Fortify Security Report

Jun 17, 2015

dkwakkel

Executive Summary

Issues Overview

On Jun 2, 2015, a source code review was performed over the poi code base. 3,213 files, 178,513 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 2340 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order

Refined by: category:"weak encryption\: inadequate rsa padding" AND category:"weak encryption\: insecure mode of operation"

Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level. The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

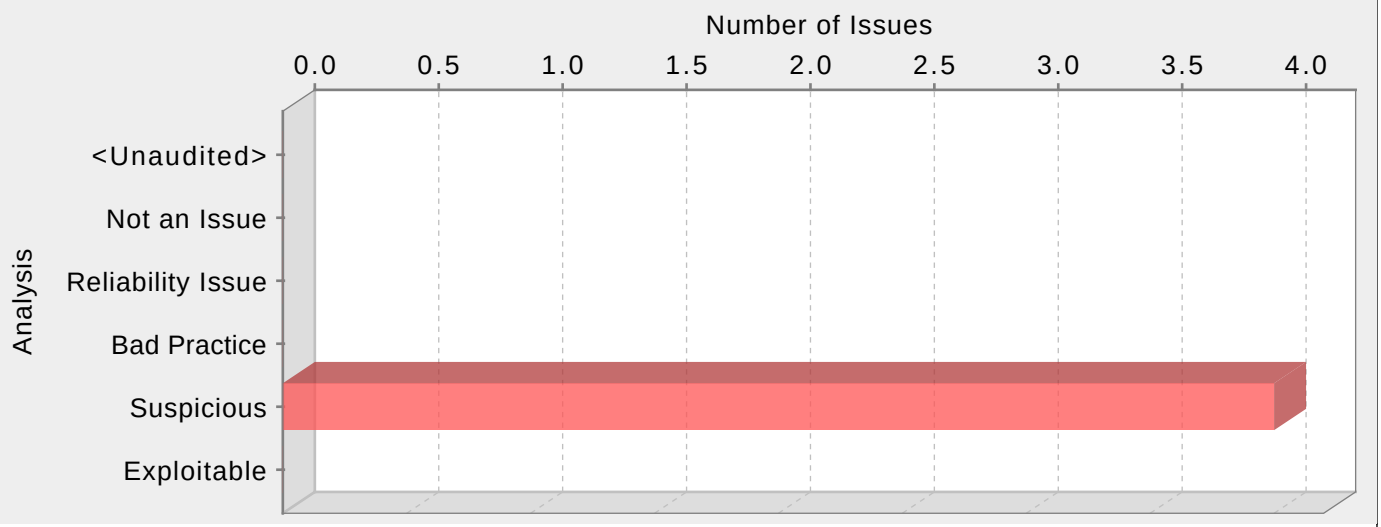
Results Outline

Overall number of results

The scan found 2340 issues.

Vulnerability Examples by Category

Category: Weak Encryption: Insecure Mode of Operation (4 Issues)



Abstract:

Cryptographic encryption algorithms should not be used with an insecure mode of operation.

Explanation:

A mode of operation of a block cipher is an algorithm that describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block. Some of the modes of operation include ECB (Electronic Codebook), CBC (Cipher Block Chaining) and CFB (Cipher Feedback).

ECB mode is inherently weak, because it results in the same ciphertext for identical blocks of plaintext. CBC mode does not have this weakness, making it the superior choice.

Example 1: The following code uses AES cipher with ECB mode:

```
...
SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
cipher.init(Cipher.ENCRYPT_MODE, key);
...
```

Cipher Transformation Modes:

The first argument to Cipher.getInstance is a string parameter transformation in the form "algorithm/mode/padding" or "algorithm". If the mode is not specified, then the mode selected is the provider-specific default, which is likely ECB (electronic codebook) mode for Java and Android.

ECB mode is inherently a weaker encryption mode because identical blocks of plaintext is encrypted into identical blocks of ciphertext. CBC (cipher-block chaining) mode is superior because it does not have this weakness.

Example: gaining a Cipher instance with the weak ECB transformation mode:

```
Cipher c = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

Example: gaining a Cipher instance with default transformation mode, which could be the weak ECB mode:

```
Cipher c = Cipher.getInstance("AES");
```

This finding is from research found in "An Empirical Study of Cryptographic Misuse in Android Applications". http://www.cs.ucsb.edu/~chris/research/doc/ccs13_cryptolint.pdf

Recommendations:

Avoid using ECB mode of operation when encrypting data larger than a block. CBC mode is superior because it does not produce identical blocks of ciphertext for identical blocks of plaintext. However, CBC mode is somewhat inefficient and poses serious risk if used with SSL [1]. Instead, use CCM (Counter with CBC-MAC) mode, or, if performance is a concern, GCM (Galois/Counter Mode) mode where they are available.

Example 2: The following code uses AES cipher with CBC mode:

```
...
SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding", "BC");
cipher.init(Cipher.ENCRYPT_MODE, key);
...
```

AgileEncryptor.java, line 204 (Weak Encryption: Insecure Mode of Operation)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	The function confirmPassword() in AgileEncryptor.java uses a cryptographic encryption algorithm with an insecure mode of operation on line 204.		

Sink: AgileEncryptor.java:204 getInstance()

```
202                 header.setEncryptedHmacKey(encryptedHmacKey);
203
204                 cipher = Cipher.getInstance("RSA");
205                 for (AgileCertificateEntry ace : ver.getCertificates()) {
206                     cipher.init(Cipher.ENCRYPT_MODE, ace.x509.getPublicKey());
```

Analysis: Suspicious

AgileDecryptor.java, line 222 (Weak Encryption: Insecure Mode of Operation)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	The function verifyPassword() in AgileDecryptor.java uses a cryptographic encryption algorithm with an insecure mode of operation on line 222.		

Sink: AgileDecryptor.java:222 getInstance()

```
220             if (ace == null) return false;
221
222             Cipher cipher = Cipher.getInstance("RSA");
223             cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
224             byte keyspec[] = cipher.doFinal(ace.encryptedKey);
```

Analysis: Suspicious

CryptoFunctions.java, line 215 (Weak Encryption: Insecure Mode of Operation)

Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	The function getCipher() in CryptoFunctions.java uses a cryptographic encryption algorithm with an insecure mode of operation on line 215.		

Sink: CryptoFunctions.java:215 getInstance()

```
213             } else if (cipherAlgorithm.needsBouncyCastle) {
214                 registerBouncyCastle();
215                 cipher = Cipher.getInstance(cipherAlgorithm.jceId + "/" + chain.jceId
+ "/" + padding, "BC");
216             } else {
217                 cipher = Cipher.getInstance(cipherAlgorithm.jceId + "/" + chain.jceId
+ "/" + padding);
```

Analysis: Suspicious

CryptoFunctions.java, line 217 (Weak Encryption: Insecure Mode of Operation)

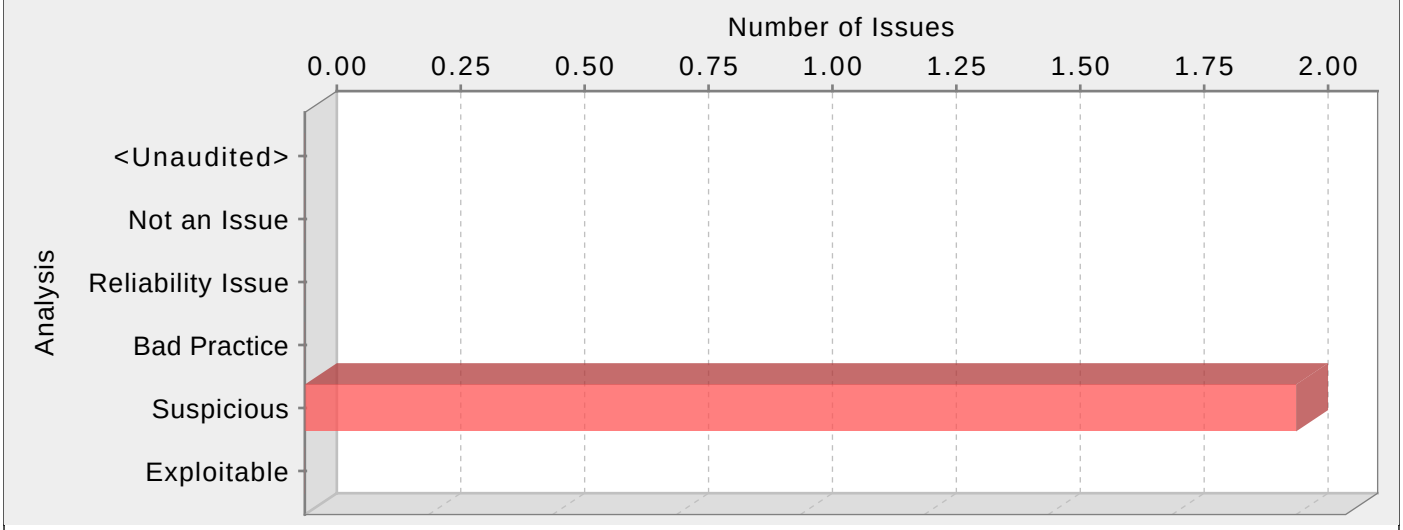
Fortify Priority:	Low	**Folder**	Low
Kingdom:	Security Features		
Abstract:	The function getCipher() in CryptoFunctions.java uses a cryptographic encryption algorithm with an insecure mode of operation on line 217.		

Sink:	CryptoFunctions.java:217 getInstance()

```
215                     cipher = Cipher.getInstance(cipherAlgorithm.jceId + "/" + chain.jceId
+ "/" + padding, "BC");
216                 } else {
217                     cipher = Cipher.getInstance(cipherAlgorithm.jceId + "/" + chain.jceId
+ "/" + padding);
218                 }
219
```

Analysis:	Suspicious

Category: Weak Encryption: Inadequate RSA Padding (2 Issues)



Abstract:

Public key RSA encryption is performed without using OAEP padding, thereby making the encryption weak.

Explanation:

In practice, encryption with an RSA public key is usually combined with a padding scheme. The purpose of the padding scheme is to prevent a number of attacks on RSA that only work when the encryption is performed without padding.

Example 1: The following code performs encryption using an RSA public key without using a padding scheme:

```
public Cipher getRSACipher() {
Cipher rsa = null;
try {
rsa = javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");
}
catch (java.security.NoSuchAlgorithmException e) {
log("this should never happen", e);
}
catch (javax.crypto.NoSuchPaddingException e) {
log("this should never happen", e);
}
return rsa;
}
```

This category was derived from the Cigital Java Rulepack. http://www.cigital.com/

Recommendations:

In order to use RSA securely, OAEP (Optimal Asymmetric Encryption Padding) must be used when performing encryption.

Example 2: The following code performs encryption with an RSA public key using OAEP padding:

```
public Cipher getRSACipher() {
Cipher rsa = null;
try {
rsa = javax.crypto.Cipher.getInstance("RSA/ECB/OAEPWithMD5AndMGF1Padding");
}
catch (java.security.NoSuchAlgorithmException e) {
log("this should never happen", e);
}
catch (javax.crypto.NoSuchPaddingException e) {
log("this should never happen", e);
}
return rsa;
}
```

AgileEncryptor.java, line 204 (Weak Encryption: Inadequate RSA Padding)

Fortify Priority:	Low	Folder	Low
Kingdom:	Security Features		
Abstract:	The method confirmPassword() in AgileEncryptor.java performs public key RSA encryption without OAEP padding, thereby making the encryption weak.		

Sink: AgileEncryptor.java:204 getInstance()

```
202                 header.setEncryptedHmacKey(encryptedHmacKey);
203
204                 cipher = Cipher.getInstance("RSA");
205                 for (AgileCertificateEntry ace : ver.getCertificates()) {
206                     cipher.init(Cipher.ENCRYPT_MODE, ace.x509.getPublicKey());
```

Analysis: Suspicious

AgileDecryptor.java, line 222 (Weak Encryption: Inadequate RSA Padding)

Fortify Priority:	Low	Folder	Low
Kingdom:	Security Features		
Abstract:	The method verifyPassword() in AgileDecryptor.java performs public key RSA encryption without OAEP padding, thereby making the encryption weak.		

Sink: AgileDecryptor.java:222 getInstance()

```
220             if (ace == null) return false;
221
222             Cipher cipher = Cipher.getInstance("RSA");
223             cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
224             byte keyspec[] = cipher.doFinal(ace.encryptedKey);
```

Analysis: Suspicious