Why You Need To Shield Your Apps

zSHIELD™

ZIMPERIUM®

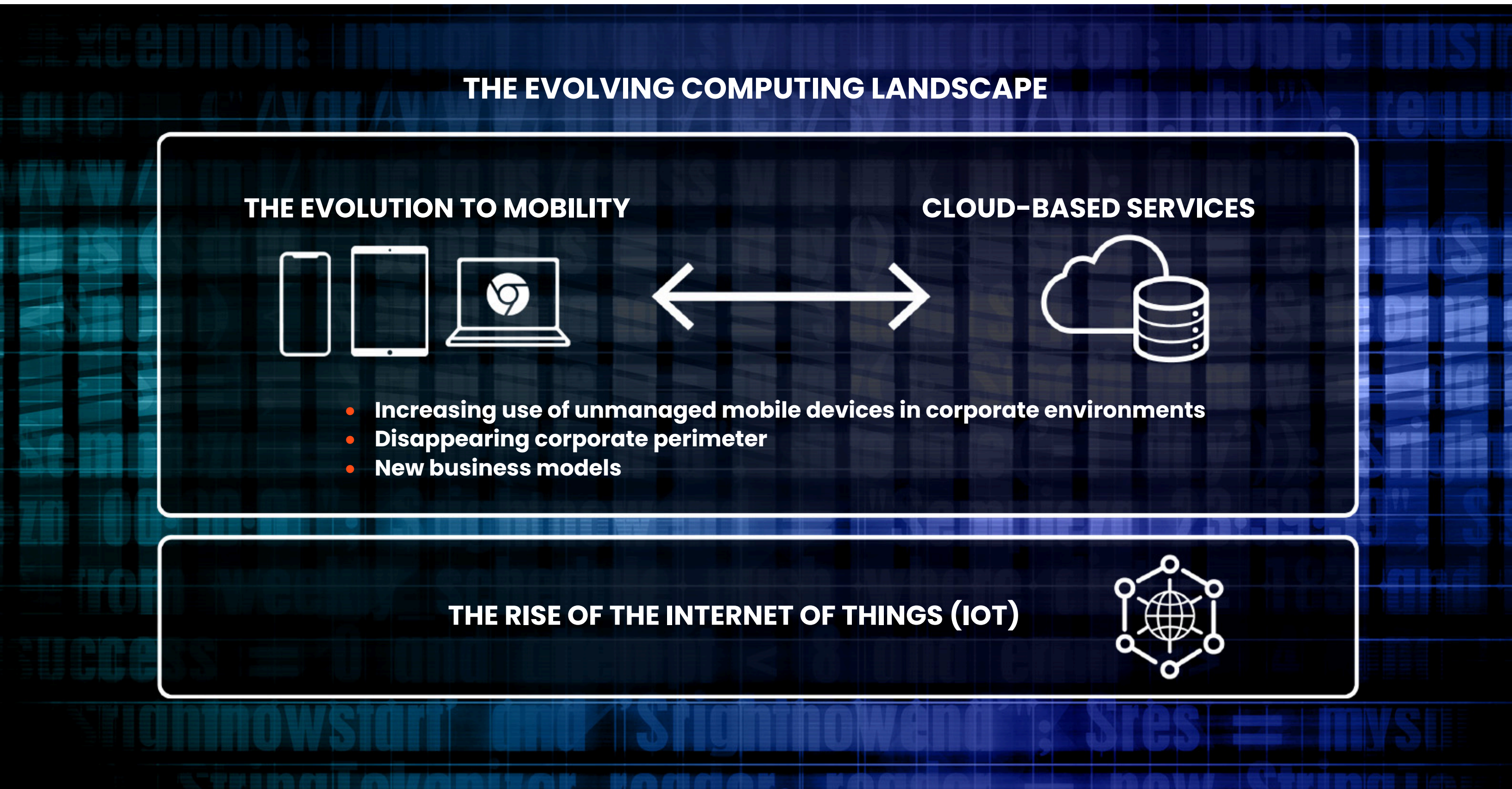
Executive Summary

The world of computing has changed. Security is not just about physically secure data centers and corporate controlled computing assets. Instead, end users have gone mobile, connecting to cloud enabled services, often with their own personal devices. And with the rise of the Internet of Things, there will be billions of connected computing devices on the planet in the next several years. These changes significantly impact the way organizations are providing services to their customers, enabling new business models and new ways to do business.

But these changes also create new opportunities for hackers who have unprecedented physical access to these devices. Hackers have a wide variety of goals including bypassing business logic, stealing intellectual property or sensitive data, obtaining cryptographic keys to steal content, masquerade as users, snoop on secure communications, or hack a device as a stepping stone to launch attacks against other devices. If they are successful, then the consequences can include financial loss, impact on brand reputation, and exposure to liability.

The way hackers go about achieving their goals starts with reverse engineering software to find vulnerabilities they can exploit, data they can extract, or ways to modify the software to do something it was never intended to do. As hackers get increasing access to mobile and IoT devices, this threat also increases. As a consequence, it is becoming increasingly important for organizations to deploy **application shielding** technology that makes it difficult for hackers to reverse engineer and tamper with software.

This white paper describes these threats in detail, and describes Zimperium's market leading application shielding product, **zShield**. zShield provides application developers with a comprehensive suite of anti-reverse engineering and runtime application security protections to help protect your applications. zShield is easy to use and requires no significant changes to the code itself or the existing build chain. Since zShield secures source code before it is compiled, protected builds can easily be delivered to an app store, end point, mobile device, connected car, and other types of IoT devices.


THE EVOLVING COMPUTING LANDSCAPE
THE EVOLUTION TO MOBILITY
CLOUD-BASED SERVICES
Increasing use of unmanaged mobile devices in corporate environments
Disappearing corporate perimeter
New business models
THE RISE OF THE INTERNET OF THINGS (IOT)

The Problem

Computer and information security is one of the biggest problems that businesses face today. Gartner is forecasting worldwide spending on information security, and risk management technology and services will grow 12.4% to reach $150.4 billion in 2021. That comes on top of a 6.4% increase in cybersecurity spending in 2020. But the majority of this budget is still being spent on legacy infrastructure and endpoints, leaving mobile and modern apps more vulnerable than ever.

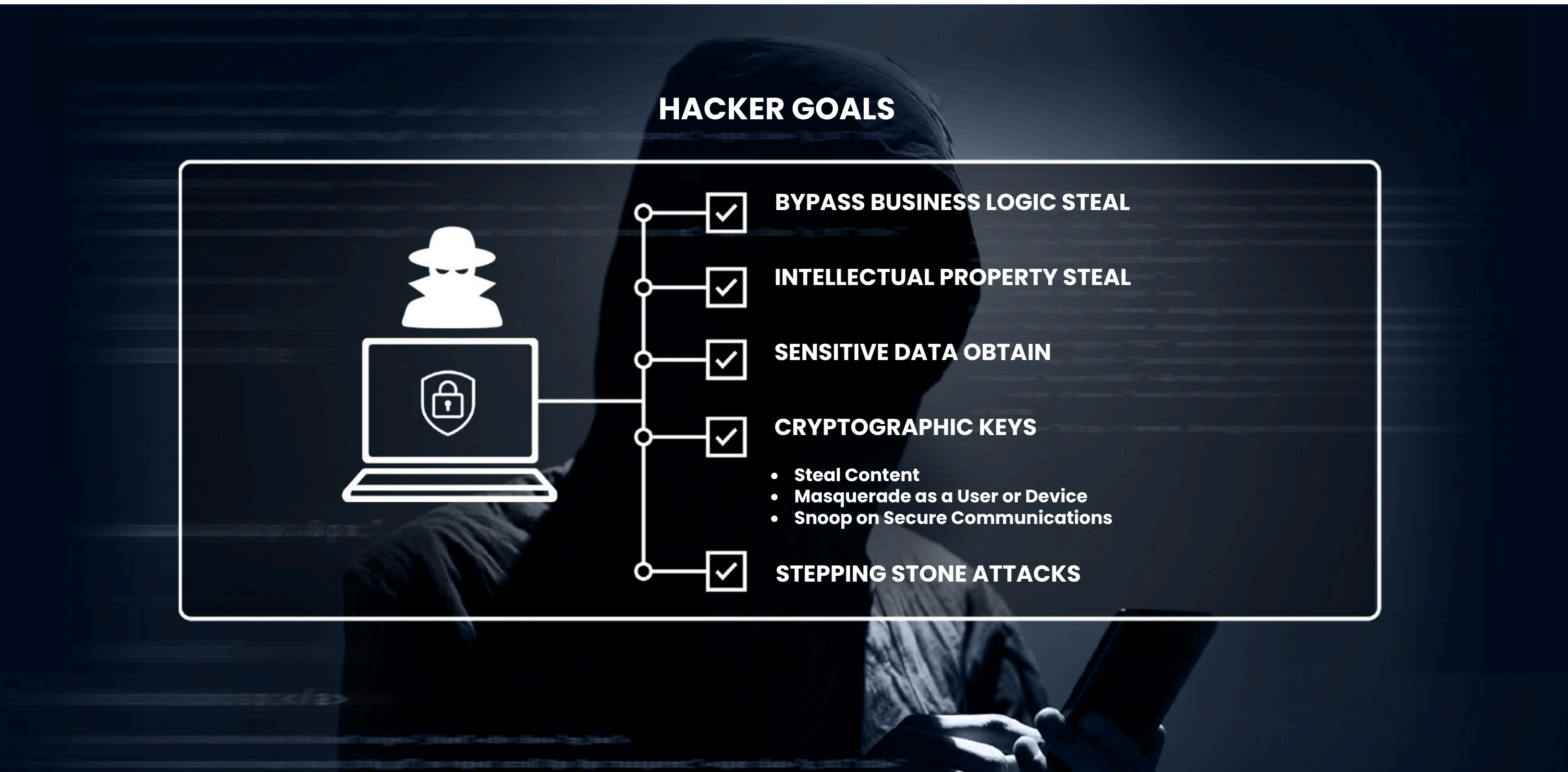
There are approximately 7 billion smartphones and 1 billion tablets running over 6 million apps from the stores. And this meteoric rise is due to the ability of these applications to combine data and a fantastic user experience to remove friction and deliver value to our lives every day. But the downside is that these applications are left protected. The proprietary code and sensitive data within these applications make them lucrative targets for cybercriminals and competitors.

The focus of this white paper is on shielding your application to protect your business. The primary consequences of compromised applications include financial loss, destroyed brand reputation, exposure to liability, and regulatory risk.

What Are Hackers Trying to Achieve?

In order to understand threats, we must understand what hackers are trying to achieve. Hackers will mount different kinds of attacks to achieve different kinds of goals. And so, defending against hackers in the context of application security may involve defending against many different kinds of attacks.

Hackers might be interested in bypassing business logic. For example, they might want to bypass controls that let them cheat at a video game or violate the terms of a software license. Of more serious concern is the potential for hackers to bypass controls in safety critical systems. It is not inconceivable that lives could be at risk if a hacker were able to hack a medical device, connected car or some component of critical infrastructure, such as a wind farm[1], a coal or nuclear power plant, a power grid, or a water treatment facility.


HACKER GOALS
BYPASS BUSINESS LOGIC STEAL
INTELLECTUAL PROPERTY STEAL
SENSITIVE DATA OBTAIN
CRYPTOGRAPHIC KEYS
Steal Content
Masquerade as a User or Device
Snoop on Secure Communications
STEPPING STONE ATTACKS

Many organizations spend millions of dollars developing applications that contain millions of lines of code. According to a recent study, automobiles today run systems that have more than 100 million lines of code.[2] Those applications often contain valuable intellectual property, which hackers would rather steal than develop. For example, they might be a competitor (or nation state) with inferior technology attempting to improve their own products in order to compete more effectively.

Hackers might also be interested in obtaining valuable pieces of data that are managed within the application, such as music or video, financial data, or privacy sensitive health data.

While data can be protected with cryptography, this only shifts the problem from protecting the data directly to protecting the *cryptographic keys*. Cryptographic keys are not only used to protect data. They can also be used to create a secure identity for a device. A device may need such a key to authenticate to a cloud service. If a hacker were able to obtain this secret, they might be able to masquerade as that device or as the owner of the device. Cryptographic keys are also used to establish secure communications. For example, HTTPS is a familiar protocol that uses SSL/TLS to secure communication to websites. If a hacker were able to obtain these keys, they could snoop on or alter supposedly secure communications.

For all of these reasons, hackers are highly motivated to steal cryptographic keys embedded in or controlled by an application.

Finally, sometimes hackers aren't interested in the application itself, but using the application as a digital stepping stone to try to achieve some other goal. Often hackers are interested in obtaining root access on the device the application is running on, so they can install malware or use the device as a launch pad to attack something else.

How Hackers Work

Hackers employ two fundamental techniques when attacking: **reverse engineering** and **tampering**. If the hacker is trying to bypass business logic, they have to find where in the application the business logic resides. That requires reverse engineering. Then they typically must tamper with the application to bypass that logic.

If the hacker is trying to steal intellectual property, sensitive data or cryptographic keys from an application, they have to know where to look in the application. Unless those secrets are obvious, hackers need to reverse engineer the application to find them.

If the hacker is trying to create a stepping stone attack, they often use the workflow shown in Figure 1. First, they find some vulnerability in the application, which again requires reverse engineering. Then, they craft an exploit that takes advantage of that vulnerability. Finally, they attack by launching the exploit to the application. In a remote attack like the popular SQL injection attack, this may involve sending the message to the application over the internet. But if they have physical access to the device, which with mobile and IoT based systems can be as easy as a trip to the store, then they can directly tamper with the device. You must expand your threat matrix into this new reality.

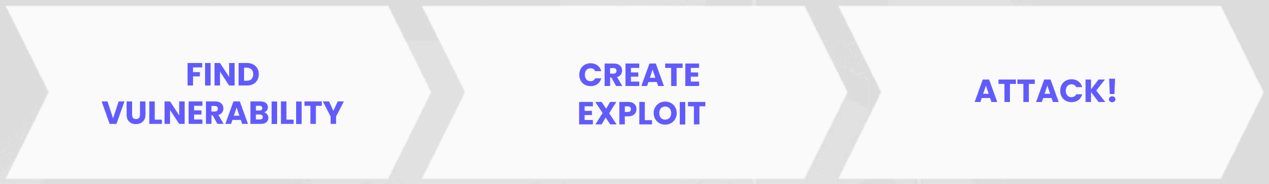


Figure 1: Active attack workflow

The Solution

In the previous section, we showed how reverse engineering and tampering with code are fundamental tools hackers use to achieve their objectives. We also showed how hackers have more opportunities to have physical access to devices because of the explosion in mobile devices, new business models, and the Internet of Things. These trends mean hackers have more opportunities to have direct physical access to applications, where the hacker can reverse engineer the code to steal intellectual property or sensitive data, can tamper with the application. The solution to these problems lies with application shielding, which prevents reverse engineering and tampering with code.

Zimperium's zShield provides the industry's best application shielding solution available on the market. zShield is a comprehensive code protection solution intended for hardening software applications on multiple target platforms. It adds tamper resistant characteristics to applications by applying code obfuscation, integrity protection, anti-debug, and anti-piracy techniques to application code (see Figure 2).

zShield can protect any standards compliant C/C++/Objective-C/Swift or Android Java source code and **requires no significant changes to the code itself or the existing build chain**.

Most of the security features are automatically added to the application, and the configuration of individual features and security strength is easily accomplished using the intuitive graphical user interface. All functions are available through command line for integration into automated build systems.

A highly specialized form of anti-reverse engineering is whitebox cryptography, in which a special cryptographic library is provided that provides strong protection for cryptographic keys. In whitebox cryptography, the underlying mathematics of the cryptographic operations are obfuscated in such a way that the keys never appear in the clear. Standard operations such as encryption, decryption, secure key unwrap and digital signature creation and validation can all be done with whitebox cryptography techniques, protecting the keys even if the device is jailbroken or rooted.

Zimperium's zKeyBox provides an industry leading whitebox cryptography solution to this problem and protects against modern side-channel attacks like DFA/DCA. A full discussion of whitebox cryptography is beyond the scope of this paper. For a detailed overview of zKeyBox, visit our website and companion white paper.

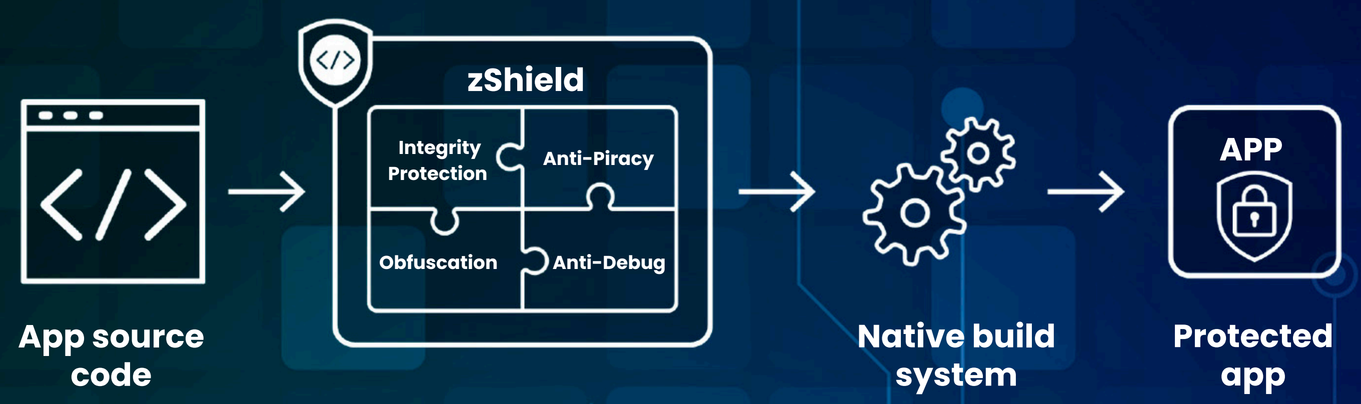


Figure 2: Zimperium's zShield Overview

Anti-Reverse Engineering

By making code difficult to reverse engineer, hackers will have a harder time finding vulnerabilities. Thus, application shielding not only protects against stealing intellectual property or sensitive data, but it can also protect your application by making it hard for hackers to find vulnerabilities.

Our customers's objective is to protect their applications for as long as possible against all threats. We enable this by make the hackers' job of reverse engineering code as difficult as possible. zShield provides the industry's best anti-reverse engineering functionality.

Advanced Obfuscation

One of the key techniques used in anti-reverse engineering is code obfuscation. The term "obfuscate" means to render obscure, unclear or unintelligible. The goal is to remove as much of the structure as possible that would be familiar to reverse engineers, to make the code as confusing as possible while keeping functionality the same.

Control flow obfuscation modifies the basic structure of how subroutines are called. For example, calls to subroutines could be replaced with computed jumps and functions can be inlined. Symbols for function names can be replaced with random strings. For example, in the Objective-C programming language, commonly used on Apple platforms, the function names are generally preserved after compilation, which gives hackers a valuable piece of information about the structure of the code. We provide a feature to encrypt this Objective-C Metadata to further frustrate hacker efforts. Dummy blocks that serve no purpose other than leading the hacker down a dead end are also frequently used.

zShield is capable of in-lining static void functions with simple declarations into the calling functions. Such operation increases the obfuscation level of the final protected code and makes it more difficult to trace. The overall result is increased security of the protected application.

Another important technique is control flow flattening, in which routines are not called directly from other routines, bur rather a dispatcher controls the control flow as illustrated in Figure 3.

Most code is made up of a series of basic blocks, which are some set of non-branching instructions (arithmetic or logical operators) followed by a conditional or absolute jump. The structure of a function can be viewed as a graph of basic blocks, kind of like a flow chart. Basic blocks can be rearranged, split or combined. For example, the basic blocks of all the functions can be intermingled, so that the code for any one function is not contained in a contiguous region of the executable.

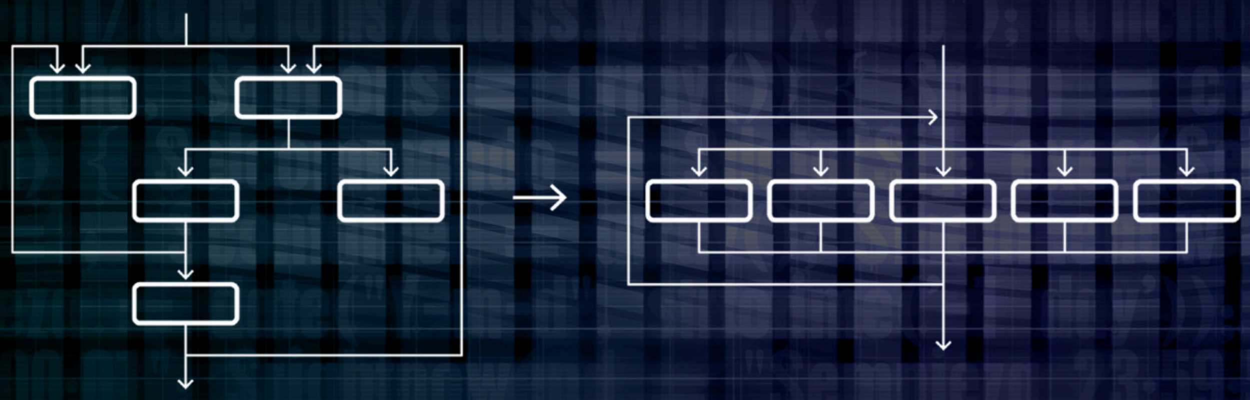


Figure 3: Control flow flattening

An absolute jump can be replaced with a computed jump adding edges to the graph. For example, consider the following code fragment:

```
if ((x*x) & 2) {
      foo();
} else {
      bar();
}
```

This seems quite straightforward. It takes the square of the number x and branches to function `foo()` if the second least significant bit is one, and to `bar()` if that bit is zero. However, it turns out that the second least significant bit of the square of any number is always zero, thus `foo()` will never be called. Why would we do that? We could replace `foo()` with another real function in the code, one that looks important. Static analysis tools cannot generally figure out that this jump will never be taken, and so the attacker may spend valuable time trying to understand what this part of the code is doing.

Dummy basic blocks can be introduced increasing the size of the graph, and thus the complexity of understanding the code. Imagine inserting thousands or even millions of such indirections, creating a tangle of function calls resulting in epic hacker frustration. It's a good feeling.

Another feature to obfuscation control flow is Objective-C Message Call Obfuscation. Objective-C is designed so that messages to object instances are resolved only at run time. Because of this fact, message calls are stored in plain form in the binary code. This is an attack vector that hackers can use to manipulate the execution logic.

zShield provides a powerful security feature that obfuscates message calls in the binary code, thus making reverse engineering more difficult. zShield provides a security feature that obfuscates a large portion of string literals (including Objective-C string literals, which are NSString pointers) in the code and deobfuscates them only before they are actually used. This feature increases protection against static analysis.

Finally, metadata obfuscation is also important. Executable files often come with metadata in the headers that can provide useful information to hackers about how to reverse engineer code.

An Objective-C executable contains metadata that provides information about names of classes and methods, as well as method arguments and their types within the executable. This information can aid attackers when they are statically analyzing the program with a disassembler.

"It seems that every time we introduce a new space in IT, we lose 10 years from our collective security knowledge. The Internet of Things is worse than just a new insecure space: it's a Frankenbeast of technology that links network, application, mobile and cloud technologies together into a single ecosystem, and it unfortunately seems to be taking on the worst security characteristics of each."

– Daniel Miessler, OWASP IoT Top 10 Project

Anti-Debugging
One of the primary tools hackers use to reverse engineer code is debuggers. Normally used by legitimate software engineers to find bugs in code, debuggers give hackers a powerful tool from which to reverse engineer code. Debuggers generally work by setting interrupts at specific points in an executable, thus modifying the executable. When the instruction counter reaches that point in the code, an interrupt is raised and the program stops at that point. Debuggers like IDA Pro do other things like disassembly and decompilation.

zShield inserts numerous anti-debug checks into your protected application. These checks take into account the unique indications of the target platform that may identify the presence of a debugger. When a debugger is detected, an appropriate defense can be taken.

These anti-debug checks use kernel syscalls, thereby bypassing usermode hooks the hacker may have inserted. This forces an attacker to modify their kernel, which significantly raises the bar for a successful attack.

Binary Packing
Binary packing is a trick that hackers use themselves when infecting a platform with malware. The code being downloaded is encrypted and unpacked at runtime. This makes it hard for endpoint detection to find malware because the malware is encrypted with a different key each time so there are no recognizable patterns in the packed code for endpoint protection systems to detect.

Diversification
A common challenge with application security is that if a hacker can successfully break one instance of an application, then they might be able to create an automated tool that can break any other instance of the application. This is known as the **break once, run everywhere** (BORE) problem.

The obfuscation techniques discussed above can be done using randomization. As a result, it is possible to create instances of code that are different from one another, thwarting break once, run anywhere attacks. This technique is known as **diversification**, and can be a powerful tool in protecting deployments of applications. Diversification can be done across a population of applications, so that conceivably every application instance is different. But it is also a useful technique for diversification of versions of code. For example, if a hacker successfully reverse engineers Version 2.3 of an application, they will have to start all over again once Version 2.4 is released. Updated code is functionally similar but the surface of the code is unique and dramatically different in shape and structure.

Diversification can be an effective tool in mitigating risk associated with break once, run anywhere attacks.

Anti-Tampering
The second major set of features of zShield include various ways to prevent hackers from tampering with your application. This technology is sometimes referred to as Runtime Application Self-Protection (RASP).

Integrity Checking

zShield uses a patented technique for making sure code has not been tampered with. The basic premise is shown in Figure 4. Small pieces of code, called checkers, are inserted into your application. Each checker tests during runtime whether a small segment of the executable has been tampered with. If tampering has been detected, a number of possible actions can be taken including notifying a user, generating a log message, or immediately shutting down the program.The scheme is illustrated in Figure 4. Here checker 1 is doing a checksum of interval 7. If a hacker attempts to modify checker 1, then checkers 3 and 4 will detect that a change has been made.

The executable is completely covered by thousands of overlapping checksum regions. This way, even if a hacker were to find one checker and disable it, there are several other checking parts of that region. And, several checkers checking that first checker, so now the hacker has to find those checkers and disable them. Those checkers, are in turn, checked by several other checkers. Each checker is diversified, so no two checkers look the same. And so effectively the entire set of checkers must be discovered and disabled at once in order to defeat the self-checking mechanism.

The checkers are inserted automatically without requiring any change to the code. Because of our profiling features, checkers can be inserted to have minimal impact on performance.

Anti-Method Swizzling

Method swizzling is a technique used in Objective-C applications, commonly used on Apple platforms, where the executable is modified so that certain method names are mapped to different method implementations. It is frequently used to replace or extend methods in binaries for which the source code is not available. This dynamic program modification is similar to **monkey patching**, a concept supported by other dynamic languages.
Although method swizzling is often used for legitimate purposes, it can also be used to attack an Objective-C application and modify its behavior in an undesirable way. zShield provides a feature that allows the protected application to detect method swizzling and execute a defense action.

iOS Jailbreak Detection

Jailbreaking is a process of gaining root access to an iOS device and overcoming its software and hardware limitations established by Apple. Jailbreaking permits a hacker to alter or replace system applications and settings, run specialized applications that require administrator permissions, and perform other operations that are otherwise inaccessible to a normal user.

Normally, a cracked or modified iOS application can be run only on jailbroken iOS devices. zShield provides security mechanisms that will execute the defense action if a jailbroken device is detected. The objective of the defense action is to prevent piracy and to help ensure that the application is run only on valid iOS devices.

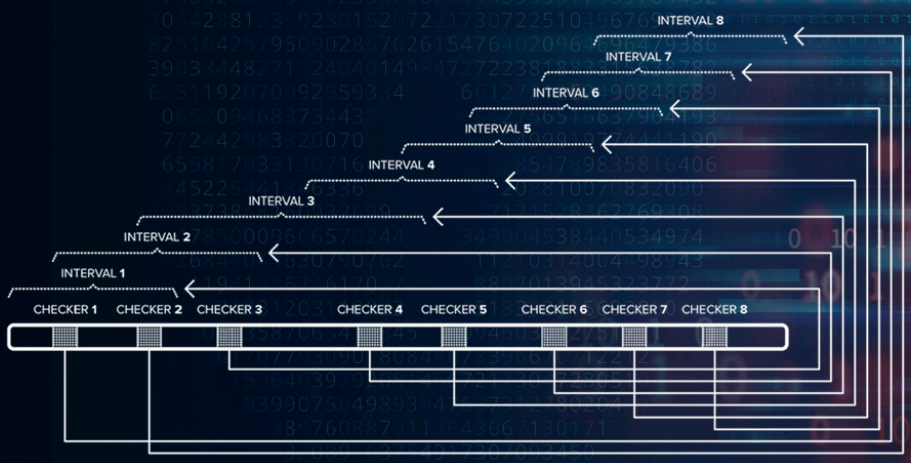


Figure 4: Integrity protection using checksum checkers

Android Rooting Detection
Rooting is the process of allowing users of Android devices to attain privileged control of the operating system with the goal of overcoming limitations that carriers and hardware manufacturers put on the devices. Since users of rooted Android devices have almost complete control over the device and data it stores, a successful rooting of Android is a security risk to applications that deal with sensitive data or enforce certain usage restrictions. zShield provides a specific anti-rooting feature that will execute the defense action if a rooted device is detected.

Function Caller Verification
An executable application file contains a number of functions. Normally, there is a predefined logic how and when these functions are called at run time. However, a skilled hacker can analyze the binary code, find vulnerabilities in the execution logic, and alter the original flow of the program by calling some functions in an unexpected way, for example on Windows, by using DLL injection.

zShield guards against such manipulation of function calls by creating a whitelist of modules (*.dll or *.exe files) that are allowed to call certain sensitive functions of the application code. Signatures of these authorized modules are stored within the application binary and used at run time to verify function caller modules.

Shared Library Cross-Checking
One way hackers can attack an application is by replacing or modifying the shared libraries it uses. zShield provides a feature called cross-checking of shared libraries that renders this type of attack significantly more difficult.
With shared library cross-checking enabled, you can select specific shared library files (*.dll or *.so) from your application, and zShield will calculate cryptographic signatures of their binary code and embed these signatures in the main application during the protection phase. Then, at arbitrary places in the application code, you can invoke a special function that checks if the signature of a particular shared library loaded in the memory matches the previously recorded signature. In other words, this function will check if the loaded shared library has not been modified or replaced.

Mach-O Binary Signature Verification
Every macOS, iOS, and tvOS application distributed via the App Store is signed with Apple's private key. This prevents piracy and unwarranted distribution of the apps. However, any user who is a member of the Developer Program, can re-sign any application with his own private key included in the development certificate, which allows the application to be run on corresponding development devices. Normally, this does not create a significant piracy risk, but there are several services on the Internet that employ re-signing of apps as a means for illegally distributing paid apps for free.

zShield provides a security feature that seeks to prevent unwarranted re-signing and distribution of apps in the Mach-O file format (used by macOS, iOS, and tvOS apps).

Google Play Licensing Protection
Application piracy is one of the primary concerns for Android developers. Although Android provides an anti-piracy library for verifying and enforcing licenses at run-time, this java library can be easily cracked and removed.
zShield provides a security feature specifically for addressing certain piracy vulnerabilities of Android apps. The security feature relies on an alternative implementation of the Google Play license verification library written in native code, which is difficult to reverse engineer and modify.

Customizable Defense Actions

By default, when zShield detects a threat, it corrupts the program state, which results in an application crash. Optionally, you can configure your protected application to execute a custom callback function defined in the source code and choose whether the program state should be corrupted or the application should be left running.

For example, if you are protecting a game, in case of an attack, you may want to secretly corrupt some data (like the game map) instead of crashing the application. In this way, a hacker would think that the game is cracked whereas in reality it would be transformed into an unplayable state.

You can set a different callback function for each of the following threat types:

- code or data tampering
- debugger
- jailbroken iOS device
- rooted Android device
- method swizzling

Select Use Cases

Mobile Payment Systems

Mobile banking is becoming the most important deciding factor when consumers switch banks, with 60% of research respondents citing this over fees (28%), branch location (21%) and services (21%).

The main defense against mobile banking malware starts with mobile app developers who need to adequately understand the risks that proliferate in the mobile data, connections and transactions ecosystem.

When an app is distributed to millions of devices and mobile banking users, it’s not guaranteed that those devices are safe environments even when running security software. This is especially true with the trend toward jailbroken devices. By hardening the app with application shielding during the application development process, the app is **able to bring security with it no matter where it goes**. The solution is application shielding.

Healthcare

The consequences of broken cryptography and unprotected applications are especially problematic in healthcare applications where doctors are moving toward mobile apps to address patient care issues. According to the Robert Wood Johnson Foundation, health apps will be on half of all mobile devices worldwide by 2018; however, personal health data requires higher standards for security and privacy because of the 1996 HIPPA requirements. The higher prevalence of broken cryptography suggests there will be unintended data leakage and other issues as the demand for mobile healthcare applications continues to rise.

Mobile device use is relatively new to the healthcare industry. In 2013, only 8% of doctors used mobile devices to manage patient data. By 2015, the number had grown to 70%, and now an estimated 90% of healthcare providers are using mobile devices in their medical practice. Patients are also using their devices to make and confirm appointments and to access medical records through mobile apps.

The U.S. Department of Health and Human Services reported more than 260 major healthcare breaches in 2015 with 9% of those breaches involving a mobile device other than a laptop. This number is expected to grow substantially in the future.

The majority of FDA-approved apps lack binary protection and have insufficient transport layer protection. Applications should have in-app security measures to protect against threats in the highly distributed mobile environment.

To protect patient data, it is essential to secure APIs that the mobile app uses to communicate with the server. Make sure to hide cryptographic keys within the application, and don't store the keys in memory, as this is a common path to back-end servers.

Develop an app that stores the most sensitive information server-side rather than in the mobile app to reduce liabilities, or if you must store secrets on the device, use whitebox cryptography to ensure their security.

Automotive

Today's car has the computing power of 20 personal computers and features 100 million lines of programming code. The connected car, controlled by software and high-tech features, may be one of the more significant advancements from the past few years. Features such as web browsing, Wi-Fi access points and remote-start mobile phone apps, help to enhance the enjoyment of the vehicle while *adding more opportunities for advanced attacks.*

In real life, we've seen thieves hack keyless entry systems in the UK to steal cars. Software recalls of cars have doubled within the past year and soon they will match mechanical recalls.

Stealing Personally Identifiable Information (PII): Connected cars collect a significant amount of data and interface with multiple after-market devices. Financial information, personal trip information and diagnostics can all be accessed through a vehicle's system. **Protect it!**

Manipulating a Vehicle's Operation: Catastrophic incidents resulting in personal injury and lawsuits may be in the near future. Famously, Charlie Miller and Chris Valasek demonstrated a proof of concept by hacking a Jeep. They now work for GM subsidiary Cruise ensuring customer confidence is strong for that effort.

Unauthorized Vehicle Entry: Car thieves now have a new way to gain entry into locked vehicles. Methods of obtaining entry include intercepting the wireless communication between the vehicle, or intercepting the fob signal from the driver. Many vehicle technologies have opted to replace physical ignition systems with keyless systems using mobile applications or wireless key fobs. In addition to gaining access to the vehicle, flaws in mobile apps have also led to controlling features independently, as discovered when Nissan **had to pull** its NissanConnect EV app for the Nissan Leaf in February 2016. The poor security of the app allowed security researchers to connect to the Leaf via the Internet and remotely turn on the car's heated seating, heated steering wheel, fans and air conditioning.

Media and Entertainment

Global entertainment and media companies have increased their value through innovative global streaming services, programs, live concerts, daily behind-the-scenes interviews, live sports broadcasts and a variety of music and news events that can be viewed on mobile devices. More importantly, consumers can now view specific entertainment content on their own devices just about anywhere, including planes, taxis, and other forms of public transportation.

To protect the content from being stolen, digital rights management (DRM) systems must be in place, and to protect the players' apps themselves, mobile security app solutions are a necessity. Developers should add a layer of protection to prevent hackers from reverse engineering and tampering with the service, content, or connected applications.

Next Steps

Contact us to see how Zimperium can help you shield your apps, get a demo, start a free trial, or to learn more.

https://www.zimperium.com/contact-us/

Sources

1 https://www.wired.com/story/wind-turbine-hack/

2 http://www.visualcapitalist.com/millions-lines-of-code/

About Zimperium

Zimperium secures mobile devices and mobile applications so they can safely access sensitive data and systems. We are an advanced machine learning-based solution with a privacy focus, supporting iOS, Android, and ChromeOS platforms.

Zimperium's Mobile Application Protection Suite (MAPS) helps enterprises to build secure and compliant mobile applications. It is the only unified solution that combines comprehensive in-app protection with centralized threat visibility.

ZIMPERIUM®

Learn more at: zimperium.com
Contact us at: 844.601.6760 | info@zimperium.com

Zimperium, Inc
4055 Valley View, Dallas, TX 75244