Corellium

Video Webinar

[Episode 8: Deep Dive into Reverse Engineering on Android]

Full Transcript

(00:16)
Brian Robison: Good morning. Good afternoon everybody. Thank you for joining us today on our first of a two-part series on reverse engineering deep dive. Today's discussion is going to be on Android. I am joined by esteemed Corellium Researcher Steven Smiley, as you've seen in previous webinars with regard to application pen testing. And it is our absolute pleasure to welcome you to today's webinar. And a lot of people joining in still, but we're just going to go ahead and get started. We have some great content for you today and we really, really, really want this webinar to be interactive so you all are muted, but we do highly, highly encourage engagement. In fact, we're going to actually probe you with some questions, so please stay tuned for those. But we also want you to ask questions of our speakers in the Q&A tool. So please feel free to do that.

(01:11)
I'm going to take the first few moments here and then I'll hand it over to Steven. So when he's speaking, I will attempt to address questions, but also we're okay to stop and answer questions verbally with you. So please do feel free to ask those questions and engage with us. We are here to help you guys be successful and we look forward to the content we're delivering today. The webinar is being recorded. As always, it'll be available on demand shortly after via the Corellium events website, so you'll be able to share that with colleagues and folks who may have missed it or if you wish to get in and review something that you saw today. As always, our presentation consists of a few slides, but mostly real world hands-on type of demonstrations, and in some of those demonstrations, the fonts on the screen might be a little too small to see, might have some issues, but we're going to try to do our best to make sure that things are visible.

(02:16)
And where it’s not too visible, we're going to walk you through what we're doing and why we're doing it. And then again, of course you have the 1080p resolution recording to come back and watch at a later time. Awesome. So today I am going to do something a little bit different. Normally I do a little spiel about, Hey, jump into our contributor program. We want you to create blogs and content and get your brand out there and get paid to do it and all that kind of stuff. But

today I want to talk about a couple of really cool new features that we are working on adding, and they're available for select customers in beta right now. But I want to talk about them for a few minutes, and I'm actually going to show them to you and they're really cool. So the first one is something we call Netdump.

(03:01)
Obviously there's a better name for this and we'll probably have a better name for it coming out shortly. But for those of you who've used the Corellium platform before, we've had Network Monitor, which is a really fantastic tool to capture HTTP and HTTPS traffic on ports 80 and 443. But what we have never had is a full protocol dump, networking protocol dump analyzer, and all these kinds of cool things. So really what Netdump is is the ability to dump every bit of communication going in and out of the device on any part on any protocol and be able to then filter through it and look through it. So in the UI, there are two options for filtering. You can do a server-side filtering, which means, oh, I only want to do a Netdump on a specific process or a PID on the phone.

(04:01)
Or you can dump everything and then filter it on the client side, essentially using the WebUI to say, okay, now only show me capture everything, but only then show me the traffic coming to and from a specific process or a PID or to and from an IP or by protocol or whatever. The really cool thing is that it produces a Wireshark pcap next generation file. They're able to download and view offline in Wireshark for much further analysis of the actual content and the traffic that's actually happening. So this is really cool and I'm going to show this to you in a few seconds. It's also available in both the UI and all of the different APIs as well. So this is really cool. You can check out the beta documentation that's on our support center if you wish and read about it, and if you're interested in taking a look at this, let us know and we can work on making it potentially available in your client. The second feature is updating the iOS version live on a device. So this is not an OTA Apple update, so it's not a Delta update. It uses the full IPSW, but what it does is it allows you to debug not just your app in how it deals when you're upgrading the OS, but also potentially in the Kernel as well. So if you're into Kernel research, seeing what's different during those debug processes and things like that is a pretty darn cool thing.

(05:52)
And again, it is available via the UI as well as the API. And so this, again, it basically uses the full IPSW, but it's going to knock the phone into an upgrade version, meaning it's going to keep all the settings on the phone the same, it's going to keep all the apps on the phone the same. The user data on the phone is going to remain the same. So this could also be very useful when you are testing pre-releasees, things like that, you can update the version of iOS. So that is something that is coming out as well. Very quickly, we're going to step over and take a peek. So bear with me a second here while I get my browser back and I got to close a couple of these windows.

(06:49)

Steven Smiley: While you're doing that, Brian, there is a question as well. Will there be a webinar regarding CHARM? So I dunno if you want to answer that while you're doing this.

Brian Robison: Ah, yes, CHARM, the developer kit. Yeah, we do plan on talking more about that more publicly, but we don't right now have anything that is currently set up for that, but we will look at that and see when we could add one. Definitely good question. OK, so let's talk about Netdump first. One of the first things that you have to do with iOS, so Netdump is a pretty darn heavy load type of thing because it's actually capturing all of the network traffic going to and from the device and basically decrypting it for you and all this other kind of good stuff. So there are new power-on options. So if you want to start an iOS device with process network monitoring turned on, which is kind of the, I think it might be the better name, although I kind of like Netdump. Makes it easy. But basically you need to start the device with essentially this extra piece added to it.

(07:58)
And then once your device is up and kicking what it's going to look like, it's in the Network Monitor tool. Just like before, it's located in-network and by default you get your standard network monitor, which is HTTP and HTTPS only. But now you have this All Traffic tab and this is where things can get very interesting. So if you want to do some server side filtering, you can click on the little gear and you can nail it down by port or range, IP range, process name, the PID, if you know them, all those kind of things like that. You're able to narrow down all of the server site. So it's only going to watch for these specific things. The other thing you can do is go ahead– Oh, we are not seeing the UI. Standby. Hang on, you are right. That is my bad.

(09:01)
I was not sharing my desktop. There we go. Now we can see the right thing. Sorry about that folks. So this is what the new UI looks like. Again, you click on the gear, you can set up the server side filtering to say only capture information based upon any one of these settings, or you can go ahead and just capture everything and then use the client side, the search, and the filter results to do stuff like that. So let's just go ahead and we're going to start the monitoring and one of the things that you're going to see pretty quickly is on these iOS devices…is it capturing? It is not capturing right now. I may have booted this phone up incorrectly. No, it has process monitoring enabled.

(09:50)
Well that's maybe why this feature's in beta. We should be having– oh, there, it was just catching up. So yeah, we're getting a ton of traffic. We're going to restart the monitoring. So basically yeah, we're getting a ton of traffic. You can sort and filter, you can see all the different information coming in. The processes that are running it…it disappeared for a second. Might be delayed a little bit in this version, but basically once you get, you can see the process, the network stuff, flowing in pretty much live. And then of course you can click on download and you're going to download this awesome pcap file that you can then pop open in Wireshark and

look at it in much, much, much more detail inside Wireshark as you would like to do for full on network monitoring. So this is a quick preview of this new feature.

(11:01)
It's going to be pretty darn cool I think. And it is, again, it's capturing every communication off of any port. So we're no longer just limited to port 80 and 443, we're going to be able to throw some mobile malware on a device and see things like command and control, all of those kinds of cool things that are not using standard communication ports to do their work. Alright, and then the other feature that I'll show while I'm in here is the ability to upload and update your phone. So let's see here, I thought this was enabled on my system. So here you have the ability to basically update the iOS version. This one's running the 16.5 release, so there is no version to update. So let me pop into this device.

(12:28)
So yeah, you can basically select which iOS version you want and this one you can upgrade from 16.4.1 to 16.5. As you can see there's an available IPSW on the server to do this. And then you're able to basically go through the update process. Again, it's not the Delta OTA upgrade that everybody would think it would be, but it uses the IPSW, but it does put the device into an actual update mode instead of a full restoration. So it allows you to keep all the information, all the data. So if you're an app pen tester, you go through and maybe let's say you rip out our version of Frida that runs on the device and you put your own, you add some custom tools, instrumentation, maybe some scripts, things like that, you can actually upgrade the OS when a new version comes out rather than rebuilding all of that from a fresh VM.

(13:23)
So this is going to be kind of a cool feature to help us out and save us a bunch of time when it comes to utilizing multiple VMs and multiple OS versions. Alrighty, so just a real quick overview. For those of you who are not really familiar with Corellium, we are in the business to help basically solve three core problems, which is why is mobile security research so darn difficult? Well, it's difficult because especially now, it's getting more difficult to get the proper devices with the proper operating systems that we can successfully jailbreak and we can successfully work with and get to the levels that we need. And those are the issues that essentially Corellium solves as you saw before on the previous screen, not just an iPhone X running 16.4.1 jailbroken, but an iPhone 14 Pro Max running 16.5 is jailbroken as well.

(14:25)
So because we are a virtualization platform and not reliant on public vulnerabilities and exploits, it's very easy for us to have any version of iOS on any version of the models that are already jailbroken and ready. So it's much easier than trying to do some sort of shady deal with somebody in a Walmart parking lot or Facebook meeting place or something like that to try to get one of these older phones that we can physically jailbreak. The virtual jailbreaks are so much easier and much easier to deal with. So that comes down to the time, right? We're no longer having to scour the earth for these very specific older devices that we can upgrade

maybe to the latest version and maybe get a reliable jailbreak going on it. But this is becoming even more difficult now with the newer devices and the newer OSs. iOS 17 will most likely provide its own challenges to the physical world, but it doesn't in our world.

(15:27)
So that's why we exist and again, we are a true virtualization platform. We're not simulation or emulation. We are running ARM-based OSs truly on ARM-based servers both in the Cloud and on-prem. We have an appliance that we sell. We even have a brand new desktop appliance that is available as well. So if you can't get into your IT environment and 2U rack mount appliance, we have a nice little itty bitty tiny perfectly silent desktop version that allows you to do your vulnerability research or your application pen testing on a much, much smaller device that can actually sit on your desktop and you won't even notice it's there. But again, we are virtualization, not simulation. Think of us as kind of like VMware. We're bare metal hypervisor, but we don't run ARM-based windows or ARM-based Linux. We run ARM-based hardware like iPhones, Android devices, IoT devices, Raspberry Pis, those kinds of things like that are what our platform is designed to run. So with that I am going to do a quick poll really quick.

(16:43)
Steven Smiley: Actually I want to answer this. You can do the poll, but we had two questions as well, which I think you're qualified to answer here while the poll's going on.

Brian Robison: Yeah. So we're basically going to jump a poll here before I switch over to Steven and for some reason my Q&A window is outside my screen, so I will fix that and I'll get to answering those questions. So one question I can see is how could malware potentially identify this environment during execution? There are changes that we make to the OSs. So if malware detects, let's say, that a device is running in debug mode or developer mode and may choose not to run, we will trigger that kind of thing in the malware. However, there are ways—and actually using some of the techniques that Steven is showing today—you can actually feed false information into any application and bypass some of those triggers like a rooted detection or a jailbreak detection and even malware. If it's trying to detect something, you might be able to even interject in there and provide it false information to trick it into running.

(18:05)
Also over the past couple of months we released on the Android side. On the iOS side, you can have jailbroken or non-jailbroken iOS devices. On Android, we've always had User Debug Mode. We've always had rooted versions, but we've recently released and they have the ability to run non-rooted Android Rancher builds as well now. So you can run rooted or non-rooted on the Android side, kind of like you can do jailbroken, non-jailbroken. So let's see, let's go ahead and end our poll and we'll share our results. The great thing is yeah, not too many of you have actually attended a previous webinar from us, so welcome. We're very, very happy to have you sitting with us today and we do hope you get a lot out of this first of the two-part series, but also would very, very highly encourage you to go back and review. I think, Steven, you and I have

done maybe what four or five webinars on different app pen testing techniques and things like that.

(19:08)
Steven Smiley: Yeah, lots of good information out there. They're all recorded, right?

Brian Robison: Yep, they're all available on our events website under Webinars for on demand. I did one here a couple months ago, or was it just last month, on mobile malware, which was really fun, very engaging. We had a great big audience and a lot of activity, so if you want to look at that one. I actually executed some ransomware and we did some pretty cool stuff. So check that out. And with that I'm going to go ahead and jump off camera and get going on the real content with Steven. So Steven, go ahead and take it away.

(19:48)
Steven Smiley: Perfect, thanks Brian. And while I'm doing that, if you want to just answer that last question, type it out.

Brian Robison: Yep. I'll be getting into the Q&A here once I get the window resized where I can see it.

Steven Smiley: Perfect. I'm going to go ahead and share my desktop now.

(20:05)
Again, as Brian said, thanks for joining us today. We're going to talk about Android reverse engineering, and based on the poll, it looks like a good percentage of you guys are new here, which is great. There are a couple slides that you might've seen in previous sessions. Obviously with reverse engineering being such a key component of a mobile test and the start of dynamic analysis, being able to build scripts and things like that, it's just such a key piece that we just want to make sure everybody gets a good understanding of. So reverse engineering, just quickly we're going to talk about what it is and what is trying to be achieved. So really, what is reverse engineering? Well, it's the process of breaking something down to understand how it works. So in mobile, that involves either deconstructing, analyzing, observing the compiled apps to understand the underlying functionality.

(20:58)
So the whole goal is just to take apart that application and really just kind of take a look at the code and figure out exactly what it's doing. Find vulnerable sections, find vulnerable blocks of code, things that you can exploit, hardcoded secrets, so a whole bunch of stuff. So there's two main types of reverse engineering and the way these are labeled, I just want to explain them because by itself, dynamic and static don't really sound like reverse engineering terms, but static which we'll look at first is modifying the local files and looking for hardcoded values. There's two real things here. One, potentially looking for hardcoded values. Are they hardcoding email addresses, usernames, passwords, API, keys, things like that. So you can reverse

engineer the application and take a look for some of that data. Some of that involves searching. There's a handful of ways to get that information.

(21:51)
The other part of static is actually going to look at the static code to be able to modify or patch out an application. So as Brian mentioned, you might have a root detection, jailbreak detection, things like that. Maybe tamper proofing, maybe certificate pinning. So you can actually decompile the application, modify the code, patch it out, repackage the application, and that is one way to bypass it without having to use maybe other frameworks, scripts, things like that. So just a different way to go about it. The other main type is dynamic where you are getting those scripts. So you're actually decompiling the application, looking at the code, finding vulnerable sections and then building scripts to kind of exploit that, whether it's pulling out certain data, decrypting data, bypassing certain components or parts of an application. There's a handful of different ways depending on what you're doing with the scripts. So what is being achieved? And sorry Brian, I do see a bunch of questions come in, so if anything it relates to me, just interrupt me.

(22:58)
Steven Smiley: Yeah, the one that just popped in would probably be a good one for you. So basically they test applications that require Bluetooth devices. So we do emulate the existence of Bluetooth and things like that. The problem with a physical Bluetooth device is there's no way to connect from a virtual phone to the Bluetooth device. However, for a lot of the companies that we work with, they actually will virtualize the Bluetooth device in some cases as well and actually have it running in the environment. So there are options to do Bluetooth to Bluetooth in the virtual world, but right now there just is no magic fun way to do it where you're connecting through an emulated Bluetooth adapter through your Mac or your PC to the virtual device. Not at this time. We do have the ability to plug USB in over the network, so if you have a physical USB app or a device or whatever, you can actually connect it to the iPhones that way. But as far as Bluetooth, that's a little bit outside of our realm unfortunately.

(24:24)
Steven Smiley: Yeah, absolutely. OK, perfect. So we've got a couple other questions coming in, but I'm going to finish this and then we'll move on and then we'll answer those. So just leave them open if we need to. So what is being achieved? We kind of talked about this already, but it's just really getting a better understanding of the application, how it's working, how things are running, how it's processing your data, things like that, looking for those hardcoded values, finding a path to exploit the application and then just gain further insight to build those additional scripts beyond the scripts you get offline or wherever that are already built. You want to build some of those custom ones, especially for applications that have been tested multiple times and it can be harder to find some vulnerabilities.

(25:09)

So the Android file structure. Just quickly, if you guys don't know, the Android, the APK file contains application data in the form of a zip file essentially. So you can actually unzip that if you want. You won't be able to see the code quite as well for everything. Apktool is a really good tool, which we'll kind of discuss next, which can decompile that a little bit better. But here's some of just the common directories you'll see. So you've got the META-INF directory, which contains verification information that is generated when the app is signed. If you're looking for some signature information, you can kind of go there. You have your assets, which is a variety of things: images, videos, documents for the application. Sometimes some additional stuff gets stored in there depending on your application. You have the lib directory, which contains native libraries with compiled code for different architectures, which I think we did have a question relating to shared libraries, if I wasn't wrong. I briefly looked at it. Yeah, there was a question around shared libraries. Brian, can you just type an answer and provide my email for that because I can follow up on that. I don't want to answer it directly in this one because it's not related, but I can follow up a hundred percent.

Brian Robison : Yep, absolutely. We'll get back in.

(26:22)
Steven Smiley:
And then you got your Android manifest as well. So the resource directory—sorry, I missed that—contains predefined application resources, XML files. It's a good spot. We actually see network information sometimes depending on how they implement network protections. So you can actually use that to also modify some of those configurations and bypass some of that network configuration, user certs, system certs, whatever they're accepting. Or if there's hardcoded certs for example. You have the Android manifest, which is obviously a big piece that probably most people have used if they're in the Android world. App package, name, activities, resources, permissions—there's a number of things in there. Great starting point for a lot of stuff. Then you have classes.dex, which contains all the Java classes in a Dex format.

(27:15)
OK, so tools for reverse engineering in Android, it's a lot simpler. iOS, it gets really complicated. There's a lot of different tools, assemblers, things like that. So we have a webinar, we're splitting it up. We have a webinar next month on iOS. Great one if you're interested in iOS as well. Register for that one. But for Android specifically, there's not a whole ton of tools. Very commonly everyone's using the same tools. JADX, which is a Dex to Java decompiler to review application code for Android in more of a human readable Java source code. So if you have any experience with Java, looking through the code when you use JADX will be fairly simple. It will make a lot more sense. Apktool, like I mentioned in the previous slide, can decode and rebuild APK files. It's a great option if you just want to decompile the application, look through the local code, patch out the binaries and rebuild them.

(28:12)

It's a good option. So you should have it. Radare2, not talked about as much, but it is a powerful reverse engineering tool to search within the binary and potentially patch out. This can be used a lot more in heavily obfuscated applications. You can kind of follow the pass and navigate the application a little bit better. It is a complicated tool to learn. I recommend checking out some videos if you really want to learn about that because it's not easy, but very powerful. Frida, which I'm sure most people have heard of if not used, which is a dynamic instrumentation tool. They can basically take what you found through reverse engineering and then you can build scripts or use scripts that are available online to exploit the application you're working on.

(29:00)
So there's one question about recommending a way to make an app tamper-proof. I assume the question was relating to how it can actually be added into an application. There are some third party libraries—you go on GitHub—of ways to do that or checks to put out there to look. The best thing that I've seen a lot in tamper proofing is how was your application signed? Did they start looking at the signatures and things like that? Was it installed? Was it installed from the App Store? Was it installed a different way? There are some checks. I think there's third party libraries that can actually do that on GitHub. I'm not a developer by trade, so I don't know exactly which ones are the best ways to do that. I can look up some resources. I know a lot of developers who do this mobile stuff. If you want to follow up, we can chat about that more.

(29:56)
Android app repackaging. If anybody's heard of the UnCrackable app, I don't know if anybody's actually used it or not. We've used it a little bit in some past webinars and I think some blog posts as well. But UnCrackable has four levels for Android. Each level has two things. One, a root detection and then a secret that you're trying to find. So one, you have to bypass the root detection and the same script will not work for all four. So there is an increasing level of difficulty as you go through. So if you're really interested in this sort of stuff and want to decompile an application and look, the link is there, you can go take a look. It's on GitHub. We're going to go through the first level today in a couple scenarios, but definitely a good opportunity if you want to keep learning to go take a look at those for the future.

(30:43)
OK, so level one is the one we're going to look at. Root detection upon launch. You see on the right hand side I got a screenshot root detected. So as soon as you launch the application, it's going to actually shut down. It's not going to allow you through. So there's two options to kind of get past this secret one. You can run a Frida script on launch if you want to. So if you have a root detection bypass script that is applicable to this application. You can actually run that at the same time as you launch the application and then it will get past the…there's not an issue with that, it's just time consuming. Every time you launch the app, you're going to need to run that Frida script. So it can be kind of a pain if you're starting to troubleshoot the second part of the app, but you need to keep running this Frida script in order to get to that. So there's some complexities there. So the better way that I like to do it if you can easily is to just modify the

code. So you decompile the application, modify the code, patch out that piece, resign it and reinstall the application.

(31:55)
OK, perfect. So level one. Like I said decompile the application. Apktool is a great tool. So it's Apktool, “d” for decompile and then the APK. So actually I can just show you, I actually have it here. So if you just run an Apktool d—and if the screen is small, I apologize, I tried to zoom in as much as I could here. Just want to make sure everyone can see it, but I’ll try to explain.

Brian Robison: Looks good Steven

Steven Smiley: OK, perfect. I'll try to explain it as well if anybody has anything…sorry, one sec. So, okay, so if you do Apktool, then “d” for decompile, like I mentioned in the slide, if you do UnCrackable and then the APK, that'll just decompile, then it'll create you a folder. So you actually see in here it's going to create this folder and then all your code is there, right? So nice and easy there, a really simple way to do it.

(32:41)
We had a question, too. What was the…oh yeah, okay. So we'll ask that question in one sec Brian, just let me finish the slide. So one thing we're kind of looking for is find the root detection code. How are we going to find it? Are you going to use JADX, JD-GUIs? Kind of an older way to do it. JADX kind of replaced that, but some people still use it. You can search the small files directly if you want to. You can modify the root detection code or delete, it just depends on how it's implemented. Some applications really just are implemented standalone where it's just one class, very simple, very easy to just delete out. Other ones get a little bit more complicated where it's just integrated very well into the application. You need to actually modify some of the codes to return true instead of false and things like that. And then you resign the application. So one thing we're going to do here is I'm actually just going to load this up. I'm going to load up JADX for this app. And Brian, we can go ahead and do the next poll question if you don't mind. OK, perfect. So we're going to throw up a quick question now. It's a short answer.

(34:17)
You don't need to be too specific, you don't need to write a paragraph or anything like that. Just as I load up JADX here and we're looking for this root detection code, my question really just was what would you search for when looking for the root detection code? Do you guys have some common things you search for right away? Do you have multiple things? Where do we go from here? And I just wanted to see, get an idea from you guys what you guys do right now. If you haven't done anything, that's totally fine. But yeah, just curious. We'll leave this question up for one second

(34:49)
Brian Robison: People are answering. We'll open that up in just a second.

Steven Smiley: Awesome. We have one other question. Oh, it wasn't a question. I guess it was a statement. And there's a great option. So John actually just mentioned in the Q&A, there's a lot of good third party runtime protection. He didn't want to call it a vendor, which I totally get, but he said if you search for mobile app RASP, you'll find quite a few options from the developer side of things.

(35:19)
Perfect. I can't actually see the results. Brian, are you able to see them?

Brian Robison: Yeah, give me…well it's on my screen sharing. It's weird. I guess this is our first attempt at using the short answer one. So sorry, I guess it doesn't show the results, it keeps them private, but answers were root, is rooted, search for strings, alert message, anything that reads SU binaries is rooted, or root, or debug. All kinds of good answers.

Steven Smiley: Yeah, perfect. No, I mean it is definitely the right path. Those are the things you kind of want to search for. Search for root number one, /su, things like that. Some of the past, maybe some of the tools they're looking for, BusyBox, things like that. So there's a lot of different things you can search for depending on the application. And I did see a question come in. Have I had the chance to work with Appdome? And just as a reference, Appdome is a third party tool that I think it's a paid tool. I think it's actually pretty expensive, but where they actually implement some of these security controls for it, it's actually a really good tool. They build a whole business on this. It is good. I have worked with it, I have seen applications that use it.

(36:33)
Yeah, that's basically what I'll say on that. But yeah, we'll go ahead here. So I got JADX loaded up here. You can run a search. So as people said, you can search for things like root. You'll find classes like this where you see "Root detected". There is some obfuscation. So you would have to kind of follow this to see what else is happening, but sometimes root isn't going to give you everything you need. I think if you do /su for example, you can find a whole bunch of stuff and these are some of the things you could be searching for like /system, superuser.apk.

(37:18)
All of these options are things that could be looked for. So when you start to look at the code, you start to see these classes that are looking for different paths. You're looking for /su, you're looking for test keys, they're looking for these various binaries that are within to see if your device is jailbroken. So there are a couple of ways here. If you actually look at the MainActivity, which is where we searched root in the first place, this is kind of where they're doing the main check to see, and we actually just saw these in the previous ones. Those are the paths and stuff, the /su and things like that, how they know you're root detected, they're looking for that. So now we kind of have to patch that out to see if we can get around that.

(38:10)

Because right now if you launch the application it is going to fail, right? So what I'm actually going to do is I'm actually going to close that out. I don't think I had another…okay, so I don't have another slide on that. That's perfect. OK, so now we need to actually go look at it. So if you actually go into…so we use the Apktool, we actually decompiled the binary already. So if you actually go in here, you can actually go into the smali directory to go into the code sg and then vantagepoint uncrackable. This is the same path you saw in JADX. You can actually take that, write it out of there. So now we've got a couple. Wo we have main activity which was the one with the code.

(38:53)
Sorry, there's a question, I'll answer that in a second. So there's a couple options here. You can go into the MainActivity and there's two ways to patch this out. Really understanding the smali code and being able to actually look at it. There's some sites out there. Go Google it to see if you haven't used smali code before to know what some of these registers are to understand what it's actually doing because it is a little bit different. It's hard to read sometimes, but you can see all the root stuff in here. So there is a way you can actually delete a lot of this code in this main activity if you want anything that has to do with root detection, which is basically three quarters of this code because it is a very small application that can actually be deleted. So that's one way to do it. And I'm actually just going to close that because I'm going to show another way and this is all trial and error for a lot of it. It is very hit or miss. Sometimes you might have to repackage multiple times and things like that. And Leonardo, yes, this recording will be accessible later. I think it gets emailed as well. Brian, correct me if I'm wrong, I know it's posted on the site, I feel like it gets emailed but maybe it doesn't.

(40:05)
Brian Robison: What gets emailed?

Steven Smiley: The webinar itself. Someone just asked if the recording will be accessible.

Brian Robison: Yes, yeah, we do send out a notification when the on-demand is available as well as for those people who signed up but did not attend. They're going to get a reminder as well.

(40:22)
Steven Smiley: Absolutely. OK, perfect. So the other piece here, inside the MainActivity you can look for things like where the system exits, right? So the application crashes, right? That was number one. We saw that in the notice. The app's going to crash right away on install. So you can look for things like this where it exits and then modify this code. So this is one option where it actually exits. You can actually just delete the exit where it actually will not exit anymore in the application. We'll just check for root detection, notify, and then nothing will happen because it won't actually exit. So that's another option to kind of delete. So the goal of this is to show that there's not one answer when it comes to any of this. You could delete code, you can modify the names of the code. Maybe it's looking for pass like we showed /su/system, whatever

the case is. You can change those paths to something else that it won't find. I mean there's a handful of ways to do it. You could delete code, modify paths, you can change the way it behaves, whether it exits or things like that. So there's just lots of ways, there's no real correct answer. It is just play around, search for the code, figure out what you're doing and then repackage the app, which we'll go back and talk about quickly. Just in terms of time, I won't actually repackage this application, but I just wanted to share how that is done in case if it is ever needed.

(42:00)
Steven Smiley:
Sorry, the Q&A thing just blocked the screen. I apologize for that. So yeah, when you're resigning an Android application, so Apktool, we use that to decompile. So you can also use it to rebuild. So you can do Apktool “b” for build and then you can get a created directory and then this is the new binary, the new APK, so that it gets generated inside the disc directory of the old one. So you can actually go in and pull that new APK out. So that's how to use APK and then you need to sign it because this binary, this APK will not actually be signed so it will not install. So a couple ways to do that. One, we need to generate a local Keystore if you don't have one already using keytool. So there's a couple of things to note in this command.

(42:40)
You'll see I have android_signing.keystore. You can name this whatever you want. Just remember what name you called your Keystore. Then you have an alias, I have an alias_android. You can change this to be anything you want. Again, remember it because you're going to need it. And then I just use the 2048 key. I mean you can change that too if you want a bigger key. Sign the APK, use jarsigner. Again, it's going to call for that Keystore android_signing.keystore. It's going to call for that package name and the alias. So make sure you remember them from above. If this is old, you can always create a new Keystore if you do have any issues. And then use jarsigner to verify the APK, just verifying, make sure everything's all good. You can use Zipalign for optimal loading. Sometimes not necessary. Sometimes it's better just to do it just to be sure that you're not going to have any issues installing the application. So Zipalign -v 4, the APK and then the new aligned APK and then you can install it on a Corellium device and the root detection will be bypassed. There's a

(43:49)
There’s questions, I'll answer those in a minute. So now we're going to look at the dynamic reverse engineering piece, which actually involves Frida and Frida scripts. So for Frida jailbroken and root devices use Frida injected mode. So if you guys have little experience with Frida, Frida server just is run on the devices, interact via USB or TCP/IP. You don't actually have to be necessarily plugged in, you can do it over the network but it injects a shared library into the target process. And then JavaScript sent from the Frida tool is executed in the context of the application or the process you are connected to. So tools that leverage the Frida API, and this isn't all of them but Frida-ps. You can list processes on the device if you want to really quickly to understand what's running, what you can interact with. Frida-trace, which dynamically traces

functions and method calls. Used not as heavily on Android because we do have the ability to use JADX and things like that and look at the code a little bit better than we do on iOS. Frida-trace is more heavily used on iOS to get a better understanding for the application and what it's doing. Then you have the Frida command line interface, which runs JavaScript and scripts. And then if you have any development experience, you can write your own tools. Frida's completely open, so you can write Frida tools, scripts, whatever you need just on your level of experience.

(45:16)
So here's the level one, and we can go through this, but I have two other examples. So I probably won't actually run this one, but just as an example. So I'll just show it really quick just in the application if you guys haven't seen it. So this is the UnCrackable app with root detection already bypassed, it's going to ask for a secret string. You don't know what that string is by default. So you can type something in, click verify, it's going to tell you that's not it. So trying to figure out what that value actually was. It's using a Byte array. It's taking two values in for the Byte array. Byte array and Byte array two. There's actually more code to this. You can actually look through the application itself, but you'd want to build out a Frida script like this, which hooks this class sg.vantagepoint.a.a.

(46:11)
It's going to create these two—variable zero, variable one—to kind of mirror what is done in the code. You're going to create this flag variable which has nothing in it by default, because it's where we're going to put it. Then you're going to convert that return Byte array. If you actually look at the code, a returning a Byte array. You're going to convert that to ASCII and append that to a string and then you are going to log that out to that flag variable and then return it. So essentially what this whole thing does is takes what's being decrypted in the code, changes it from a Byte array to plain text, puts it in the console log when here. And if you actually look here, I have Frida running. So I think I have the script here. So if I attach to the application, if I actually run this script—and just so you can see the script is the same exact script I just mentioned—you can actually execute this. And I won't go as quick for the next one because it's a little bit more in depth. But for this one I just wanted to quickly show. So if I type in the wrong string now and I do verify, it's going to tell me what the right string is.

(47:31)
It’s going to do it a hundred times, I don't know why. And if you type it in correctly…if you spell correctly, it will work. IEVE. There we go. No, and then I delete it to be…I don't know why. Oh my gosh, my typing today is not great but it still works. So this is the correct secret. You can see how that script actually was able to hook that, log it out into here and I was able to get the correct answer. And the reason I don't want to spend too much time on that script is because the next one is a fairly similar script, so I want to focus more on that one. So level one, this will just show you exactly what I just did, which is load the script, run the application, hook the application and execute the script and type in something invalid and it will come back. Corellium Cafe. If you guys were not part of any previous webinar, which I think 70% of you guys were not,

you guys might not be aware. So Corellium Cafe applications: it's a fictitious coffee shop application that allows users to purchase beverages. It was developed by us and an insecure manner in order to show a lot of these things.

(49:01)
Reverse engineering, static analysis, hardcoded values, misconfigurations, a handful of stuff. There's a lot of different stuff in there in terms of a vulnerable app. But in terms of the reverse engineering side, there's two potential spots that can be exploited via Frida script and then root detection as well. So root detection, we already covered, about ways to patch out the application. So I will leave that because you can use either a public Frida script or you can patch out the application. Both will work and some of the ways we already showed. And then there's two potential spots that can be exploited via Frida script. So we're going to take a look at those now.

(49:39)
So let me just load that up quick. OK, so we have this Corellium Cafe application. It has a login. We won't show this. If you actually want to see more of the vulnerabilities relating to this app, we have a previous webinar. We actually did a webinar…I don't know what month it is. Maybe Brian knows off the top of the head. It was probably March or April. No it wasn't April, it was February-March I think.

(50:17)
Brian Robison: February, I think we covered the Android one, and then March we did the iOS one I believe.

(50:25)
Steven Smiley: Yeah, so if you haven't seen it, there are tons of vulnerabilities in there. We did a whole webinar on it. Really cool. Go take a look if you want. It's on demand, it's available. But yeah, essentially continue as a guest, you can have a cup of coffee, you can add it to your cart. But there's two spots I wanted to focus on. One, if we go to this blog component there's actually the Corellium blog. So if you actually click this, it will actually load up with our blog and with some of our blog articles within the application. But one thing I wanted to show was the actual code relating to that.

(51:00)
So I'm just going to load that up. I don't know how many directories I've moved but should be there soon. There we go. So what I'm going to do here is we're going to load up JADX again with the Corellium Cafe application and we're going to start looking at the code. So there is this WebView activity and some of this would take some searching just to figure out kind of the WebView activity is here. So this is it. Set up WebView, it's calling the blog and we've got the setup of WebView here with some of the settings. Brian, can you load the poll question for this one? I just have a quick poll question.

(52:00)
It has a photo. We've actually never done this with a photo before so I don't know if even know it's going to work. But essentially in order to manipulate the WebView, which misconfiguration will help us write a script? So I've got four options here. The bottom one is the set up WebView, which is the first thing you saw, which is a URL. There's the .loadUrl with a string, which is down at the bottom. You've got .setSupportZoom to true and .setJavaScriptEnabled to true. So just which one do you essentially think is going to be the one we want to target with a script?

(52:45)
And while we're doing that, we had one question. What's a good tool to download APK files to the computer? Really depends what you're trying to do. If you're trying to download it from the store, it's probably better to just install it on your phone and then pull it off your phone as the base APK. There's a couple tools, a couple of tutorials online to do that, but if you download the Android app from Google Play to your device and then pull it off, that's essentially probably the best way to do it. Looks like a lot of people had .setJavaScriptEnabled to true. Nobody had .setSupportZoom, which is not right, which is good. 0%. Perfect. We had 36% saying .loadUrl string and then 18% this setup WebView Corellium blog. So it depends. I mean part of this is right yet the two biggest ones .setJavaScriptEnabled to true would allow us to run JavaScript if we're kind of looking for some different exploitation, some cross-site scripting, things like that. .loadUrl string is the one we kind of wanted to focus on in this one. And I'll show you the script here just because we're kind of running a little low on time, I just want to make sure we're able to show this.

(53:59)
OK, so we have a script here. And you'll see some very good similarities in these scripts. So definitely take a look at the webinar again, when it's recorded, go back over, look at them, see what you can do because you will see some similarities that you can use in other applications. But what you'll see here is that we create this vulnerableWebView, we call this com.corellium.cafe.ui.activities.WebViewActivity which is exactly where we just were. We have this vulnerableWebView and we're going to put one variable, which if you actually look at the code, it's only taking one variable as a string right now. Set up by view takes one variable. So what we're going to do here is we're going to do inject URL. So we go to the setupWebView and we're going to inject it with something else. So in this case we have our own URL and what we can do is—let me go back here, let me execute this. The screen froze for one second, I apologize.

(55:27)
So you see here it actually pulled up a YouTube video instead of what you expected to be the Corellium blog. So essentially all the script does is hook that value. So you see that it's looking for just one string. That being an issue is getting the load.Url via the string. There's the URL, it's taking it. We're basically using the script. If you actually look at the script, we're using the script to have one variable. We're going to give it this URL instead of what it expected and manipulate that to do what we wanted to do. Now we only have two minutes left, so I will just show the

script for the other ones. If you actually look at order review activity—and I just want to show this quick—so in order review activity, when you actually try to buy something, there is a promo code field. And you'll see here there's references to invalid codes.

(56:23)
These are things you can search obviously within JADX to be able to see. But there's this decrypt function which actually decrypts it. If you actually look on the device—and I do recommend, I can't mention this enough, go look at the previous webinars with some really great stuff where we actually go into detail on some of this stuff—but on the device itself, there's a promo code that's encrypted so you can't actually see it. So what you would do is you would come back to JADX and you would start looking for things around the promo code. So you have this decrypt function which takes into inputs. So what you can do here, if you actually look at the code, instead of trying to decrypt this, trying to figure out how it was encrypted and trying to find the keys and all sorts of stuff, what we did instead was hooked it with two variables and then just logged it out.

(57:11)
It's as simple as that. We just call the function, give it two blank variables, and then take the decrypt function that comes back and put it to the log. So similar to what you saw in UnCrackable, you could do the same thing. So the scripts have a similarity with them that you guys can take to other applications and then just decrypt it out. So as soon as you try to go to it, type in the wrong promo code similar to UnCrackable, it's going to tell you the right one. And that's going to change every time. It's actually constantly changing this application. It's not the same every time. So if I showed you now, it will not be the same for anybody else. But yeah, that's essentially it. I don't think I have any more time for that. But like I said, go look at the other webinars. We did some of the webinars more detailed on this application. We're going to do an iOS reverse engineering one in June. June 21st I believe. So register for that. It's on the site. We'll go over some more stuff in detail. Was there any questions that I missed answering?

(58:15)
Brian Robison: Regarding the debug cable and how to get information pre-Kernel boot and things like that from Android? Yes, we absolutely do that. It's available through the WebUI. You don't actually have to plug a cable in. It's all built into the platform and the product. So that's a cool feature. If you are interested in seeing more about like Kernel level vulnerability research type work, I encourage you to come to our next Quick Start training, which is essentially like a 2–2.5 hour deep dive demonstration into using those types of tools like that specifically for VR work as well as pen testing and even API stuff. So please look that up and go to those. Again, as Steven said, thank you guys all very, very much. We appreciate your time today. We hope you got something valuable out of this webinar. I always do when I watch Steven do some cool stuff. Thank you very much and everybody. Have a great rest of their week. See you next month.