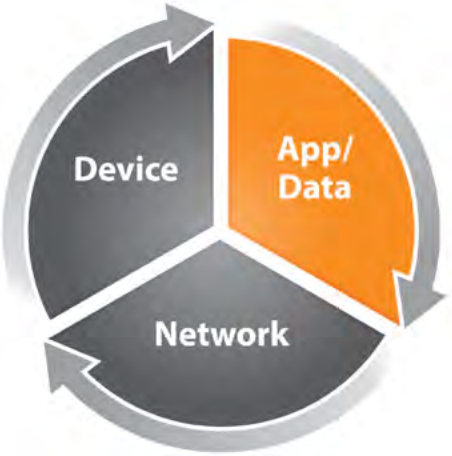
Mobile Application Protection Handbook

Defend Against Application Risks & Attacks by Protecting the Integrity of Your Binary Code

Three Key Components of Secure Mobile Solutions

To have a secure mobile solution, you need to have a secure device, a secure network, and a secure application. If any one of these is compromised, the solution is at risk.



Security is moving from the device to the application level. This handbook focuses on the risks associated with mobile applications/data and provides recommendations for how to protect them from attacks. This handbook is relevant for any organization that has or is developing sensitive or high-value B2C or B2B mobile applications that run on any platform.

This handbook was developed for resources involved in any way in ensuring the security of mobile applications (e.g., application developers, IT executives, security and risk officers).

Table of Contents

How Frequently are Application Getting Hacked? – Is There A Case For Action?

Many Applications are under Attack:

TechCrunch

Developer Spams Google Play With Rip -Offs of Well Known Apps… Again

"It's not uncommon to search the Google Play app store and find a number of knock-off or "fake" apps aiming to trick unsuspecting searchers into down-loading them over the real thing." January 2, 2014

EXTREMETECH

Chinese App Store Offers Pirated iOS Apps Without the Need To Jailbreak

"The site offers apps for free that would otherwise cost money, including big-name titles." April 12, 2013

SD Times
www.sdtimes.com

More than 5,000 apps in the Google Play Store are copied APKs, or 'thief-ware'

November, 2013

ARXAN

Arxan Research: State of Security in the App Economy, Volume 3

"Adversaries hacked 75 percent of the top 100 paid Android and iOS apps in 2014." November, 2014

Many applications are vulnerable:

HP Research Reveals Nine out of Ten Mobile Applications Vulnerable to Attack.

"86 percent of applications tested lacked binary hardening, leaving applications vulnerable to information disclosure, buffer overflows and poor performance. To ensure security throughout the life cycle of the application, it is essential to build in the best security practices from conception." *November 18, 2013*

Potential Business Impacts

- Confidential Data Theft
- Unauthorized Access and Fraud
- Brand and Trust Damage
- Revenue Loss and Piracy
- Intellectual Property Theft
- User Experience Compromise

Are Your Applications at Risk?

If you answer **YES** to any of these questions, your applications are vulnerable to a binary attack:

1.	Can someone code-decrypt this app (iPhone specific) using an automated tool like ClutchMod or manually using GDB?
2.	Can someone use an automated tool like Hopper or IDA Pro to easily visualize the control-flow and pseudo-code of this app?
3.	Can someone modify the app's presentation layer (HTML/JS/CSS) within the phone and execute modified JavaScript?
4.	Can someone modify the app's binary executable using a hex editor to get it to bypass a security control?

Key security analysts, associations, and providers are recommending that binary code be protected:

Gartner

"For critical applications, such as transactional ones and sensitive enterprise applications, hardening should be used."

Gartner report: Avoiding Mobile App Development Security Pitfalls

May 24 , 2013

Gartner

"Make application self-protection a new investment priority, ahead of perimeter and infrastructure protection...it should be a CISO top priority." *– September, 2014*

OWASP The Open Web Application Security Project

Lack of Binary Protection identified as a Mobile Top 10 Risk by OWASP

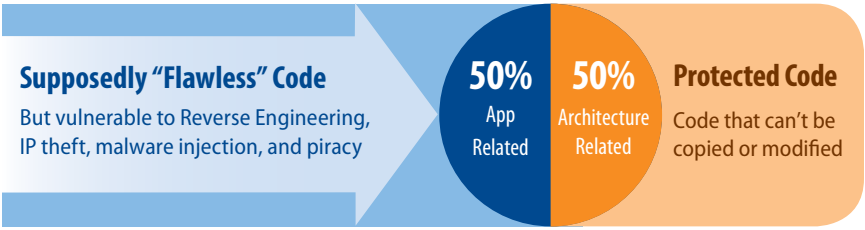
IBM

"Within IBM's Mobile Security Framework, Hardening and Tamper Proofing binary code is a key step in "Securing the Application"

How are mobile applications being attacked as a result of a lack of binary code protection?

Many developers use traditional techniques to confirm that code is secure. They perform vulnerability tests on their code and remediate areas of conventional risk within their applications. However, unless the binary code is **hardened**, the application is still vulnerable to prevailing risks (see graphic below).

Breakdown of App Risks*



Best Practice Development Process

The following steps describe how hackers are analyzing and reverse engineering applications that have not been hardened.

1

- Developer creates source code that runs application
- Source code is compiled into binary code and made available through online app stores
- Hacker buys or downloads free copy of application

2

For Android

- Unpack the Android app
- Convert Dalvik bytecode to Java bytecode using dex2jar, a freely available tool
- Decompile Java bytecode to Java source code, JD-GUI, another freely available tool

For Apple

- Decrypt the native app (if it is downloaded from the App Store) on a jailbroken device, using clutch – a freely available tool. This step removes the encryption protection of application code added by the App Store
- Disassemble the decrypted app using a disassembler, such as the inexpensive Hopper Disassembler
- Decompile the disassembled code using Hopper, which also features decompilation. That's it!

3

- Modifies code / Inserts malware
  - For example, to steal sensitive credit card information , hackers can easily discover the APIs that take in credit card information. With that knowledge, hackers can write malware that specifically targets the app and steals sensitive credit card information from users when the compromised app is run.
- Copy or pirate code, disable jailbreak-detection logic, and make a similar application available on third party sites
  - Those with jailbroken devices can run these pirated apps like normal apps
  - If code has been tampered with to disable the jail break-detection logic, then the pirated app will be able to run on any device!
- Copy critical IP and use it within other applications

What are the key application risks that you should defend against?

A summary of **application integrity risks** that are typically targeted for malicious gain is provided below:

- **Jailbreak and Root Detection bypass**
  Organizations may want to know that their code is running in a jail-broken environment for a number of different reasons. For example, they may choose to not honor a financial transaction conducted on the device due to increased uncertainty of its security environment. An adversary can force an application to run in these devices by modifying the logic of the jailbreak-detection code.

  Jailbreak detection code is notoriously difficult to implement correctly due to a myriad of evolving techniques available for an adversary to bypass or trick the code. The adversary successfully tricks the code into running in a hostile environment.

- **Repackaging**
  For an adversary to modify an app, they must first defeat Apple's code encryption and code signing technology that Apple automatically includes with each app in its App Store. Once the adversary bypasses these controls, they can modify the app and host it on a third-party site for download. Victims can use their iDevices (non-jailbroken) to download and execute these apps from these third-party sites.

  To defeat Apple's initial security controls, the adversary must first download the app from the iTunes store or through an Enterprise using an Enterprise deployment model. Next, the attacker must successfully start the app on a jailbroken iDevice. In doing so, the adversary can decrypt the app using unauthorized tools and execute subsequent steps to make the desired modifications.

- **Swizzle with Behavioral Change**
  Objective-C supports dynamic redirection of method invocations from one method to another of the same signature. This handy feature is commonly referred to as method swizzling. This feature is typically used in cases where an application needs to perform method substitution or method extension.

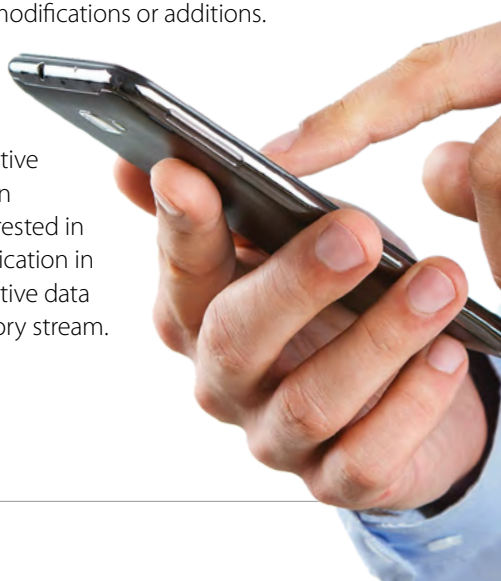
  An adversary can leverage this feature and redirect Objective-C method calls to malicious code provided by the adversary in the form of an external library. The Objective-C runtime will invoke the adversary's malicious form of the method rather than the original and safe one.

  This feature is also exploitable within Android environments through Cydia Substrate tools that facilitate such attacks. In said environments, the tool targets NDK-based applications written in C/C++

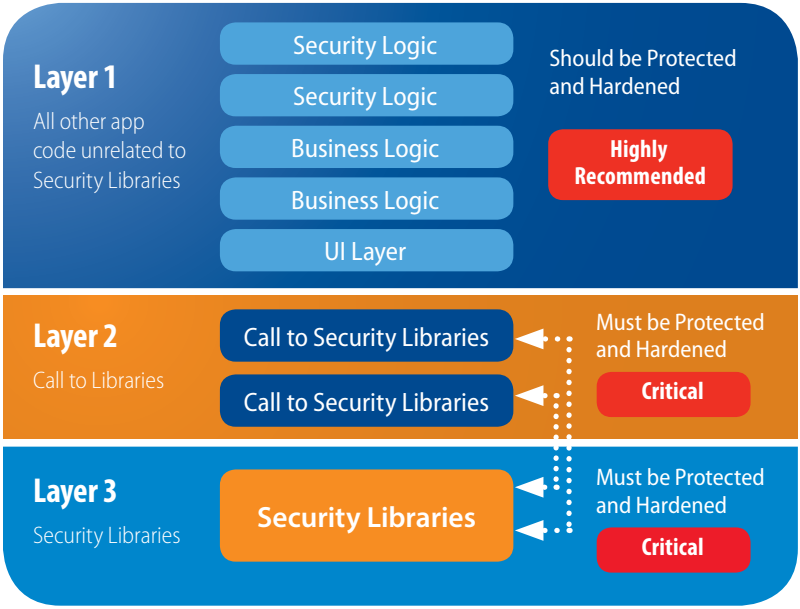
- **Security Control Bypass**
  Many apps contain decision-making control flows that guard the execution of sensitive operations. Without protection, such logic is subject to circumvention by the adversary through unauthorized code modification.

- **Presentation Layer Modification**
  Within hybrid apps, an application contains an outer shell that is typically written in Java or Objective-C. In the inner shell, the application contains a set of presentation files typically exposed as HTML, JavaScript, and CSS files. An adversary may choose to modify these presentation-layer files to perform unauthorized operations through JavaScript modifications or additions.

- **Cryptographic Key Replacement**
  Applications use cryptographic keys to encrypt or decrypt sensitive data residing in a local store or in memory. Attackers may be interested in replacing keys used by the application in order to decrypt and copy sensitive data from a local repository or memory stream.



In summary, all three layers – security libraries, calls to libraries, and all other code unrelated to security libraries must be addressed. See graphic below:



Example: Circumventing Google's License Verification Library (LVL)

LVL is a Java-based license manager that an app from Google Play can use to verify if its users have properly paid for it. Integrated as part of the app, LVL determines the licensing status on behalf of the app through the LicenseChecker class. Behind the scenes, however, for an attacker to defeat the entire licensing verification mechanism he/she just needs to change one crucial decision-making instruction in LicenseChecker's checkAccess method.

Single points of failure like the above example are common across many security controls, including those provided by Mobile Application/Device Management (MAM/MDM) solutions. MAM/MDM solutions wrap unprotected mobile apps with per-app security controls, such as password protection, data encryption, remote wiping of app data, and so on. However, these are policy code additions that only defend the app against misuse and data breach at the user level. Internally however, the decision-making logic of the policy code is still vulnerable to tampering.

Techniques and best practices for protecting binary code

The following table summarizes key business requirements for protecting an application's binary code and their impact.

Business Requirements	Impact
• Confirm your mobile application is being protected against the widest range of static, dynamic and BORE attacks • Leverage layered protection that continually evolves to thwart hackers	• Increased application security
• Leverage a solution that protects apps across all mobile computing platforms	• Ability to standardize on application security process and tools • Reduce need to leverage multiple security providers and integrate application protection solutions
• Leverage an approach that is not disruptive to the software development lifecycle or application functionality	• Ability to deploy quickly across many applications without risk of compromising the source code. • Source code developers don't need to learn a new tool/ language. Developers don't need to worry about compromising their application when addressing security concerns, since there are no source code changes, agents, or separate processes
• Extend app security from vulnerability testing to run-time protection– 'build it secure' during development and 'keep it secure' in production	• Ability to quickly gain security intelligence on attempts being made to compromise mobile apps, and react flexibly • Ability to handle attacks in a controlled manner
• Leverage an approach that offers the ability to tailor the amount of protection per application	• Attainment of a robust security solution that has minimal impact on performance
• Leverage an approach whereby Security is inserted into applications directly, so it goes where the app goes	• Ability to port mobile apps from one environment to the next

In order to deploy effective application protection, a two-step process can be taken:

First: Identify relevant risks and attack targets in your application. This forms the basis of your protection strategy.

Second: Protect apps with robust and dynamic application-hardening and integrity protection techniques, which defend, detect, alert, and react to attacks.

The inclusion of the following security techniques applied at the end of the build process will yield self-aware, self-defending and tamper-resistant apps.

Defend Against Compromise

Obfuscation: Hide structure and code flow within the application

Encryption: Encrypt parts or the whole application when stored on disk and when unused at runtime. Also, encrypt data within the application

Renaming (iOS): Rename Objective C++ metadata to frustrate reverse-engineering attempts

Detect Attempted Attacks

Checksum: Verify integrity of the application and its data at run-time

Anti-Debug: Block application from executing in presence of kernel-mode debuggers, or when executed in an emulated environment

Authentication: Cross module integrity verification of the components of an application

Value Verification: Mini Guard that compares a 4-byte expected value to runtime value within an image to detect change

Jailbreak Detection (iOS): Robust detection of jailbroken devices

Swizzle Detection (iOS): Detection of Mobile Substrate / Theos-based types of dynamic function replacement

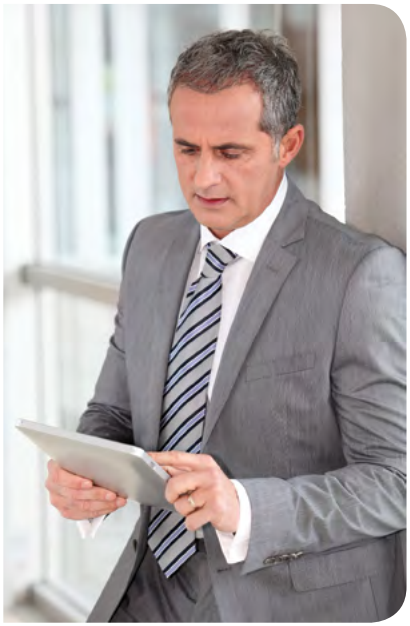
Resource Verification (iOS/Java): Verification of on-disk assets - check summing files within ipa or apk packages

Alert and React to Ward Off Attack

Repair: Replaces tampered regions of software application with original code. Can also be used to replace sensitive regions with decoy or garbage code when not in use, thus limiting window of opportunity for compromise

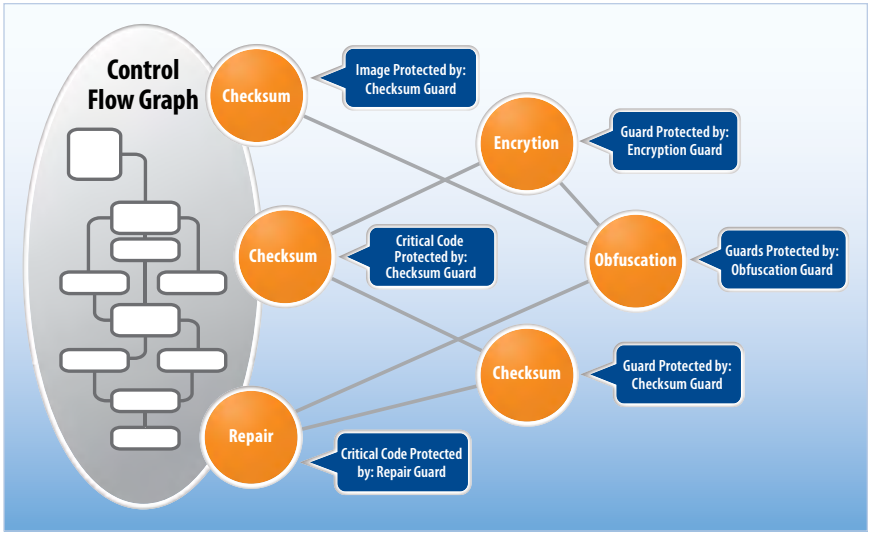
Standard Reactions - Exit, fail, notify, custom call/function callback

Alerts – Alert local or remote, integrated with Enterprise Security Management consoles



Many of the above techniques are very difficult or error-prone to apply manually, due to their binary-level nature. Fortunately, there are sophisticated (commercial) tools available that can apply these techniques on users' behalf (e.g., obfuscate the control flows of the application so they become highly convoluted). Some of these tools such as Arxan's GuardIT® and EnsureIT® can even achieve multi-layered networked protections at the binary level.

It is important to layer these defenses and implement them in a networked fashion such that the protection does not cover just the vulnerable application code, but also the protection mechanisms themselves. This is a **multi-layered, defense-in-depth approach** that has been proven highly effective against hacking attempts. An illustration of networked guards is provided below:



Best Practices for Establishing a Layered, Network of Guards

- Use multiple Guards to protect a single code segment
- Guards protect selected ranges of code
- Guards protect entire image
- Guards protect each other
- When attack is detected, Guards 'fire', and reaction is fully programmable
- Use many implementations of given Guard, so no global signature

This compiled-in approach for application integrity protection is transparent to the software development lifecycle with no modifications to source code or app performance parameters.

When applied appropriately, these self-defense techniques can ensure your app is highly resilient against attacks, even on rooted or jailbroken (compromised) devices and independently detect whether its own state has been modified, and take remedial or punitive actions as needed.

For a free risk mitigation consultation or to learn more, please visit our website at www.arxan.com

For additional information on architectural principles that prevent code modification, we also recommend that you go to https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_Ten_Mobile_Risks

About Arxan Technologies Inc.

Arxan provides the world's strongest application protection solutions. Our unique patented guarding technology 1) Defends applications against attacks, 2) Detects when an attack is being attempted, and 3) Responds to detected attacks with alerts and repairs. Arxan offers solutions for software running on mobile devices, desktops, servers, and embedded platforms —including those connected as part of the Internet of Things (IOT)—and is currently protecting applications running on more than 300 million devices across a range of industries, including: financial services, high tech/independent software vendors (ISVs), manufacturing, healthcare, digital media, gaming, and others. The company's headquarters and engineering operations are based in the United States with global offices in EMEA and APAC.

Trademarks

Arxan Technologies, the Arxan logo , EnsureIT®, GuardIT®, TransformIT®, GuardSpec®, Protecting the App Economy® and all other Arxan Technologies product and service marks (collectively "Trademarks") are the property of Arxan Technologies and other parties. All other trademarks, service marks, registered trademarks, and registered service marks are the property of their respective owners.

Content and Liability Disclaimer

Arxan Technologies shall not be responsible for any errors or omissions contained on any Arxan Technologies website or this handbook, and reserve the right to make changes anytime without notice. Mention of non-Arxan Technologies products or services is provided for informational purposes only and constitutes neither an endorsement nor a recommendation by Arxan Technologies. All Arxan Technologies and third-party information provided on any Arxan Technologies website or within this handbook is provided on an "as is" basis.

Arxan Technologies disclaims all warranties, expressed or implied, with regard to any information (including any software, products, or services) provided on any Arxan Technologies website or this handbook, including the implied warranties of merchantability and fitness for a particular purpose, and non-infringement.

Second Edition August 2015

Notes

Arxan Technologies, Inc.
6903 Rockledge Drive, Suite 1250
Bethesda, MD 20817
Phone: 301.968.4290
www.arxan.com