Corellium

Video Webinar

[Episode 12: Automate Complex Tasks by Giving GPT-4 Control over Corellium's AP]

Full Transcript

(00:00:49)
Brian Robison: Good morning, good afternoon everyone. We are going to begin in just a few moments. There's quite a few people coming in. So just as the slides say, we're going to begin in just a few moments. Grab a drink, get comfy, and we will be back on with you very shortly. Thank you very much. Good morning and welcome everybody to our webinar today. It's been a few months since I've seen a lot of you. It's great to be back in the new year and I hope everybody had a great holiday season and a new year and are getting ready to kick off and get going in 2024. So thanks everybody for taking the time out of your day to come and hang out with us and learn something interesting and I think we're going to have a lot of fun with today's session. I'm Brian Robison. I'm the Chief Evangelist here at Corellium and I am joined by a very, very special guest speaker today.

(00:03:03):
David Manouchehri from ai.moda has joined us today and David started his professional career in cybersecurity quite a few years ago after being recruited by the Department of National Defense. He's since worked for startups like Linchpin Labs and started his own company, ai.moda. And after becoming an independent security consultant in 2020, he's actually found and disclosed over half a dozen Chrome's zero-day exploits and vulnerabilities. So he now works on combining AI with real world security problems and in fact, that's what he is here today to talk about. So good morning, David, are you with us?

(00:03:49)
David Manouchehri: Thank you for the introduction and good morning as well.

Brian Robison: Awesome, excellent. So we're going to go ahead and get started with today's presentation because David has a lot to go through. So we're going to begin just with a little housekeeping for a few seconds. All of you are muted, but we do encourage engagement with us, your speakers today. So please feel free to ask questions using the Q&A tool and I will be able to ask David those questions throughout his portion of the presentation or if you have questions for me, I'll be able to answer those back to you while you're there. So if you have any questions, please feel free to engage with us Now. We are going to engage with you, we are going to ask you some questions as well. So keep an eye out for those during the presentation. Obviously the webinar is being recorded. Many, many of you who were on a few minutes early heard the presentation being recorded announcements.

(00:04:48)
So this will be recorded and it will be available for on demand very shortly after today's presentation. So feel free to share that with coworkers or anybody who didn't get a chance to attend today. And as with most of our presentations, the slides, the presentation consists of slide material as well as in this case kind of prerecorded material because some of the things David is going to be demonstrating take a little time. So he had to speed those up a bit. So they are live demonstrations, but they are prerecorded, which helps us kind of compress time and keep everybody on track. Now, we are also going to make all of the information that David is going to be talking about today available for download as well. So keep an eye out for a blog post to follow up on today's webinar with some summary information about the webinar itself as well as a link to download the materials.

(00:05:47)
So everything that you're going to see, you don't need to worry about capturing it live or anything like that. You're going to have access to it very shortly afterward. Now there is a project that David helped us with a few months ago and he helped create the first ever Corellium Terraform provider. So any of those who are using HashiCorp Terraform out there, this gives you a great way to interface with Corellium. You don't even have to learn Corellium's APIs or anything like that. David and his team did all the hard work for you. So you can review the blog on our website, you can also download it, the provider itself from GitHub and get in there, begin playing around with it. So we invite you to check that out and thank you to David and his team for helping us create that. And really with that, I am going to pretty much hand this thing over to David because he's the person you came here to see today. So with that, David, I'm going to go ahead and stop my sharing and you can go ahead and get going. Thank you very much everybody for attending and have some fun.

(00:06:58)
David Manouchehri: Thank you. And while I set up my screen sharing, we're going to take a poll to begin with just to see where everyone's prior experience level is at. Let's see.

Brian Robison: And like you said, we're going to ask you guys some questions so hopefully we can get some responses.

David Manouchehri: Perfect. So can you see my slides?

(00:07:26)
Brian Robison:

Yeah, we can see the slide. The audience should be seeing the poll over top and it's almost completed and I'll turn the answers over. David, are you able to see the poll?

David Manouchehri: Yes.

Brian Robison: Alright, we're going to share the results.

(00:07:50)
David Manouchehri: OK, so it's a good mix. So we're going to go over three example use cases for each topic. We're going to go into how to make it actually production ready, not just a, this could maybe work. No, we're actually going to show how to make it work. So the first example we're going to go over is the simplest. We're going to demonstrate how to interact with Corellium's API using GT-4. The second example is the one I'm most excited about and I think it's what the audience will find the most fun because we're going to be using input fuzz with GPT-4V(ision) feedback. This opens a lot of possibilities that I couldn't really imagine doing even two years ago. And then the final example, we're going to show how you would take something like the previous example and deploy it in the cloud as a production workload.

(00:08:47)
So with that, let's go into the first example. As I mentioned before, the first topic, we'll be using natural language to communicate with Corellium API. This is a fun beginner task. It doesn't take that long and you can see some really cool results quickly. So our goal for this will be to have GPT-4 to be able to answer the question what devices are currently running on our Corellium account. For this, you're going to need to have an active Corellium count, an OpenAI API key and Python 3. We're not going to cover creating the OpenAI API key because that's covered many other places online already and it's pretty easy. So first step, whether you're on an enterprise or cloud version of Corellium, you're going to need to your profile and generate an API token. We're going to use this same token throughout the whole presentation.

(00:09:43)
And then now we're going to need just two Python dependencies. I'm hitting the versions here so that if you're watching these slides say in 2025, this should still work. You can probably get away without putting the versions, but no promises. I tested all this with these versions. I know it works. Yeah, it's a good practice to pin. First we're going to need to download Corellium's OpenAPI specification. The JQ command is not necessary, but it just makes it a little more readable for us humans. So we're going to need to define the server URL manually for LangChain. This is technically optional by OpenAPI 3 specifications, but LangChain requires it to be defined or at least time of writing this. So you go from this. And the only thing we need to modify manually, and this is a one-time thing, you're going to need to add your endpoint. So for us, we have an enterprise account. This is ours. Yours will be different. All right, onto the programming part, we're going to start by importing all the packages and libraries we need.

(00:10:59)

And just to take a side note, what is LangChain? It's a framework to assist developing applications with large language models. Today we're going to be using one of its prebuilt functionalities, which is to communicate with an OpenAPI endpoint. Let's also take a poll to see if anyone here has used LangChain before. While that poll's going on we're mostly going to go onto the next slide. So this is now using LangChain where we're going to load the specification and also reduce it. And this is probably the only one of the magical parts where we want to be able to give, we need to be able to authenticate ourselves to Corellium API. So what we do is we create this little wrapper that adds the authentication header for us.

(00:12:02)
And then finally, this is how we go ahead and use the OpenAI chat interface or sorry—how we set up the client. And based on the poll, yeah, so I think this might be…that's good that not many of you have used it before because this is you're learning something. So another side note I want to take for all of our examples today, you can use OpenAI or Azure OpenAI. They are pretty much the same. My personal preference is to use Azure OpenAI. The reason being is that Azure in our experience has a bit better reliability and also multiple regions. So for example, if the US West server is down for a couple seconds, the Australia server is up. You can use either when you want, but that's just a tip from us from our experience. And if you do go the Azure OpenAI route, which is something also I'd recommend. Optional, but I would recommend it. I would look at LiteLLM. This is a really, really useful open source library or application where what it does is it allows you to take multiple Azure OpenAI endpoints, load balance them with failover across multiple Azure regions. We're not going to cover it because it's not necessary for this talk, but it is something that if you do decide to deploy this at your workplace, it's something I'd really recommend looking into. It saved us a lot. Alright, back to the coding. Now we're going to create the OpenAPI agent.

(00:13:46)
And finally we're going to give our example prompt, which is what devices are currently running. And then we're going to need to set our three environment variables, which is our OpenAI API key, the Corellium API token. And then optionally, depending on if you're using an enterprise or the regular cloud version, you also need to send your Corellium API host at url. It's the same one that we set earlier in our json file. Alright, so now we're going to show, this is the fun part, the demo of what it looks like. But as Brian mentioned, this is prerecorded, it was sped up slightly, but it is pretty darn fast, so I wouldn't really…it's more of if you're staring at it, it can get a little boring, but in production as long as you're not staring at it, I don't mind the speed. So we're just going to run it and I already have the set. So here you can see the logs of how the LM is thinking internally and it goes through multiple steps. And there we go. You can see that we have two devices currently running and those have the names. And yeah, that's it for the first section. So at this time, before I go into the next section, is there any questions from the audience that we have queued up?

(00:15:15)

Brian Robison: I'm not seeing anything yet David, so if anybody has any questions, please enter them into that Q&A widget and we will get on 'em as quick as we can.

David Manouchehri: Awesome. Yeah, that was relatively simple example. Now we're going to go on to the next section. We're going to take a poll while I'm going over this. So what we're going to be talking about is, and I think this is the most exciting, we're going to show how to make a basic input fuzzer and use GPT-4V(ision) for feedback. So prerequisites are almost the same as the previous section. The only added section is we're going to need an S3-compatible storage endpoint. So based on the poll, quite a few of you have not used GPT-4V(ision) before. I'll just go into what that is. So it's very similar to GPT-4, same underlying model, but with the added ability to take text but then also take images to analyze. So that's the reason that we need an S3 or it could be any storage API. I just find S3 is the industry standard and the easiest to use. What we're going to be able to do is we're going to be able to give GPT-4V(ision) a screenshot of our Corellium device and have GPT-4V(ision) tell us about what's in the screenshot, which is really exciting.

(00:16:52)
You don't need to do OCR, it already can do that itself. Alright, so our goal is we're going to try to figure out automatically what user inputs are needed to reach a desired state in an iOS app. So we're going to just take a random app and this would be an example of what inputs that the user would need to do to open a quick start menu and enter our example app. So this is our example app. It's one that I just happened to have on my iPhone so I decided to use it. So on the left is our starting state when you first open the app. And then on the right, this is kind of what I want our desired final state to be. We're making this example a little bit simpler so we can just focus on the concept more rather than getting too wound up in the details. But there's not really any limit. You could do a simpler example so that you could actually find an answer within a minute or two. But if you want to go through really complex user inputs in series, there's no reason you can't do that. You just have to wait longer.

(00:18:06)
So this is probably, if you want to take one thing away from the talk, this is probably the most useful trick. Currently models like GPT-4V(ision), they do struggle with providing accurate positioning. Things like pixel offsets, it's not the greatest at. Now when you're interacting with a smartphone, usually you're using XY touch coordinates. This isn't a problem though because what we can do is we can use the keyboard to control our volume virtual iPhone instead of using the touch screen. This is something that when we were working on this webinar at first we were trying to use touch and it was being really inconsistent, not super accurate. And as soon as we switched to using the keyboard, it suddenly started to work on the first try, which is really exciting. So I'm just going to quickly go over how you enable that on a Corellium device or Corellium virtual iPhone.

(00:19:10)

So first you just go to the settings, you scroll down to accessibility, you go to the keyboard section, you go to a full keyboard access, you enable full keyboard access. And then now the way you can tell this is enabled is that you'll see a blue outline of what we currently have selected. Now also to make our screenshots a little easier to understand what just happened, we're also going to go to the auto hide and disable it. That's it for setup. This is a one-time thing. You can also automate this too, but for the purposes of the webinar, I figured I would just go through it manually. So let's take a moment to talk about our strategy here to find the user inputs needed to reach that desired state we talked about. First we're going to generate a list of random key presses to navigate through the iPhone app UI and then we're going to loop through each key press one at a time. We're going to press the key, wait a second, and then take a screenshot. And then finally we're going to use GPT-4V(ision) to check if any of our screenshots match the desired state. If none of them match, then we're going to go back to the beginning and repeat the process until it finally does.

(00:20:30)
We're going to keep this a little simpler today. In practice, you would want to improve this by also having a feedback loop where if it doesn't match the desired state, you would ask GPT-4V(ision) to also guess what inputs to try next. Instead of always randomly guessing, you won't have the randomly guessing. And then also asking for feedback. You would also want to add a function that uses GPT-4 to reduce the number of inputs needed. So if you can reach it in three key presses, that would be probably better in most use cases than having 10 key presses. That's something we're not going to cover today, but it shouldn't be too hard to implement from the code that we're going to provide.

(00:21:16)
First, we're going to install our target app. Again, this can be automated but we're just going to use the UI for this to keep it simple. And yeah, we've installed our example target app. So as I mentioned, we're going to need to have somewhere to store screenshots. You can use any public storage provider you need. It does need to be something that generates a URL that is publicly accessible. Personally I like using Backblaze B2, so we're going to use that today. But if you prefer Amazon, Amazon S3, feel free to go with that. If you do go to the Backblaze route, this is how you can create a bucket that automatically deletes old screenshots. And then this is also how you would generate S3, API key for it. Don't worry, these keys aren't going to work. They've already been deleted, but you're welcome to try it if you don't believe me. Alright, so our Python requirements, we will be using a pinned version of Corellium’s Python, API, same thing as before, best practice. And we're also going to use boto3e, which is our library for interacting with S3. And also we're going to use this JSON-repair library. I'll go into why that library is useful a little bit later.

(00:22:37)
Now we're going to install the requirements we just defined and now we're back to coding for our implementation. So first we're going to set up our imports. These are all the things we're going to need. And now we're going to create our loggers. For this, we're just going to be

logging the standard out. There's no reason you can't log to a file, whatever happens to be easiest for you. Alright, so here I'm hard coding our INSTANCE_ID and APP_BUNDLE_ID. Sorry, one second, I just lost my place. We're hardcoding it to keep it simple.

(00:23:27)
You can make these environment variables or command arguments. The thing I do want to point out at the very end, this SLEEP_TIME_SECONDS, what this is is we are going to have to wait between when you press a key in an app, it doesn't instantly render. This is the time we're going to wait between key presses and screenshots. So for this particular example on the device we tested about a second was fine, you can play with this and tweak it for performance. We're also going to need to create s3_client to interact with our storage bucket.

(00:24:11)
And this is the reason, so don't worry, I promise I won't keep doing this with these massive code sections without explaining it in depth too, too much. But this is just uploading. If you provide a file path, it uploads the screenshot to our S3 bucket. This isn't particularly interesting, but it takes in a file path and then it returns a temporary URL. This is the important part. It returns a temporary URL that is valid for, I think I've defined it as, yeah, returns a URL that can be visited for a day publicly, which is more than enough for our use case because we're going to be using these URLs to talk to GPT-4V(ision).

(00:25:01)
Also, as Brian mentioned at the beginning, don't worry about trying to take a screenshot of this, we'll be providing the code in a blog post later. Now this is probably one of the more important functions. This is how we actually go ahead and type in our crawling device. So the way I have things set up is first because it's an iPhone and it can have the lock screen we're going to be using, we're first going to make sure the device is unlocked. After the device is unlocked, then we're going to want to fill the app if it's running and then we're going to want to relaunch the app. The reason being is that because we're going to be calling this function multiple times, we want to make sure we're always starting with the clean state. So that's why we're killing the app and then restarting it that way.

(00:25:50)
We always know it's going to be the correct inputs and it's not going to be missing information. And then what we're going to do is we're going to take a screenshot to begin with before we press any buttons. And then we're going to step through our inputs, each key press and we're going to press a key, wait for about a second and then after a sec we're going to take a screenshot and then again loop through. And for every single key press, so we're going to get, if you had nine inputs you want to press, you would get 10 screenshots in total because the first screenshot is before we press anything, alright? This is our very, very lazy fuzzer where we're going to define some key presses up, down, left, right, space, back and forward.

(00:26:47)

These are the keys that we need to be able to control our virtual iPhone. And just something that for people who have used their iPhone with a keyboard before, you might know that back and forward. If you use those literal keys, they don't actually go back and forward. The way you would go forward is you actually press tab and the way you go back is you press shift tab. So we're going to create this little helper function that replaces the back and forward with the actual keys. I like using back and forward. And the reason why we want to use words like back and forward is because later on when we're explaining to GPT-4V(ision) what key we pressed, it's going to have a harder time understanding what shift tab does compared to what back does. So you sometimes have to think of not what is technically correct, but what would an AI guess is correct.

(00:27:52)
And in this case to most AI models a key that says back makes more sense than shift tab. So that's the purpose of this. And to segue into that, we're going to talk a little bit about prompt engineering. So our goal, we want to have a text prompt that looks like this where our prompt is going to be to GPT-4V(ision). We're going to say which screenshots have the quick start dropdown menu open. You must answer in JSON like this [=1, 2, and 3. For the screenshots that you want to select. The first screenshot is considered index one. If no screenshots are correct, return an empty list. It's important to tell it that it can return an empty list. We don't want to try to encourage it to return fake answers. And we also want to tell it that explicitly that it should start indexing at one.

(00:28:55)
If you don't, sometimes it might decide to start at index 0 and then you'll be off by one. So that's why we're being very explicit. And then we're going to want to give a numbered list where we have a description for each key press. So the first one will always be static because there is no key presses. So it'll just be an initial state, no keys pressed yet. And then whatever key presses we happen to press, we're going to list them and then say it was pressed. Now when you're talking, don't worry about trying to read this, but when we're passing, when we're using GPT-4V(ision), the way the API expects it to be formatted is you're going to have first this text object which has the prompt that we just in the previous slide. And then it's going to have in order, and the order does matter. We're going to have the URL that the picture can be, the screenshot can be accessed at here. I've just truncated the previous example, the two inputs so that you could actually even attempt to see what the slide is. But for the rest of the slides it will be 10 inputs.

(00:30:15)
So we're going to have to make some text formatting functions. The first function we're going to create is one that it's going to create a list of screenshot descriptions. So the first one again will always be “Initial state, no keys pressed yet.” And then for everyone after that, every input after it's going to say whatever key was pressed and it'll return a list of that. And then we're now going to have to combine the initial text prompts or we give it, tell what the goal is and we're

going to want to make a numbered list as well. And also give it the URLs of each image in the format it expects. That's what this function is doing.

(00:31:10)
And yeah, it returns the object that we're going to be using in a second when we talk to GPT-4V(ision). So now we're going to be creating the Corellium client and instance. And by the way, whenever you see the "..." that means I'm not giving you the full function. It means that in the next slide there's more to continue. So if you're wondering what the "..." means, that's the reason. Continuing the main function, we're going to give it our beginning of our template for our prompt where we give it instructions and we're going to start a loop. So as I mentioned before, because we're using the keyboard, this does work shockingly well. I was finding we were getting an answer usually on the first or second iteration. In more complex examples, you would want to go more towards the hundred. When we were using random key presses, sorry, touch inputs, we were actually needing closer to a hundred.

(00:32:18)
Again, with the keyboard you could probably get away with a lot less than a hundred, but that's kind of worst case scenario. And for more complex examples, you would want that. So we're going to loop through and then once we have the prompt in the format that's ready for an API call, we're actually going to go and make that call to GPT-4V(ision). Now earlier in one of the previous slides when I mentioned there was a library called json--repair. The reason that we're using that library is we're instructing GPT-4V(ision). We're asking it nicely to return JSON, but that is not…so we're asking to return valid JSON, but that is not a guarantee. So what json-repair is, it's a library that if OpenAI returns us an invalid json_object, but it's close enough. For example, they have a trailing comma which is not allowed—what this json-repair library will do is it'll remove that trailing comma. Now if for those who have experience using GPT-4 without V(ision), just regular GPT-4, there is a JSON mode for GPT-4 text. GPT-4V(ision) does not currently have that JSON mode. So that is why we're using json-repairs. If you're watching this in a year, there's a good chance that OpenAI will have added a JSON mode, which guarantees is valid JSON. But until that's added, that's why we're using json-repair.

(00:34:01)
And this is just the entire main function, all the previous things, but just combined into one slide. So you can see it's honestly not that complex. The logic is not bad and we haven't skipped any over code. This is it. Now for setup, we're going to need to define our environment variables. So the bucket name, the bucket endpoint, the keys for our S3 bucket, the OpenAPI key, and then also our Corellium environment variables. Alright, so now it's demo time. We're going to run it. So we can see here it is now pressing keys, one at a time, taking a screenshot, and then it's also logging out what the URL is just for debugging purposes. So give it a second. We're doing nine key presses in total and there we go. So you can see in the response that on the first attempt it found a screenshot that had the quick tools open and it found it in the last three screenshots there.

(00:35:23)
So starting at screenshot eight, that's when our desired state was visible on the screen. I also just want, while we have the slide open, if any of you are wondering the cost for this, it's really not bad. So you can see that we use about a thousand input tokens and close to zero output tokens. So this example to run this one iteration costs us about one to two pennies really not bad. So if you run it a hundred times, you're only looking at about a dollar. So let's confirm that what we did actually works. So remember we had screenshots one through 10, and here I've just downloaded the screenshots one through eight because eight is when it said it started to see the answer. So we're only going to look at the first eight and here I've just mapped out what keys were pressed along with what the screenshot ended up looking like.

(00:36:26)
So you can see we pressed up right, left. None of these screenshots show our desired state, we hit back, we hit right, right, and as soon as we hit space, there we go. Our quick start menu is open. So this is indeed correct. That was our goal and yeah, it works. So we're going to cover some suggested improvements on how I would recommend doing this if you want to do fuzzing here. You'd want to create device, the Corellium device on demand and install the IPA for the iOS app automatically. We do have this covered in our previous blog about how to install IPAs program from Python. You can go to Corellium’s blog for this and it's in this blog where we talk about how to install an IPA instead of going through the UI manually.

(00:37:25)
Another suggestion I would make is I would to avoid making the script too long, I kept everything somewhat simple for performance reasons. I would recommend separating the input generation screenshot logic from the GPT-4V(ision) processing stat. For example, our script, it doesn't attempt to take more screenshots until we get a response from OpenAI. There's no reason that we can't do a thousand runs, which is going to be, it's going to take less time than the time it takes to get a response from OpenAI. So I would recommend separating those. The logic there, you could also do things like you could spin up a device, a Corellium device for 10 minutes, pay for those 10 minutes, have a bunch of runs, and then turn the device off and do processing with OpenAI in another thread. And then you're not paying for resources you're not using.

(00:38:29)
Something that's really exciting about this is here we were just using one device, it was already pretty quick, but there's no reason that you can't use Corellium to turn on a thousand devices. Try to brute force and kind of fuzz all the inputs, find out all the paths for the UI and you could do it in a couple minutes. You don't need to be using one device and waiting for days to get to all the input states. That's something very exciting that with physical phones you don't want to buy a thousand iPhones to use for 10 minutes at a time. With Corellium, yeah, you can just turn on a thousand.

(00:39:17)

So that sums up that section. Now the section we're going to go into no—actually we're first going to take a poll to see if any of you have used Google Cloud Run before. Alright, the poll looks like it's up. So the topic we're going to go into, this is more of a DevOps discussion about how we could take something similar to the previous example and do nightly testing for deployment. We're going to use Google Cloud Run. I found it to be one of the easier deployment options. You're not limited to it, but I find it easy to use and our goal will be, oh cool—so most of you have not used Google Cloud Run before.

(00:40:18)
I promise it won't be too hard. So our goal will be to get an email every single night checking if our app is working as expected. So the example I'm going to give is after a series of predefined key presses have we successfully done a DNS query for Corellium.com because that's the demo app we're using. That's one of the features it has…can do DNS query. So we're just going to see every night is it working as expected? And remember, this is without having to make complicated unit tests. So here I don't have the source for the app, but I'm still able to make very nice test runs and to keep you all interested, I'm going to show you what the end result will look like. So this is the test I was doing last night. Here we have an email, and again, this is all AI generated. This is an example email that I got last night when the unit test ran as expected and here I'd have this screenshot attached as well. And here's an example email where the unit test didn't work as expected.

(00:41:36)
So hopefully that keeps you interested enough to listen through the next bit. We're going to now talk about the prerequisite. So again, similar to the previous example, you're going to need a Corellium account, OpenAI and S3 compatible storage, Python 3. The two added requirements because we're going to want to be sending a real email, you're going to need an SMTP or some sort of email service API. I recommend, and I have a feeling most of the people in this webinar are probably from a security background. The reason I recommend AWS is that it allows you to create credentials that are very narrowly scoped. So I'm going to demonstrate how to actually create that API key so that you can create an API key that can only email certain people. Again, you don't have to use AWS SES, it's just what we prefer. And then you're also going to need a Google Cloud account. The reason being is that we're going to walk through how to deploy this on Google Cloud Run. So let's talk about our implementation. So we're going to take the previous script we wrote and to begin with we're just going to delete main and then recreate it with what we have here on screen. One second, I'm just adjusting. Yeah, so here we're going to predefine key sequence.

(00:43:12)
The key sequence list is kind of like our unit test and then we're going to take a screenshot just of the final state and we're going to upload it and we're also going to create a function. That cool email I showed you. That's what this function's helping create. So we're going to do two API calls to OpenAI. So the first one will be to GPT-4V(ision). What this is going to do is it's going to instruct it to write an email body describing if the screenshot below shows a successful

response of a DNS query to Corellium.com. That means the app is working. We're also going to tell it the local time and we're also going to instruct it to say something about the current time just to make it feel a little bit more natural. Again, that's optional. It's a personal preference. It's just kind of cool when you get an email in the morning and it says Good morning. Similar to the previous because we're only doing one screenshot here, we don't really need any big helper functions. We just straight up put the text prompt and then also the URL. Something I didn't cover in the previous, but I think I should just point it out now before I forget.

(00:44:40)
In the image URL object, you'll see that I have detail and for all the previous examples as well, I had detail set to low. The reason I have details set to low is that it is quite a bit cheaper to do that. What OpenAI is doing behind the scenes is it's downscaling images to be a maximum of, I believe it's 512x512. I have to pull it up to check that, but I believe it's that which because we're using an iPhone SE, the screen is already relatively small. The resolution and detail at low is more than enough and it ends up being much quicker and much cheaper just to put detail to low by default, I think it's an auto, but for all of our examples, I would send it to low. That's the first one where we get to look through the screenshot and write an email body.

(00:45:35)
And then the second one we're going to do, if you run just the previous example by itself, what for all of you who have used ChatGPT-4 to write emails, what you'll notice is that even if you instruct it to not do this, it'll often put placeholders in the email and it looks super unprofessional to have an email sent saying, dear [user]. We don't want that. So what we're going to do is we're going to do a second API call, which we're going to use GPT 3.5 Turbo. We really don't really need GPT-4 for this because all the hard stuff, all the heavy thinking has already been done. We're going to be doing a second API call, which is purely just really for formatting. And the benefit of this as well is that like I mentioned before when we were using json-repair, because GPT-4V(ision) does not guarantee that you'll get a valid json_object back, but because we're doing the second API call without GPT-4V(ision), we're just using straight up GPT 3.5 Turbo.

(00:46:52)
This is guaranteed to return valid JSON. Another thing is that the reason why I recommend splitting these two prompts is the more instructions you give—and again, over the years this will improve and not be as much of an issue—but the longer prompt you give, the less likely it is that the LLM will follow your instructions. So I really recommend if possible to split them up to give shorter instructions. It'll be a lot more likely to follow. So the previous, if I go back one slide here, I made the GPT-4V(ision) response really as simple as it could be so that it would really focus on just giving the correct answer. And then for the second API call taking the correct answer and then doing the formatting I found from testing this works quite well.

(00:47:52)
And then finally we're just going to do, because we get a json_object back, we're going to use json.loads and then we're going to take the message body. This function isn't particularly

exciting, but this is going to be using AWS SES to send an email. And then before we send the email, we're going to be building the email body. So Python has an email message like library already for you. We're going to set our to, from, subject, and we're going to make it a plain text version and an HTML version. Only thing maybe to note here is that for the HTML version, our response from GPT from the second API call, it's going to be having new lines like /n. In the HTML version like /n isn't going to make a break. So that's why in the HTML version, we replace all the new lines with the break tag and then it renders out really nicely on Gmail.

(00:49:03)
And then at the very end we just send the email through SES. If you were to use a different SMTP provider, you would change it. This is all somewhat generic. You could easily swap it out for a different provider if you'd like. And this is what our full main function will look like where, so in the previous one, we make the Corellium API instance. We're going to define the sequence list, take a screenshot after we take, sorry, we press the keys, we take a screenshot, we upload the screenshot, then we create the text email body, and then we go ahead and send the email. Now I won't spend too much on this. I mentioned earlier the reason why I like AWS SES is that it allows you to create email API keys that are very limited in scope. This is something that took an annoyingly fair amount of Googling. So I'm just going to cover really quickly how you can limit an API key from SES to only be able to send from this address to this particular address. So you go to your IAM profile, you don't attach any groups or policies to begin with. You'll see that the permission summary is empty here after the user has been created.

(00:50:29)
Then we're going to create an inline policy, and this is the inline policy that's very restrictive where it can only send from this particular address to this address. This is really nice when you're doing things in production because if this user was compromised, this AWS account, it would only be able to phish me from a predefined address. So it's not like we would end up having our partners get phishing emails from my address. It's something that, yeah, I haven't seen…unfortunately it’s not that common among other email providers. Thankfully Amazon does have that. This is what it would look like. You just copy paste what I had in the previous slide and then now you're just able to create the policy, create the access key, and once you create the access key, save it. And now we're going to blaze in the last couple of minutes how to deploy it on Google Cloud Run.

(00:51:26)
The reason I'm including these slides is because Google doesn't do a great job of documenting how to do it. Once you figure it out, it ends up becoming super easy. But that's the purpose of this. So the reason I love Google Cloud Run compared to something like AWS is that Google Cloud Run allows you just to make a Docker file and it just deploys the Docker file. So this is a really simple Docker file for our example. The only two files we need are the requirements and then the main Python file and we need to install the requirements. We're also going to define the initial command. We're going to create a new project on Google Cloud. And then this is the part

where, yeah, Google does a terrible job of documenting it. Hopefully I will help you here. You're going to want to use artifact registry and create a new Docker repository.

(00:52:21)
You're going to want to define the region. So the reason we're sticking with North Virginia is because that's where we're putting our S3 bucket and where we're going to be deploying everything to. You can make it multi-region, but let's just stick with one for now. Once you've created it, you should see something like this. And now we're going to go ahead and set up, we're going to connect, we're going to go to Google Cloud Build, which will connect to our private repository that I've put this code in, and this will do continuous builds. So here they support GitHub, but you can also use Bitbucket or any other. They have support for quite a few here. This is the name of the other repository I have. And then once you have hit next, you're going to want to create a trigger.

(00:53:11)
This one, anytime you push to Maine, it's going to do a build. The most important piece of this section is the image name. The default will not work. That's the image name. The thing there that will work for what we're doing. That's the only thing that I've really changed and that took a couple of minutes of filling around to figure out once you've created it, then once you do a push or you manually cause the trigger to run, you're going to see…I've made a few mistakes, but you'll see a check mark once you have a successful build. Alright, now we have a successful build. We're going to go to Google Cloud Run and we're going to create a job, which means we can run this every X amount of minutes or hours here. We're going to use our container URL, that was from the previous step.

(00:54:09)
We're going to give it a name and we're also going to keep it in the same region. You're going to want to define your environment variables very similar to how we were doing with the command line earlier. And you're always going to want to define your secrets. I have not put my secrets on screen for obvious reasons. Alright? And this is what it looks like when it's running. We're also going to make this a nightly job. So we're going to create a trigger where every day at 8:05 PM my local time, it's going to run this. Once you create the trigger, we're going to wait a few minutes until it's 8:05 and you'll get an email with the result. Thank you. I believe that's all my slides. We're going to open it up to questions.

(00:55:09)
Brian Robison: Thank you, David. That was a lot, but it was really, really, really awesome. So I really appreciate you taking the time to create all of this and put it together and then present it today. So that's very, very much appreciated. Does anybody have any questions? There's still quite a few of you that have hung out, which is great. If you have any questions, feel free to type 'em into the question and answer section. And we have a few minutes left so we can do that. I'm going to go ahead and just pop up our contact info on the screen. So again, while there's questions, please feel free to take note of this. Again, all of this will be available for you later. So

someone just asked David, they'd like to see some pen test cases for iOS and Android using AI. So I think we might be able to do some of that stuff in a future either in a webinar or even a follow-up blog post for you.

(00:56:21)
David Manouchehri: Yeah, that's part of the reason I went over the input fuzzing because at least the way I've done it, when you're pen testing an app, one of the first things you want to do is explore all the functionality. Instead of you sitting there tapping through every single menu, it's kind of nice to just be able to have a loop that just tries everything. And then after a couple hours you find out all the weird sections of the app you didn't know existed. You don't have to go through it manually. You can also, with Corellium, when you're using the keyboard, you can have it type things. So it's possible to have, say you have an app that has a login, you can use GPT-4 along with Corellium API channel. If there's a login window, here are the credentials. So you can have automatic login logout. You do need to put some safeguards around that because one of the examples I was going to use was WhatsApp, but then I realized that eventually the AI will find a way to get to the account settings and delete the account. So you do need to put some safeguards if you're using accounts to make sure it doesn't, if you have some delete account and make sure you give it some instructions, like never click that.

(00:57:42)
Brian Robison: Yeah, that's a great point. That's a great point. So what tier is required to follow this demo manually? Not exactly sure about what you mean by what tier.

David Manouchehri: I believe they're referring to what tier of Corellium.

Brian Robison: Perhaps? Yeah, which license category?

David Manouchehri: Brian might be able to confirm, but I believe professional.

Brian Robison: Pretty much every one of our accounts, whether you're an enterprise customer or even a personal user, has access to the full API. So even with a professional, which is relatively low cost, is able to use that. So if that doesn't answer your question, feel free to reach out to me, drop me an email and I'll get back to you on that. Have you had any success using text-based source code, decompile code fuzzing with GPT-4 with a start and end function rather than using images?

(00:58:53)
David Manouchehri: Yeah, so internally we've hooked up Ghidra’s decompiler with GPT-4. It does work pretty well. Something that we haven't done yet that I think would be really useful is

combining both the decompile version along with screenshots of what the app is doing. Another thing that, there's something we haven't tried yet, but one of the nice things with Corellium is that it has Frida pre installed. So I think a really cool use case would be using a decompiler along with using Frida and telling GPT-4, Hey, here's an example Frida script that can do something. Could you write a Frida script that tries to find reachable functions? No, that's a good question.

(00:59:47)
Brian Robison: Definitely. And have you found any real world software vulnerabilities using this approach? You don't have to be specific.

(01:00:03)
David Manouchehri: Yeah, so there's one vulnerability that we found manually maybe a year or so ago, and we were able to find it again. This is all relatively new. We've been working on this for maybe the last couple of weeks. So we've been able to reproduce vulnerabilities. We haven't yet done it on a larger scale yet because we're just kind of validating. The answer is sort of. We have found vulnerabilities that we found manually. And yeah, that was kind of cool to see that it does work. Yeah.

(01:00:44)
Yeah. That's really awesome. Alright, well with that, we are out of time for today. David, thank you very, very much for this wonderful content that you have behind the scenes developed and presented today. I really appreciate it. Corellium appreciates it, and I really hope that all of you on the line today have enjoyed this presentation, maybe had a little fun, give you some ideas to begin doing some other levels of research and work within your own projects. Reach out to us if you need anything. And with that, we'll sign off for today and wish everybody a fantastic rest of their week and see you on the next webinar in February. Thanks everybody, have a great day. Take care.

#####