Corellium

Video Webinar

Episode 2: *Mobile Vulnerabilities Exposed: Data at Rest*

Full Transcript

(00:05)
Brian Robison: Good morning, everybody. Good afternoon, good evening, depending on where you're coming from. Thank you so much for joining today's webinar. My name is Brian Robison. I'm Chief Evangelist here at Corellium, and I am honored and humbled to be joined by one of our security experts, Steven Smiley, who's going to be doing actually the bulk of the presentation today and pretty similar to our first webinar in the series. The bulk of it is actually going to be live demos and live discussion around today's topic—which is data at rest vulnerabilities in iOS and Android applications. So, welcome Steven. How are you doing today?

Steven Smiley: Thanks, Brian. I'm excited.

Brian Robison: Awesome, excellent. So let's go ahead and get started into today's session. And as I said, this is episode number two essentially in a new series that we call the Change What's Possible webinar series. And again, we're going to be focused on data at rest in iOS and Android applications today and some of the vulnerabilities that are associated with that.

(01:17):
Part 2 of this is coming next month and then Part 3 of it will be in December. So please stay tuned for those things. And just as a quick recap from last week's or last month's webinar, I want to talk about mobile app pen testing and why is it so difficult to do? Why does it cost so much to actually do it? And it's really because of—and I'll just talk through them again really, really quick—those three major limitations that impact us on mobile devices. On x86 platforms, it's really easy for us to get these things. We have VMware to virtualize them. We have operating systems from Microsoft and Linux and MacOS, and we can just run these things easily. But on the mobile device side, it's a lot different. There are limitations to getting specific devices or specific types of devices and then the operating systems and what level of access do you have inside those operating systems?

(02:14):
Are they jailbroken? Do you have access to the level of an operating system that you need? So these two are really tied together because the reason you're out there dumpster diving for older

devices is because they potentially have an older operating system on them, which might have a public jailbreak that is available to you. And then the third thing is the time. And last month we talked about the old way we used to do threat and malware research. For example, we would walk around to systems, physical boxes, and re-image them from hard discs. And it just took forever because every time you would mess up a box, you'd have to re-image it back to scratch again, and that could take an hour or so. And so with physical devices in mobile we're presented with essentially the same thing. And what really changed that world was the invention of virtualization where we could take a snapshot of that device and restore it within seconds back to a pristine known good state.

(03:20):
And essentially, that's what Corellium is bringing to the mobile security space as well. We treat these devices as virtual machines, just like VMware where you can create machines that are completely virtual. You can configure them, you can clone them, you can split them up, you can snapshot them and you can move them back and forth all at freewill. Those are essentially where you save the time with a virtualization solution. Just like what Corellium is building, it's not simulation; it's not emulation. Those tools like that aren't allowed on those platforms until you go to total virtualization. So again, that's just a real quick overview of what we do. And so I'm going to go ahead and have the honor to turn the rest of the conversation over to Steven at this time. Steven, I'm going to go ahead and stop my share so that you are able to. Let's verify that you can do that.

(04:20):
Steven Smiley: Thanks, Brian. I can go ahead and try there.

Brian Robison: Yeah, go ahead and take the share. Awesome, you've got it. And I am going to go ahead and shut up and let you go ahead and take the conversation. So folks, we do want this to be an interactive session, so if you do have questions, I will be monitoring the Q&A tab. Feel free to ask. If I can address them, I will. If not, then we'll have some time towards the end reserved for Steven. So with that, Steven, I'll let you go ahead and take it over. Thank you, sir.

Steven Smiley: Thanks, Brian. So like Brian said, we're going to talk about data storage, data at rest, how there's potential vulnerabilities with that data, how we can access that data with Corellium and third party tools, integrations that we have, and that sort of stuff. So we're going to start with iOS here. Primarily insecure data storage is all about how your application data is being stored and protected within your device. Every application, whether it be a marketing app, whether it be a financial app, is taking some sort of data from you and storing it. Now that could be non-sensitive data—where it could just be an email address, maybe your location, something very simple—to something as complicated as your credit card, your address, your date of birth, or your personal information as well. So in iOS, some of the common places you're going to find that data, your Keychain for one NSUserDefaults, application databases stored locally.

(05:57)

And then PLIST files are property files. So we're going to go over each of those sections individually, talk about them, and then just go over a quick demo of how you can kind of do that with Corellium or third party tools. For this one, for iOS specifically, we are going to be using the DVIA-v2 application. It's freely available on GitHub if you want to go install it on your own time and take a look at some of the challenges as well. You're more than capable of doing that. This is really just a vulnerable application that highlights not only local storage but a bunch of other vulnerabilities. So there's a really good opportunity to go through that if you do get the opportunity or the time to do that.

(06:39)
So first, we're going to talk about the iOS Keychain storage. So is the Keychain secure? You'll see in a lot of documentation and a lot of stuff online, they'll tell you to store your sensitive data within the Keychain, whether that be your personal information, credit card, authentication tokens, a handful of data—but just storing in the Keychain is not enough. You can configure that data protection for the Keychain items by setting this kSecAttrAccessible key. So I actually can show you here just briefly. This is on the OAS GitHub page. You guys can actually look this up as well. But there are a ton of attributes that you can apply to any Keychain item that tells the device how accessible that data is. If you need biometrics first for accessing that data, if it's when your device is unlocked (if it is [unlocked] at all times), if it's when your device is unlocked the first time after a reset, things like that. So there's a bunch of attributes, and depending on how that data is entered in the Keychain and which attribute is assigned to it, that data could be accessible to anybody who has access to the device.

(07:53)
One of the number one answers is just encrypt that data within the Keychain and then even if it's accessible to people who have the device, depending on the level of encryption, that can be quite difficult to access. So that brings us to a good question: how can someone access that data? Now, if you have physical access to the device, you can use things like iOS Keychain Dumper, which is the tool—it is freely available on GitHub—that you install on your iOS device, and you actually run that to dump the Keychain. One that I like and one that we're going to kind of demo out today is using an objection security framework to be able to pull those entries as well for your application. So what is Objection? That is a runtime exploration. It's a runtime mobile toolkit powered by Frida. So if you've done some mobile work already, you're probably familiar with Frida or even Objection, but it allows you to inspect and interact with container files and systems bypass security controls.

(08:52)
There's built-in commands to do that. Things like jailbreak detection, certificate pinning—just some various different controls. You can easily run some scripts to bypass those. And then dump the Keychain, which we're going to focus on today. There are some other options we're going to go through as we get into some of the next slides about some of the additional options beyond dumping the Keychain. But for this part we'll just focus on the Keychain. So how is that performed? Well, if you're using the Corellium Cloud, you would just connect to the VPN and

then you would ensure USBFlux is running so that you have that basically physical connection. It's as if your device is plugged directly into your machine and then you would hook the application with Objection, which I just have a command there, which we'll kind of go over in the demo.

(09:43)
But once you hook into the application, then we can kind of run from there. So we'll focus on a demo now. So on that Keychain, I actually have my iOS phone running here on Corellium. It's a 15.5 device, iPhone 7. I have the DVIA application running. I'm actually just going to connect to the VPN as I've mentioned there. That'll take two seconds. Once we're connected, we can start USBFlux here, and you'll see the devices are running one remote and then you could do things like this. So this is actually just a Frida command, which is checking the processes of the USB device. So it is seeing the USB device and all the applications that are installed and which ones are running. So you can see that I do have the DVIA application, it is running, I can grab the package name here and then I could run something like Objection, which this is the command that you saw in the slide. -g is for gadget, you would put in the package name, which can either be gathered through Frida, that command that I just ran, or you can see that within the Corellium interface as well.

(11:02)
So if you go to apps, you can actually see it right here. So once you do that, what'll happen is that the application will run on this phone and then it will hook into it here. So now I have a connection. I'm hooked into that application. So what I'm going to do is actually go over to this device, and I'm going to go to the local storage challenges. And like I mentioned, there's a ton of challenges here. So I definitely recommend installing the application if you get a chance to go through some of these. But for this one we're going to focus on the local data storage. We have your Keychain here. We can put anything in here and save it to the Keychain just as an example.

(11:47)
And then we just go over here and we'll do something like an iOS Keychain dump, and you will see that the Corellium_Secret is saved here in plain text and it is when the device is unlocked. So anytime the device is unlocked, if you run this, you'll be able to get access to that data and pull that down very easily. Now in this application, obviously I clicked a button to enter that Keychain entry in any other application. You're not sure when that's actually happening. Maybe you're saving your data locally, maybe a saved credit card, maybe a saved email address, or phone number. Really you don't know what data is being stored in there. So it's really a good opportunity not only if you're doing a pen test or some sort of AppSec assessment, but to look at the Keychain entries for all applications just to know what's being stored, how it's being stored, is it encrypted, is it not? Things like that. There was a question. Do we need to zoom in on something?

(12:55)

Brian Robison:
Yeah, there's a question. The text and things are a little small on the screen, but on your viewing side, as you're viewing what Steven is showing, under view options, you can pull down to view zoomed in modes, things like that. In the future, we'll try to broadcast it a little bit lower resolution on our screen so that we can see things bigger. But right now go ahead and just use the zooming within the Zoom webinar itself.

(13:26)
Steven Smiley:
Okay, perfect. Yeah, I can try to zoom in on something too if we need. That's no problem. So iOS NSUserDefaults, it's kind of a confusing name. What is that? So the iOS—it's basically an iOS system that allows an application to customize its behavior to match a user preference. Now that sounds like a generic sentence. That could mean anything, but really what it means is anything from themes, units of measurements, application preferences. For example, you have an application that has a light mode, dark mode—what does a user kind of prefer when they log in? Unit of measurement could be things like kilometers, miles, US dollars, Canadian dollars, something like that. So there are those differences. Application preferences could just be that a user wants to see a different screen when they log in. Maybe they want to see their news alerts, maybe they want to see their profile, different options within the application.

(14:24)
So really, what the main goal of NSUserDefaults is, is to give a user a customizable plan for your application. The ability to be able to see the application the way you want it, not just a generic application. Now when insecurely sensitive data could be stored in these—and I've seen this quite often—people think this is kind of a hidden feature, that they can store sensitive data in there while they can use it to store data. Obviously it's the same practice as your Keychain. You do want to encrypt that data because it is easily accessible to users. Now how can they do that one objection? Again, we could just run a simple command to kind of pull that data or there's actually sometimes a preference file in the local data directory. We can kind of pull those defaults as well and look at them with a text viewer. So a couple different options there. So we'll just look at a quick demo on that one. So if we go to the user defaults here within this application, we can type any text in here, Corellium_Secret or something like that, and we can save it in there. So what I'm actually going to do, I actually still have Objection running, so I can just use NSUserDefaults. So you can run this command, and this is kind of small, so let me see if I can—oh, there we go. I can make that a little bigger.

(16:00)
Okay, so what we can do is we can do iOS NSUserDefaults, and what that'll do is that we'll actually just get the defaults for this application. So you can see that the Corellium_Secret is there. That might be hard to read, but it is there. You can see things like language as well, location—there's all sorts of stuff that it's actually used for regular purposes for an application, but obviously in an insecure method, something like that. Like a Corellium_Secret could be

stored in there. That could be username, password, anything like that. And that data becomes at risk.

(16:44)
So iOS application databases, that's really kind of the next option for storing your data. Now they're usually stored within the local data directory on your device, which is only accessible via the application unless your device is jailbroken. So a lot of times you'll see in the public, people believe that information stored in your local data directory for iOS is safe. It's fairly protected, only the application has access. There's all this talk about applications being sandboxed and only being able to access their data. And while that's true—your application can only access your data and other applications can't access that data directory—what happens when you have a jailbroken device and you can access everything? And Corellium, for one, allows root access. We can jailbreak any version all the way up to 16, and anytime there's a new version, we have the Jailbroken devices ready to go for users. So if you're working with Corellium especially, you're going to have access no matter what the device is to be able to access that data. But in the public, anybody who is on a breakable version, that data is potentially at risk.

(18:01)
So there are multiple database options, SQLite, Realm, CoreData for iOS realm data. That's kind of a newer database as you'll see more and more through Android and iOS. And we'll talk about it on Android as well. SQLite has been around forever and that's probably your most common unless they're moving over to the Realm. Now obviously, data stored within plain text is vulnerable to anyone with access to your device. And like I said, with geo-breakable versions or with Corellium, that data is still at risk and isn't as protected as people like to think. So how is this performed? Well, one: identify the path to the application data directory, which can be a little complicated on iOS. So we'll kind of talk about that. You can search the local data directory for database content. Could be multiple databases, could be in different folders, so you want to search for that. There's some kind of manual intervention there to kind of go through and find some of that data. And then again, just review the databases you do find for that potentially sensitive information.

(19:17)
Okay, so iOS application databases. Where is that data directory that we kind of talked about? So on iOS, it's in VAR/mobile/containers/data/application, and then there's an <ID>. Now how do you get that <ID>? They're unique for every application. There's really two options to get that one. I mean there's a third technically: clicking through and seeing which ones are there. But that's kind of boring. There's a lot of data there, and it can kind of be complicated, but there's two main easier options. One, the Corellium file tab, which I can actually show you here if you're in Corellium. If you go to the files tab, they have this last modified column. And if, when your application was installed, maybe it was the last application you installed, you can use that last modified version when you get to this directory. When you get to the VAR mobile containers directory to the end, you could sort by date modified, find the application that you last installed

or based on the date and likely that is to be the folder of your application and look through and make sure it is the other option.

(20:27)
The foolproof option is to use Objection and run this ENV command. So it just stands for environment. It will actually show you the entire environment for that application, your bundle path, your cache directory, your document directory, and your library directory. So what you'll see from the cash documents in the library, they all have the same path and there's the <ID>, so the 057…I'm not going to say that whole thing, but that is the directory for this. So now we can take that either into Corellium, or if we're going to SH, the device to be able to actually go into that directory and pull that data.

(21:09)
And again, here's just the command if you want to hook an application. So it is relatively easy. So what we'll do here is we'll kind of take a look. So you can go to Core Data here, which is going to save a database, a local…I just want to make sure that everything is working there. So you can enter data here and save it and then you will find it…let me go to my files tab VAR/mobile/containers/data/application. And it is this one. So this is your data directory for this application. And on Corellium, you can copy this file path as well. So if you want to pull down multiple files, you want to use some SSH tool or whatever you want to use, you can just copy this file path and then pull down any data you want or you can just click on different files and you can download this stuff directly.

(22:13)
So within here we've got a bunch of directories. You'll see your library directory, you'll see a bunch of information, tons of data in here, HTTP, there's application support, which is kind of the one we're going to look at in a second. But you'll see there is a ton of data here. And if you go back, there's also documents which have some other information as well. So in this case, I've already actually downloaded this file just to kind of save some time, but once you enter this data here, it gets saved in this SQL database unencrypted. So if you actually launch this database, you'll be able to browse the data. And that is hard to see. I don't know if you'll actually be able to see that.

(22:56)
Brian Robison:
You might try command plus, plus, plus to see if you can make that text bigger.

(23:01)
Steven Smiley:
That's actually, that doesn't zoom in, but what you'll see here is just the data in plain text. So there's your email address, your name, your password, and your phone number, the same data, which is actually stored here, name, email, phone, and password. And that's stored in an unencrypted database. So what we really wanted to show here was just the ability to quickly

download this. So if you click these files, you can just download the database right away and just really quickly, you can select multiple files if you want to. You can go into your folders to quickly review all the data, and as we mentioned, there could be databases stored in multiple locations. So Corellium will give you the ability to kind of go through and click this very quickly. So we have the application support. Obviously if you go into documents, there's another Realm database there as well, and there are some other folders that have some additional data. So just the ability to kind of click through all the data, find what you need, and then from that data, just download it, review the data, and figure out where the vulnerabilities are, if there are any.

(24:13)
So PLIST files. Last common storage place for iOS is a PLIST file, which is a property file. It's really just a settings file for iOS that typically contains critical information regarding the application configuration. So most commonly, if you've ever unzipped or decompiled an IPA or an actual iOS application, you would see the info PLIST. That's probably the most common. It has to be there for every iOS application. And data contained within there obviously is if you want your iPhone version, iOS or iPad, depending on which device you kind of want it installed on, what the minimum versions are, landscape, portrait, network settings, potentially API keys, permissions, custom URL schemes, things like that. So there's a whole lot of data stored there. And then obviously within that same directory, sometimes applications do store other PLIST files that could be sensitive in nature where they have SDK implementations and they have API keys or credentials and things like that.

(25:23)
So it's always good to look at those, but that's just one location to look for PLIST files. They are also stored within the data directory. Now this doesn't always happen, but there are applications that believe PLIST files because their Apple files are not as easily readable, and that's really not true. They're just an XML file. You can actually open it with a text editor. It doesn't look very pretty. But there are tools on Mac already built in PLIST Editor, PLIST Buddy, Xcode opens it by default as well to kind of read those. A handful of tools that actually do that. So obviously storing them locally is an issue. That data becomes just as accessible as any of the other data we kind of looked at. So if you're storing API keys, usernames, passwords, anything like that, that data is kind of accessible.

(26:15)
So how are we going to go about exploiting those local PLIST as well? You got your two locations. One, your IPA. You're going to unzip your IPA, which I'm not going to show because we don't have an application that actually does that, where they actually expose sensitive data. But for an IPA, you can just unzip that and go into the package contents, the actual app file, and then you'll just see a file called info.PLIST or potentially look for other PLIST files within there. Or we can go look at the data directory, which we've already looked at, VAR/mobile/containers/data/application, find that directory. In this case it's documents within there and then you just look for that file. So on a quick demo here, we can go in and we can look for this PLIST data. If we store Corellium username, Corellium password, and we save this

in a PLIST file, if I actually go in and I go into documents, there is this userinfo.PLIST. That's where they're storing that info and it's going to launch an Xcode. Which if anyone's used it, Xcode is not the fastest tool to open. So just take a second here. But what you will see is the data being stored here as well.

(27:34)
I'll just give it one sec. And there you go. So it brings it up. It's easily readable. Like I said, PLIST, they're not really protecting your data in any way. It's easily accessible to read that data within Xcode. There are additional tools that do that as well. I think you can actually preview the files. You can actually preview the files right in Mac and you can actually see that data as well. So there are a few ways to get that data and your data is really not protected unless it's being encrypted. Now you could use a lot of the mechanisms we kind of talked about and just while I talk, I'm going to power up this Android device. A lot of the methods we kind of talked about in iOS really can be used by applications, but your data still needs to be protected in some way.

(28:29)
You do need to encrypt it. If you have a database, encrypt the database or encrypt the data within the database. If you're going to use the Keychain, use the correct keys to kind of protect that data or encrypt it in the Keychain or do both. If you're going to use NSUserDefaults, I mean typically I wouldn't store data in there anyway, but if an application is going to use it, take the next step of trying to encrypt that data, try to protect it. Because what you'll notice from this—and I think we spent, I don’t know, 25 minutes—but what you'll notice from that time is that we were able to exploit data from a bunch of different sources within iOS with minimal effort—less than half an hour. Now I know this is a vulnerable app, but those same concepts apply to any public application. So it really just ensures that your data is obviously protected because it is easily accessible, it is easily reviewed, especially within Corellium or third-party tools. So I'll just stop for a second and see if there's any questions on the iOS stuff before we jump into Android.

(29:34)
Brian Robison: There aren't any in the Q&A, but again people, if you'd like to ask some questions, please go ahead. The questions that have been answered so far, the screen is difficult to read. So if you go back to terminal, Steven, just click command plus a few times to make the font size big.

Steven Smiley (29:50): Yeah, that sounds good. So it looks like we did have one question here from Sean. What is the purpose of having so many places to store the data? Why not just show the information like themes and other config data in the Keychain? I mean, it is a great question. Really, iOS gives you the ability to kind of do this depending on the sensitivity level of the data. So what iOS recommends, if you kind of read their documentation, what they recommend is if your data is sensitive, you store it in the Keychain, right? You want keys, OAuth tokens, credit cards, passwords, whatever you need to store that's sensitive. It should be stored in the Keychain and protected. Now, how that should be stored in the Keychain is up to them.

NSUserDefaults should be used for themes and just general preferences and stuff because it's an easy place to access no overhead, it's very quick.

(30:44)
But again, people took that as the ability to store other data in there. So should it be done? No, but is it done? Definitely all the time. PLIST files, same thing. It is really the preference files that are meant to be used for data. So if you look at the info.PLIST, it's used for things like iPhone versus iPad, the orientation of your application, the minimum version, just some generic stuff for your application, some configuration options, some settings basically. But people have taken that to say, okay, well, if other stuff is stored in there, why can't I store my data in there? So they're doing it. Well, you're right, we should store the data in one. The answer really is overhead. So Apple kind of separates it—their settings, their themes, there's security stuff, everything for them. And then developers took that as, okay, well I can go ahead and actually store other data there, whether it's because it's quicker for them, or because it's easier to integrate with.

(31:44)
There could be a number of reasons. I'm not a developer so I can't really figure out why they've done it. But what I can tell you from a pen testing perspective is that tons of applications store data in every sort of way. So in a pen test or an AppSec assessment, you kind of have to go through every one because you don't know what people are going to do. And I wouldn't be surprised. You'll find stuff everywhere and that could be sensitive stuff, it could be nonsensitive, but you will find data all over different applications. I hope that kind of answered that question.

Brian Robison: Yeah, there's a follow-up question there as well. Steven, if you'd like to address that.

(32:25)
Steven Smiley: Oh, sure. When a developer submits an application to the AppStore, does Apple ever comment on things like this? For instance, a rejection application? Typically, no. Unless they see something glaring, Apple really has their policies and stuff like that. So as long as you're following everything that Apple wants, not over obfuscating, not trying to do something malicious, not trying to do something that goes against their policies, they really don't care what you do with the rest of your data. If you decide to store your stuff in a PLIST, that's up to you. If you decide to put your stuff in NSUserDefaults go right ahead. If you decide not to encrypt your data, they're okay with that. So Apple's really not going to stop you unless you're going against their kind of terms and conditions.

(33:14)
But yeah, pretty much for the majority, for everything I showed, Apple pretty much isn't going to stop you if you store your data any of those ways. We saw that Keychain data can be listed as well. So do we need to store credentials in Keychain, in encrypted format? Yeah, so really there's two steps to that. One, you should go to Keychain first for your data. For sensitive data,

that should be where it's stored. That's just a common practice for Apple and for iOS, that's normal. Now, if it is really sensitive data, we saw how easy it is to dump that Keychain. So there's a couple options. One set a key of how you want to protect that data, but there are some user issues with that because you could be requesting biometrics. So if someone's in the application and they need to load their account and see their credit card and that's stored in the Keychain, they may be prompted for the fingerprint and they might not like that. So there's a user experience thing that also comes into play. So what I would suggest and what I've seen a lot of applications do is encrypt that data just within the Keychain. Then what you can say is you can set a key that says, okay, as long as the device is unlocked, everyone has access. That's totally fine, but the data is encrypted in such a way that nobody can really decrypt it or figure out what is actually being stored there.

(34:30)
Is it possible to install an application from TestFlight in Corellium? So we don't actually support TestFlight directly. There's some FairPlay encryption stuff. I think we have an article on it. Maybe Brian can quote that if he has it handy, but we don't support it directly. So you will need the IPA whether you guys are testing through a third party and they can provide you the IPA or your dev team can pull it out and do it. And then if there's issues with signing, we can work with that. Or we've written a recent article as well that you can review about how iOS signing is done and some of the ways around that if you need to get that IPA and there's some concerns there.

(35:11)
Brian Robison:
Yes, basically due to the DRM, we can't crack the DRM for you. There are multiple ways of doing it as well as accessing, but basically, as Steven said, you need an unencrypted IPA file. Most people get those from the developers themselves. But if your customer is forcing you to use their public app store version—which is an interesting concept because if you're doing app pen testing on a public app store version, then what value are you really providing to that business, right? You're too late. Essentially, these companies need to look at building their app pentesting into their DevOps strategy so that the app gets pentested before it gets posted to the public app store. But in any case, there are ways that you can essentially get around some of the Apple encryption, but we do not do that for you. And as Steven said, yes, actually Steven did post a blog that was posted earlier this week on our blog site, so please check that out about app signing and issues with Apple and installing apps on Corellium. Virtual devices, all those kinds of things are covered there as well. So keep an eye on our blogs. There's actually a couple more blogs coming out this week as well, so keep an eye there. Back to you.

(36:36)
Steven Smiley:
Awesome. So yeah, if there's no more questions, I'll continue with Android and then we will go over some questions again at the end and we'll leave some time for that. And Brian, just to note, you're really quiet for me. I don't know if that's for everybody else, but I just thought I'd let you know.

(36:51)
Brian Robison:
Fixed.

(36:52)
Steven Smiley:
Perfect, there you go. Okay, so we'll move on to Android data storage. So Android, same thing. It's all about how your data is being stored and protected within your device. Now for Android, they don't have the Keychain, it's not as easily accessible. They do have the KeyStore which is actually quite well protected and actually the default spot that you should be storing any sort of sensitive data. But what we've seen on that topic is that there are three common places where we've seen data be stored, and that is the shared preferences, the external storage, and different application databases. So for this one, we're not going to use the DVIA application, although there is one for Android and you can actually go get one if you do want to try it out. But what we're going to try to focus on is the MASTG-hacking-playground application. This one is also freely available so you can actually get it on GitHub, but this one was kind of built to go over the mobile security testing guide from OWASP and some of their common vulnerabilities. So it does a really good job of showcasing what we want to go through today. So yeah, we'll just hop into that.

(38:06)
So the first one is the Android shared preferences. So the shared preferences is really just an XML file to store private primitive data in key value pairs. That data could be anything from Boolean, floats, integers, long or strings, it really doesn't matter. You can pretty much store everything. So you'll see a large amount of data within shared preferences for every application. Now, similar to what you saw with iOS is that, when you're in your shared preferences, there should just be common data in there, just really simple things, whether it be preferences, whether it be anything you kind of saw in iOS that was not sensitive, but what they shouldn't be storing here is obviously anything sensitive—configuration data, API keys, user information, credit cards, location, anything really that should obviously not be stored within that application. Now the shared preferences can be world readable, so it is one thing to check, it doesn't have to be, it can be actually private if it is world readable. That word speaks for itself; it's just accessible to every application and anyone on the device can access that and read the data that's in there. So obviously storing something sensitive with world readable not where we want to go.

(39:34)
So where's the shared preferences stored? Where would you find it? That's in the data directory, which if you guys have ever been on Android and kind of ran through the data directories, it's much simpler to find. So you always go in /data/data that's common. And then there's a <package name>. Now most package names are fairly obvious. You can actually go through and scroll. A lot of times it's a com.example.app or something like that. So a lot of times even just scrolling through, you can figure out the name. And then there's the shared preferences

directory. Now, similar to iOS, if you aren't aware of the package name—which iOS is actually harder to find the package name than Android—you can use the Corellium apps tab, same as we saw on iOS. You'll be able to see the package name of any app that is installed, or you could just scroll through the list because typically it's fairly obvious which one it is just based on the name of your application.

(40:33)
But those are two options. You can also use Objection as well. Same thing for iOS. I'm actually going to show that, but you can run the ENV command for this one as well and it'll actually show you. So there is that possibility. Now for the shared preferences, when they're used incorrectly, sensitive data gets stored in those XML files. Typically, like I said, generic preferences are what should be there. A lot of times what you will see there is if someone has installed an application and started running it. Let's say it's a banking app and I logged in with user one and a password. Sometimes what they'll do is they'll store that data in the shared preferences, maybe at least the username, maybe the password, even the passwords kind of less often. But a lot of times you will see things like usernames in there, maybe some API keys, maybe some preferences of whether you like biometrics enabled, things like that. So there is some additional information that can kind of be gathered using the shared preferences.

(41:31)
So I have my Android device up. This is the application, which by the way, if you've never seen it, is a really good application if you can see how many tests there are. There's just so many, but it’s a really good application to kind of install and go through if you have some time. But for this one, we'll share preferences. And what's kind of cool about this application is that you don't actually need to enter anything. What happens is these files actually get created by default when you click the challenges. So if you actually go to /data/data and then this application, see most of them are com.android or something depending on if it's a system one or if it's something you install publicly. These vulnerable ones have different names where it's like sg.vp, things like that.

(42:21)
Typically a lot of them are com.examples or something like that. So for this case, we're going to go to this app, this is the package name. We'll go to shared preferences and you will see this key XML that got added in. And within that, that is hard to read. So I don't know why it wants to open with Xcode, but what you'll see here is the password is super secret and the username is administrator. So obviously storing, like I said, password in shared preferences is not ideal. That is not where data should be stored. Now a lot of times you will see even username, which still is a risk depending on your application. Obviously if you use an email address that's kind of fairly public, so is that really an issue? Probably not. But if you use some username that you selected or that's very generic, there is a potential risk there to have brute-force attacks and things like that. So you're relying on other protections of your application to protect you in other ways. So you need to have rate limiting and things like that to actually protect against brute-force attacks

because you have already given up the username. So those attacks are going to happen. So obviously storing any sort of data is bad.

(43:44)
Moving on to that, we're going to talk about Android external storage. So new devices probably don't have SD cards, at least I think most. I don't think there are many new devices that have an SD card anymore, but every Android device still contains external storage. It's not something you see on iOS. And this can either be an SD card or it can just be added internal storage. So you will see a folder if you ever go into the file system of Android, you will see a /SDcard/storagedirectory that does have some stuff in there: documents, downloads, pictures, stuff like that. So you'll see a whole bunch of data in there. One thing to kind of note with external storage is all files are world readable. So where this comes into play, some applications do allow you to download data or store data within there, or they'll do it off the top of my head, I'm thinking of banking apps where you download your statements or credit card apps and they download locally to your device.

(44:45)
That's going to be in your external storage. If those files have sensitive information in them, account numbers, personal information, that's world readable. So any other app or person on the device can access that data. Where do you access it? Well, I mentioned it. So /sdcard is common. /storage/emulated/0 is another option. It really kind of depends on your device. So that kind of does get moved around. And one thing to note is this data, does it not get deleted when the app gets removed? So if you have data in the data directory, like your shared preferences, it's sensitive. If you delete the app, it's gone. Nobody's going to be able to access that. If you have data stored within here, you downloaded stuff or your application downloaded stuff or however it's being stored, that's not going to get deleted. So it's still at risk for as long as you pretty much have your phone. That data is always going to be there unless you manually delete it.

(45:51)
So if we look at the data here and we look at external storage, it's actually going to create a file. So if you actually go to /sdcard, there is this file called password.txt which I can actually make bigger unlike everything else. So there we go. You can see the secret password is being stored there. Now most applications, like I mentioned, do not store stuff there by default or without asking you. Typically it's a scenario where you're going to go ahead and download something from there, whether it's a message attachment, credit card statements, bank statements, things like that. So just be wary of where you're storing data and if you're storing it, just be aware that other apps technically do have access to that and can read that data. I don't think there's many applications out there going to try to look in your directories to look for your personal information and exploit that. But there are malicious apps in the play store and out in the public that are probably doing that exact thing where they're trying to look for data, trying to read data, trying to steal data. So just something to be wary of when storing your kind of data there.

(47:05)
Android application databases. There are basically two types: unencrypted or encrypted. So we'll talk about unencrypted first, and there are multiple types of databases, kind of like we saw on iOS. You have your SQLite databases. Pretty common, it's been around for a long time. You'll see it in a lot of applications. Realm: fairly new. You'll see some newer developers who are trying to move to these newer technologies that are starting to use these. And those two have some very common practices together that we'll kind of go through. And then you have Firebase as well. So you'll notice that most Android applications have a Firebase project and they are using Firebase for something, whether that's actually to store data or whether it's analytics or any number of things they can use Firebase. Now Firebase is easily tested because what you can actually do is you can take the Firebase link, just decompile your application, you can look in the Android manifest, you can look through some of the code, you will see it, you can just do strings and search for Firebase and you'll probably find it.

(48:11)
But there are a handful of ways to find that URL. Once you do, just put it in your browser, you can add /.json, and then you'll be able to see if you have access. A lot of times they are blocked. It's not very common anymore. There have been some public exploits where they've leaked data. I think I've seen it a couple of times where it's been conference apps or these big applications where they're storing a large amount of data. If they use Firebase, they don't protect the database and then somebody has access. So that is all kind of a possibility. But for the other ones, they're stored in your local data directory /data/data, your <package name< as we looked at /databases. So when left unencrypted, this data is easily accessible to anybody on the device. And I'll just kind of show that really quick.

(48:58)
We are also kind of running out of time, but if you look here, if you go to /data/data and then go down to the application…that's not the application…that is! /databases. There is this private, not-so-secure database which is going to open in a text editor for some reason by default, it has no extension on it. But if you open this with a database reader, you can actually see it's in database format as well. You will see, and actually I can make this bigger too, you will see <adminAdminPassadminAdminPassadminAdminPass. So if you actually open this up in a database reader—which just for the interest of time, I'm not going to do it just to save a couple of minutes here to test the next one—but you can see the sensitive data is stored there in plain text. You can actually not read it.

(50:03)
Now, encrypted databases, the other side of it. So obviously the answer would be to encrypt the data within the database or encrypt the database itself. Now, if you encrypt the database itself especially, that can still be vulnerable depending on the implementation. Now that's all going to be dependent on a key or a passphrase. So where is that being stored? It's probably going to be stored in the binary somewhere. It doesn't have to be. It can be stored on the web server outside and make a request for it. And that's kind of a more secure way to do it. Or you can

store it in the Android key store, but if you store it locally, it will be exploited. It's only a matter of time. If you leave something, a local application is going to be in the hands of thousands, hundreds of thousands, millions of people.

(50:53)
And if that's the case, it's just a matter of time before that data gets exploited. So storing anything locally that isn't in some sort of a proof storage like the Android KeyStore is going to be exploited. It literally is a matter of time. So we'll take a look at that. So if you look at the SQLite encrypted, there is an encrypted challenge which will create this encrypted database. So we can actually download this. And what I'll do is I'm actually going to, for some reason that's—sorry, give me half a second because I dunno why I did that.

(51:37)
Okay, so it prompts you with a password, right? It's not going to let me in the database. That is step one. Now where is that password? We can kind of take a look for that. I won't go through all the mechanisms as you search through. And as you do more reverse engineering, you'll see how to go about searching for some of this data, where to look for secrets and things like that. In this case, I've already done the research so I know where it is. If you actually go to /data/app and then I believe it's this one, and then lib arm. So basically this application in their app directory as well, not just their data directory, they have these shared objects. So there's something called libnative-lib, which is a shared object on Android. And if I can't exit because I can't spell, so if you do strings on lib-native, so if you run—trying to see if I can zoom in here.

Brian Robison: Yeah, in Terminal if you use the command plus.

Steven Smiley: That's what I still did.

(52:51)
Can anyone see that? You guys can see that maybe?

Brian Robison: Yeah, it's making it better.

Steven Smiley: Okay, so if you run strings on the command, you'll get a whole bunch of strings. And actually, if you look through here, there is this secret string which is being stored here within this shared object. Obviously this developer thought that nobody would look through their shared object or nobody would actually look through their data to find that string. But once you do, you enter it in, get access to the database, and then there's an admin that's also kind of hard to see, but there is an admin password that is encrypted, now decrypted and ready for use. So obviously there's another level here because you're going to have to do some reverse engineering. You're going to have to find that key, whether it's stored in actual, in this case it be a shared object, could have been stored in shared preferences, could have been stored in another database, could have been stored anywhere. But obviously anything that gets stored is eventually going to be exploited. In this case, we found the secret string, and we're able to decrypt it. So obviously using the KeyStore would've been the better access.

(54:15)
That kind of wraps it up. So we'll kind of leave it for questions. We got four minutes so we can ask a few questions and kind of go through some stuff.

(54:28)
Brian Robison:
Awesome. Steven, looks like there's one question in the Q&A.

Steven Smiley (54:33):
Without route access, can the data in the shared preferences be accessed by another application? No. Quickly, the answer is no. Data is what iOS and Android have kind of done recently is gone with the sandbox approach where your applications are kind of protected in a way. Now, if you decide, if the developer decides to store shared preferences as globally readable, any application will be able to read that data. That's just on them. But typically no, if they make it private, they won't be able to see that data. Obviously if you have a rooted device, which on Android is far more possible than on iOS, that data is very possible. Anybody's Android phone can be reflashed, add a new firmware, jailbreak it, root the device if you need to get access. So it is more possible, right? It's not possible on iOS, you have to have specific versions. Corellium kind of allows you to do that, but in the public it's not as possible. So yeah, it just depends kind of how that data is stored. If it's world readable, accessible to everybody or if it's kind done privately. But if proper development practices are kind of taken into account that data will be protected from other options.

Brian Robison (55:56):
And just kind of on this too, Steven, and I think that what a lot of people run into is not necessarily bad design or bad engineering. A lot of times there are shortcuts taken during early phases of development that are meant to be gone back and fixed. And sometimes that doesn't happen. And that's how things like AWS credentials get shared in public apps, things like that. So a lot of this stuff is really just double checking that engineering developed the app properly, protecting the data properly, but as an app pen tester, being able to actually verify that and not relying on device security is just good practice. Hands down, it's never a good idea to put all of your data security eggs into one basket because as we all know, device security is not impossible to overcome. And I think Steven, there is one last question that we have time for before we need to end for today.

(57:09)
Steven Smiley:
Sure. As a tester, is there a way to identify whether the shared preferences is global readable by doing static analysis of the code? Yes, there are a couple ways. If you're using Corellium within the file browser, as you're going through, you can actually see the permissions and then you can

see what permissions are actually set, who has access to that, if it is world readable. If you're going through, you could use a DB to kind of do that. Once you're in there, you can access the actual file structure and then you could check the permissions of the file. No different than you would on a regular terminal or command prompt where you'd actually list the files and list the permissions kind of associated with those. So there are a couple kinds of options for taking a look at that. And then if you're running it through some sort of test, you could use something like MobSF. If you want something more automated like a MobSF Scanner can also flag those kinds of vulnerabilities as well if you have permissions. But it just depends how you want to look at it or what kind of tools you're using as you go along.

Brian Robison (58:21):
Outstanding. Thank you very much, Steven, for all of the wonderful content from today. Thank you everybody for hanging out on episode two of our webinar. Again, check our Events Page at Corellium.com/events for recordings of past webinars as well as where to sign up for new webinars that we're going to be doing a month from now. The week before Thanksgiving here in the US will be our next webinar and we're going to be discussing vulnerabilities where data is in transit, data in motion, so over the network. And Steven and I look forward to seeing you on our November webinar. Thank you all very much. And have a fantastic day.

#####