SANS | Research Program

PRODUCT REVIEW

Collaborative Mobile App Security Development and Analysis

Written by **Jeroen Beckers**

May 2025

CORELLIUM

Introduction

Many companies implement a mobile-first strategy wherein the focus of interaction with their customers is via mobile applications. These applications range from silly games and social media apps to healthcare, financial, and official government applications.

Before the age of smartphones, security was focused on the back-end API. This makes sense, because web applications are typically thin clients that don't store any sensitive information locally. The API is responsible for performing authorization controls, securely storing any user information, and enforcing proper authentication. Although API security remains as relevant as ever, mobile applications have introduced an entirely new attack surface. On both Android and iOS, mobile apps typically store data on the device itself and often allow the user to authenticate using local authentication methods such as biometrics.

Thus, it is paramount that mobile apps undergo stringent security audits to ensure they are protected against many different types of attacks. The industry standard for mobile application security assessments is the OWASP Mobile Application Security Verification Standard (OWASP MASVS).[1]

Unfortunately, performing a security audit of a mobile application comes with many challenges:

- **Wide range of supported OS versions—**Every year, new versions of Android OS and iOS are released. Although most users will migrate to the latest version, this is not always possible depending on the device they have. As a result, companies must support older versions of both Android and iOS to support the largest possible customer base. New security features should be implemented on new versions, while maintaining the best possible security baseline on older devices.
- **Wide range of supported hardware—**Even if everyone used the latest OS version, the hardware could vary widely. Android enforces some minimum hardware requirements, but there are still important differences in the support for biometrics, cryptographic key storage, and tampering detection.
- **Localization—**Applications that require the most stringent security controls and the highest resiliency typically deal with very sensitive data. Government or financial applications often require users to identify themselves using secure authentication methods such as eID or registered phone numbers. This means that in order to properly test the application, the tester must be able to authenticate securely, which is often possible only if the tester is an official resident of the country for which the app is developed. On top of this, mobile apps are often only available in a local language without an easy way for the tester to translate the UI into their preferred language.

[1] "OWASP Mobile Application Security," https://mas.owasp.org

- **Availability of root access to device—**Although it is possible to perform mobile security assessments on non-rooted or non-jailbroken devices, it is much more complicated than when the tester has access to the underlying OS as the root user. Direct access to the file system, the iOS Keychain, and Android's KeyStore, and the ability to dynamically instrument applications without having to repackage them makes performing an assessment much more straightforward. Unfortunately, it's not possible to obtain root access on just any device. Android devices may have a locked bootloader that can't be unlocked; for iOS, no public jailbreak allows root access on the latest iOS version (iOS18) or even the previous major version.
- **Sharing of physical devices—**iOS devices with a version of iOS that can be jailbroken are a very valuable commodity. Due to Apple's aggressive update policy, the inability to perform an OS downgrade, and the unavailability of a public jailbreak for recent iOS versions, it's getting more and more difficult to obtain an iOS device that can be jailbroken. This leads to device sharing between testers of the few exploitable devices that are available, often having to plan projects around the availability of devices and having to ship devices cross-country.
- **Unavailability of iOS emulators—**While a tester can choose from a plethora of Android emulators, this simply does not exist for iOS. Xcode, the IDE for iOS apps, does contain an iOS simulator, but this is only useful during application development. It can't be used to perform a third-party application assessment because the application must be specifically built for the iOS simulator.

These challenges can make it very difficult to quickly and thoroughly perform a security assessment of a mobile application. Additionally, Apple's tight control over iOS devices makes it difficult to organize training around mobile application security. Due to the aforementioned challenges, it's simply not feasible for a training instructor to provide iOS devices, often requiring students to bring their own devices. This means that, in practice, most mobile trainers simply skip iOS or are limited to static analysis for iOS applications.

In this product review, we'll look at how Corellium can help overcome these challenges and how it allows us to perform efficient security assessments by a global team. Although the focus of this product review is on mobile application security assessments, we'll also look at some more advanced features that are useful for exploit development.

The SANS SEC575 course has already been utilizing Corellium for many years now, and it's an invaluable asset to teach students about iOS security analysis and instrumentation. For this review, Corellium has provided us access to the latest features so we can see what their cutting-edge solution offers.

Overview of the Corellium Offering

The power of Corellium lies in its CHARM software. CHARM (Corellium Hypervisor for ARM) allows Corellium to virtualize ARM code, including both Android OS and iOS. Corellium offers two different environments, depending on the goal:

- **Corellium Viper—**Automated tooling to help security researchers perform security assessments and malware analysis
- **Corellium Falcon—**Powerful low-level control over the device for vulnerability research and exploit development

For this review, we will focus on the Corellium Viper environment.

Getting Started

It couldn't be simpler. After clicking "Create Device," we can choose the hardware model as well as the version of iOS. Corellium actively tracks new OS releases, and releases are typically available very quickly after the official release by Apple and Google. The list of supported devices is quite long, and you can go as far back as an iPhone 6 on iOS 10.3. For this test, let's go for the latest and greatest: an iPhone 16 Pro Max on iOS 18.3.1 (see Figures 1 and 2). Corellium claims to add support for the latest iOS releases within two weeks of their release, so you should always have access to the latest version as long as you choose a compatible device.

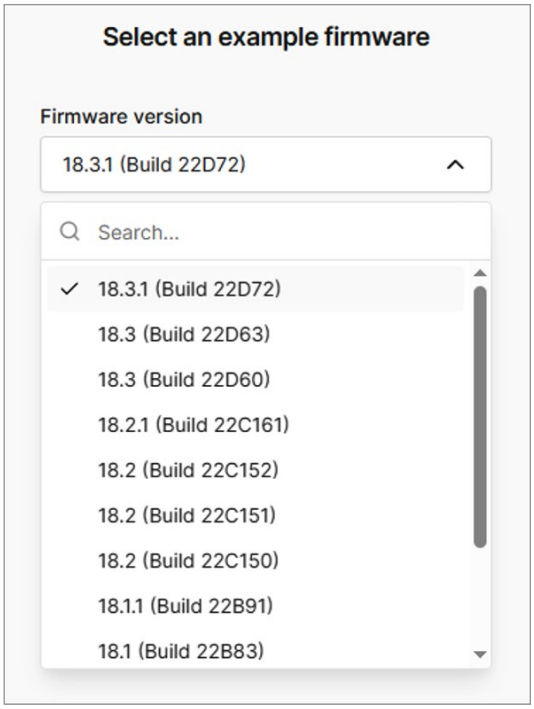

Figure 1. iOS Versions for the iPhone 16 Pro Max

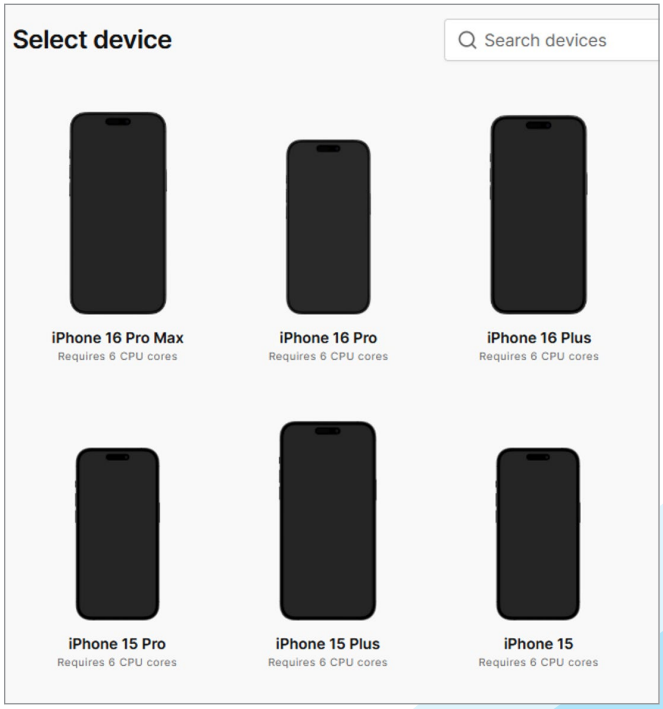

Figure 2. Recent iOS Device Versions Corellium Can Create

It takes a while for the device to fully initialize, but luckily this only needs to be done once. After the device has been created, restarting or restoring from a snapshot only takes a few minutes.

The main UI of Corellium contains the device we've created and a whole range of tools on the left side of the screen (see Figure 3). The device itself can be interacted with directly through the UI, and you can use your keyboard and mouse to navigate and type.

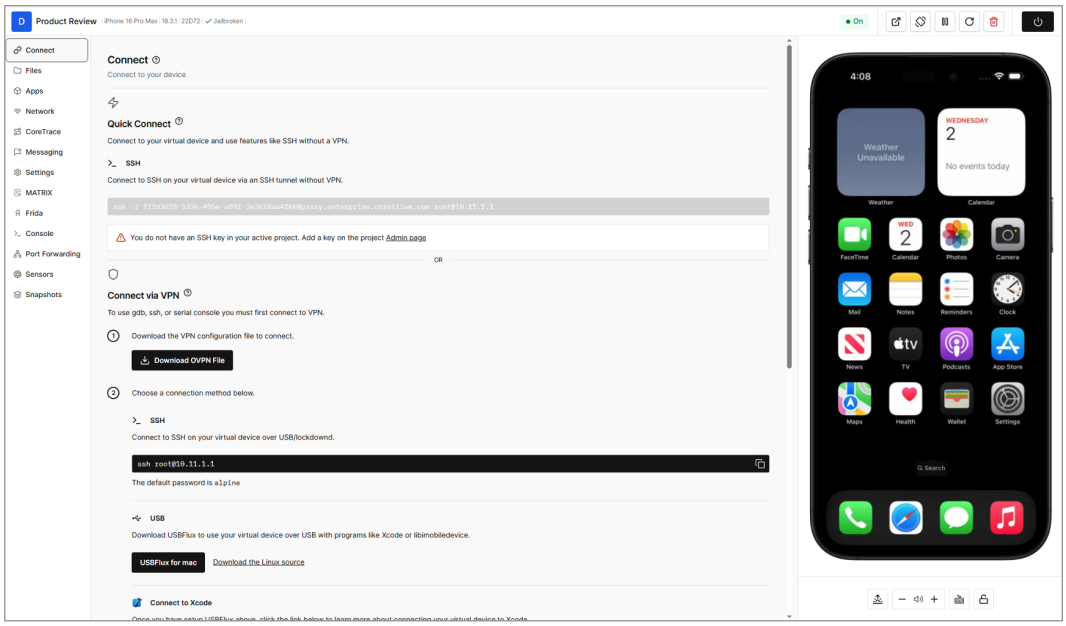

Figure 3. Main Corellium Interface

The fact that we can control the device through the Corellium interface is already an awesome feature. It allows remote teams to collaborate easily, and it allows an instructor to troubleshoot a student's device during online trainings just as easily.

Initial Impressions

We can interact with the created device in various ways, either through the Corellium UI or through one of the many connection options.

```
→ ssh root@10.11.1.1
root@10.11.1.1's password:

Product-Review:~ root# id
uid=0(root) gid=0(wheel)
groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmod),
20(staff),29(certusers),80(admin)

Product-Review:~ root# pwd
/var/root
```

Figure 4. Accessing the Device via SSH

Each Corellium device is created in a 10.11.1/24 subnet, and we can download an OVPN file to join that subnet. From there, we can access the device over any protocol we want, including SSH (see Figure 4).

For physical devices, modern jailbreaks are called "rootless" because they no longer allow you to modify the root filesystem. However, because Corellium has full control over the hypervisor, this is not an issue on a virtual device (see Figure 5).

```
Product-Review:/ echo "Oh no" > SANS.txt
Product-Review:/ cat /SANS.txt
Oh no
Product-Review:/
```

Figure 5. Full Access to the Root Filesystem on iOS 18

Although this isn't very useful for a mobile security audit, it is very useful while doing vulnerability research with Corellium Falcon, and shows the power of virtualized environments, even over jailbroken physical devices. We'll look at other connection possibilities later in this paper, but first let's discuss the other tools in the Corellium UI.

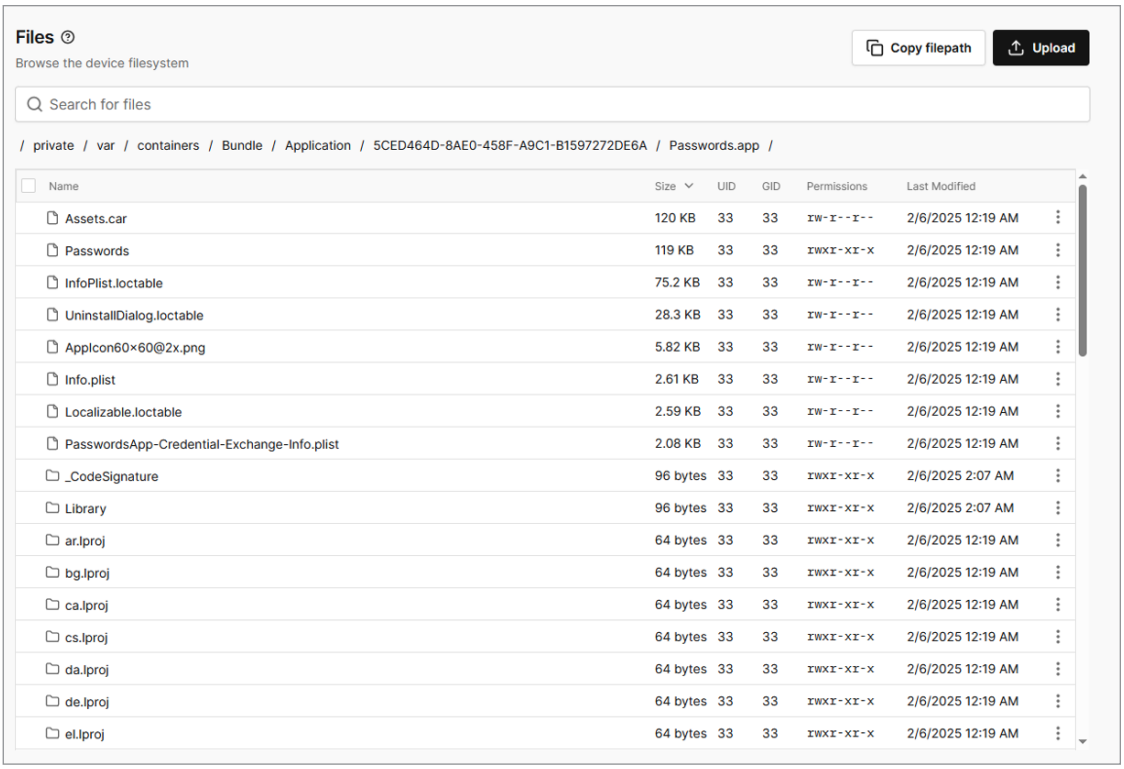

Figure 6. File System of the Passwords Application

One very important part of a security audit is analyzing the locally stored data. This is usually done by using SSH to connect to the device, injecting Frida and using Objection, or taking an iOS backup. Corellium offers a much nicer approach: a live file browser for the entire filesystem! It's easy to navigate, and you also can easily upload files (see Figure 6).

Corellium also provides an easy way to view, launch, and kill installed applications, as well as install new applications (see Figure 7).

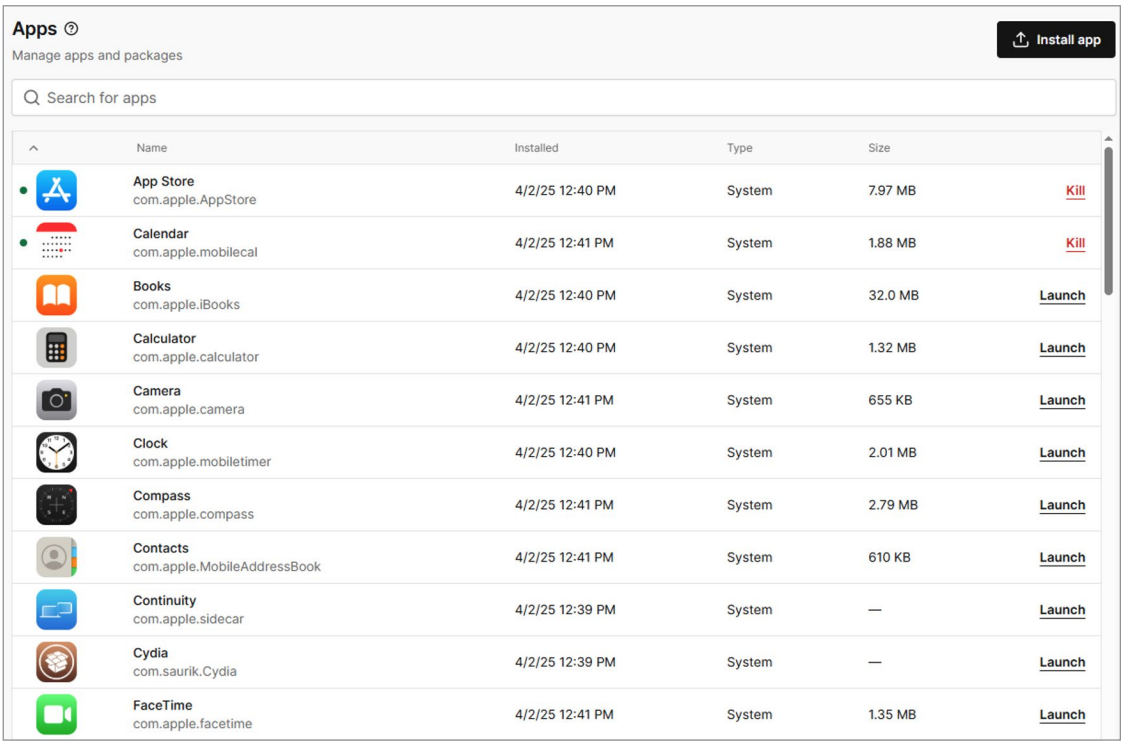

Figure 7. Overview of All Installed Applications

The most useful feature here is probably the ability to easily install new applications. Corellium does require your IPA to have a valid signature, which requires a valid Apple Developer account. This can easily be fixed on a physical jailbroken device using AppSync Unified, but because AppSync Unified doesn't support iOS 17 or iOS 18, it cannot be used on our virtualized iOS 18 device.

Corellium does provide one additional trick here: If your IPA is properly signed for at least one device, you can change the UDID of the virtual device to match the UDID from the signature (see Figure 8).

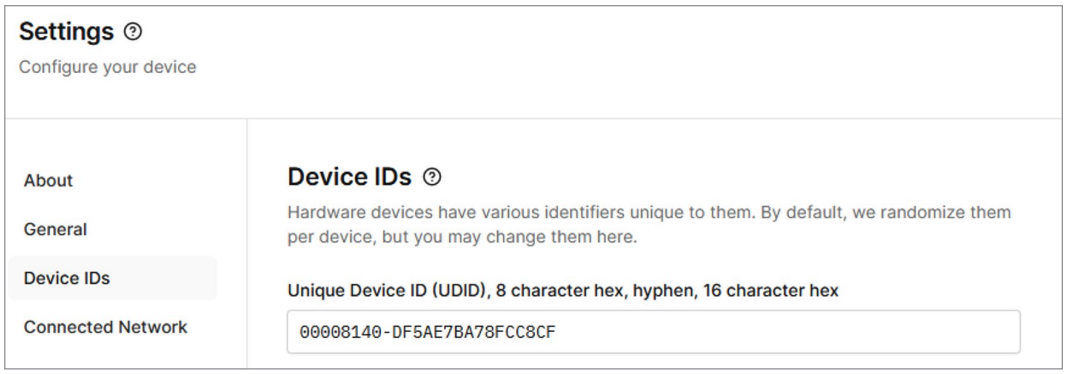

Figure 8. Changing the Device's UDID to Pass IPA Signature Validation

This way, you can use any ad-hoc signed IPA as long as the signature itself is still valid

Moving further down the UI, there is also a Console section that prints out the system's console logs. It might not be obvious at first, but this Console window is a terminal, too. It's not very convenient to use because the normal log output constantly interrupts your commands—but it's nice if you need to execute a few quick commands on the device (see Figure 9).

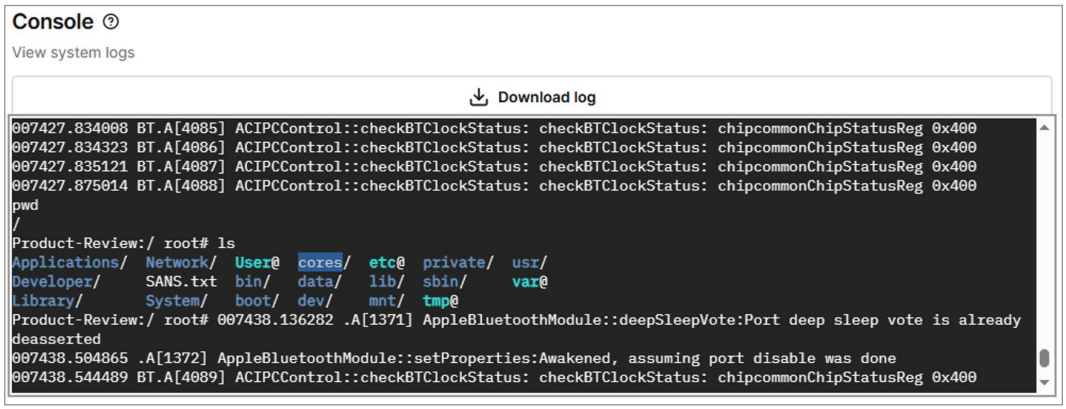

Figure 9. Console Access via the Main UI

Finally, users can create and share snapshots. This is a very useful feature, both for testing and for teaching. Being able to share the snapshot of a preconfigured device is invaluable during training, because we don't want to spend time troubleshooting the setup. Additionally, if you're analyzing malware or performing a security assessment, it's practical to be able to load a "known good" state before triggering the malicious or vulnerable behavior. The snapshot restore allows easy repetition of specific tests—for example, while you're fine-tuning your Frida script.

A few other features are available through the Corellium interface, but we'll explore those later.

Just Like a Local Device

One very nice feature of Corellium is that you can connect your device directly to your local host and make it appear as a connected USB device. This is done through the power of USBFlux, which is available for both macOS and Linux.

After connecting to the provided VPN and installing the required USBFlux software, we can interact with the device using the libimobiledevice suite, just like if the device was connected via USB (see Figure 10).

```
→ ~/tools/usbfluxd/tools/usbfluxctl add 10.11.1.1:5000
SUCCESS
→ idevice_id
00008140-DF5AE7BA78FCC8CF (USB)
```

Figure 10. Virtual Device Appearing as a Locally Connected USB Device

Because of this setup, all our tools can treat the virtual Corellium device as a normal device connected over USB without the tools sensing the difference.

Out-of-the-Box Support for Frida

This brings us to tooling. As explained before, our tools shouldn't even know that they are connecting to a virtual device rather than an actual physical device connected over USB. This means we can easily use our standard tool set (xCode, MobSF, Objection, Frida, and many others) without having to change our setup. Let's give that a shot. Frida is supported out-of-the-box by Corellium so we don't even need to do anything special: no repackaging, no installation of Frida-server, no port-forwarding! See Figure 11.

```
→  ~ frida-ps -Ua
 PID  Name      Identifier
----  --------  --------------------
1196  Calendar  com.apple.mobilecal
1584  Settings  com.apple.Preferences

→ frida -U -f com.apple.Preferences
     ____
    / _  |   Frida 16.5.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to iOS Device (id=00008140-DF5AE7BA78FCC8CF)
Spawned `com.apple.Preferences`. Resuming main thread!
[iOS Device::com.apple.Preferences ]-> ObjC.available
true
[iOS Device::com.apple.Preferences ]->
```

Figure 11. Pre-installed Frida, Available Even on iOS 18

This is a very useful feature from Corellium because we can use Frida on recent iOS versions without repackaging the target application.

Network Traffic Interception

A major aspect of mobile application security is making sure that the back end is secure as well. This typically requires setting up a machine-in-the-middle position so that all the traffic runs through a web application proxy (e.g., Burp Suite). Corellium offers a few interesting features to make our lives easier.

Certificate Validation Is Disabled by Default

On all iOS versions, Corellium disables SSL certificate validation in many popular SSL libraries. This is similar to what SSL Kill Switch can do on a jailbroken device, but built in from the start! You can even view the collected HTTP(S) traffic directly in the Corellium interface (see Figure 12).

Unfortunately, this won't always work. This isn't Corellium's fault, because SSL Kill Switch has the same limitations. Sometimes applications just do their absolute best to prevent a machine-in-the-middle position. In total, SANS SEC575 has three modules on intercepting traffic on both Android and iOS, discussing all the different pitfalls and potential bypasses. This is typically where you start reverse-engineering the binary and create your own custom Frida scripts.

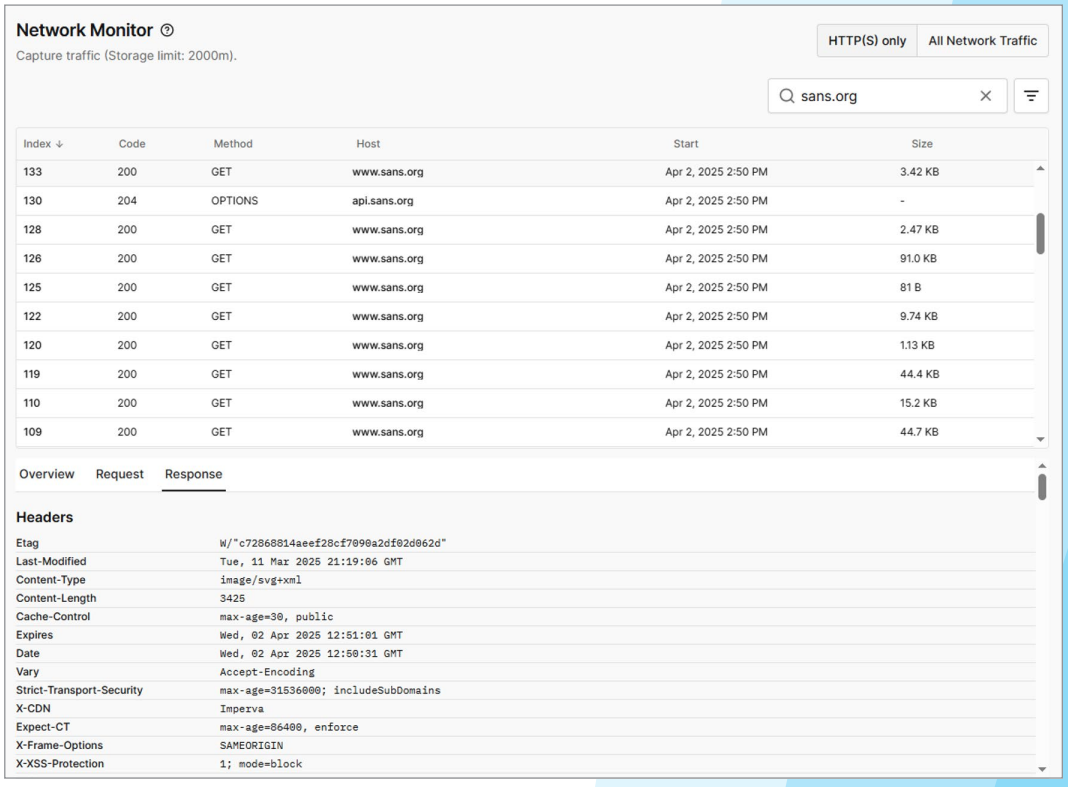

Figure 12. Corellium's Network Monitor

When troubleshooting these issues, it's important to collect the network traffic on the device so you can figure out what is going on. And luckily, Corellium can help us there, too.

Full Network Monitor

In addition to logging HTTP(S) calls, Corellium also can collect a full packet capture of everything leaving (and entering) the device. The output is a packet capture (PCAP) file that we can then analyze using Wireshark. This saves a lot of effort in comparison with a physical device, where we would have to create a custom hotspot or perform DNS spoofing. See Figure 13.

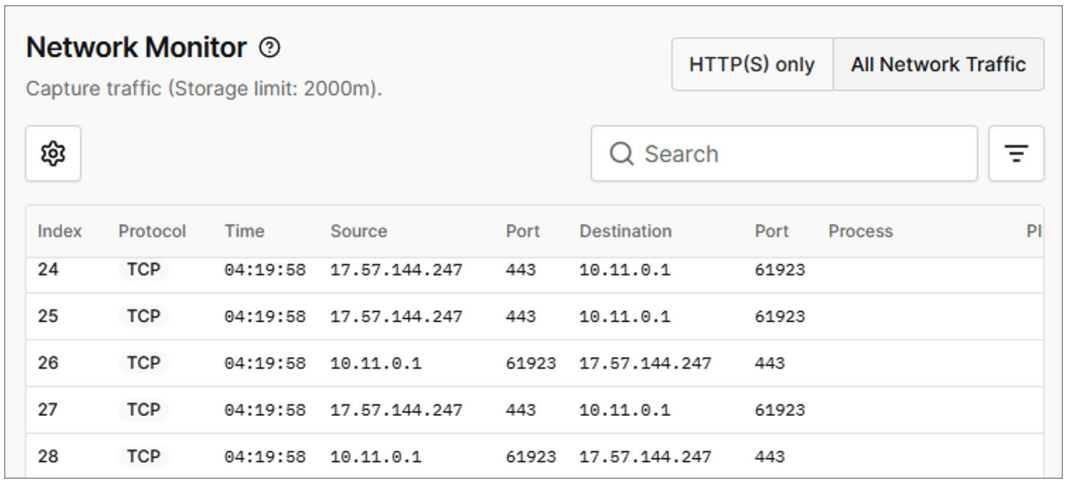

Figure 13. Full Traffic Captures

Enter the MATRIX

In addition to supporting a pentester during a security assessment, Corellium also offers MATRIX for automated security testing. This technology allows you to specify which sensitive data it will track and will then monitor you while you interact with the application. During the interaction, the goal is to trigger as many code paths as possible so that the dynamic analysis engine can collect as much information as possible.

After the manual interaction with the application, MATRIX performs static analysis, too, finally resulting in a unified report with all passed and failed checks (see Figure 14). In this example, we tested MATRIX on iGoat-Swift, an intentionally vulnerable iOS application.

At the time of writing, MATRIX executed 58 tests and identified eight different artifacts. Each of the tests is mapped to compliancy frameworks including the OWASP Mobile Application Security Verification Standard (MASVS), OWASP Mobile Application Security Weakness Enumeration (MASWE), Common Vulnerabilities and Exposures (CVE), and Common Weakness Enumeration (CWE).

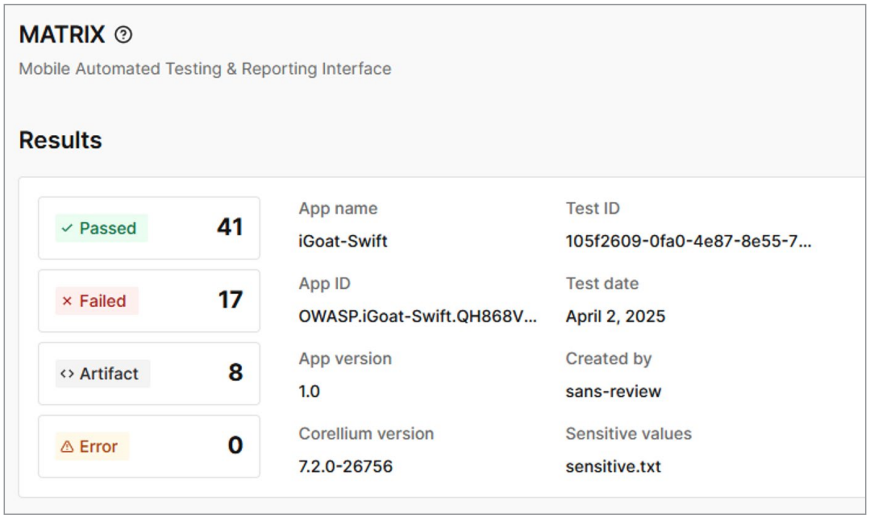

Figure 14. MATRIX Results of the OWASP iGoat-Swift Application

Of the 17 failed tests, only two were false positives, but luckily MATRIX includes the information it used to draw its conclusions, so it's easy to look and figure out whether something is an actual issue. For example, the issue shown in Figure 15 indicates a hardcoded API key. However, the identified key is a magic string belonging to the WebSockets spec, and it will be included in many different applications.

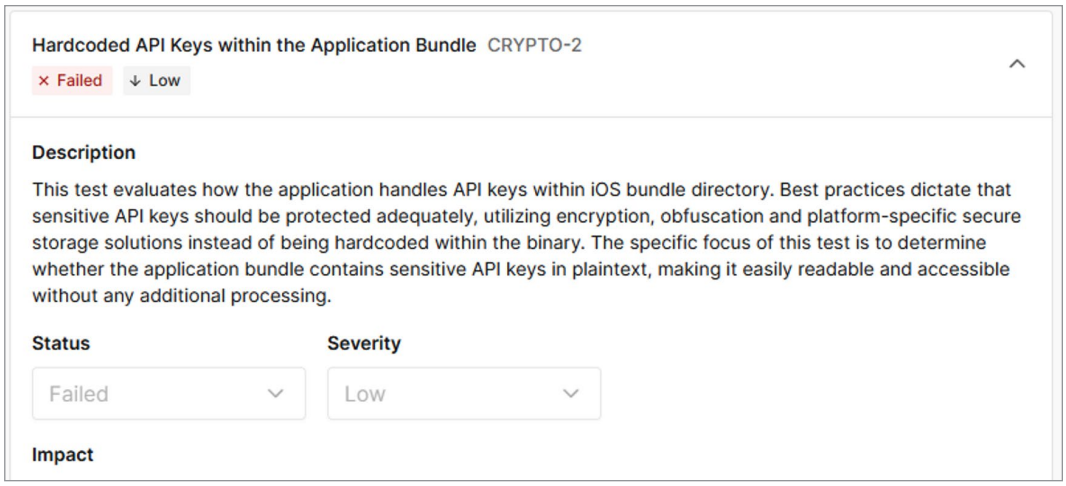

Figure 15. False Positive for a Hardcoded API Key

On the other hand, MATRIX identified several real issues with a high impact. The issue in Figure 16 describes a biometric authentication bypass, something we often see during mobile security assessments. It indicates insecure usage of cryptographic APIs as well as insecure storage of authentication tokens/ credentials.

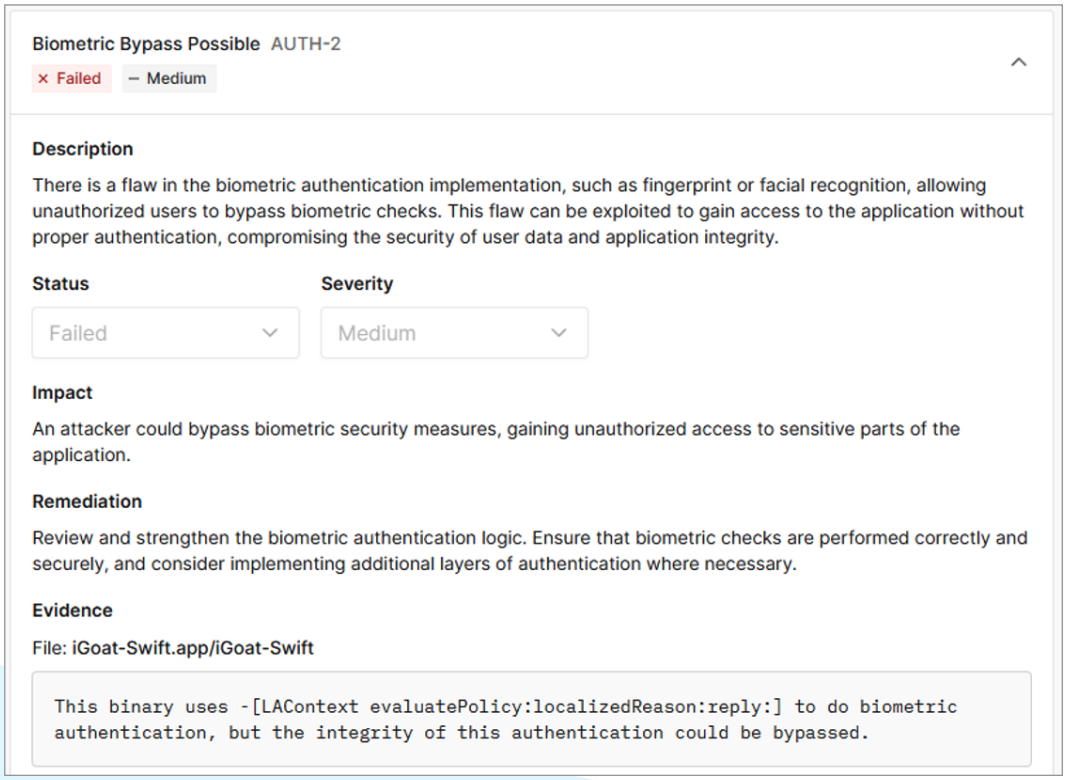

Figure 16. Corellium Identifying Insecure Biometric Authentication

MATRIX contains a few tests that we hadn't seen before; for example, a test for a specific vulnerable version of a protobuf library (Nanopb). This is an interesting test, because dependencies are often overlooked by automatic scanners. This is most likely because identifying version information is quite tricky from compiled binaries, but it looks like these vulnerable dependencies are on MATRIX's radar. See Figure 17.

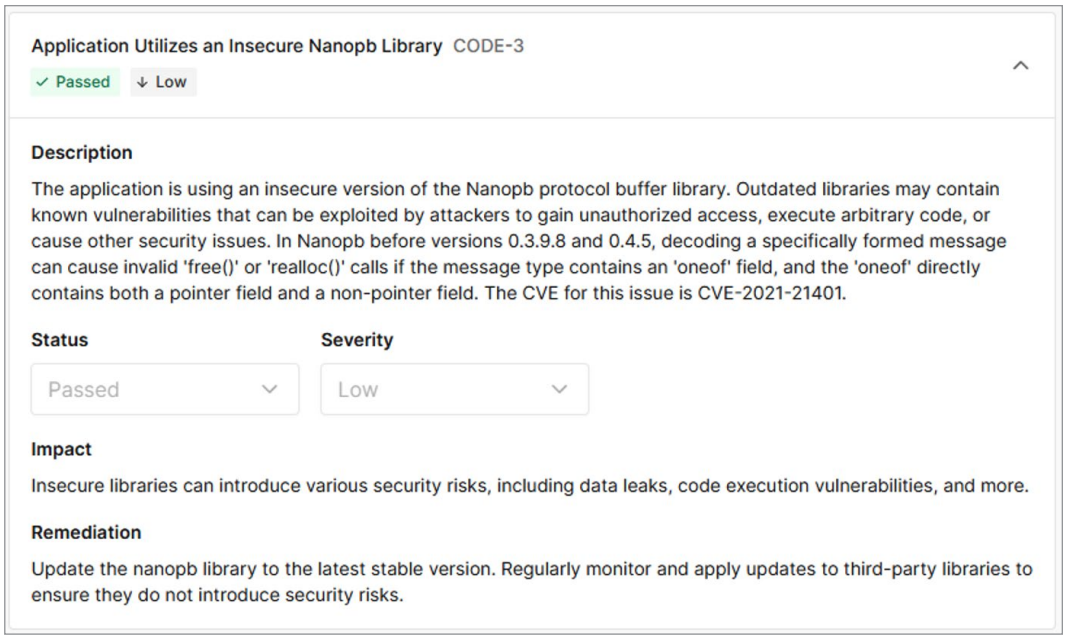

Figure 17. Corellium Checking for Insecure Libraries

Will MATRIX make manual security audits obsolete? Definitely not. But neither will any of the other commercial mobile security scanners. Reverse-engineering is (so far) an unsolved problem. We can't just give our compiled code to an automated algorithm and be sure that it fully understands each line of code, each path through the code, and each possible button press and text input.

However, MATRIX can point us in the right direction for potential issues, and it gives us information on how to quickly dig deeper to see why certain conclusions were made.

Advanced Use Cases

The focus so far has been on application security assessments and interacting with a targeted application. The real power of Corellium, however, lies in the vulnerability and malware research world with Corellium Falcon. With Falcon, you can have advanced access to the device's kernel, trust cache, device tree, and Ram disk. You can even supply your own custom kernel, apply different kernel patches to get as close to a real device as possible, or add your custom modifications wherever you want.

For mobile security assessments, the Corellium Viper environment does give you access to CoreTrace, allowing you to monitor which syscalls an application is triggering. This is useful during resiliency assessments, where the goal is to identify vulnerabilities in the application's Runtime Application Self-Protection (RASP) controls.

Is It All Great News?

Unfortunately, there are some downsides to Corellium when compared with a physically jailbroken device. The main downside is that Corellium virtual devices do not have the Apple App Store installed. Therefore, it isn't possible to download third-party applications directly from the App Store. In practice, you must obtain the decrypted IPA file in a different way, for example, via an older jailbroken device.

Although we couldn't test this, we got a sneak peek into a planned Corellium feature that allows you to install certain popular third-party applications without having to obtain the decrypted IPA files yourself. This would remove one of the biggest downsides of Corellium and would make the platform even more interesting to use.

The second downside is latency. This is not uncommon for emulators, but the fact that it runs in the cloud does make the interaction with the device sluggish at times. This will most likely also depend on how far you are from the Corellium data centers, but even with a great connection, a physical device will always be more responsive. If latency is a major concern, there is also the possibility to have an on-site appliance, which additionally gives you the assurance that all your research stays private.

What About Android?

Although Corellium's focus is on iOS, it does offer Android devices as well. In general, Android is a much easier ecosystem to perform security assessments in. You can buy a device with an unlockable bootloader, and there are many different options for Android emulators, too. Still, Corellium offers the same nifty features as we saw for iOS, including adb access, snapshots, a file browser, and automatic SSL pinning bypasses. This makes it more convenient to interact with your Android device, especially when you're just getting started.

In SANS SEC575, we use the Corellium Android device extensively to learn about Android security. The possibility to share snapshots, take over someone's device, and the standard Frida integration make it a very convenient tool to teach about core mobile security concepts without having to spend too much time on troubleshooting and setup.

Conclusion

In this paper, we've looked at many different aspects of Corellium. Some of the Corellium features really stand out and give unprecedented access during mobile security assessments, while for others the advantage over a local (Android) emulator or jailbroken iOS device is more limited.

The fact that our remote device can behave like a local device is a great feature, because we can easily use our own tool set in this new environment. Additionally, the default availability of Frida takes away a lot of the setup that is normally required for any Android or iOS device.

Having access to the OS internals on iOS 17 or 18 is simply not possible without Corellium. Performing a mobile security assessment on these versions of iOS is inherently much more difficult, because it always requires repackaging the application with a Frida Gadget and dealing with Apple's signing requirements.

MATRIX can identify some of the low-hanging fruit of an application, but manual analysis is required. Still, any tool that can quickly identify true positives will save us time and effort in the end. If you're responsible for mobile app security assessments, then this is worth considering for your toolbox.

Sponsor

SANS would like to thank this paper's sponsor: