ZIMPERIUM®

# 10 ways to shield and protect apps

In-app protections modify an app to make it more resistant to reverse-engineering, tampering, and monitoring.

Below are our top 10 recommended techniques, from the most basic obfuscation up to equipping your app with autonomous attack detection and response.

## 1 Modify names

Change identifiable variable and method names into meaningless character strings to make them confusing to a hacker. There are several free obfuscation tools that rename classes, fields, and methods.

## 2 Insert dummy code

Add extra, non-functional code into the application to make reverse-engineered code more difficult to analyze. Many of the free obfuscators also insert dummy code.

## 3 Remove unused code and metadata

Eliminate dead code, debug information, and non-essential metadata from applications to reduce the information an attacker can obtain and use.

## 4 Obfuscate Objective-C calls

Messages called in Objective-C are resolved at run time, which means they are stored in the code in plain form where they can be exploited. Protect Objective-C code by obfuscating plain text message calls so that they are not easily readable and editable.

## 5 Use binary packing

Pack your code to protect against static analysis. Packers encrypt and compress code, adding a stub that unpacks it at runtime. This makes it difficult for hackers to reverse engineer as they can't run the code through a disassembler or decompiler.

## 6 Protect against debuggers

Hackers use debuggers to understand how an application works and to reverse engineer code. For basic anti-debugging, insert API calls to query process and system information to check for debugger presence. For stronger debugger protection, use a tool that detects debugger-inserted breakpoints and automatically executes defensive actions.

## 7 Diversify your software

Create diverse software instances that are functionally identical, but where the code is uniquely different in shape and structure. This forces hackers to break each copy of the application separately, rather than using a single attack across all instances of the application.

## 8 Obfuscate the control flow

Modify the basic structure of how subroutines are called to make the code indecipherable. For example, inline functions, replace calls to subroutines with computed jumps, and flatten the control flow by converting tree-like conditional constructs into flat switch statements.

## 9 Protect against tampering

Deploy in-app protections that prevent attempts to modify and hijack application code—insert overlapping checksums to test code integrity, add iOS jailbreak and Android rooting detection, use shared library cross-checking, anti-method swizzling, function caller verification, and other anti-tampering protections.

## 10 Make your app self-defending

When your application identifies a tampering attempt, it should trigger a relevant defensive response. If possible, incorporate real-time, automated defense actions rather than generating alerts that need to be manually remediated. Common defense actions include blocking account access, halting the execution of commands, deletion of sensitive data, and shutting down the application entirely. To avert additional attack attempts, corrupt select elements of the application so a hacker believes they have been successful, but they actually only have minimal access.

**Zimperium helps organizations to build secure and compliant mobile applications. It is the only unified solution that combines comprehensive in-app protection with centralized threat visibility. Click here to learn more.**

ZIMPERIUM®

4055 Valley View Dallas, TX 75244
844.601.6760
info@zimperium.com