Corellium

Video Webinar

Episode 10: Intro to Custom Device Modeling with Corellium

Full Transcript

00:00:14
Brian Robison: Good morning. Good afternoon. Welcome everybody to our webinar. Thank you for sharing with us. We took a short break last month. Hopefully everybody has kind of enjoyed their summer break and are ready to kind of get back into the swing of things here in the fall. Again, thank you for joining us today. I'm Brian Robison, Chief Evangelist here at Corellium and I have a guest speaker with me today, Chris Williams, who is on our team and is going to do some interesting things today and give you kind of an introduction to our what we call our CHARM Developer Kit.

00:00:48
And I've seen quite a few of you have asked some questions about that in past webinars. So hopefully those of you who asked are here to learn more today. So thanks, Chris, for joining us. And he'll be on in just a few moments. I do want to make everybody aware this is a webinar format, so you are all muted.

00:01:10
But we do encourage engagement, so please feel free to use that Q&A widget. While I'm speaking, Chris might be able to answer some questions on when he's speaking. I'll try to do my best to answer questions. Please, please ask them. We can also we want to ask questions live. So again, we want to encourage engagement. So please ask questions.

00:01:32
And you know, I can ask Chris those questions as we go through the webinar. As you heard, it is being recorded and it will be available to you shortly thereafter. It will also be available within a day or two on our website as an on-demand webinar. So keep an eye out for that if you want to share it with your teams or review any of the details that are going on on the screen.

00:01:57

Speaking of screen sharing, we do a quite a bit in our webinars. We don't just go through slides. We do a lot of on-screen things, and some of those things might have tiny fonts. We do our best to make sure things are somewhat legible, but again, those recordings of these webinars are going to be helpful for you if there's something that is maybe a little too small that we're trying to explain that you want to go back and review.

00:02:22
So why do we do kind of what we do? And I'm just going to go through a few slides here very quickly before I hand it over to Chris. But, you know, why is this area of mobile security testing and research, why is it something that it's so difficult to do? It's not too dissimilar from, you know, testing on an x86 software or something like that, but it's just a little more difficult to do because we have, in our world, we have kind of three limitations that are making this a little bit more difficult than it really needs to be.

00:02:57
The first thing is access to the devices and what I mean are, you know, these are physical devices. They're easy to get sometimes as far as physical devices. But getting them in the proper mode that you need them to be can be the more difficult thing. So really, this is kind of a combination of devices and the OSs, and what we mean by that is really gaining access to these devices outside of their walled gardens.

00:03:27
This is the key thing when trying to do security testing on devices, their OS, or even the apps that run on them, is that the manufacturer errors have created a pretty, you know, significant walled garden that you have to kind of break to be able to get access to the data vaults that are stored at rest on disk to be able to get to, you know, things like network interception with, you know, stripping out SSL and TLC. Those kinds of things like that is really why, you know, a solution like Corellium exists because it's difficult to gain access to devices with specific operating systems whereby we can, let's say, jailbreak these devices and gain access outside of that of that known manufacturer of walled garden.

00:04:13
And the third is time: the time that it takes to acquire these, update and maintain them, you know, flash them up, flash them down, restore them, jailbreak them. All of these things take time. And what it does is it takes time away from the security research that you actually should be doing when you basically are now, you know, a physical device manufacturer.

00:04:47
You have to build and hold these physical devices or whatever or maintain them and manage them. They're in your desk drawers or whatever. So any time that those devices, the time that you spend on them, takes away from your security research, it really impacts your ability to deliver good results and efficient results. And so that's what Corellium actually is.

00:05:11
So we are a virtualized platform. We're not an emulator. We're not a simulator. We actually built Type 1 Hypervisor that runs directly on Arm servers so that we can run Arm operating systems directly on on those Arm chips. So think like VMware, what they did for x86 virtualization with the ESXI platform. We are basically the same for the the the virtual phone platforms or other IoT-type of devices.

00:05:50
So if it's an Arm operating system and I'm not and I'm not talking about things like Linux Arm or Windows Arm or Mac OS, I'm talking about very specific types of Arm OSs like iOS, like Android, like other types of things. And that's kind of what we're here to talk about today is yeah, the Corellium that, you know, today does iPhones and Androids but it also does a lot more things around devices.

00:06:21
And that's where we're going to be kind of taking today's discussion. So as, again, you know, like I said, you know what Corellium can do today by offering different iPhone, different Android models. This is what most people see when they purchase Corellium. But what happens if you have other types of devices like these that are not your run-of-the-mill, you know, iOS and Android-type of devices?

00:06:52
Maybe they run a completely proprietary operating system. Maybe they run again, proprietary apps, those things like that, that cause you to need to have a different type of virtual machine. And really what that kind of introduces us into is today's topic, which is what we call the CHARM. And this is an interesting thing. You might hear sometimes when you're around the Corellium circles and it basically stands for Corellium Hypervisor on Arm.

00:07:22
So that's what we call it, is our CHARM platform Arm. And today we're going to be discussing the developer kit, which basically allows you to create your own custom virtual device models and then put them into production on the Corellium platform. Now, why would you want to develop your own models? That's a big question. So if you think about the analogy of the VMware world, VMware built one type of virtual machine that everybody makes use of, and that's a generic x86 platform.

00:07:55
Yeah, you can change the number of CPUs, how much RAM is allocated to the machine storage, other types of peripherals that you can attach, you can even pass through physical hardware into that VM. But there's only one type of VM in the Corellium world, each type of different iPhone, for example, is a completely different type of VM model.

00:08:21

And then when you expand that into things like the Raspberry Pi model or certain other Arm-based circuit boards and components, or let's say, you know, I'm a router manufacturer and I build a board. I want to virtualize that board so that I can then develop my OS. I can develop, you know, on the OS, I can write applications for that OS, but I don't want to have to ship, you know, every single developer in my company, a board or my tester, or one of these boards, especially early on in the development, the availability of those circuit boards is going to be very, very limited.

00:09:02
So how do we get a jump start in the application development in the OS development, debugging, QA testing, all that kind of thing? Well, the answer really is to virtualize the actual board itself and run it in a virtual platform like Corellium. So what is this kit? It's kind of funny, because actually on my screen I think my picture here is actually a little bit bigger than the kit.

00:09:27
If you look at the size of the ports on this thing, you're going to get an idea for kind of how tiny this thing actually is. But the developer kit tactically is a small appliance that you get from us. It's designed for an individual developer to be working, you know, locally on their machine. The platform itself is the in video or in platform, which is a little five inch by five inch box.

00:09:53
I wish I had one here to show you, but they're really tiny and they're super quiet and it comes with the Corellium software and all of the appropriate licenses and things like that ready to go and installed. It does have 10 VM cores for you to play with. So this is, you know, when you build a VM model, you're going to again assign a number of CPU cores to it so that it can boot and run all your different OSs and things like that.

00:10:22
It also is going to include the CHARM SDK, which is libraries and tools and examples and documentation and things that are all available. They all come pre-installed on this little box and then you're going to get access to some really cool training that Chris is going to be talking about a little bit more in depth because he's one of the people we have that go out and do this type of training.

00:10:49
But you basically have some really excellent training that's available to you from Corellium as well as support that is available to you as well. So in some cases, you may end up speaking with one of our modeling engineers who can really kind of help you out in this kind of thing. So I think that took a little less time than Chris was expecting.

00:11:15
But at this point, we're going to turn the conversation over to Chris and are you ready, Chris?

00:11:24
Chris Williams: I am.

00:11:24
Brian Robison: All right, cool. Well, I'm going to go ahead and start my share so that you can get going. I will be available on here for Q&A, if anybody has any questions they'd like to ask otherwise. Chris, go ahead and give us a preview of what this SDK is all about.

00:11:44
Chris Williams: All right. Thanks, Brian, though, for this webinar, we'll start with an overview of hardware, accelerated virtualization and then introduce the hypervisor for Arm and discuss the anatomy of a module. Next, we'll walk through setting up the built environment using Docker and then we'll walk through some examples of extending the built-in iPhone models to run some custom code inside hypervisor.

00:12:09
Before we get to all of that, we need an understanding of the types of knowledge and experience that will set someone up to be a successful model. Development of modeling devices requires understanding the hardware software interface of the target device along with the Arm architecture, reverse engineering tactics and the various mechanisms that are used to communicate with peripherals.

00:12:30
Experience with firmware development or Kernel development is quite helpful as a device model can be seen as the counterpart of firmware and drivers. So many concepts are the same or substantively similar. So virtualization allows for running multiple virtual machines on a single system. In modern times, there's often architectural support for virtualization, which provides hardware acceleration and enforced separation of resources.

00:12:58
A hypervisor coordinates the virtual machines and their access to hardware on ARMv8, the hypervisor runs in exception level two or L2, and it's kind of like its own operating system, which provides services to other operating systems that run in L1. The line between virtualization and emulation is a little bit fuzzy, and there are some different definitions out there though for our purposes, emulation refers to a software implementation of a system or a full system emulation, something like QEMU.

00:13:29
This allows for running across architectures such as running Arm on Intel, but on a sizable performative virtualization expands on emulation to allow direct execution of the guest code directly on the CPU without being translated, which is much faster. This requires the Hardware's assistance to enforce the separation between the guest. For example, by using another layer of page tables which track which guest has access to which memory regions.

00:13:58
Some aspects of our model are in fact emulated under these definitions, but don't get caught up in the distinction. So for the most part, it doesn't matter. For our purposes, this webinar is all about CHARM the hypervisor for Arm, which as Brian said, is a custom Type 1 or bare metal hypervisor. The CHARM SDK provides the ability to develop custom models by providing a framework for handling some of the complexity of a device by leveraging the rich, battle-tested APIs that Corellium engineers used model iPhones, Android devices, IoT devices, development boards and so on. The term CHARM development kit, or CDK, is the hardware component that is required for modeling devices. Again, as Brian said, it's derived from a NVIDIA Orin development kit and runs the full Corellium stack. But this hardware is meant to be deployed to an individual developer's desk operating as a headless server.

00:14:56
Since developing models can sometimes result in EL2 panics, it is not recommended to have multiple users on a single CDK unit and having access to the physical serial console is preferred so that the crash logs can be seen and investigated.

00:15:11
Brian Robison: We have a couple of questions that have popped in. How do you see CHARM being useful for security testers while they examine things that are running on Arm processors, that kind of thing? So why would somebody build a unique model, I guess is kind of a better question to ask, is what is the need here?

00:15:39
Chris Williams: Well, I'll give the already existing example of the iPhone models. So for example, the Kernel debugger, there's really no way to get a Kernel debugger on an iPhone. Having a model allows for that sort of…it's free functionality and that would be extended to other hardware as well. If you're modeling a router or an IoT device that and maybe it's JTAG excused all. So you can't just attach your debugger to that.

00:16:11
And actually it allows for scale so you can have multiple people looking at the same type of device even if you only have one piece of physical hardware.

Brian Robison: Right, so it's the same essential use case model as like why people would use it for iPhone debugging. It's because you get access to that iPhone hardware in a way that can be introspective, that is not possible from the physical world. And the second part of that question is, yes, that the different models can be deployed into any of Corellium deliverable platforms, which include on premises servers as well as our cloud environment and some of the other methods of delivery are, you know, whatever cloud platform you have that is running Corellium or an on-prem, you can deploy the models that you build into those different environments.

00:17:15
Chris Williams: Just a quick clarification on that. By at least policy, if you develop a model on a CDK, it cannot then be deployed to standard cloud because that would be a rather large security risk. We can't have people running code in EL2 on a shared infrastructure like that, but can be deployed to the Rackmount server or private cloud.

00:17:40
Brian Robison:
Right. That's what I meant. Yeah, you're right. I forgot. Your device is not going to become available to other customers through our purchasable platform. But if you decide to put a platform in the cloud or whatever, that's that's up to you.Go ahead, Chris.

00:17:56
Chris Williams:
Was there a third question?

00:18:01
Brian Robison:
Oh, yeah, there is. So can this be run on an M1 Mac?

00:18:08
Chris Williams:
Meaning the development of models or CHARM itself?

00:18:13
Brian Robison:
I think the question is as can CHARM be run on an M-Series mac?

00:18:20
Chris Williams:
It cannot. It needs to be Corellium provided hardware, which is either the entire Rackmount server or the CDK hardware. But in terms of doing the development itself, I run on them on Mac. I do everything in a Docker container and then I upload and run the models on the CDK.

00:18:39
Brian Robison
Right. It's just not the the CHARM platform itself that is able to run on the on your local machines. All right. Go ahead.

00:18:49
Chris Williams:
All right. But once the model is ready for production use, it can be deployed to full size Corellium

servers for scaling purposes. A model in CHARM encapsulates a number of components which are then packaged into an extension. At runtime, an extension is used to create a domain which is the unit of a virtual machine.

00:19:16
And we'll go through each of these components in more detail on the next slides. So the domain file aggregates everything needed to create the VM, essentially naming all of the other necessary components and some configuration. The file defines the type of the model, in this case rpi4b. When creating a VM, this is the name that used to tell CHARM what to create, and this will generally be the same as the file name prefix.

00:19:42
So this file is rpi4b.domain. The file that enumerates all the other components, including the loader daemons, and then the CHARM module itself. CHARM modules run inside of Arm itself within EL2. This is where the bulk of the modeling occurs and they're responsible for defining the CPU configuration mapping, memory handling system registers, emulating peripherals and so on.

00:20:11
Since they run on the EL2, causing a panic, we'll crash the entire system, which again is why developing models on a full appliance is not recommended. Daemons provide services to the VM but run as regular userspace processes in a Linux environment called dom0. So 0 is the “host” operating system. If you switch into the CDK, this is what you're connecting into.

00:20:37
Technically, it's also a VM, but it's privileged and can interact with CHARM in order to manage other VMs. There are a number of built-in daemons, some of which are required for the to operate at all, and then some are optional. So for example, the cons daemon provides the glue between the console tab in the user interface and the UART serial port model inside the the device model. Custom daemons are also possible and the full course involves implementing a few of those.

00:21:05
The Corellium Hypervisor Interface or CHI provides a mechanism to send messages between dom0 daemons and a VMS module. The loader is a special type of daemon that takes firmware from disk and then loads it into the memory. So for example, by parsing an alpha file. This fills the same role as a bootloader on a physical device. There are built-in loaders for standard Linux Kernel formats as well as flat binaries and ELF files.

00:21:37
For the raw loaders, another file must also be provided that tells the loader where in memory to place the contents. Custom loaders are also possible, which are useful for more complicated devices. One of the example models provided in the CHARM SDK uses a custom loader to

patch the device's Kernel and load time in order to provide the Core Trace system IP and functionality.

00:22:01
So an extension.json is a higher level description of the model containing information needed by the web interface in order to make the VM usable. There are many possible settings in this file, but some common or noteworthy ones are working on the slide taken from the Raspberry Pi. For example, one of the most important fields here is the flavorId, which must match the name of the domain.

00:22:23
This is the glue that ensures that the correct device is created when it's selected from the create device carousel. And some other important fields are cores and ram which tell the user whether they have the resources available to start a VM of this type. There are also quite a few feature flags that control the UI. For example, if the dynamidHardwareChanges setting is enabled, then the user will be allowed to change the number of cores and the size of RAM from within the UI.

00:22:50
Even for a device that's already been created. This is not an exhaustive list, so see the full documentation in the SDK for many more settings. And finally, another major component is the firmware. This is treated separately from the rest of the model. It's not packaged into the extension as there can be many variants or versions of firmware for each model.

00:23:12
So a firmware bundle is a zip file containing generally a Kernel, ramdisk, devicetree and NAND along with a JSON file to specify which model the firmware goes with. The device identifier is the important field again matching the name of the domain file. The unique identifier is used to allow updates to the firmware, so if a new firmware zip is uploaded with the same unique identifier, it will be considered an update to that version rather than a separate firmware.

00:23:38
And then the build and version fields are for display only and are not used for matching the listed file names here. Kernel, ramdisk, etc. are common but not necessarily required except for info.json. So the loader is responsible for knowing how to handle the firmware bundle as a whole and may define a different structure inside the SDK bundle delivered with the full training, there is a directory called Docker Buildings.

00:24:08
This contains a Docker file for the standard term SDK build environment. In particular, the cross compiler is needed to CHARM code, assuming Docker is installed. You simply build the image from within this directory and name it cdktraining and then once it's built you run a new container from the root of the handouts directory and change to the SDK directory. For Mac OS users, this has been successfully tested on both Intel and Apple Silicon systems, but for Apple Silicon it will

run under Intel emulation with a small performance hit.

00:24:37
Linux systems can also use the Docker container or install the simple chains natively, and developing CHARM modules on windows is not recommended, but your mileage may vary. Doing logs is an important part of working with the term SDK, and there are multiple places for different types of logs to go, so knowing exactly where to look for something specific can be a little bit tricky at first.

00:25:05
From within a CHARM module, again that's EL2 code, regular printf will log in a CDK's physical serial console—the physical micro USB port on the hardware and also to the chi-dmesg utility. This is like regular message but focused on logging from CHARM. There are a couple variants. Qprintf will only log the chi-dmesg. If you're not using a physical serial connection to your CDK, this might be the better one to use in terms of buffering and flushing of data and so on. And cprintf logs to the VM console in cyan text which is visible in the WebUI. This facility is not available during the early stages of device creation, so you'll need to use printf or qprintf inside of the module's create function and any functions called from it.

00:25:53
Production modules generally shouldn't have any separate tests left other than perhaps in exceptional situations. Logging for CHARM will go to the physical serial console, which requires connecting a micro USB cable to the CDK hardware. If an unhandled exception in EL2 occurs, a crash dump will be logged to this console, so it's important to have access to it.

00:26:15
If the whole CDK locks up, that's probably what happened. But generally it's recommended to keep chi-dmesg running and giving it the -f flag for follow. This won't catch EL2 crashes, but most other logging from models will be visible here. And for daemons, the logging occurs in CHARM. Since this is managed by system d, Journal Control is used to view these logs. In particular when a device fails to start this is where the logging will generally be. And finally, while the device logs, as in what it outputs to its serial port, can be viewed from within the web interface. They can also be found within a file on dom0, which will contain the entire history. Since VM power is particularly useful when the needed information has scrolled off the screen, some of those Kernel panics can be quite likely.

00:27:13
The file path can be requested from CHARM over its control socket as shown in the screenshot and then tailed or otherwise viewed. I will walk through developing a very simple module to add functionality to an existing device model. In this case, one of the built-in iPhone models. Since Chrome doesn't provide the source code for these devices, this is a neat capability with a number of potential uses.

00:27:44
For example, if a component of Apple's system on CHIP is needed for research but isn't implemented in the models already, it could be added in after the fact, although it would certainly be a lot of work. So this example will simply log when the module loaded and then will add a custom recall which can then be triggered from userspace code within an iPhone VM.

00:28:06
Just like defining a new model, extending a model also requires a domain file. This must match the name of the original model. For example, iphone12. The loader and daemons also need to match the original model. Although new daemons can be added, the SDK documentation contains a table of which daemons are required for each built in model. And finally, the attached line specifies a model to load.

00:28:30
This doesn't replace the module for the built-in, but it is executed after the device is otherwise created. And now we'll finally see some actual code. So this is a minimal module here. MODINFO is a special structure that CHARM uses to initialize the module similar to what you'd see in a Linux Kernel module, for example. And the name field must align with the module named in the domain file.

00:29:00
The .init and .done callbacks are called automatically by CHARM when loading and unloading the module. Init then registers another callback which is called after the normal device creation from the built-in module is completed. And remember that printf output goes to chi-dmesg. Extensions live within dom0 in chrome extensions. So we will need to upload the module in this case.

00:29:26
It's my-iphone.cmod and the domain file iphone12.domain. Then we will watch the message while booting an iPhone 12, which should show logging for our module to indicate that we have our code running. We'll jump over to our. This is the same same code we saw on the slide. There is a cascading effect of callbacks here, but essentially and it gets called first, it registers a callback for when the built-in module has completed its setup and then we'll get executed and then our callback will just print in return.

00:30:08
So we're in our Docker container here. We run make package. This will create a zip file. We then upload to our CDK and down here we have chi-dmesg running and we unzip our extension and move it into place and then we start our VM. So we see a bunch of logging from the built in module, but we also see our log in here.

00:30:44
So our code is running inside the VM and the console is taking its time, but the device is

running, our code is running. So now we will add a hypercall to the extension. Hypercalls provide services to the VM via the HVC instruction. This is somewhat like a system call, except instead of being handled by the Kernel, it is handled by the hypervisor.

00:31:38
Corellium uses these internally for a number of features, including the arbitrary Kernel and Kernel read/write feature unitcopy. A custom hypercall could, for example, be used for another to ask the hypervisor for the next test case or to pair virtualized hardware like an Ethernet interface or any number of other uses. The example in this section is somewhat synthetic. We'll simply log with cprintf and then set a register which some userspace code inside the VM will then print.

00:32:10
Within the attached callback we can register a trap specifying the HVC number, 0x1337. So when any code inside the VM, whether it's userspace or Kernel executes an HVC 1337 instruction, the handler callback will be invoked. The handler receives a pointer to the VM structure passed in as void *param that will require casting, but that provides access to the end memory.

00:32:37
The callback also receives a pointer to the frame, which includes all of the registers. In this example, the X0 register is being sent to a hardcoded value and finally multiple callbacks can be registered to the same track. So returning one indicates that the hypercall was handled and returning 0 will pass execution to the next handler. Our iPhone 12 VM contains a pre-built command-line binary which simply invokes the hypercall and then prints the contents of the X0 register. We'll use this in the example to validate that the recall is functioning.

00:33:12
Brian Robison: Hey Chris, there's a couple of questions that have popped in. One question is how does this handle TEE-backed or SC-backed features such as the Android hardware backed Keystore and Strongbox?

00:33:33
Chris Williams: Well, nothing like that is built in. That's all up to the development of the model. So for example, on the iPhone models, there is the secure enclave processor which runs as its own virtual CPU inside the VM running its own firmware. That's all modeled separately. It's up to the model to implement that.

00:34:01
Brian Robison: Awesome. And there's one more: Are presets available for common cases like iPhone and Pixel?

00:34:11

Chris Williams: Well, the iPhone models are built in, so extending them like we're doing in these examples would be there. The Pixel is not currently modeled. In the full version of this in the training, the section after this would be starting to implement the Pixel 6A and getting to the point where its Kernel is loaded, it's executing, it's enabling the MMU and it starts loading drivers.

00:34:38
It gets to the point where in it is running…so userspace execution and then that's where we stop and move on to other topics. There's no full model for Pixels at this point.

00:34:51
Brian Robison: Awesome. Thank you.

00:35:04
Chris Williams: OK. So I have another file here already that simply repeats what's in the slide. So it sets the trap for 0x1337 and gives it the handler. And so when the HVC 1337 instruction is executed, our callback will be called and it will print to the serial console. It will print into the VMS console and it will change the 0 register that will then be accessible from within the VM.

00:35:39
So I've already copied that into place and I will rebuild the extension, delete the old one and copy the new one. So now when we start our VM, we should see the same logging for the attachment piece which we have here and now. We have to wait for our VM to fully start up so we can get to our original.

00:37:00
OK. So there's way more logging than I expected, which is going to make this really difficult. But hopefully we can. That's going long enough. So I ran the command and got covered up in all this. But here's the output and we see our code over here. This is our regular printf inside of our callback and here we have our cprintf in the cyan text and then the hypercall completed line is from inside of our userspace binary. So that's just printing the line and then printing contents of the 0 register.

00:37:50
This is a brief excerpt of the full training, which is hands-on and covers many more topics, including implementing the first steps of modeling a new device, including Pixel 6a from scratch. Some CHARM SDK features such as daemons, the ARM2ARM binary transition layer and live snapshots. The full training ends with a capstone project implementing custom peripherals for a purpose-built device derived from the Cortex-M85.

00:38:16
So contact your Chrome representative to book this training and learn more about what the SDK has to offer. Any questions?

00:38:26
Brian Robison: Awesome. Thanks, Chris. Really appreciate you taking the time today to discuss with us and give us a preview of some of the things that you can do for those of you who are interested in this. Like Chris said, you can reach out to us and we can get more information about purchasing the CHARM SDK, making one available to you, all that kind of good stuff, as well as the training and the support that Chris is talking about.

00:38:56
In fact, you may be trained by Chris or somebody else, but most likely be him and, you know, we think it's so important. I think there may be some folks on here who are confused, like, you know, do you have to build a full iPhone model? Do you have to build these things? No. The like the iPhones and the Androids and all those kind of things like that are already built for you by us.

00:39:23
That's essentially what you're buying. So this is really valuable if you want to do something like extend an existing model, like what Chris is showing today. Maybe you want to attach a new piece of a peripheral to that device or something like that. That is your unique hardware that you can customize and attach it to an iPhone or attach it to an Android device.

00:39:46
Or if you are doing something a little bit more generic in the Arm world, I highly, highly, highly recommend if you're doing Arm board development or OS development, that you check out Arm's virtual hardware platform because it is essentially us underneath. So they have a series of boards that are available already for you to basically participate in and build your custom operating systems and things like that.

00:40:16
And you know, if you wanted to extend those devices into new peripherals, it's basically this type of scenario that is what you're going to need to do. So let's see here. I'm going to pop this back. So we are ending a little bit early today, but that's good. Everybody can have a little bit of time back in their schedule and again for both Chris and myself, thank you very much for taking the time and hanging out today.

00:40:48
Our regularly scheduled webinars are going to continue next month and if anybody has anything they would like to see from us, please reach out to me. Let me know. We have done a large series on mobile application pen testing for a good chunk of nearly a year now, and so there's a lot of those on demand available. If you go to the Corellium website under the events, webinars, you can view all of those on demand as well as to where this one will be here within a day or two.

00:41:19
So again, thank you all very, very much for attending today's webinar and getting a preview of some of the things that you could do if you were to if you were interested in building a custom model or extending an existing model that runs on the Corellium platform. Once again, thank you very much. And we hope to see you on the next Corellium webinar. Have a great day. Take care.