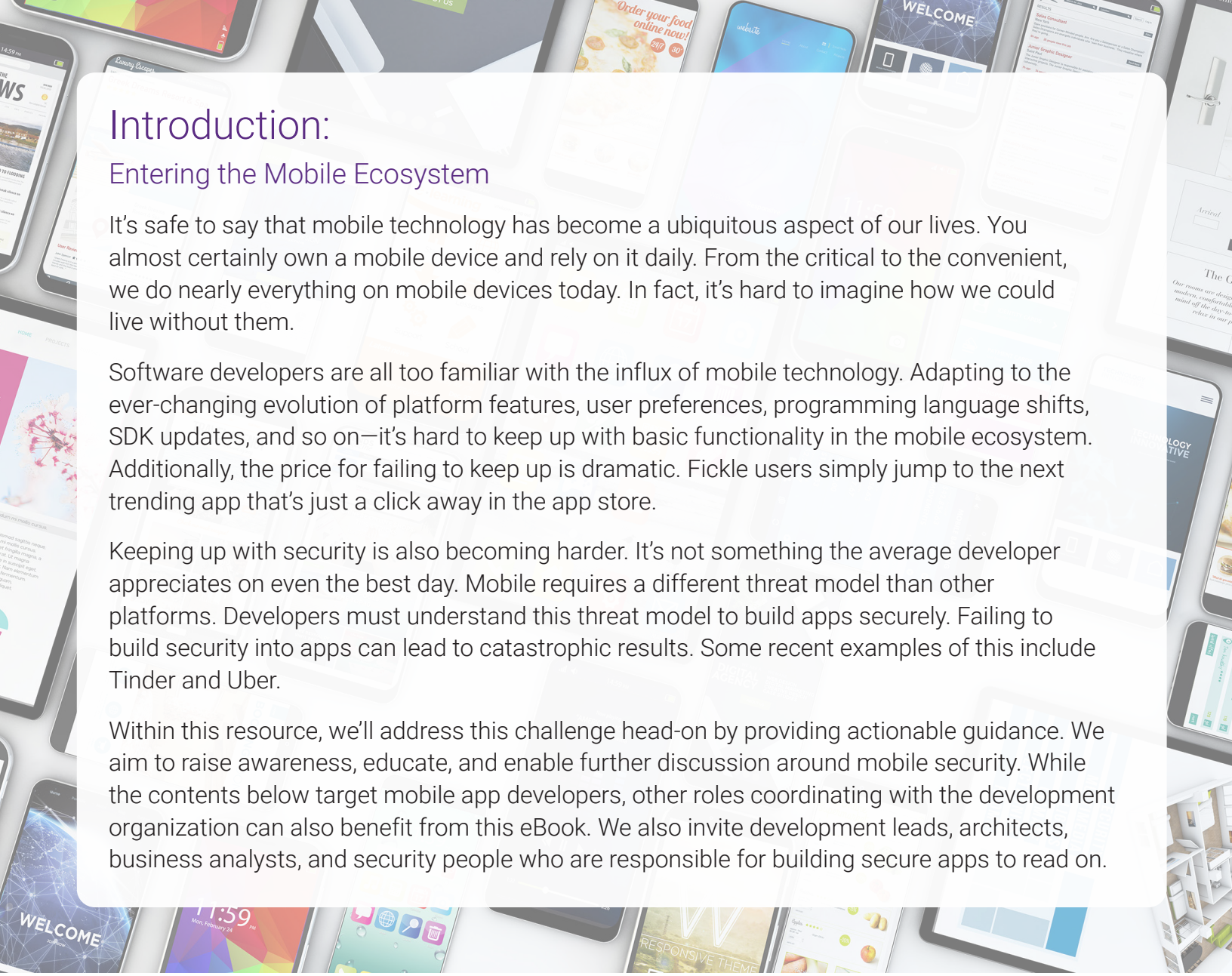
SYNOPSYS®

The Developer's Guide to Securing Mobile Applications

Part One: Threat Modeling

This eBook, part one in a series, focuses on threat modeling mobile applications. Subsequent eBooks within the series will cover mobile secure coding guidance and integrating security into the mobile development life cycle.

Table of Contents



Introduction:

Entering the Mobile Ecosystem

It's safe to say that mobile technology has become a ubiquitous aspect of our lives. You almost certainly own a mobile device and rely on it daily. From the critical to the convenient, we do nearly everything on mobile devices today. In fact, it's hard to imagine how we could live without them.

Software developers are all too familiar with the influx of mobile technology. Adapting to the ever-changing evolution of platform features, user preferences, programming language shifts, SDK updates, and so on—it's hard to keep up with basic functionality in the mobile ecosystem. Additionally, the price for failing to keep up is dramatic. Fickle users simply jump to the next trending app that's just a click away in the app store.

Keeping up with security is also becoming harder. It's not something the average developer appreciates on even the best day. Mobile requires a different threat model than other platforms. Developers must understand this threat model to build apps securely. Failing to build security into apps can lead to catastrophic results. Some recent examples of this include Tinder and Uber.

Within this resource, we'll address this challenge head-on by providing actionable guidance. We aim to raise awareness, educate, and enable further discussion around mobile security. While the contents below target mobile app developers, other roles coordinating with the development organization can also benefit from this eBook. We also invite development leads, architects, business analysts, and security people who are responsible for building secure apps to read on.

Mobile Threat Modeling

Threat modeling is a pen-and-paper exercise that identifies potential security risks in applications. This essential step in mobile application security helps identify security concerns including:

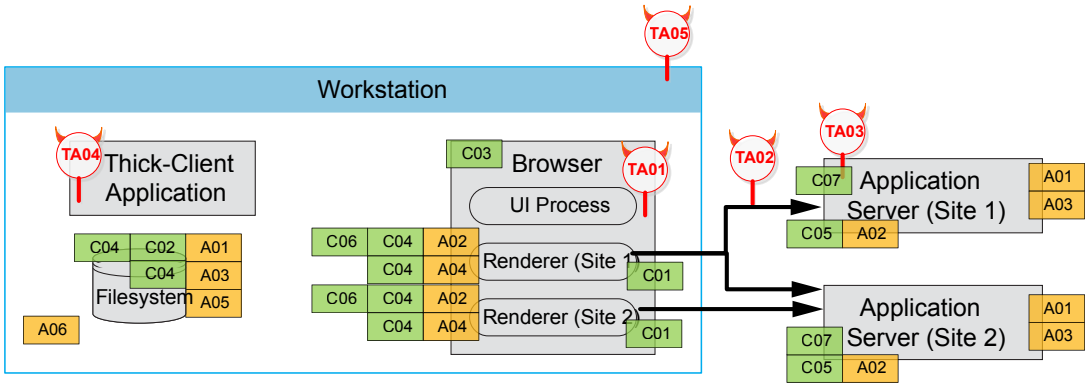
- Assets requiring protection by an application
- Security controls provided by the technologies in use
- Controls that the application needs to implement itself
- Threat agents that may attempt to attack the application

Threat modeling promotes the idea of thinking like an attacker. It models security risk by documenting the existence of—and relationships between—key components of risk. For developers, threat models identify controls requiring implementation. For security teams, threat models identify areas requiring testing. Without threat modeling, security activities turn into an endless and aimless bug squashing activity without a risk-based understanding of priority and impact.

Without threat modeling, security activities turn into an endless and aimless bug squashing activity without a risk-based understanding of priority and impact.

Web Applications

For many years, development teams have been working with Web applications that often have a common threat model. All Web applications have similar interfaces (e.g., protocols, languages, etc.). Many of the assets, controls, threat agents, and attack vectors don't change significantly between Web applications. Figure 1 presents a sample Web application threat model diagram.



Assets
A01: Application Data
A02: Application Functionality
A03: Cookies
A04: DOM Objects
A05: Users' Private Data
A06: User Workstation Functionality

Controls
C01: TLS / Certificate Verification / User Cues
C02: Cache Directives
C03: Browser Process Sandboxing
C04: Same Origin Policy
C05: User Authentication / Authorization
C06: JavaScript Obfuscation
C07: Input Validation / Output Encoding

Threat Agents
TA01: Malicious App Users
TA02: Network Attackers
TA03: Malicious Websites
TA04: Malicious Thick-Client Software
TA05: User with physical access to workstation

Figure 1. Threat model diagram for Web applications.

Table 1 presents the controls in place protecting assets from threat agents. The only controls included here are those that Web application developers can expect to be in place universally, or controls that the Web application developers can implement themselves.

Controls including filesystem encryption and access controls on end user workstations may be present in some cases. However, developers cannot expect them to be in place on all workstations. That is, unless they're creating Intranet applications for an organization in which the IT department adds these controls to all workstations.

Controls such as certificate pinning aren't present in some of the major browsers. At the time of this writing, these include Internet Explorer, Edge, and Safari. Therefore, Web application developers can't rely on these controls.

Threat agent	Attack surface	Target assets	Controls
Malicious app users	Browser	• Application functionality	• Javascript obfuscation
Malicious app users	Browser	• Application data	
Malicious app users	Application server	• Application data • Application functionality	• User authentication / authorization
Network attackers	Lan	• Application data • Application functionality • Cookies	• Tls / certificate verification / user cues
Malicious websites	Browser	• Application data • Application functionality • Cookies • Dom objects	• Same origin policy • Input validation / output encoding
Malicious websites	Browser	• Users' private data • User workstation functionality	• Browser process sandboxing
Malicious thick-client software	User workstation	• Application data • Application functionality • Cookies • Users' private data • User workstation functionality	
User with physical access to workstation	User workstation	• Application data • Application functionality • Cookies • Users' private data • User workstation functionality	• Cache directives

Table 1. Threat matrix showing mappings between common threat model entities for Web applications.

Mobile operating systems implement controls like sandboxing and permission enforcement to limit malicious application access

Web Applications on Mobile Devices

Web applications are accessible from mobile devices using mobile browsers. When accessing Web applications in this manner, some controls and threat agents are different.

Figure 2 presents a threat model diagram of these differences.

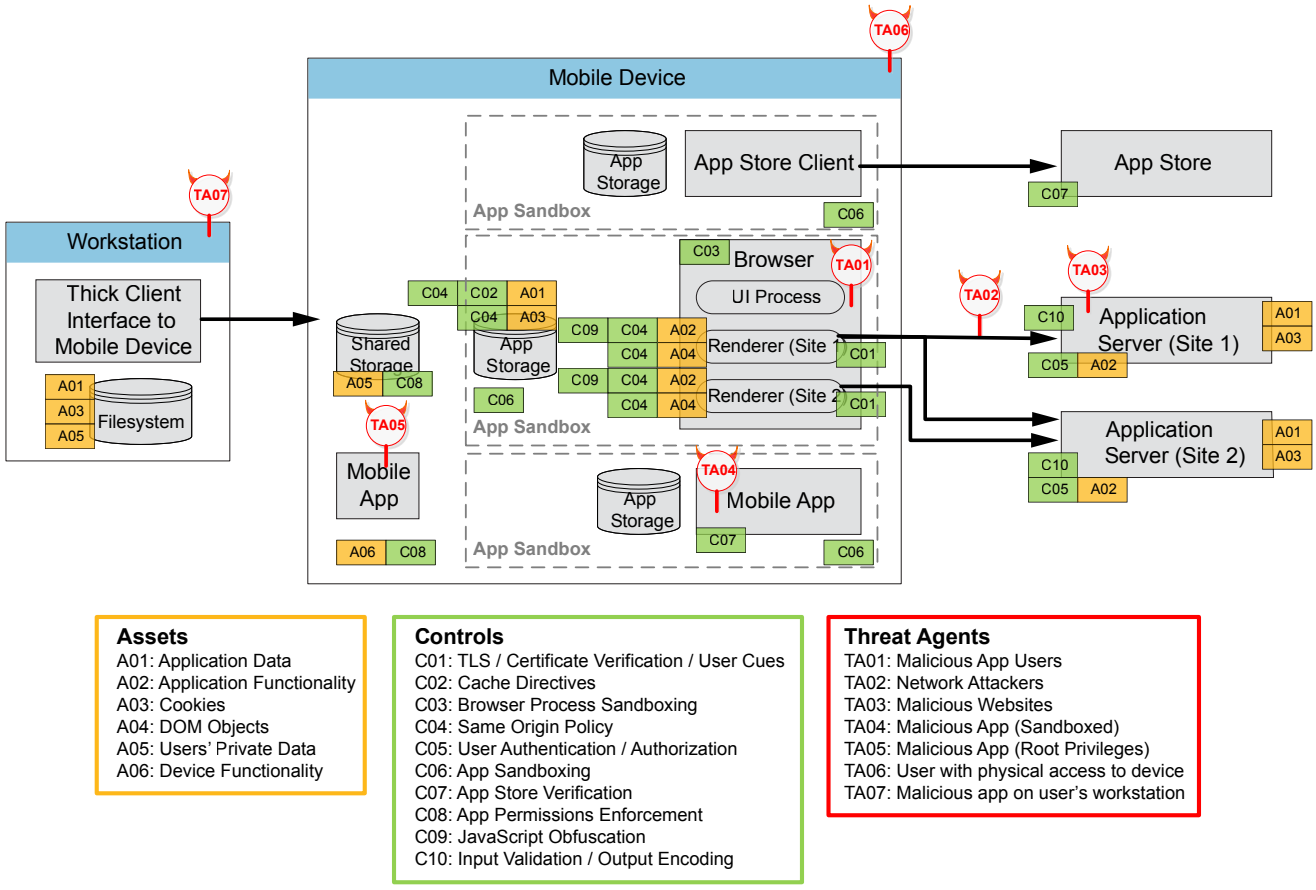


Figure 2. Threat model diagram for mobile Web applications.

Table 2 presents the concerns that vary from the Web application threat model in Table 1.

Threat Agent	Attack Surface	Target Assets	Controls
Malicious app users	Browser	• Users' private data • Device functionality	• Browser process sandboxing • App sandboxing • App permissions enforcement
Malicious app (sandboxed)	Mobile device	• Application data • Application functionality • Cookies • Users' private data • Device functionality	• App sandboxing • App store verification • App permissions enforcement
Malicious app (root privileges)	Mobile device	• Application data • Application functionality • Cookies • Users' private data • Device functionality	
Malicious app on user's workstation	User workstation (device backups)	• Application data • Cookies • Users' private data	

The additional controls that are present reduce the likelihood of malicious apps accessing other applications' data. This includes application data stored on the device, in addition to end user data and device functionality. Authors of mobile operating systems learned from the mistakes made years ago by authors of desktop operating systems. For example, if a user installs a malicious thick client application on a desktop operating system, there is a chance that it can allow full access to a user's workstation. This is especially true if the end user is an administrator on the workstation.

Mobile operating systems implement controls like sandboxing and permission enforcement to limit malicious application access. Most applications don't have administrative privileges on the device. Some mobile platforms only allow applications approved by the platform vendor for installation on devices. However, these controls are ineffective against malicious apps running with root privileges. For this reason, rooting/jailbreaking devices is discouraged for users who don't have a deep security background, and are therefore unable to identify potentially malicious applications.

Many mobile devices allow users to back up device data to their workstations. These workstations often run traditional desktop operating systems where these controls are not present. Thus, the concerns from the desktop environment don't completely disappear in the mobile ecosystem.

Native Mobile Applications

A native application is an application written for a mobile device using the SDK provided by the device's operating system vendor. As of this writing, these are applications written in Java (optionally, with some components written in C/C++) for Android, and applications written in Objective-C, Swift, and/or C/C++ for iOS.

Figure 3 presents a threat model diagram for native mobile applications.

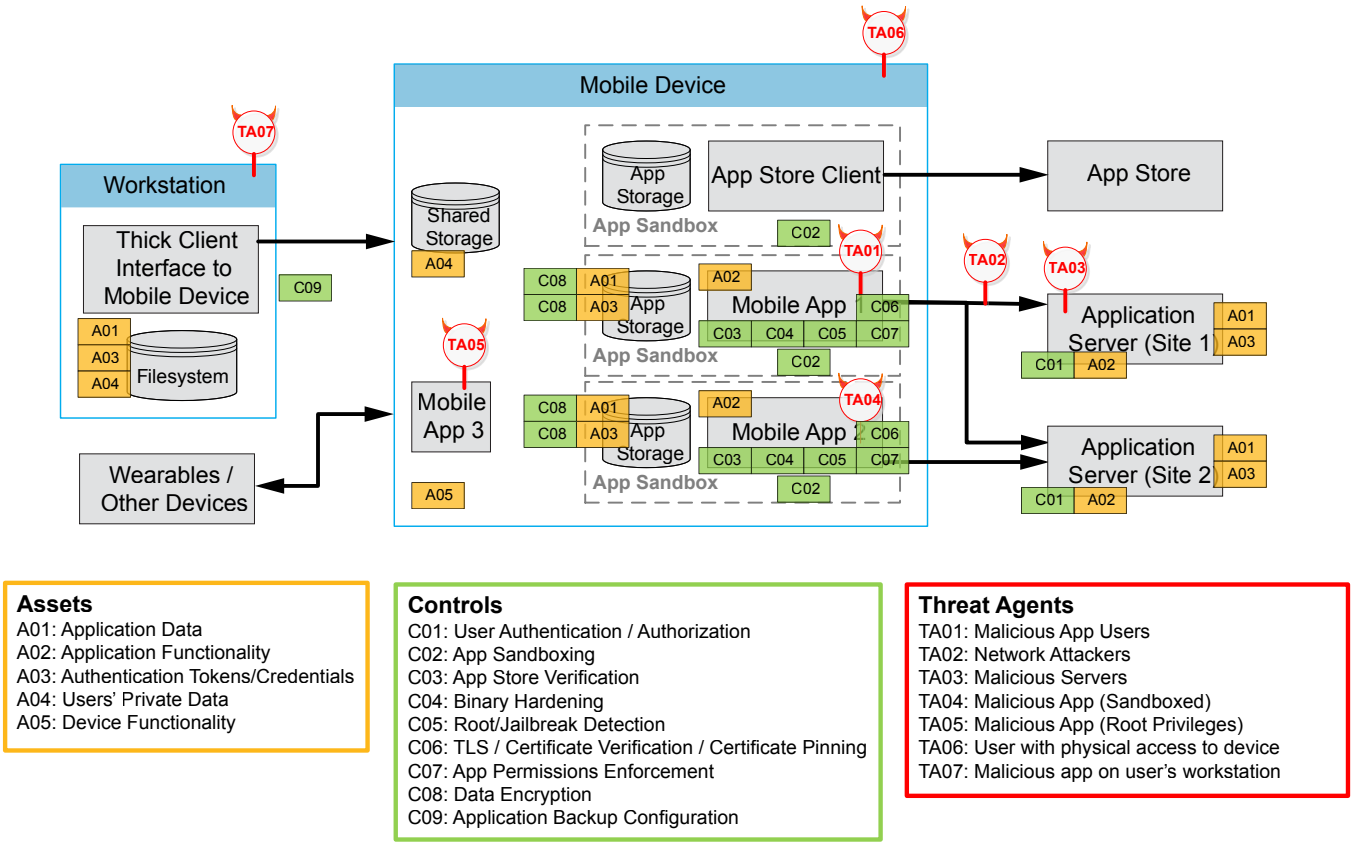


Figure 3. Threat model diagram for native mobile applications.

The same origin policy is an important control for Web applications. However, it's irrelevant for native mobile apps.

Table 3 presents the concerns that vary from the Web application threat model in Table 1. Note that platform-specific controls (e.g., iOS keychain) and attacks (e.g., Android applications executing native library functionalities of other applications) aren't discussed here.

Threat Agent	Attack Surface	Target Assets	Controls
Malicious App Users	Mobile Device	• Application Functionality	• Binary Hardening
Malicious App Users	Mobile Device	• Application Data	• Root/Jailbreak Detection • Data Encryption
Network Attackers	LAN	• Application Data • Application Functionality • Authentication Tokens / Credentials	• TLS / Certificate Verification / Certificate Pinning
Malicious App (Root Privileges)	Mobile Device	• Application Data • Application Functionality • Authentication Tokens / Credentials	• Root/Jailbreak Detection • Data Encryption
User with Physical Access to Device	Mobile Device	• Application Data • Application Functionality • Authentication Tokens / Credentials	• Data Encryption
Malicious App on User's Workstation	User Workstation (Device Backups)	• Application Data • Authentication Tokens / Credentials	• Data Encryption • Application Backup Configuration

Table 3. Threat matrix showing new mappings between common threat model entities for native mobile applications.

The code in native mobile applications interacts with the mobile operating system, rather than the JavaScript runtime in a browser. As a result, some controls provided by browsers are no longer available. This architecture enables application developers to use more effective controls than are possible in Web applications. Unfortunately, developers can also disable security controls that Web applications cannot disable.

Certificate verification for TLS connections is different in native mobile applications than Web applications. In Web applications, browsers often perform certificate verification. (In several cases, depending on the browser/OS combination, the browser offloads certificate verification to an operating system component or library.) Visual cues are also presented to users depending on whether a given site is trusted or not.

When it comes to Web application certificate verification modification, HTTP Strict Transport Security (HSTS) and HTTP Public Key Pinning (HPKP) allow developers to modify browser behavior regarding connections and/or pinning for their own website. However, support and adoption varies. These technologies don't fully work in mobile (yet).

The code in native mobile applications interacts with the mobile operating system, rather than the JavaScript runtime in a browser

Native mobile apps have a variety of options to modify certificate verification:

- Use the default certificate verification functionality provided by the platform.
- Perform certificate pinning.
- Use the certificate verification functionality provided by an arbitrary cryptographic library included with the application.
- Create application-specific certificate verification functionality.

The application's choice is hidden from the application user. The user may only see an error if the certificate verification functionality rejects a certificate.

App stores perform some of the same functions for native mobile apps that certificate authorities perform for Web applications. Instead of expecting a user to trust a Web application because a certificate authority verifies that it belongs to a particular organization, the user trusts a native app because the app store verifies that it belongs to a particular organization. Unlike certificate authorities that follow certain minimal standards, some app stores do little or no verification of an app publisher's true identity. This is a common problem with Android app stores.

The same origin policy is an important control for Web applications. However, it's irrelevant for native mobile apps. This is because each server-side application has a separate native client-side application. Application sandboxing for native mobile apps provides some of the same protections that the same origin policy provides for Web applications. It's important to note that these two protections are not equivalent. For example, the same origin policy prevents websites from being able to read responses from other origins. This is the case unless the other origins expose cross-origin communication mechanisms.

On the other hand, application sandboxing for native mobile apps doesn't prevent them from reading responses from other origins. There is no concept of origin in native mobile apps. The only readable content from other origins is content available to unauthenticated users—unless a malicious native mobile app obtains the user's credentials or cookies for other sites. This ability to read unauthenticated content from arbitrary sites is one of the reasons why many organizations don't allow mobile devices on their internal networks.

Web applications can't verify the environment that they are running in because they don't have sufficient access to the underlying operating system. Therefore, they can't attempt to detect clients that have malicious software installed. Native mobile apps are generally sandboxed and protected from other malicious applications that aren't running with root privileges. They can also perform checks to identify whether the device they are running on has been rooted or jailbroken.

If client-side functionality requires protection from reverse engineering, Web applications are limited to JavaScript obfuscation. However, native mobile apps have a variety of advanced controls available to them. These include obfuscation, anti-debug, and anti-tamper.

Native mobile applications also have more options available to them for encrypting locally-stored data. They can also configure backup options to prevent sensitive data from being copied to user workstations where fewer protections may be available.

Hybrid Mobile Applications

A hybrid mobile application is a combination of a native mobile app and a mobile Web application. Some components are built as a Web application. These components display in one or more WebViews inside the rest of the application that is built as a native mobile app.

Figure 4 presents a threat model diagram for hybrid mobile applications.

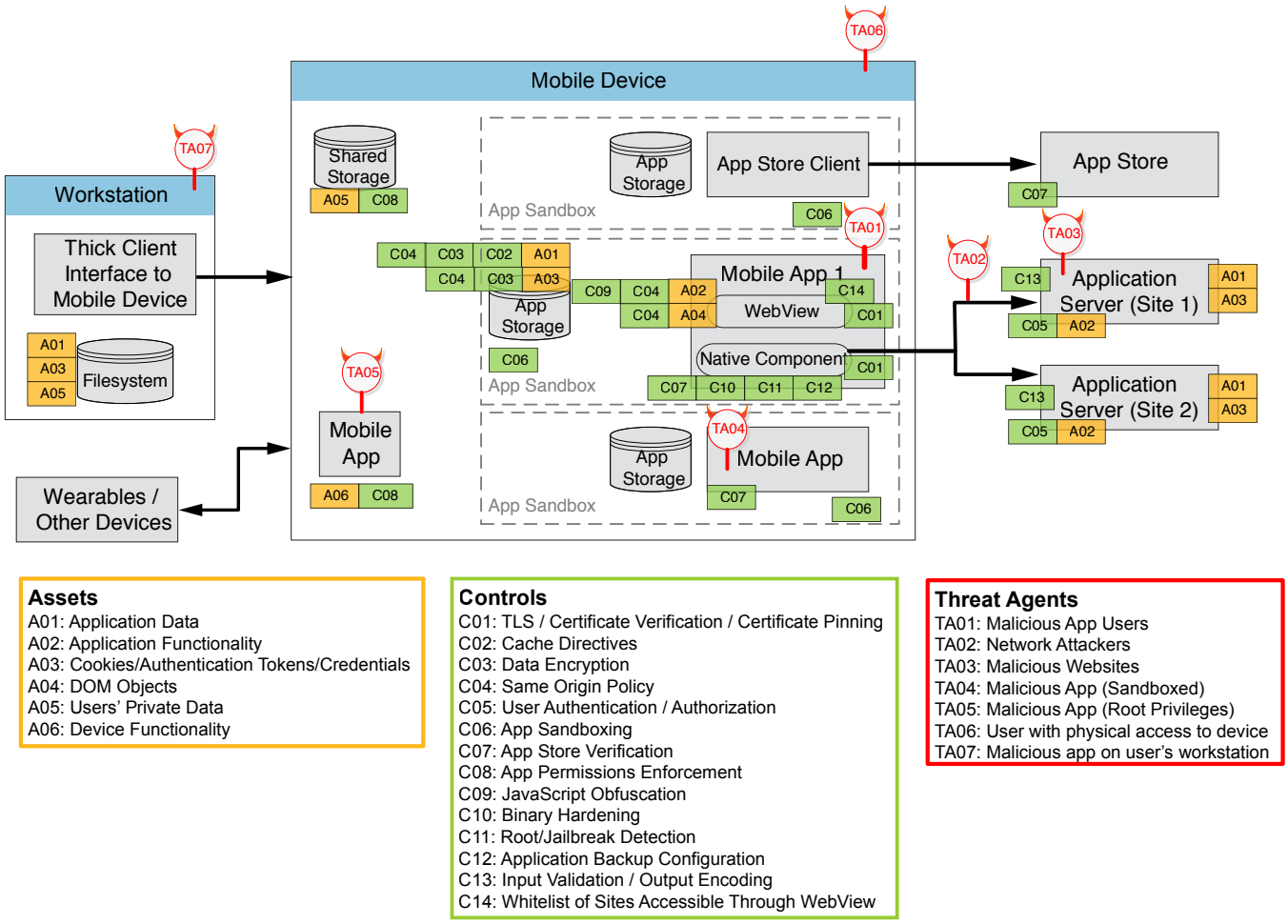


Figure 4. Threat model diagram for a hybrid mobile application.

Since hybrid applications contain aspects of both native mobile apps and Web applications, the concerns and controls from both threat models apply

Table 4 presents the concepts that are different from all previous threat models.

Threat Agent	Attack Surface	Target Assets	Controls
Network Attackers	LAN	• Users' Private Data • Device Functionality	• TLS / Certificate Verification / Certificate Pinning
Malicious Websites	Mobile App	• Application Data • Cookies/Authentication Tokens/Credentials	• Whitelist of Sites Accessible Through WebView • Input Validation / Output Encoding
Malicious Websites	Mobile App	• Users' Private Data • Device Functionality	• App Sandboxing • App Permissions Enforcement

Table 4. Threat matrix showing mappings between common threat model entities for hybrid mobile apps.

Since hybrid applications contain aspects of both native mobile apps and Web applications, the concerns and controls from both threat models apply. The threat model for native mobile apps applies to native components. Meanwhile, the threat model for Web applications (mostly) applies to the components in the WebViews. However, there are some differences.

- **Some standard browser-provided security controls aren't present.** This includes the standard user cues corresponding to certificate verification status for TLS connections. Just like native applications, hybrid applications can often utilize certificate pinning.
- **Although the same origin policy is present, it's only applicable to components in WebViews.** Many hybrid applications contain a mix of locally-stored HTML/JavaScript content in the application sandbox on the mobile device, and HTML/JavaScript content stored on a server. This complicates the same origin policy. Vulnerabilities have been discovered in the past that are a result of locally-stored files that weren't subjected to the required same origin policy restrictions.
- **The impact of unauthorized JavaScript code running in a hybrid application can be greater than unauthorized JavaScript code running in a browser.** Hybrid applications often contain native components with which JavaScript code can interact. For example, JavaScript code in hybrid applications can often access the application's locally-stored data. In these cases, the same origin policy isn't enough to protect the application's data and cookies, authentication tokens, and credentials from malicious websites. Thus, applications need to implement a whitelisting mechanism to ensure that only authorized websites can be opened in its WebViews.

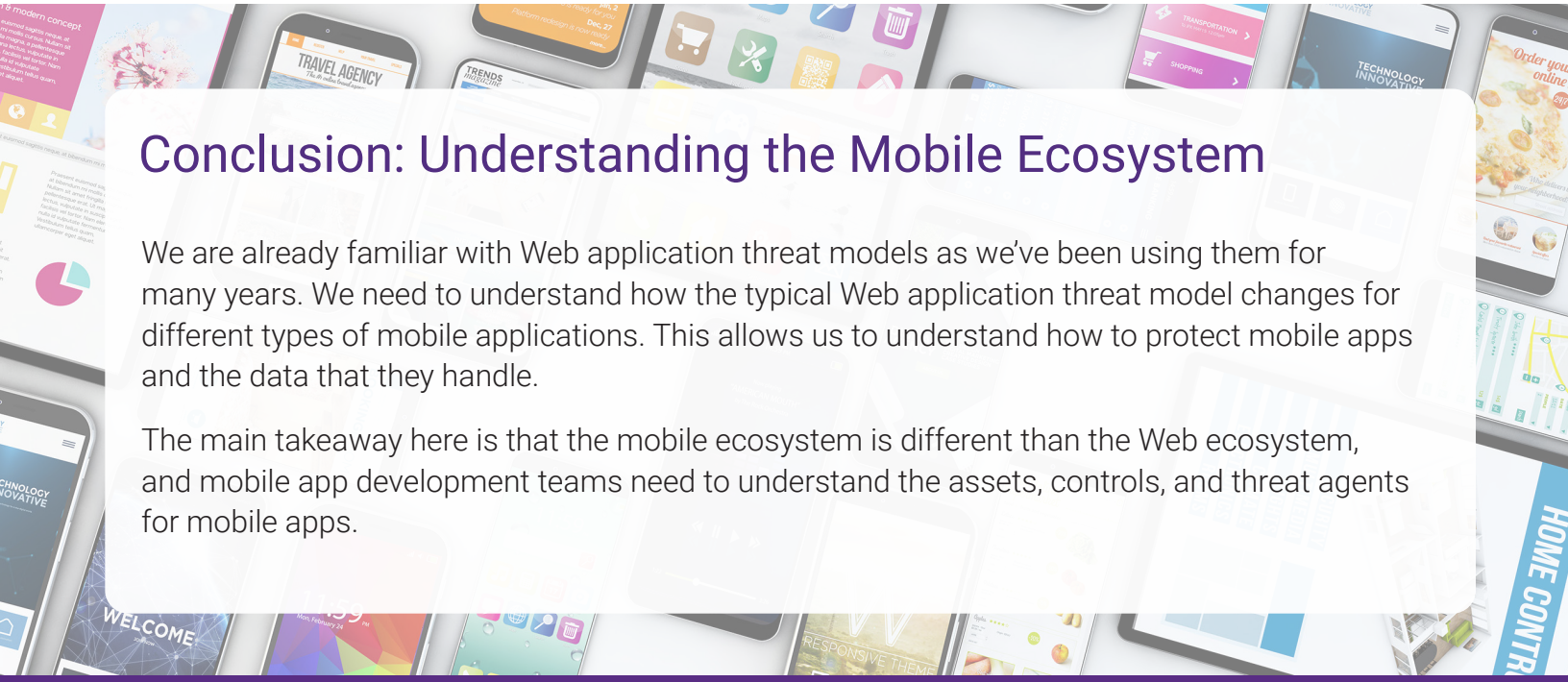
Cross-Platform Mobile Applications

Most Web browsers are similar enough that Web applications can be written once and accessed from any browser. However, when writing mobile apps using native platform APIs, organizations face a challenge. They need to create a different mobile app for each platform (e.g., Android, iOS, etc.) that they want to support. The increase in development costs is significant. Additionally, keeping applications on different platforms in sync, and hiring developers with the right skills, are difficult tasks. As a result, many cross-platform development frameworks have become popular. These frameworks allow organizations to write a single mobile app that will run on all platforms.

The cross-platform frameworks take care of most platform-specific details. This is similar to how Web browsers take care of most platform-specific details of the underlying operating system.

There is no single threat model for cross-platform mobile apps as each framework is different. Three examples of this include:

- Apache Cordova creates hybrid mobile apps in which most application code is HTML/JavaScript. This code accesses device functionality through plugins written in native platform APIs. The application code renders in a WebView. The threat model for hybrid applications applies here.
- React Native creates mobile apps in which most application code is JavaScript. A JavaScript engine interprets this code and calls native modules that directly call platform-specific APIs. (WebViews don't need the same involvement as they do in Apache Cordova.) The threat model for native mobile apps mostly applies here. However, the threat model for hybrid applications may apply if using WebViews in the application. Consider JavaScript-specific issues in React Native applications even if foregoing WebViews.
- Xamarin creates mobile apps in which application code is C#. The runtime is different on Android and iOS. On Android, the code compiles to an intermediate language that is interpreted using a Mono VM implementation for Android. On iOS, the application code, along with Xamarin framework code, compiles to native code that runs like a native mobile app on iOS. Regardless of the platform-specific implementation details, the threat model for native mobile apps applies here. However, the threat model for hybrid applications may apply if the application is using WebViews.



Conclusion: Understanding the Mobile Ecosystem

We are already familiar with Web application threat models as we've been using them for many years. We need to understand how the typical Web application threat model changes for different types of mobile applications. This allows us to understand how to protect mobile apps and the data that they handle.

The main takeaway here is that the mobile ecosystem is different than the Web ecosystem, and mobile app development teams need to understand the assets, controls, and threat agents for mobile apps.

As you travel further into the mobile ecosystem, ensure you're doing all you can to protect your applications.

Learn more

Expert contributors:
Amit Sethi, Neil Bergman, John Kozyrakis, Corey Gagnon, Joel Scambray

The Synopsys difference

Synopsys helps development teams build secure, high-quality software, minimizing risks while maximizing speed and productivity. Synopsys, a recognized leader in static analysis, software composition analysis, and application security testing, is uniquely positioned to apply best practices across proprietary code, open source, and the runtime environment. With a combination of industry-leading tools, services, and expertise, only Synopsys helps organizations maximize security and quality in DevSecOps and throughout the software development life cycle.

For more information go to www.synopsys.com/software.

Synopsys, Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com

SYNOPSYS®
Silicon to Software™