approov

Mobile Threat Lab

Mobile App Security Report

Exposing the Security Vulnerabilities of Top Finance Apps

Contents

The Q1 2023 Report

Welcome to the Approov Mobile Threat Lab Security Report. This quarter, we look at the State of Finance Apps and expose their not very well hidden secrets. We downloaded over 650 of the most popular Banking, Financial Services, and Crypto apps from the US, UK, France and Germany. We examined the applications to see if secrets like API keys and other sensitive information could be located using popular open-source tools.

Adoption of mobile technology has been rapid for financial use cases including banking, money transfers and crypto-currency. According to Google, 73% of smartphone users have used a financial app. The average smartphone user has 2.5 financial apps installed per device, and this number is growing rapidly.

Financial apps have access to valuable and sensitive personal data, therefore security is paramount to protecting end-users. We focused on the top 200 financial apps in each of the four countries, as this was deemed a representative sample. However it is worth noting that there are over 100,000 financial apps in existence and this number is also growing.

Despite many of the claims about improved cybersecurity awareness and better development and testing tools, most apps still expose secrets in their code. As we will show, this is still a major security issue, as hackers can easily exploit known vulnerabilities. We went beyond a simple static analysis to investigate what we could uncover about how well applications are protected at run-time by using an automated inspection of the application packages.

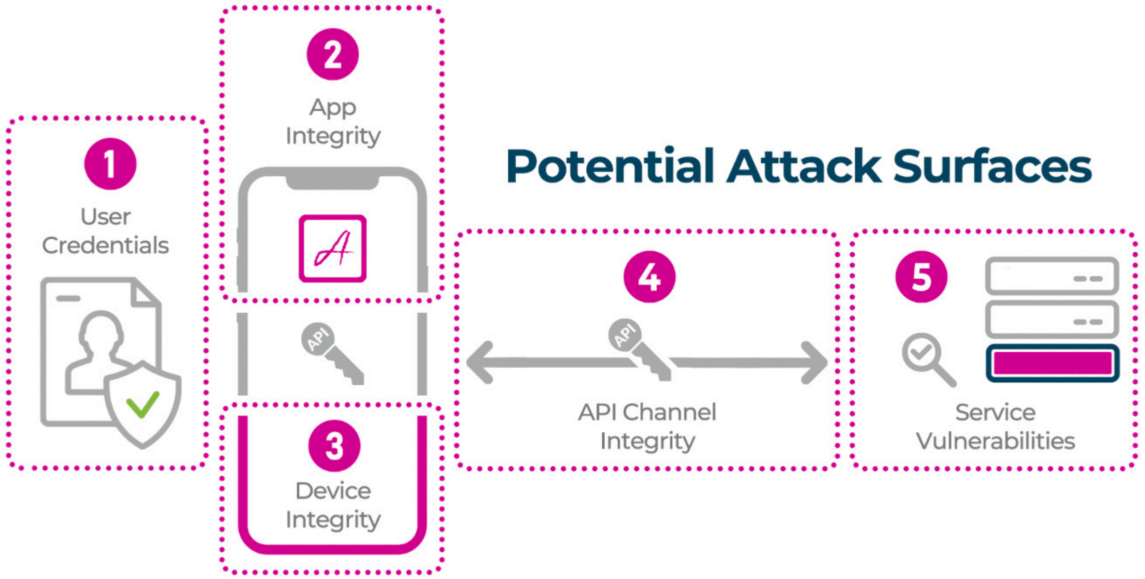
Hackers like to manage and prioritize their efforts by looking for obvious vulnerabilities. Once an app is released, you can be sure that it is being scanned and inspected by a variety of hacking groups using a multitude of inspection tools. Many apps are so poorly protected that they immediately divulge their secrets. Some apps may initially seem harder to crack, however we will show that clues can be gathered from the package about where the vulnerabilities lie. Skilled hackers rely on these clues and run-time techniques that often prove effective in giving them a priority list of the most exposed apps.

Future Threat Lab reports will look at additional geographic regions, and other classifications of mobile apps. Our goal is to shed light on mobile security issues across industry verticals, covering specific threat surfaces and risks. These reports will highlight security issues that make it easy for enterprises to benchmark their security against competitors, compare security best-practices, and ultimately improve their security to protect the data of their valued clients.

It's as if potential thieves were driving through a neighborhood. Some houses have alarms on the wall or signs that show they are well protected. A potential thief also sees houses that have no cars in the driveway, and no sign of anyone home, inviting them to walk in and help themselves. Does this sound unrealistic? That's the message a significant number of the top financial apps are communicating to potential attackers. They will be the targets for the hackers attention.

Mobile App Security and API Keys

Secure mobile app operation is built on the requirement that only legitimate users without malicious intentions are accessing the service(s). Safety measures should ensure that they are using a genuine version of a mobile app, running on an uncompromised device, communicating directly with the API server via a secure channel and the API cannot be accessed by any other way. These are the "attack surfaces" targeted by bad actors.



A mobile app is a critical element of the overall application, accessing back-end elements to conveniently deliver the services. Both the mobile app code and the network environment it runs on are exposed and can be manipulated. Attacks can directly target mobile apps to steal data or can directly attack APIs using secrets acquired from mobile apps or elsewhere. Both types of vulnerabilities are commonly exploited. See below for a detailed description of five common mobile app attack surfaces.

API keys are commonly used to secure access to backend data based in the cloud. Typically, when apps use a third-party API, they sign up and get allocated a key. This identifies the app to the backend API and authenticates the app that's making the call so the API backend knows which particular app is making the request. This blocks any anonymous traffic and can be used to limit data requests.

Key Findings

Only 8% of 650 apps scanned did not immediately reveal valuable secrets

23% of the apps immediately revealed high-value secrets

Only 4% of the apps implemented TLS certificate pinning

20% of the apps were vulnerable to Man-in-the-Device attacks

Only 4 apps were protected at runtime against both MitM by using pinning and against Man-in-the-Device manipulation by using a packer or protector

It is sometimes argued that if valid user authentication tokens are required by the API then API keys don't actually need to be kept secret since they only identify the app and are secondary to user authentication. However, if obtaining a user login is easy, anyone, including attackers, can sign up and use an app and access APIs using stolen keys.

The presentation of the API key is therefore sufficient to gain access to the API, so API keys should be kept secret. If the API key becomes known, the mobile app can be impersonated and the API calls can be made by other parties or bots. What can go wrong if API keys are exposed? The risk depends on the capabilities of the particular API whose key has been compromised. Recent examples show major data breaches via scripts using stolen API keys.

Hackers systematically acquire keys and secrets and use them to access backend systems. Recent high-profile cases are due to secrets exposed in public repositories such as github. As we show, if secrets end up in mobile app code, they can easily be extracted and exploited. This can be done in an automated way at scale without ever needing to launch the actual app.

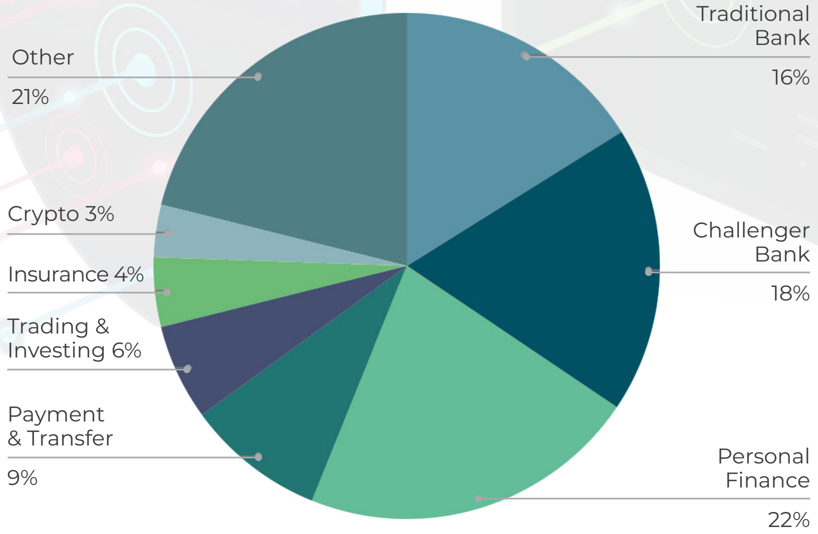
Even if keys and secrets cannot be easily reverse engineered from the mobile app code, hackers can get another opportunity to grab secrets at runtime by manipulating the app, the environment and/or the communication channel(s).

There are two main ways to steal secrets from a mobile app. The first is static analysis, inspecting the source code and other components of the mobile app package for exposed secrets prior to runtime. The second is during runtime, when the app is compromised at execution, by instrumenting the application, modifying the environment, or intercepting messages from the app to the backend via Man-in-the-Middle (MitM) attacks.

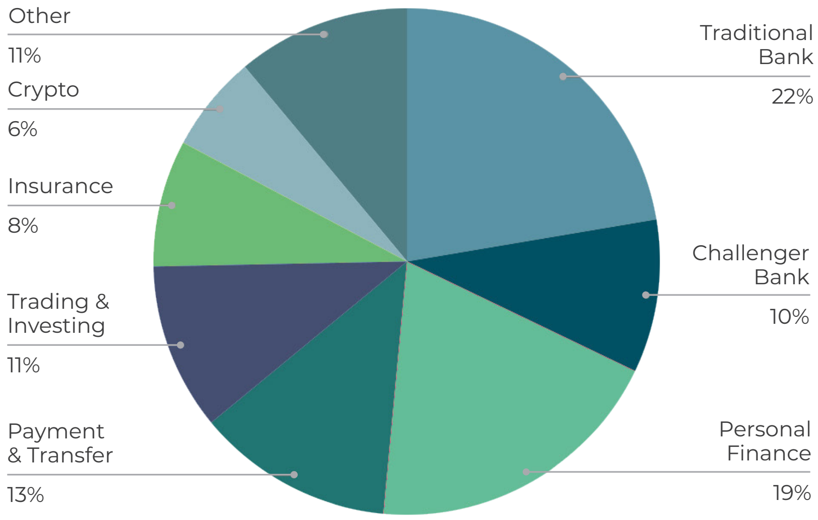
Obfuscation or code hardening, provides protection against static reverse engineering. It provides some protection against bad actors using simple tools to statically reverse-engineer keys out of the app, possibly deterring hobbyists and less serious attackers. More sophisticated techniques can render obfuscation ineffective. Obfuscation does not prevent runtime manipulation of the app or device. Other techniques must be in place to prevent app cloning or jailbreak/rooting, tampering or client environment hooking.

Apps Researched by Region

USA
170 apps



Europe
480 apps



Most importantly, obfuscation does not protect against Man-in-the-Middle extraction attacks where attackers manipulate traffic to and from the backend to steal data or keys. This is a popular and rapidly growing mechanism, as it is effective even if the user has implemented two-factor authentication (2FA).

The Methodology

We used Sensortower to identify the top 200 financial services apps for each of the four countries. We downloaded each App APK from the Google Play Store. First the APKs were decoded using the apk_api_key_extractor. This decodes the APK with the bundled apktool and then extracts the secrets and classifies them with a neural network. Lastly, we ran the Gitleaks and Trufflehog scanners against the folder containing all the decoded APKs.

After the scanners completed the extraction of secrets and information, we ran a custom python script to process the findings of each scanner, using a combination of whitelists and blacklists to eliminate false positives and classify the secrets found.

Secrets found were categorized into three levels (High, Medium and Low). Low Value Secrets are not considered “service-impacting” e.g. API keys associated with crash or installation analytics. High Value Secrets are those we consider extremely dangerous if exposed. Some examples: private keys, keys for payment or transfer services and keys that included “authentication” or “attestation”.

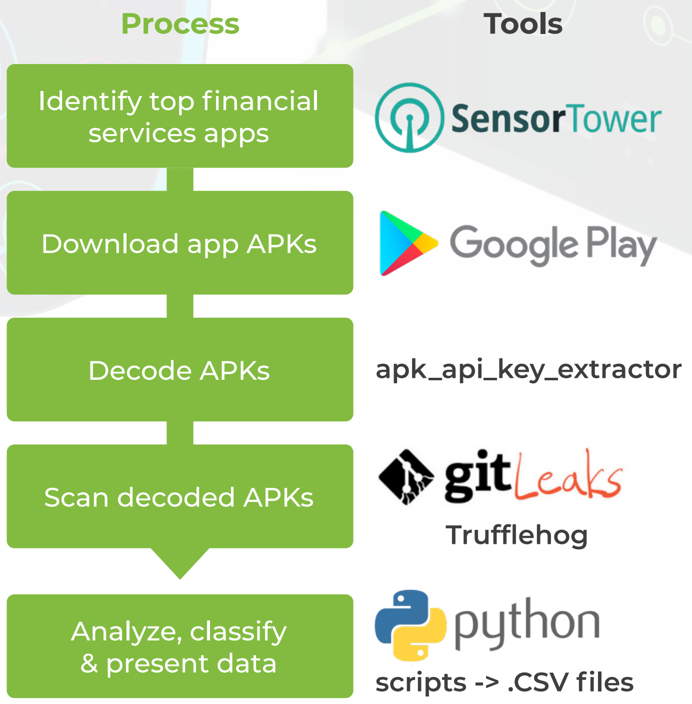
Takeways from the Report

Overview

For this study, we manually classified apps according to their business sub-sector: Traditional Bank, Challenger Bank, Trading and Investment, Payment and Transfer, Cryptocurrency App, Personal Finance Management and Other. This classification gives a means of looking at trends between different kinds of financial apps.

As far as the hackers are concerned, all Banking, Finance and Crypto apps all have valuable and attractive data, and the majority require APIs to other backend systems and services to access that pool of valuable data. We used fully automated static analysis techniques to extract sensitive information directly from the app packages and did not perform an analysis on the running apps. A static analysis can provide valuable insights into how well-protected applications can be at run-time.

Threat Lab App Analysis Process



Protecting Secrets at Rest: Code Protection

We looked at the number and type of API keys and secrets we could extract from the code using static analysis and assessed whether any obfuscation techniques were being applied. This told us what a hacker could uncover rapidly using limited effort from a mobile app package.

Protecting Secrets in Transit: Preventing Man-in-the-Middle

Our analysis allowed us to assess how well defended each app was against Man-in-the-Middle attacks at run-time, another way secrets can be stolen even when traffic is encrypted.

Device Integrity: Preventing Man-in-the-Device

We were able to discover from a scan of the app packages whether apps implemented any checks on the integrity of the run-time environment at run-time to check if hackers had installed frameworks and tools in the client that could steal data or change the operation of the app. From our static scan, we could tell if such runtime checks were in place.

Findings

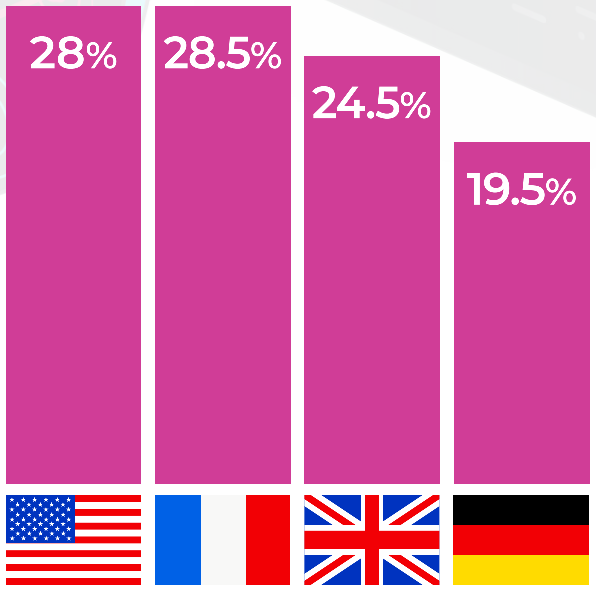
Secrets Exposed by Static Scan

- Only 8% of 650 apps scanned did not immediately reveal valuable secrets.
- 23% of the apps immediately revealed high-value secrets.
- Statistically, the use of obfuscation techniques had little impact on protecting secret. The average number of exposed secrets were the same in obfuscated and non-obfuscated apps.

Runtime Protection

- More secrets can be acquired at runtime via Man-in-the-Middle (MitM) attacks. A more advanced automated scan of the app package indicates which apps are vulnerable to MitM attacks.
- We found only 4% of the apps implemented TLS certificate pinning and therefore were well protected against MitM attacks.

Percentage of Apps Exposing High-Value Secrets



The percentage of apps exposing high-value secrets which could be used to attack APIs was significantly lower in Germany.

- 20% of the apps were vulnerable to Man-in-the-Device attacks.
- Only four apps (less than 1%) were protected at runtime against both MitM by using pinning and against Man-in-the-Device manipulation by using a packer or protector. This means that more than 99% of the apps were signaling that they were exposed to attacks at runtime.

Comprehensive Protection

- None of 650 apps had high scores for both static secrets protection and runtime security.

US vs Europe

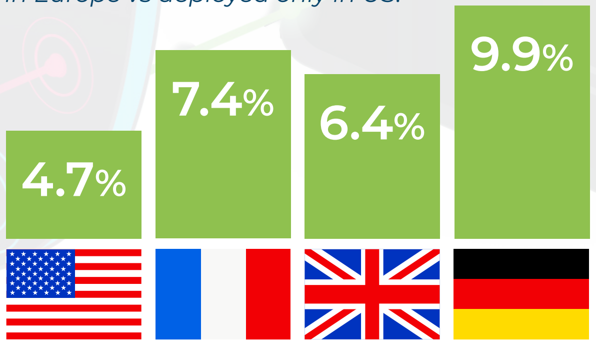
- We were interested to see if Europe's strict GDPR security rules had an impact and compared US-only apps with apps released in Western Europe. We found clear differences for both static and run-time protections, all showing that security was taken more seriously for apps destined for the European market.
- The percentage of apps in Western Europe revealing no secrets was double that of the apps deployed only in the US.
- For MitM attacks, the difference between apps deployed in Europe and those only available in the US market was also striking. Twice as many US-only based apps are open to MitM than European deployed apps.
- Only 8% of European deployed apps were protected against Man-in-the-Device, but that was three times better than US-only apps.

Data by Type of App

- For Leakage of Critical Secrets, crypto apps were the worst with 36% of apps leaking high-value secrets. Personal finance had 18%.
- For Channel protection, insurance apps were the worst with 8.5% vulnerable to MitM versus only 1.6% of trading and investing apps.
- For Man-in-the-Device attacks, traditional banks are twice as likely to be well protected than any other sector, reflecting the use of packers and protectors to protect against run-time manipulation.

No Secrets Leaked

The percentage of apps leaking no secrets was shockingly low across the board, but we did note that it was better for apps deployed in Europe vs deployed only in US.



Summary Table

Category	Exposed High Value Secrets %	Open to MitM %	Open to Man-in-the-Device %
Traditional Bank	26%	4%	32%
Challenger Bank	26%	2%	18%
Personal Finance	18%	3%	17%
Trading & Investing	29%	1.6%	14%
Payment & Transfer	23%	5%	6%
Insurance	21%	8.5%	26%
Crypto	36%	6%	8%
Other	13%	4%	27%

Detailed Results

Protecting Secrets at Rest: Code Protection

We set out to determine how many secrets we could immediately extract from code with an automated static analysis and found valid secrets exposed in almost all of the apps. These were real secrets that could be used to access a variety of backend APIs.

Secrets found were classified into Low, Medium and High Value. Although Low Value Secrets are not "service-impacting", hackers could still access these types of APIs and post incorrect and misleading data, undermining the quality of analytics, or causing support teams to waste time tracking down seemingly widespread phantom bugs.

High Value Secrets are those we consider extremely dangerous if exposed. Some examples: private keys, keys for payment or transfer services, and keys that included "authentication" or "attestation.

Protecting Secrets in Transit: Preventing Man-in-the-Middle Attacks

The communication channel between apps and APIs presents a rich target for hackers via Man-in-the-Middle (MitM) attacks, which can be used to bypass advanced security measures such as MFA. In the mobile use-case, deploying Transport Level Security (TLS) alone is not sufficient since tools installed in the device can easily intercept encrypted communications.

MitM attacks occur when an attacker intercepts or manipulates mobile device communications to gain access to sensitive information. They give attackers the ability to see any communications, modify messages, steal login details or certificates from encrypted traffic, and intercept sensitive commercial/personal data,. They can easily launch a denial of service attack against the service being accessed through a mobile app.

Static protections, such as obfuscation, do not prevent MitM attacks at run-time. A static analysis of the Network Security Config File can identify how susceptible the app is to these kinds of attacks.

More info on MitM attacks can be found at approov.io/product/dynamic-cert-pinning.

We scanned each decoded APK to find the apps network_security_config.xml file and then parsed it. This file lets developers customize an app's network security settings in a declarative way separate from the app code.

How Apps Fared at Protecting Secrets

Only 5 apps exposed no secrets to our analysis while 150 exposed high-value secrets.

5 ✓ **Acceptable** No secrets exposed	48 ⚠ **Low** Only low value secrets exposed
444 ⚠ **Medium** Medium value secrets exposed	150 ⚠ **High** High value secrets exposed

To classify the communications channel, we inspected the following:

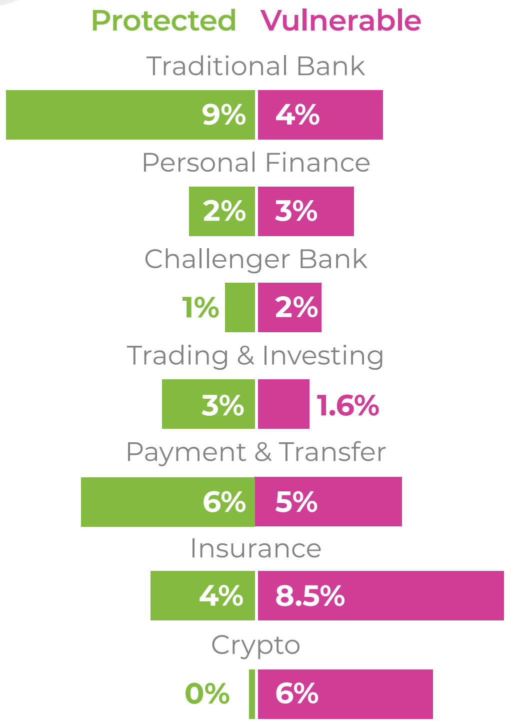
- **Are there indications that the application uses certificate pinning?** Certificate pinning is a way of limiting the set of acceptable certificates and is declared in the network security config file. Use of pinning shows that efforts are being taken to protect the channel.
- **How limited are the types of Certificate Authorities trusted by the app?** Information captured from the network security config file can tell us which Certificate Authorities (CA) are trusted for an app's secure connections. For example, the app can allow particular self-signed certificates or can restrict the set of public CAs that are trusted. Configurations extending trusted CAs beyond system trusted CAs, especially to self-signed certificates, can be a signal that the channel may be open to attack, when a user is tricked into installing custom certificates from a rogue public wifi portal, thus allowing all communications to be intercepted by an attacker who can extract secrets and understand the communications between app and server in order to be able to impersonate the app from a script.
- **Does the app permit cleartext traffic?** Allowing the use of the unencrypted HTTP protocol is strongly discouraged and a developer should avoid it as much as possible. In the most recent versions of Android it is disabled by default. If cleartext is enabled that is a clear signal that this channel is open to attack at runtime.
- **Version of Android:** Older versions of Android have less protection against MitM attacks, and using an older version can be used to an attacker's advantage. We can verify if older versions are permitted.

We used the indicators to categorize apps:

- **Protected:** We observed signs that solid defenses were in place. These apps will be low on a hacker's priority list. Apps fall in this category if pinning is detected in the network config file and no other red flags are raised.
- **Vulnerable to MitM attacks:** We observed one or more serious vulnerabilities that would lead us (and attackers) to conclude that the app is open to run-time MitM attacks. If any of the following are true the app is placed in this category:
  - User-generated trust anchors permitted
  - Cleartext traffic its permitted for non localhost domains
  - Android SDK API level below 25 is declared in the Android Manifest

Man-in-the-Middle Attacks

MitM attacks occur when an attacker intercepts or manipulates mobile device communications to gain access to sensitive information. Apps were considered protected if pinning is detected in the network config file and no other red flags are raised. They were rated vulnerable if (1) user-generated trust anchors were permitted, (2) cleartext traffic its permitted for non-localhost domains, or (3) Android SDK API level below 25 is declared in the Android Manifest.



- **No conclusion:** We were unable to determine the level of run-time channel protection. These apps would be a second priority for hackers after “Open to MitM attacks” apps have been targeted.

Device Integrity: Preventing Man-in-the-Device Attacks

Although rooting, jailbreaking or other manipulation of the mobile device environment do not always mean that malicious activities are taking place, these are common techniques used by attackers and pentesters to bypass security mechanisms and limitations imposed by the original version of the OS.

Rooted and jailbroken devices pose a threat to device integrity because these actions enable the in-built security mechanisms to be compromised. The threat extends to mobile app integrity because the mobile app is now running in an environment that cannot be trusted. Another form of code tampering is to inject code at runtime by using an instrumentation framework.

Such frameworks are used to hook into the key functions that, when manipulated, will produce different app behavior than expected or will change input parameters or output results. Fraudsters can intercept and modify genuine user instructions.

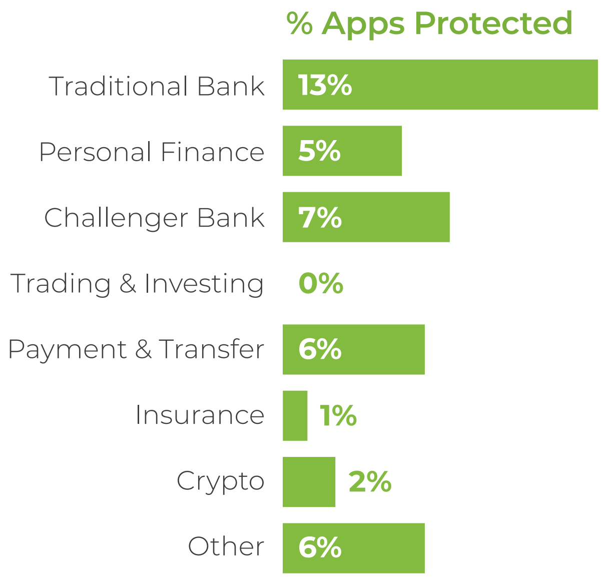
To prevent this form of attack, the mobile app must employ runtime self-defense code to detect rooted/jail-broken mobile devices. Ideally, this code will also detect the presence of all known instrumentation frameworks.

The readily available scanners we used can perform some basic evaluation of the types of protection in place to prevent runtime manipulation of the environment:

- **Debuggers:** Debuggers are important tools for development and test phases but should never be in place in a production environment where they could be used for malicious purposes, allowing attackers to observe an application in action under different conditions. It is best-practice to deploy anti-debugging techniques to keep programs safe. The scanners we used indicated whether or not apps checked for the presence of debuggers at runtime.
- **Use of a packer:** Packers are also known as “self-extracting archives”. This is software that unpacks itself in memory when the “packed file” is executed. If a scanner detects the use of a packer on the app code, this is viewed as a step towards protecting against runtime tampering.

Poor Protection Against Man-in-the-Device Attacks

A focus on the well-protected apps data shows that traditional banks are at least twice as likely to be well protected against run-time manipulation than any other sector, reflecting the use of packers and protectors. Overall, more than 87% of the apps are underprotected. In some sectors, it is more than 95%.



- **Use of a protector:** A protector is software that is intended to prevent tampering and reverse engineering of mobile app code. An attacker will be faced with protective layers around the payload, making reverse engineering difficult. Techniques can involve encryption or code-virtualization. Again if a scanner determines an app uses a protector, this is deemed to be a positive contributor to runtime protection.

We used these indicators to categorize apps:

- **Well protected against Man-in-the-Device:** We observed signs that solid defenses were in place. Hackers won't waste their time here. Apps are in this category if we were able to determine that packers and protectors were in use.
- **Open to Man-in-the-Device:** We observed serious vulnerabilities that would lead us and attackers to conclude the app is open to run-time manipulation. Apps are placed in this category when no anti-debug checks are detected.
- **No conclusion:** We were unable to determine the level of run-time protection. These apps would be a second priority for hackers after "Open to run-time Man-in-the-Device attacks" apps have been targeted.

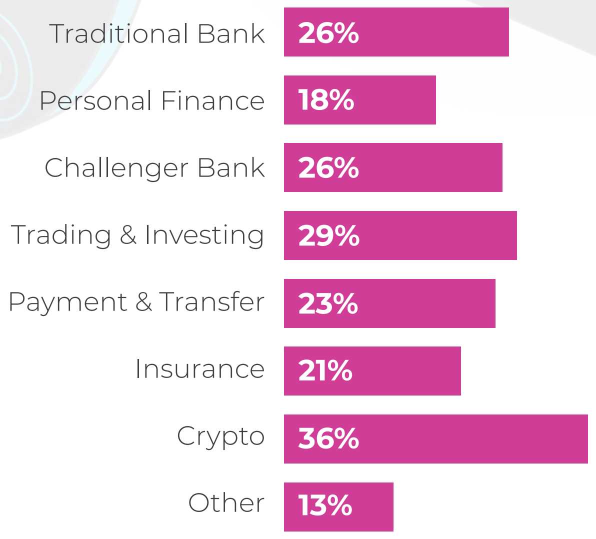
Conclusion: A Comprehensive Approach to Runtime Security is Needed

Obfuscating APIs within the mobile application can provide some level of security by making it more difficult for attackers to discover and access the APIs. However, this approach is far from foolproof, as determined attackers can still reverse engineer the application and find the API endpoints. Using automated scans, we were able to immediately extract-sensitive secrets from almost all the inspected application packages. The use of obfuscation techniques had little impact on the number or type of uncovered secrets.

Moving API keys to a secure cloud and delivering them to the mobile application at runtime can provide a higher level of security. In this approach, the API keys are stored on a secure server, and the mobile application requests them at runtime. This way, the API keys are not hardcoded into the application and are not as easily accessible to attackers. Our investigation found that 99% of the app packages inspected indicated they were not implementing protections to prevent secrets such as API keys from being extracted from the running app.

The percentage of apps leaking dangerous secrets and exposure was consistently high — above 20% for the main app categories. Crypto had the highest percentage at 36%. The major app types were consistently low in terms of leaking no secrets. None achieved better than single digits.

High-Value Secrets Exposed



Commonly implemented security practices focus on shift-left approaches to identify and remove vulnerabilities during development as well as applying code obfuscation to mobile apps prior to deployment. Neither of these approaches proved effective in eliminating the risk of "zero day" vulnerabilities nor preventing loss and abuse of API keys at runtime.

A more comprehensive approach to runtime mobile application security is needed, one that protects the app, the client environment and the communications channel from the app to the cloud. Such a solution should:

- Systematically check that any request to the API is genuine and not coming from a bot, a script or a repackaged app.
- Be able to detect any unsafe operating environments on the client device, such as rooted/jailbroken devices, apps running under debuggers or emulators, or whether malicious frameworks are present.
- Protect the path from the app to the API from Man-in-the-Middle attacks by systematically implementing certificate pinning.
- Use a secrets management solution that manages API keys and certificates securely in the cloud, delivers them "just-in-time" and allows them to be easily rotated via over-the-air updates as required.

Implementing a runtime security solution with these features will keep mobile apps and APIs safe from abuse, and safeguard the valuable data of consumers who depend on these applications to safely manage all their payments, banking and investment assets.

About This Report

One requirement for this research was to be able to classify the risk level of secrets found without manual intervention. The automated classification methodology may introduce some uncertainty in how the secrets found were classified, however this approach eliminated arbitrary criteria or judgment errors. Overall the percentages occur within a reasonable margin of error, consistent with other reports, and give an indication of the level of immediate exposure.

In terms of uncovering information about the risk of runtime secret loss, we used commonly available scanning tools, and were subject to the limitations of these tools in terms of completeness and accuracy of the results. There may be protection mechanisms in place which are not detected by these off-the-shelf scanners. However, we remain confident that the investigative tools did identify, with a reasonable level of accuracy, the corner cases (i.e. well protected vs. poorly protected) for both the runtime attack surfaces we investigated.

For more information about Approov Mobile Security, call us at +1 (650) 322-5300 or visit us at approov.io.

US Headquarters
165 University Avenue
Suite 200
Palo Alto, CA 94301 USA

UK Headquarters
181 The Pleasance
Edinburgh, Midlothian
EH8 9RU, United Kingdom