Branch: master ▾ Find file Copy path

ms-intune-app-sdk-ios / Documentation / MAM-iOS-Security.md

msintuneappsdk 12.4.0

a958afe on 21 Apr

1 contributor

Raw Blame History

113 lines (64 sloc) 11.9 KB

Security in MAM iOS

iOS model

Apple has a great white paper on this - https://www.apple.com/business/docs/site/iOS_Security_Guide.pdf

Our model

Our current model is based on ensuring that admin defined policies are in effect. To that end, the two areas of most concern to us are data exfiltration and policy circumvention. These two things are mostly accomplished by gaining root access to the device. In the iOS space, this is refered to as ...

jailbreaking

iOS 'jailbreaks' are exploits found inside Apple supplied code that allow priveledge escalation attacks aginst iOS itself. From here, the user is able to gain root access to the device and modify some of the iOS behaviors that we depend on, such as code signature verification and encrypted keychain storage.

Where do we find the latest updates

Believe it or not, we've found the most reliable, up-to-date source of information for iOS jailbreaks to be https://www.reddit.com/r/jailbreak/

Jailbreaks (JBs) are often community driven, so reddit really is the best place to stay up to date with the latest attack. Additionally, for most of our customers, this is where their users would also go to learn about the latest JBs.

iOS jailbreaks tend to move in bursts, as finding a jailbreak is often time consuming, and Apple tries to fix any vulnerablities within the next iOS release. There are often specific releases of iOS that have jailbreaks, while others do not. For example, 12.0.1 and 12.0.3 might have a jailbreak available, while 12.0.2 does not.

Our policy for JBs is to verify whether or not our current logic can detect the newest JB. MSIT is not a fan of JB software, and would prefer you use an isolated machine for this task. Additionally, it can be time consuming to successfully get a device into the JB state, as some rely on memory corruption in order to succeed.

How we detect

Detecting the JB state of an app is a Cat-and-Mouse game. We may never be able to confidently say we can detect a tampered or jailbroken device, or that we can totaly prevent data exfiltration.

If we detect any evidence of jailbreaking, we have very limited response options, as Office is unwilling to let us indiscriminately terminate the app. There are some ideas to improve this that are covered in a later section.

Heuristic based

Our primary method for detecting the JB state of the device is a list of heuristics. We generally follow the same heuristics as this site: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/jailbreak-detection-methods/

You can also see the full listof actions at: `CommonUtility\Utilities\CUTRdUtil.m`

Swizzle detection

One novel approach that we use is to detect when our functions have been swizzled. Neil came up with the idea to verify the address of our funciton implementations to ensure that they remain inside our binary. For swizzled functions, we should see that their implementation pointer is outside the range of the loaded MAM SDK. Because this kind of swizzling is only possible on a JB device, we can use this as a marker. The policy for this is called SBIV, for Swizzle Based Integrity Verification. The code for this is located in `AppRestrictions\Utils\SecureFunctionUtils.m`

What's a Tweak?

The jailbreak community refers to modifications to the system software as tweaks.

What is Frida?

https://www.frida.re/docs/ios/

Frida is an open source project for enable iOS instrumentation. It is of interest to us because these developers are working to enable some previously unfeasible actions (swizzling) without jailbreaking the entire device (kind of - you still need to re-sign the app).

We haven't yet done a full investigation into this framework for several reasons. Although it provides a comprehensive framework for swizzling and modifying app behavior, the non-jailbreak approach requires downloading and re-signing the app. While this is not impossible (our App Wrapper essentially does the same thing), it does raise the barrier of entry. Additionally, Apple encrypts app binaries downloaded from the app store, which again although not impossible raises the abrrier of entry passed normal users.

This has been cropping up more often during pen tests as the go-to tool for testers, as it allows them more options to explore the SDK vs Swizzling tweaks.

What happens in a PEN test

During a Pen test we (or a customer) pays other engineers to poke holes in our product. You can ask the PMs for a copy of the most recent one, it should give you a good sense of what they're all like. They check a variety of things, including

- Are they able to circumvent policy
- Can they intercept communication between the MAM SDK and the MAM Service
- Can they get data out of the "MAM Sandbox"
- Can they jailbreak the device without MAM noticing
- Can they see encrypted data at rest

This is a good resource for the process they'll likely follow:

https://github.com/OWASP/owasp-mstg

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

What are some things we could do to make MAM more secure ?

User Supplied Encryption Key

We currently depend on Apple's hardware-driven encryption to keep data we hold on the device secure. This includes everything from corporate data, to access keys, to download policies, to encryption keys. All of these are stored in the device keychain, and protect by the device passcode. If an attacker were to subvert this passcode, the device and all data it contains is comprimised. In the case of MAM (as opposed to MDM), IT Pros are unqble to specify password requirements for the device, other than that one is present (only enforced by our own conditional launch policy for device passcode). For certain less-secure (4-digit) device passcodes, the attack time is sufficiently low that a dedicated attacker may be able to brute force the device passcode and gain access. One improvement we could make is to require the user to supply a sufficently complex password in order to generate a master encryption key held only in memory, with which we could encrypt/decrypt other sensitive data. We have so far declined to pursue this option as it presents a serious reduction in UX, as the app would require a complex passcode at every launch.

Double Encrypt

As stated above, we currently depend on the device passcode for encrypting sensitive data. If we ignore the UX hit we take when generating a secure key, we can easily solve the encryption problem for the relatively small data set that includes our own internal keys and policies. However, we also rely on device encryption for a large set of corporate data, which for files and databases can grow very large. We have tried this "double encrypt" method before (Vinay adapted the Android encryption hooks and tested this scenario), and the performance hit was deemed a non-starter for rolling this out to the apps. Apple hardware has the advantage of on-chip encryption which is much faster than anything we would be able to duplicate. If we were able to utilize this in a future release, as by some system API where we would ultimately hold the key, it would give us the opportunity to shut down some long standing concerns that large financial customers have held.

MAM CA Tie In

Our JB detection is currently toothless - our hands are tied with regards to what actions we're able to take if we detect a JB device. Because of the high risk associated with falsely flagging a device as JB, specifically for a personal app that has no enterprise account associated, we're unable to take automatic actions such as force-closing the app. We also rely on a JB policy to determine whether or not the IT Pro is concerned about device JB state. This leaves us open to attacks agianst these mechanisms, such as altering the policy in transit via a MITM attack as the policy comes in over the wire, disabling the blocking code or tampering with the policy via swizzling, or any other avenue we haven't thought of (or seen, as these examples were executed by pen testers already).

We do control MAM CA access to the Enrollment ID though, which gives us some control. For cases where we've determined that our SDK has been tampered with, we can revoke the Enrollment ID stored in the keychain/NSUserDefaults. This has the effect of preventing the app from continued access to corporate resources without impacting the personal account experience.

Function Obfuscation

One thing that comes up fairly regularly in Pen tests is function obfuscation. Function obfuscation is a compiler step that take our function names and automatically make them non-human-readable. This would serve to delay vanilla swizzling attacks by making it harder for a human to determine what to change. We intially did some research into this area, but were unable to procure a suitable 3rd party solution, and unwilling to fund the development need to create this ourselves.

What are some concerns we have about security?

In general, we've seen more customers adopt MAM. This is good. Some of these customers have higher expectations of the MAM security model than we can reasonably provide. This is not good. There are a couple of reasons for this.

They come from a different provider that had a much more restrictive product

Some `Good` examples of this are other MDM vendors who have developed buisness specific apps, and thus don't have to contend with the buisness/personal split, or having adopted design patterns that we've been reluctant to do (such as complex user supplied password). These 3rd party MDM providers are able to take steps that we cannot, giving them a slight advantage. Our current position, as Microsoft, is that our benefits (MAM, Office, UX) outweigh the concerns that they have. Usually, the role for engineering in these situations is to work with PM to determine the best, most accurate message that we can use to alleviate customer concerns.

They have an unrealistic expectation of our ability to respond to differing threat levels

MAM security is only as good as the app, which is unreliable even when we have dedicated teams of Microsoft engineers working on them. An advanced attacker can simply create their own app, integrated the SDK, then do whatever they want with the data. A non-advanced attacker can copy data out of the system manually, using a pen and paper.

In addition to this, we're not prepared to stop a dedicated attacker with a sufficient level of funding. A nation-state sponsored attack would be able to circumvent Intune MAM protections; MAM is not an appropriate safeguard to data that would fall into this category. Below that, we have seen Pen testers able to circumvent MAM protections, signalling that MAM is not sufficient to prevent attacks by industry experts. MAM should be considered sufficient to prevent accidental data leakage, or malicious attacks from non-technical, non-persistent persons.

In addition, we are most concerned about sufficiently weaponized exploits

As stated above, MAM is not currently sufficient to protect data from a dedicated, proffessional attacker. This is acceptable for a large percentage of current use cases, as it still covers 98% of the targeted workforce, being the non-technical IC. One of our major concerns is a widely published exploit, such as a JB "tweak" specifically targeted at Outlook. We have already seen this in the wild, as a tweak like this did appear on reddit. We were fortunate that we were able to deploy the swizzle detection code, which we had previously thought too volatile given the attacks we saw. This attack vector remains a continuing concern for us though, as we cannot acurately predict when or how the next exploit will occur.

Teams Channel

There's a teams channel that coordinates PM responses to incoming Pen tests.

https://teams.microsoft.com/l/team/19%3ae8406eab6ea14463bf7a13e5d3d10f6f%40thread.skype/conversations?groupId=d1f2187d-4dd5-4e5f-883e-583a2decea59&tenantId=72f988bf-86f1-41af-91ab-2d7cd011db47