Corellium

Video Webinar

Episode #4: Mobile Vulnerabilities Exposed: Reverse Engineering

Full Transcript

Brian Robison:
Good morning, good afternoon. Welcome to our webinar. This is our final webinar of 2022, and we'd like to thank you all for taking the time out of your busy day to come listen to our third episode of Mobile Vulnerabilities Exposed. This is our reverse engineering session. I am Brian Robinson, Chief Evangelist here at Corellium, and I have Steven Smiley, one of our researchers joining us, and he's going to be doing the bulk of the presentation today. So again, thank you so much. Happy holidays. For all of those of you who celebrate, we do appreciate you coming out. There were two other sessions in this track. The first one was on data at rest and the vulnerabilities that could be posed in there. And then last month in November, we looked at data in motion and both of those previous webinars are available on demand on our Corellium events page.

(01:03)
So let's go ahead and get started with today's webinar. A little bit of housekeeping, everybody is muted. We are in webinar mode, but we do encourage engagement. In fact, we actually have a few polls for you to participate with today. So we look for you to engage with those. And we also ask that you please ask questions using the Q&A tool widget. We would like you to see you participate and get some good questions from you, and we've had some fantastic questions in the past, so please continue to do so. The webinar is being recorded. We will make it available shortly after the session, and this presentation does consist of both slides and demonstrations. Now, some of the demonstrations, the fonts might be very tiny, but that's usually because the tools we're using may or may not respect screen size and blowing up fonts.

(01:55)
So we will spend a lot of time kind of explaining what we're doing and how we're doing it so that you can get an understanding of it and perhaps later during the recording you're going to be able to see that and make better view of that text. So we do apologize for that. First off, I would like to begin by announcing our new Corellium Contributor Program. It's at Corellium.com/contributor-program, and this is a way for mobile security professionals to actually get paid for contributing content to Corellium. And you can see here there's lots of

things that we'd like to have. We'd like to have research examples, topics in app pen testing, IoT and R&D. We want demos, blogs, tutorials, even co-presenting with me on these webinars, things like that. It is a lot of things that are available to you and you can get paid. You can see here some of the compensation that's available depending on the type of content that you are working.

(03:04)
So hit that website. If you're interested in creating content, maybe you've got some original research that you're looking for a way to publish, we can help you do that and we will be happy to pay you as well for your time and your efforts. So please look into that. We'd love to see you become a Corellium contributor and be part of the team. So hopefully we see you soon. Next, I'd like to announce that we have released our Mobile Malware and Mobile Threat Research Solutions Guide. And really these are, it's the same platform that we use for everything else, but it focuses on how to use it in the world of doing dynamic analysis on mobile malware and doing mobile threat research. Now, in the past we have been limited in mobile malware especially because of the device issue of the time issue and things like that.

(03:58)
And so now we have the ability to actually do this in a virtualized environment. And virtualization made a huge difference back in the day with x86 malware research because we no longer had to spend our days restoring systems and doing all these kinds of things after we detonated the malware. And we can do the same thing for iOS and Android and even other Arm platforms with our Arm virtualization platform. So I encourage you to hit the website, check out this new solution brief if you're in this area of security research and how we can here at Corellium really help you speed up and do things that we've never been able to really do before in the mobile threat space. So just for those of you who haven't been around Corellium for a while, I just want to explain very quickly why is security research so hard with mobile?

(04:52)
And there's a lot of reasons. I mean, one, these operating systems aren't designed to be viewed and introspective at the levels that we normally would, but really it kind of boils down to three different areas that I'd like to focus on, and that is access to the devices. And that means a wide range of iOS devices. It means a range of Android devices, both iOS and Android in the same platform with essentially the same tooling, all those kind of things. But most importantly, we're looking for specific operating systems on those devices. We're spending a lot of time going out there and sourcing older devices because they have older operating systems on them that potentially have easy ways to jailbreak them to get rooted access on them. But basically we're looking for very specific devices because to do security research, we need access to the operating system at that root level.

(05:50)
And if we don't have access at that root level, then our visibility into data at risk, data in motion, things like that is very, very difficult and is severely hampered. So finding specific devices with

operating systems and specific operating systems that can be jailbroken is another reason why this is so difficult. The third is if you literally take the time that it takes to do all of that kind of stuff—procuring devices, upgrading them, downgrading them—all these kinds of things that happen, this is just a time-consuming effort. If you're a global team, you're shipping devices around the world potentially and you're jailbreaking them when they land somewhere and it's just a time consuming effort. And that is the third limitation really, because I call it a distraction. And basically every minute that you spend not doing security research is a minute that is wasted from advancing your program forward.

(06:50)
So how does Corellium help you solve that? Well, really because we give access to any type of virtual device that you want, and we are a virtualization platform. We are not an emulation platform or a simulation. We run true Arm-based operating systems like iOS and Android on Arm CPUs. We run them directly through our Type 1 Hypervisor and we're the only Type 1 Hypervisor designed to model mobile devices and other IoT Arm-based devices where we're actually modeling the device itself, not just a generic platform designed to boot a server or something like that. So you're not going to be running Windows Arm or Linux Arm on our platform, but you are going to be running very, very specific device types like the iOS and Android devices. So what really sets us apart is the time savings that this virtualization platform gives you. As I mentioned earlier, when doing mobile malware and mobile threat research back in the day, we used to have to physically image these systems and restore them from physical hard drives.

(08:00)
And you could literally spend an entire day or maybe even a better part of a week restoring a whole lab back to known good status on the hardware. And so when VMware was introduced and you could all of a sudden right click and say restore a snapshot, it was so much faster and it allowed us to scale so much more when doing that mobile research or doing that malware research for instead. And so what we do with Corellium is we bring that scale and that power of time savings to the virtual device world. So probably the best feature that I like the best in the platform is the ability to save a snapshot of a device at a specific point in time and then be able to restore that device and its OS and everything else back to a very specific point in time that makes my research go much quicker because I'm no longer having to reboot the device, potentially reflash it if I've corrupted it somehow and jail break it and do all these kind of things.

(09:04)
I can literally within just a few seconds have my device backup where it was before I began my testing by using that snapshot feature. So it's a fantastic feature to use and we really look forward to you being able to use it. Now, there was an issue during the recording yesterday. In fact, I didn't get the recording started on time, so we missed a few of Steven's early slides. So I'm going to recreate the slides for you and then we're going to jump over and pick up where Steven left off. So I do apologize for that, but I try to get you all the content here. So Steven's going to talk about reverse engineering and the process of disassembling and decompiling a

mobile app to reveal its code, its internal logic and other things like that. So in today's session we're going to learn about some techniques and topics around reverse engineering.

(10:05)
We're going to look at the iOS and the APK file structure and we're going to look at some basic reverse engineering principles and techniques for both iOS and Android. And then Steven's going to show us lots of demos and examples. We're going to use Frida, we're going to use a lot of different tools that are in this area of static analysis. And really we've focused on dynamic analysis for the last two webinars because that's where the Corellium platform really shines is in actually running the code. But we would be remiss if we didn't focus as well on probably one of the equally important concepts of reverse engineering and looking at the source code itself. So what is reverse engineering? Essentially reverse engineering is the process of breaking something down to understand how it works. Mobile apps—basically you need to deconstruct them, analyze them, observe the compiled apps to understand the functions and things that they do.

(11:04)
And there's really two types of reverse engineering dynamic, which is again where Corellium shines because we're actually running the app on a virtual device. You can attach debuggers to it, you can look at its behavior, you can interact with the application, you can do all that kind of stuff. And then there's static reverse engineering as well. We're going to look for strings, we're going to look for URLs, we're going to look for interesting things that are in the app that we would see potentially while the app is running in dynamic mode, but we can look at it from a static standpoint, look at the source code. So to do this, we're going to gain a better understanding of the application along with the inner workings of some of the logic. We're going to specifically look for hard-coded values that are stored within the application as a lot of things like AWS hard-coded credentials are leaked in mobile apps especially.

(12:00)
And maybe it's just because the engineering didn't remember to go back and remove it or obfuscate it, but it is something that's happened a lot. There's recent news stories about it. We talked a little bit about it back in our September webinar where we introduced the Corellium platform to y'all, but it's a big problem and a lot of times it's just forgotten, right? The engineer means to go back and remove these hard-coded credentials, but in the end they didn't get back to a time or just didn't make the cut or whatever. So being able to find those hard-coded values in the application is something that we can do with reverse engineering. Also looking for specific things to exploit in the mobile application. Perhaps there's function calls and things like that that we can actually spoof some information and get a different result out of it, or we're looking for the output of that function and we're able to then take that value and do something with it so we can look at different ways to exploit the application and then we can move into decompilation versus disassembling.

(13:13)

This is a big discussion area and what is the real difference between decompiling something versus disassembling something? And on the Android side, it's easier to get back to really good human-readable code because the decompilers that exist basically translate the compiled binary low level code into human-readable high level code. And this could be converted back pretty decently into maybe the original programming language. And depending on the level of obfuscation in an application, a lot of times a good chunk or if not the vast majority of that source code can be revealed and back into human-readable format where we can actually see some of the names of these function calls and things like that. Now this assembly is a little bit different. Unfortunately it doesn't provide as much level of visibility into the original code and this is what is utilized for iOS applications. And essentially that machine code gets converted back to human-readable assembly code. So it's a lot of best-guess type of things to get converted back. It's way more complicated. And with that, we're going to go ahead and join back into the webinar that was in progress yesterday. Again, I apologize for screwing up on the recording, but the rest of the webinar is Steven talking about these concepts. So please enjoy the rest of it. Thank you very much.

(14:42)
Steven Smiley:
…and a handful of different tools. You could use those to gain information that can be used within the disassembly tools to kind of make it a little bit easier because it is more difficult. So we'll talk about the Android file structure. So the Android package here is the APK contains all the data in the form of a zipped alvik executable file. Now you've got, when you unzip an APK file, whether you use APK tool, which is a great tool to do that or whether you just unzip it, there is a handful of folders in there.

(15:25)
So the META-INF folder, it just contains verification information that's generated when the app is signed. So you see some signing certificates there. If you're looking at Android vulnerabilities, like different signing issues, you can look in there for the certificates and see them. There's a lot of other tools that pull up that information as well that you don't actually have to manually go in and do that. If you look at the assets folder, it just really contains assets. The developers bundle with the application. That can be images, videos, documents. Sometimes it's good to take a look at just to see what's there from the developer perspective. Maybe they stored something they shouldn't, it doesn't happen as often. The Lib directory just contains native libraries with the compiled code for different device architectures. So you can go through there, look at some different libraries. There are some different files. I've seen use cases where actually sensitive information or keys actually get stored in the libraries. A little strange, but it does happen.

(16:19)
And then you got the Res, which is just the resources that contain predefined application resources. It could be XML files that define network connections, interface layout, a handful of things. Really good folder to go take a look at just for some generic information and potential information relating to your network configurations, which we talked about in the previous

webinar. You have the Android manifest, which I'm sure everyone is fairly familiar with. It's just a manifest that contains the app's package name, activities, resources, permissions, things like that. So there's a lot of information in there. Obviously a good place to start in any assessment is to look at that. And then you have the Classes.dex, which contains all the Java classes in a .dex file format. For iOS, much simpler, there's not a whole lot of information, but iOS iPA files are just a zip archive. They can be renamed.

(17:10)
I've seen some people do that. Just rename it to a zip and then unzip it or you can just unpackage it with any really unzipped tool, Mac, Windows, wherever you want to. But within there you have the App Binary itself, which is the executable file, that compiled unreadable source code. So that's what's used as part of disassembly. You could use that actual binary to be able to load it up into a disassembler or DRE or whatever tool. You want to be able to take a deeper look. You have the info.plist, which has the configuration information such as bundle ID, the version number network configurations, a handful of information and potentially sensitive information if your application's coded wrong. You also have the framework folder, which is a list of dynamic libraries for your application. And then your Embedded.mobileprovision, which is your certificate for application signing, which is also the file that's going to block you if you are incorrectly signing or incorrectly installing on a certain device.

(18:09)
So we do have ways around that. If you check out our previous blog posts and webinar and take a look, we really talk a lot about the Embedded.mobileprovision file tools for reverse engineering. And by no means is this an exhaustive list. There are other options, there's other tools depending on what people like to do. For Android, there's a lot less because there are some really good tools. JADX being one, which is just a Dex to Java, decompiler. APKTool, you can decode and rebuild APK files. There are things like DEXtoJar and JD-GUI, things like that. Those are kind of outdated as JADX came into play. That one's just kind of better for the most part. For iOS, you have things like Otool, NM, you have Strings, Hopper, there's a bunch of Disassembles, IDA Pro, Radare, Ghidra, BinaryNinja, a handful of options. We have a quick poll actually I believe Brian, just around the tools and what kind of people are using currently. I have a feeling the Android one, I know the results are almost unanimous. We'll put the poll up really quickly. I have a feeling this is going to be pretty unanimous. The iOS one's going to be more interesting I think. So we'll just give 2 seconds.

(19:28)
Brian Robison: If you're using other tools, please send us the name of that tool in the Q&A widget if you would, because what we'd like to do is potentially put together webinars on these tools and their usage and things like that in the future. So if your answer is other, please do send us that in the Q&A and we can look at adding. We have a lot of guides and things like that on our support site and would like to always add more.

(20:01)

Steven Smiley: Yeah, lots of stuff coming and part of these poll questions, they're not just for this webinar to get information. It’s to get information just around the industry and what people are trying to use so that we can guide some content and some future creation to more advanced webinars and blog creation and things like that. So for Android, it looks like the majority, and you can put up the next poll while I talk about this, but the majority was on JADX. There was some on jd-gui and Dex2jar that is an older tool. Depending on your research or if you're going online, you may see some references to Dex2jar and JD-GUIi. JADX is much quicker. You'll see some demos we're going to go through that just show kind of how quick it is and how really good it is for that tool. But yeah, good to see people are still mixing it up a little bit.

(20:48)
So yeah, we're going to do the iOS reverse engineering question. We're just talking about what your favorite disassembler is. There's so many different options. This kind of depends if you have a license, if you're a corporate customer, things like that. There's a handful of people who use different things. It looks like iOS one came back, it's pretty split. Hopper is up there, just barely ahead. Ghidra obviously being a free option is quite up there, which I expected. IDA Pro still has quite a few votes. IDA Pro being more of an enterprise level tool, you don't see, depends on the consulting level, but IDA Pro can be tough to get your hands on. So I'm not surprised to see Ghidra and Hopper towards the top, but we're going to kind of look at Hopper really quickly today. We're not going to spend a whole lot of time on that, but we do have a quick demo on that.

(21:38)
Brian Robison: And one of the folks sent into the Q&A when they answer others–

(21:43)
Steven Smiley: R2, yeah, we should have put that in the list. That was an oversight on my side.

(21:48)
Brian Robison: And VS Code, Visual Studio Code with the APKLab plugin. That's a cool one. And also just so that there is also a Corellium tool with VS Code that allows you to do things like start and stop devices and other things like that directly from VS Code. One of our researchers wrote that. So that's actually kind of a pretty cool integration of that tool.

(22:13)
Steven Smiley: Yeah, yeah, definitely. OK, so we'll get into some demos. We got about 25 minutes left so we got some demos as well to go through both for Android and iOS. So for the first one we'll kind of look at the hard-coded value just being stored locally. So for this demo we're going to be using the injured Android, which is available on GitHub. There's the link. If you haven't tried it before, feel free to try it. It's a really good tool, especially for Android. There is a ton of options for vulnerable apps. So this by no means is the go-to. There's a lot of different options, a lot of different use cases. This one has a good one for reverse engineering, you get

this code. So it just felt like a good option for that. So I'm actually going to load up my Corellium Android device.

(22:59)
I have the injured Android actually running here and congrats I got the flag, but let me just kill that. So this FlagOneLoginActivity, you enter a flag and submit, nobody knows what it is. By default you would kind of take a look that's obviously incorrect so you don't know what it is. How are you going to find that information? Well, you can use some reverse engineering to kind of take a look. There's probably some other ways to also look for this, but what we're actually going to do is I'm going to load JADX right here so we can kind of all take a look together.

(23:40)
That's not it. Oh, one sec. I'm in the wrong directory and then we'll load injured Android here. So I'm going to load this up in JADX. I'm going to try to zoom in as much as I can on most screens so that we can kind see. If you guys have any questions, if that's something you can't see, especially when we get into the Hopper, just feel free to put it in the Q&A. We're more than happy to answer that or elaborate on anything like that. So for this one, you see all the code here. There is some obfuscation, you see A, B, C, D. Typically we're going to go into the actual application itself to look at that code. So we look at the injured Android and there's a bunch of stuff in here. There is some obfuscation, A, E, F, but you do see a couple flags here.

(24:27)
So Flag18Activity, FlagFive, FlagNine, FlagOne. So if you actually go back here, we're looking at FlagOneLoginActivity, right? So we'll try to look and see if there's anything associated with that. You actually look here, there's a class called FlagOneLoginActivity. So you can actually go take a look and then you just look through the code. So in this case you obviously see a reference right here. The flag is right under your nose and you look through some of the stuff they're doing and then you just kind of look down. There's actually right here, they actually call it out as FlagOne. Now there is a couple other ways to do it. That's a super easy way to look for it. It might not always be that easy to know which class it is.

(25:16)
The search functionality within JADX is pretty good. So you can obviously look…let's see if that's going to process quickly. If not, that's OK. It might actually freeze. OK, there we go. Sorry, apologies for that. That took two seconds. That was having some weird issues there. But within the search functionality you can search for a number of things and this should be used kind of in every pen test depending on what you're searching for. In this case I search for the word flag. You can see a bunch of stuff here, a bunch of information, there's a hundred or more. So there's lots of things to go for. I'm trying to think what else was in there. FlagOneLoginActivity. So normally in a pen test you kind of look, you could look for things like secret or SecretKeySpec is really good, if you're looking for crypto. You can look for things like that where you can get a lot of information just about how the application is processing, how the application is doing crypto,

how it's doing decryption, anything like that. And JADX is having all sorts of issues today where it's just continuing to fail.

(26:28)
That's okay, that demo's done anyway, but then you just enter the flag. So the same way you found it, you just enter it in here, hit submit and it will solve that flag. Same as you see here: congrats you found the flag. Android repackaging app repackaging. So as you're doing reverse engineering, one thing that you could do, part of iOS or Android, but a lot of times it's done kind of more in Android, is you can actually repackage the application and modify the code. So there's a handful of reasons why you'd want to do that. It could be bypassing a security control or certificate pinning, root detection, things like that. It could be patching a certain functionality to pop out, maybe a password, a window. There's a handful of reasons. Not a lot of applications do that anymore, but there are a lot of use cases where you're trying to bypass potential security controls or look for additional information. So one good example of that, if you use the UnCrackable application, they actually have four levels. We're only going to go over one of them today just in terms of time, but go check it out if you guys want to challenge to solve the rest of them. But we'll go over level one, which if you look at it, actually I have it here.

(**27:46)**
If you launch the application right away, it's going to have root detection, which is unacceptable. It says the app is now going to exit. Now there's a couple ways to bypass that. One, you could use a Frida script. Now if you use a Frida script like that where it checks on launch, if you do that through Corellium, you'll actually need to either do it outside of Corellium, which is one option with your Corellium device. It's totally possible or you can actually use the console directly in your device here. If you actually go to this console, Frida is actually running here. So you can actually run Frida right through here.

(28:20)
Sorry, I'm just waiting to get to the actual command line. But yeah, you can run Frida. Frida tools are all right here. So you have Frida, FridaTrace, everything is all running right here as well. So you can actually upload a script to your device and just run it this way if it checks on launch because this actual Frida console, you need to actually hook the process. If you see here, if I launch this, it's already too late, it's already checked for root detection. So if I hook the process, it's not going to be able to actually bypass that. So that is one option. Or you can manually modify the code as I mentioned, right? So you can decompile the application, modify the code, resign it, and then install the application. So if you look at this one here, I'll actually decompile the Android application, which is using APKTool and then the d flag and then just decompile the application. It basically unzips it.

(29:13)
You can go into the directories and then you just find the root detection code. So similar to what I mentioned in the last demo, if you use JADX you could search for various things. Now if you're looking for obviously root detection, you're going to look for things like root or /sbin or

/exposed.su, different things that potentially you would see as they're detecting root detection. And then you would just find that level code. Now once you find that code, you can actually go into the Smali code into the directories and give a couple options. You can either manually edit that if you have knowledge of small code, we could change it to be true if it shouldn't be or be false in the case that it actually found something. You could do that. You could change out the classes they're looking for, the binaries they're looking for to different names that they aren't going to find you could actually delete the code.

(30:10)
That kind of depends on how intertwined it is with the rest of the application. Maybe deleting a part might not work because if it's intertwined with the rest of the application you might get a bunch of errors. So there is a whole lot of trial and error with that. If you actually look in the top right here on this slide, you'll see this is the code in the main activity where it was actually doing the root detection and there was additional code beyond this. So you can actually go take a look at this if you have the binary and then you just resign the Android application so that we can install it. So if you're resigning APKTool to rebuild the APK, there's the command APKTool and then the B-flag -F-D just so that we have a directory.

(30:51)
And then just call it the binary. The new APK automatically gets generated in the directory of the decompile folder you already had. And then you just, we need to re-assign it. You just generate a local Keystore. This is just using my name. So Android signing Keystore, you can change that to whatever you want. There's an alias, you could change that to anything you want and you can change the keysize. I just use 2048 just as a quick example and this will work. And then you would sign the APK with that Keystore you created. So you are calling that Android Keystore again. So make sure if you change the name, you're changing it here as well and the alias. And then you would just use jarsigner to verify the APK. And then lastly you just use Zipalign for optimal loading, make sure everything's going to work on every device and then you would just install it. So if you actually look, I won't go just for time purposes, I won't go through all that. All the commands are there. So you can look back at the recording and grab any of that if you need it. But if you actually look here, I have a different device here where I've actually patched it out already and you can see there's no route protection.

(32:04)
So beyond that, there is now a secret string you need to solve, which gets us kind of into the Frida conversation of things. So if you actually look, this is the code that came out of JADX as you see from the screenshot. So SecretKeySpec as I mentioned earlier, really good opportunity if you're doing reverse engineering for Android, look for things like SecretKeySpec. A lot of crypto is handled through that and it can kind of guide you right away to classes that pertain to any sort of decryption. So that could help you obviously look for potential hardcoded IVs, potential hardcoded data or just understand how they're doing decryption and then be able to write a script like this. So here's just a quick script that actually finds the class. It calls the function, it converts the return byte array to ASCII and then it leaks it to the log.

(32:59)
So we can actually load that up here. So if we actually go to Frida, select the process, now we can actually, like I mentioned before, with the root detection being checked right away, we weren't able to hook it, but now we can now that it's bypassed. So we hook onto that. I have a script here running which the script's available in the slides as well, so you guys can get that as well. But you can see the script was loaded. If I put a secret in here, anything I want, if I put test for example, you're going to see it returns “I want to believe…” which is the answer in here. And that will actually solve that. iOS reverse engineering. We'll just quickly talk about this, we'll just use the iGoat-Swift IPA to kind of look for some hardcoded secrets as well. When it comes to iOS reverse engineering, there's so much to do and you get into disassembly, it kind of gets super complicated.

(33:58)
So we won't dive a whole lot into that. Just to talk about some generic methods for that and what you can do. So here's an IPA you can go download if you want some practice, you want to go take a look. Now if you actually look here, so one thing you can do is just run the strings application. You can also run Otool and things like that. There's a couple other tools to gain some basic information, which you always should be doing prior to putting it in a Hopper or Ghidra or something like that. The more information you can get, the more insight you can get into the application, really guides how you use Hopper and these other tools to gain even more information. You know what to search for, you know what to look for, things like that.

(34:39)
So if we actually, I have iGoat actually running here. I have a folder, so I actually already unzipped all this, but so you just go into the folder, into the app folder itself. You can see the binary is right here, iGoat-Swift. So then you can run strings on and then look for things like—and this could be anything you can search for in a normal binary. This could be not just secret, but things like password, any sort of crypto thing you're looking for, any key search terms you have, anything like that you could search for. This is just generally to get an understanding of the application. You can see here there's something called “You found the secret!!” So that's obviously a really key thing. We want to look for this application. This could be vastly different for any other application, but in this case what you would do is you can actually open this up in Hopper or Ghidra, whatever your favorite tool is, really doesn't matter.

(35:38)
And by the way, this is going to be hard to see, so I'll try to explain this the best I can to try to avoid this. But it's processing everything right now with the binary. Looking at everything up here, you have a couple of different options to look for, label strings, things like that. So one thing you can look for now that you have some insight using things like strings or Otool or whatever the tools you're using outside of the disassembler, you could look for things like secret. And then you see “You found the secret here!!” which is I know probably hard to see again, but it does say “You found the secret!!” and that'll bring you down here. You can also look for

references. So in this case, I'm going to look for a reference to the “You found a secret!!” There's one I can go to that.

(36:26)
You can see “Congratulations. You found a secret!!” That could be extremely hard to read. You can also switch to things like pseudo code where you can actually read it a little bit better and get a better understanding. So you can see, well you probably can't see, but it does say “Congratulations, you found the secret!!” There is some information here. So this is just—generally Hopper is a really advanced tool. Same with Ghidra or any of these Disassembles. There is a lot of complicated nature to it in terms of actually reading the code, understanding what's there. But the goal of reverse engineering with iOS or even Android is just to gain as much information about the application as you can. That's why using things like Strings or Otool or any of these other ones like NM or there's a handful of different tools that you could use to gain an insight and then use that information and what's being done, what crypto are they using, are they using any secrets, are they using things called password, whatever. And then use that information to search for things in here to get an understanding of where it could be. And then start looking through the code to gain more information. And this could lead you to potentially be able to build out additional scripts, potentially Frida scripts, use R2 to patch out the binary if you need to. There's a handful of other more advanced things you're going to move on to from here. But this is kind of just obviously the start.

(37:55)
I'm not going to say that that was the demo. So we'll talk about Frida really quickly. Frida is obviously a dynamic instrumentation toolkit for reverse engineering security researchers and mobile pen testers. So why should you use Frida? What can be done? You can observe and trace application functionality using Frida-trace. You can execute code in the application context, explore the application internals without the source code and modify the behavior of an application at runtime. You saw some of that as we did the Frida demo with the UnCrackable. So how does it work? Well, on jailbroken root devices, they use Frida injected mode, which the Frida servers run on the devices. You can interact via USB or the TCPIP stack, but Frida injects a shared library into the target process. And then JavaScript is sent from the Frida tool and then executed in the context of the process.

(38:51)
So there's a handful of tools beyond this that leverage the Frida API, things like Frida-ps, which you can list processes on your device as well, which by the way, you can get a lot of that information right at a Corellium just on the apps tab. But Frida-ps is a really good opportunity to gain that information. If you need process IDs, package names, just the name of the application running, you can use things like Frida-trace, which can dynamically trace function and method calls, which we'll talk about quickly in a second. And then Frida CLI, which you can run JavaScript within a target process. So within Corellium we have all of this, but the Frida command line and the ability to write a script and run it really quickly against an application a second to none, you won't find anything else as quick or as stable to be able to do all that stuff.

(39:40)
And then obviously with development experience, you can write your own tools, scripts, anything you need to with Frida because everything is open, so Frida-trace, it can be used heavily as another opportunity to gain additional insight to reverse engineering or to potentially leverage yourself to perform additional attacks. So, Frida-trace is just a component used for dynamically tracing application function calls. So when Frida-trace is started, it'll automatically enumerate the functions for that given module. Now here are a couple examples just as some generic information. This isn't exhaustive by any means. You can use Frida-trace to do a ton of different things. It can be very customized depending on your application. But here are just some generic things you could do. If you're looking for the network, you look for HTTP calls, you look at the local file system, how is your application actually working with the local file system, the NSFileManager, things like libcommonCrypto look for things like that for crypto functionality. But again, there's tons of other opportunities as well. And just as a quick example, the iGoat-Swift application, it actually has a secure messaging system which can use Frida-trace to gain information. So if you actually look, let me go to that iOS device. So this is the iGpat application, it's already running. They have this, sorry, lemme just start this as well.

(41:16)
One second. I need to connect to the VPN. OK, so that means a cloud instance as well. But they have what they call a secure messaging, obviously not secure because it is an insecure application. And the point of this is to kind of target this, but if you put some fake information in here, random message doesn't matter. And send this, it's actually being sent over the open URL and you can actually, let me see if this,—sorry while I do this, Brian, there was a question too. Can you read it?

Brian Robison: Yeah. Does the Corellium Frida feature depend on the Frida developers when a new iOS version is released? Do we need to wait for the devs to update Frida to work with new iOS versions?

Steven Smiley: So we just—do you want to answer this or do you want me to answer this?

(42:14)
Brian Robison: Go ahead.

Steven Smiley: Yeah, so we just updated. So for our devices we're on what, 16.0.2 I think?

Brian Robison: Yeah, I believe.

(42:21)
Steven Smiley:
I believe so. Yeah, so we're pretty new. So we're not the absolute latest. We don't stick with the absolute latest obviously in terms of there could be bugs, there could be some issues with that.

But we do try to keep up as best possible to the latest releases. So we obviously fully test that, make sure everything's working, make sure everything's running, and then we make those migrations to the newer versions to keep up. I think publicly right now they're a little bit higher. I'll actually check my free version. I actually just updated mine I think to the latest, but so Corellium will always be a little bit back, but in terms of the major version, they should always be at the latest as those releases come out. And there shouldn't be any issues with that. But if you are having any concerns about the Frida version that's being run on the device, you can actually use your own Frida like I'm doing now.

(43:13):
You don't actually necessarily need to use the console; it's directly in the Corellium virtualized device. You could take that outside and use USBFlux for iOS or use it for Android and be able to use your own Frida and your own Frida version. So that's a possibility as well if you have compatibility issues with other tools or anything like that. So quickly here, just while we were answering that question, I just ran. So what I'm actually doing, I'm using USBFlux to connect to the virtualized device, which is this device running. I did a Frida-trace on this open URL for the iGoat-Swift application. It had said it was tracing one function and then I actually went ahead and actually sent this message. And then you can see that the application is actually using that. Here are some pointers. So you can use this information and Frida-trace to gain information about what the application is doing, where it is doing these things, and then take that information to your other tools to be able to do more advanced analysis.

(44:15)
So you're going to take it into Hopper or Ghidra and start looking at these locations and start seeing what is the code doing, potentially build something else, build an additional script. But you do have that functionality. And this can be done right in the virtualized device as well through the console. I just already had it set up right here as well. And Frida, lemme see right now because I'm on 16.0.8 and this one is on 16.0.2, I believe 16.0.1. So it's a couple minor versions back, but for the most part it's going to stay pretty up to date. And this should probably update fairly soon I would imagine. And that's kind of it for me. We'll leave it open. I think we have three minutes. We can leave it open to questions. If anybody missed something, anything, we'll open the floor.

(45:12)
Brian Robison: Yeah, if there's any questions, feel free to go ahead and enter them in the Q&A. If not, feel free to bail off and get back to your day. We'll just go ahead and hang out for a few more seconds. But just wanted to say thank you very much for everybody attending. Again, happy holidays for all of you. Hopefully the end of your year is going well and we will see you on the other side in 2023. I hope everybody has a great rest of your day, rest of your week, and finish out the year strong. Thank you everybody, have a great day.