626</td>
<td>414</td>
<td>2,907</td>
<td>1,805</td>
<td>1,988</td>
</tr>
<tr>
<td>Finance</td>
<td>11</td>
<td>41,541</td>
<td>441</td>
<td>3,866</td>
<td>1,890</td>
<td>2,139</td>
</tr>
<tr>
<td>Health</td>
<td>3</td>
<td>50,550</td>
<td>504</td>
<td>4,217</td>
<td>3,029</td>
<td>1,917</td>
</tr>
<tr>
<td>Library</td>
<td>11</td>
<td>28,831</td>
<td>570</td>
<td>2,727</td>
<td>2,988</td>
<td>217</td>
</tr>
<tr>
<td>Security</td>
<td>39</td>
<td>22,426</td>
<td>253</td>
<td>1,480</td>
<td>794</td>
<td>460</td>
</tr>
<tr>
<td>Social</td>
<td>23</td>
<td>62,658</td>
<td>594</td>
<td>4,317</td>
<td>3,386</td>
<td>1,861</td>
</tr>
<tr>
<td>Utility</td>
<td>11</td>
<td>29,673</td>
<td>303</td>
<td>2,346</td>
<td>1,424</td>
<td>2,379</td>
</tr>
</tbody>
</table>

TABLE II: 200 Android projects partitioned into 11 groups with the average number of Lines of Source Code (LOC), classes, methods, fields and commits
Eclipse Plug-in Screenshot
```java
void objMethodInvocation() {
    try {
        PwDatabaseV4 newManager;
        newManager = createDBv4();

        SecretKeySpec keySpec = new SecretKeySpec(newManager.getFinalKey(), "AES");
        byte[] iv = new byte[128 / 8];
        SecureRandom prng = new SecureRandom();
        prng.nextBytes(iv);
        Cipher cipher = Cipher.getInstance(algorithm); // Select a cipher
        // algorithm
        cipher.init(Cipher.ENCRYPT_MODE, keySpec, new IvParameterSpec(iv)); // Encrypt
        byte[] ciphertext = cipher.doFinal(plaintext.getBytes());
        System.out.println(new String(ciphertext));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```
public byte[] getFinalKey() {
    return finalKey;
}

public void makeFinalKey(byte[] masterSeed, byte[] masterSeed2, long numRounds) throws IOException {
    // Write checksum Checksum
    MessageDigest md;
    try {
        md = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new IOException("SHA-256 not implemented here.");
    }
    NullOutputStream nos = new NullOutputStream();
    DigestOutputStream dos = new DigestOutputStream(nos, md);
    byte[] transformedMasterKey = transformMasterKey(masterSeed2, masterKey, numRounds);
    dos.write(masterSeed);
    dos.write(transformedMasterKey);
    finalKey = md.digest();
}
Eclipse Plug-In Output

[DBG] Found at line: 46, AndroidFinalKey.transformMasterKey USES crypto.spec.SecretKeySpec(ARG1, ARG2)
[DBG] Found at line: 24, KeyUseWrong.objMethodInvocation USES crypto.spec.SecretKeySpec(ARG1, ARG2)
  getFinalKey of PwDatabaseV4 finalKey at 10
  finalKey=md.digest() in makeFinalKey in PwDatabaseV4 at 30
FIREBUGS - Quick Demo
Conclusion - FIREBugs

The ultimate goal is to provide tool-assisted capabilities to automatically reduce software maintenance burdens of the number of outstanding software defects that exceed available resources.
Thank You!

This work is supported by the following:
Dr. Myoungkyu Song, Computer Science, IS&T
Dr. Harvey Siy, Computer Science, IS&T
Dr. Rui Zhao, Cybersecurity, IS&T

Graduate Research & Creative Activity (GRACA 2019)

6th IEEE/ACM International Conference on Mobile Software Engineering and Systems
MobileSoft-2019 Vision Track
MobileSoft-2019 SRC Track

Mutual of Omaha Insurance Company