Mobile Top 10 2014-M6

Broken Cryptography

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	**Exploitability EASY**	**Prevalence COMMON**	**Detectability EASY**	**Impact SEVERE**	**Application / Business Specific**
Threat agents include the following: anyone with physical access to data that has been encrypted improperly, or mobile malware acting on an adversary's behalf.	Attack vectors include the following: decryption of data via physical access to the device or network traffic capture, or malicious apps on the device with access to the encrypted data.	In order to exploit this weakness, an adversary must successfully return encrypted code or sensitive data to its original unencrypted form due to weak encryption algorithms or flaws within the encryption process.		This vulnerability will result in the unauthorized retrieval of sensitive information from the mobile device.	This vulnerability can have a number of different business impacts. Typically, broken cryptography will result in the following: • Privacy Violations; • Information Theft; • Code Theft; • Intellectual Property Theft; or • Reputational Damage.

Am I Vulnerable to Broken Cryptography?

Insecure use of cryptography is common in most mobile apps that leverage encryption. There are two fundamental ways that broken cryptography is manifested within mobile apps. First, the mobile app may use a process behind the encryption / decryption that is fundamentally flawed and can be exploited by the adversary to decrypt sensitive data. Second, the mobile app may implement or leverage an encryption / decryption algorithm that is weak in nature and can be directly decrypted by the adversary. The following subsections explore both of these scenarios in more depth:

Reliance Upon Built-In Code Encryption Processes

By default, iOS applications are protected (in theory) from reverse engineering via code encryption. The iOS security model requires that apps be encrypted and signed by trustworthy sources in order to execute in non-jailbroken environments. Upon start-up, the iOS app loader will decrypt the app in memory and proceed to execute the code after its signature has been verified by iOS. This feature, in theory, prevents an attacker from conducting binary attacks against an iOS mobile app.

Using freely available tools like ClutchMod or GBD, an adversary will download the encrypted app onto their jailbroken device and take a snapshot of the decrypted app once the iOS loader loads it into memory and decrypts it (just before the loader kicks off execution). Once the adversary takes the snapshot and stores it on disk, the adversary can use tools like IDA Pro or Hopper to easily perform static / dynamic analysis of the app and conduct further binary attacks.

Bypassing built-in code encryption algorithms is trivial at best. Always assume that an adversary will be able to bypass any built-in code encryption offered by the underlying mobile OS. For more information about additional steps you can take to provide additional layers of reverse engineering prevention, see M10.

Poor Key Management Processes

The best algorithms don't matter if you mishandle your keys. Many make the mistake of using the correct encryption algorithm, but implementing their own protocol for employing it. Some examples of problems here include:

- Including the keys in the same attacker-readable directory as the encrypted content;

- Making the keys otherwise available to the attacker;
- Avoid the use of hardcoded keys within your binary; and
- Keys may be intercepted via binary attacks. See M10 for more information on preventing binary attacks.

Creation and Use of Custom Encryption Protocols

There is no easier way to mishandle encryption--mobile or otherwise--than to try to create and use your own encryption algorithms or protocols.

Always use modern algorithms that are accepted as strong by the security community, and whenever possible leverage the state of the art encryption APIs within your mobile platform. Binary attacks may result in adversary identifying the common libraries you have used along with any hardcoded keys in the binary. In cases of very high security requirements around encryption, you should strongly consider the use of whitebox cryptography. See M10 for more information on preventing binary attacks that could lead to the exploitation of common libraries.

Use of Insecure and/or Deprecated Algorithms

Many cryptographic algorithms and protocols should not be used because they have been shown to have significant weaknesses or are otherwise insufficient for modern security requirements. These include:

- RC2
- MD4
- MD5
- SHA1