Automating Infrastructure as Code with Ansible and Molecule

featuring Matthew Heckathorn as Interviewed by Suzanne Miller

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Suzanne Miller: Welcome to the SEI Podcast Series. My name is Suzanne Miller, and I am a principal researcher in the SEI Software Solutions Division. Today I am very happy to be joined by Matthew Heckathorn, an integration engineer with the SEI CERT Division. We are here to talk about his recent work on developing Ansible roles, a particular type of role in infrastructure as code and DevSecOps with confidence.

Welcome, Matt. May I call you Matt?

Matthew Heckathorn: Yes.

Suzanne: Great. You are new to our podcast series, and what we like to do before we get into your work is tell us a little bit about yourself and what brought you to the SEI and what is it that an integration engineer does that is really cool?

Matthew: Well, so, I have been at the SEI now for 13 years. I have been here

for quite a long time. I actually started out as an intern. I did my master's degree over at the University of Pittsburgh, which is right up the street from the office. There was a man who taught a class there named Sid Faber. He used to work at the SEI. I ended up as an intern for him after I had him as a teacher, and I never left. I have been here for a very long time. I have bounced around from a bunch of different teams here. I actually started out assisting with the Vulnerability Disclosure Department here at CERT. Then, I switched to the Network Situational Awareness team, and I did some network monitoring work here. I actually taught a few classes at FloCon on SiLK, which is our NetFlow monitoring tool. Then I got really into automation and in particular, I got really into system administration automation. I started dabbling my toes into Ansible and into Terraform and some other tools. I ended up on a team that managed a data center for a government sponsor here. That was where I became an integration engineer. My primary role is mostly in automation, in particular, around this idea of DevOps. It is mostly infrastructure automation is what I mostly do. I dabble in things like CI/CD [continuou integration/continuous development] pipelines. I dabble in infrastructure-as-code, like I mentioned, Ansible and Terraform. It is all about automating the deployment and configuration of IT data centers. I get to work with some cutting-edge stuff. I get to work with some particularly really interesting tools. I get to automate as much stuff as I want, which is always really fun. Yes, that is what I do here.

Suzanne: OK. One of the things in my experience working with folks like you is that you are wonderful at automating the things that are routine and boring. So kicking off a little, there is a whole bunch of things in system administration as well as in managing DevSecOps pipelines that are very repetitive and things you have to do a lot. You have to do things like destroy the old environment before you can have a clean environment to test in again and things like that. That is really the point of some of these tools like Ansible and some of the other tools you are going to talk about is to help engineers to kind of get that boring-stuff-but-has-to-be-done automated so that they can really focus on the intellectually challenging aspects of whatever it is that they are developing, right?

Matthew: Yes. You are pretty on point. The main thing that I am mostly tasked with here is standing up fresh environments that are configured the way that we want them to be configured. We want them to come up correctly the first time every time.

Suzanne: Right.

Matthew: There is this adage in DevSecOps and the DevOps world of treating your infrastructure not like a pet but treating it like a cattle. The idea is that if there is a problem with my infrastructure, I don't bother putting a lot of effort into trying to fix it. It is not a pet. I will just get rid of the old infrastructure and create it fresh from a fresh state. Now, I don't have to worry about debugging it anymore. It saves us a lot of time and effort. That is where a lot of these tools like Ansible or Terraform, these infrastructure-as-code tools, really start to shine is they enable us to automate and be confident that it is going to come back up. We are not going to destroy something and then, *Ah! Where did everything go?* That is really their main purpose.

Suzanne: It is that confidence that we can re-create. That is one of the things that makes using the cloud and using DevSecOps pipelines as a foundation for giving us confidence in our code that are really important. The pet analogy, don't name your infrastructure. It is like don't name it if you are going to destroy it later.

All right. Getting into Ansible itself, you recently wrote a blog post that provides some wonderful guidance on how to begin developing what Ansible calls roles. Let's talk about what is the function that roles play in Ansible, and how are they different from the roles. When we usually hear the word *roles*, we think about people, systems engineer, architect, things like that. What is different about the way Ansible defines roles, and why is it critical that the Ansible roles function the way that we intend them to?

Matthew: The first part of that question is basically what is an Ansible role. An Ansible role, if you have never done any development work in your life or anything like that, is kind of like a function or an object in object-oriented programming. It is this idea that we have a reusable piece of code. In the Ansible world, that role, that reusable piece of code, is easily shareable. We package it up, and we can share it around publicly. We can share it around internally privately. It really enables us to have a common configuration for that particular role. Now that leads into your second part, the latter half of that question, which is basically what is the difference between an Ansible role and like a role, like a person or a sysadmin or something like that. They are actually kind of similar. An Ansible role would be like a web server. *This particular node, it is going to have a role of web server, and this other particular node is going to have a role of database server* and so on. It is not too different from a person having a sysadmin role or a DevOps engineer role. There is a different purpose for the node, and we give it a role. Now, a web server role, that is very generalized. At the end of the day, Ansible roles are a little bit

more specific than that. For example, if we go back to the web server example, there are different types of web servers. There is an httpd apache web server. There is an NGINX web server. There are all kinds of different web server software out there. You might have an Ansible role for NGINX, or you might have an Ansible role for Apache HTTPD. Technically, at that point, your role is less generalized. We're talking about this is the NGINX role. Its purpose is to deploy and configure the NGINX software suite.

We can generalize that and say, *Web server role calls NGINX role* if we wanted to, but we don't have to get that generalized when we are talking about Ansible. We can just simply call a role called NGINX, and now our node is a web server because the page is running the NGINX software suite.

Suzanne: I come from a different generation. I come from the…My first program was written in the 1970s. It is a whole different infrastructure mindset. The way I kind of resolve it in my own head is a role has a set of responsibilities to execute. A systems engineer has a certain set of responsibilities. A reliability engineer, who is a specialized systems engineer, is even more specific. I think about the Ansible roles that way. They are a role that performs a purpose in the way that a human does, but it has specific responsibilities. That is the part that actually gets complicated in terms of writing those responsibilities for that role, so that they are correct for the environment that you are trying to configure. Is that accurate?

Matthew: Yes. That is a pretty good analogy. Additionally, one other thing is that roles, Ansible roles, can actually rely on other roles just like people have to rely on other people. You can have requirements in your roles that are essentially like, *Before you deploy NGINX, you have to make sure that the EPEL repo is there* or something like that. That is a just super general example. You are able to kind of layer them in a way where it is like, *This role requires this role, and this role requires this role*. Another good example is we here at the SEI, we publish a NetFlow monitoring tool called YAF, Yet Another Flowmeter. YAF requires the fixed buff C library to be installed. I actually have some open-source Ansible roles for managing that software suite. My YAF role relies on the fixed buff role. There is a good example of some nested stuff there.

Suzanne: Ansible roles are strongly used, are often used in infrastructure as code for obvious reasons because they can set up a set of conditions and a set of activities that are going to allow us to configure the next test increment or the next test configuration or the next development configuration that we want to use in a particular environment. Why would you need to go beyond

the Ansible roles that have already been published? There have been a lot of Ansible roles that are published already.

Matthew: Red Hat owns the Ansible tool. There is a web service out there called Ansible Galaxy, it is galaxy.ansible.com. That is where you would go to kind of find the Ansible roles that are published out there. Under the hood, Ansible is really just doing a Git-clone or a Git pull. It actually is pulling the roles from a Git repo. You technically can get them from any Git repositories: GitHub, GitLab, any of the public ones, or your private Git repos inside your company, you can get them from here. Which kind of leads into why you might want to develp your own. For example, let's say that you are a company, and you are releasing a new application, and you want to have some common configuration for that. You might want to support not just containerizing your app through something like Docker, but you would also want to support allowing people to install it on the server itself, on the host. You would use a tool like Ansible for that. You could develop that role internally before you publish it externally for your customers. Additionally, in our line of work, we often have government contractors that have like different sets of requirements where it is never going to be public. We have to pull from different sets of resources or different things like that. That allows us to develop that role internally that way as well. Then, additionally, Ansible Galaxy, while there are roles out there for doing different things, sometimes they might not do what you want them to do.

Red Hat owns it, but the roles are really published by the community. The quality of roles is kind of all over the map. You might have some roles out there that you are just like, *Oh, this is the best thing I have ever found.* Then you might find a role for a piece of software that you are like, *Oh, OK like this doesn't do anything. It is broken. It is out of date*. There are a lot of reasons why you might sit down and write your own role. I will say though that you should check first. You don't want to sit down and just write a role, unless you are trying to learn. You don't want to sit down and just write a role for something that is out there that already does what you want. The first thing you want to do is sit down and look. That actually is a useful experience as well because you might find something that is 90 percent of the way there. You are like, *Well, I just need to tweak a few things in this role. Now it does exactly what I want*. Because it is just in a Git repo, and it is often hosted on GitHub, you can simply fork that role, add the functionality, and then see if you can get a pullback upstream. Or, if it is no longer an active role, which I have had to do before, like the maintainer has been gone, I will just maintain it myself going forward. There are a number of reasons for writing your own role. I have just gone through a whole bunch of them.

Suzanne: Well the learning function, that is another one. Sometimes that is the best way to learn something, is to try and write one on your own. One of the things that you talk about in your blog post is there are also now some tools that will help you to write roles, so that you don't have to learn. Somebody like me, I am just not ready to go there. I am going to want to use tools like you mentioned like Molecule that are going to give me some templates, give me some guidance, and are going to help me to do that. Talk a little bit about what that class of tools helps you with in terms of writing Ansible roles.

Matthew: Right. When you first start out writing an Ansible role… So, the purpose behind Ansible is it is an infrastructure-as-code tool. It falls under a class of infrastructure as code tool that is more targeted towards orchestrate, or not orchestration, I am sorry, configuration management. The other class is orchestration, and while Ansible can do that, it neatly falls under the configuration management bucket. The idea is that you use Ansible to configure a service or an application on some remote server. When you first sit down to write it, you sit down at your Mac laptop or your Windows or Linux laptop. Oftentimes, when we're talking about configuring a remote server, we're most of the time talking about Linux servers or Windows servers and so on, right? But a lot of times, we're configuring Linux. If I am sitting on my Mac, and I want to start targeting a Linux server, I now need a Linux server. Now, in order for me to do that, I need to figure out some way of standing up my infrastructure and managing that infrastructure just for testing, just for development work and testing work. You could, if you wanted to, you could sit down and you could use something like HashiCorp Vagrant tool to stand up a virtual machine on your laptop. You run a command. You get an Ubuntu virtual machine. Then, you open the port so you make sure you can get to it. Then you tell the Ansible, *OK, target this. This is the Linux machine*. You could also use your IT services infrastructure, right, instead. You know, maybe they're running a vSphere cluster VMware vSphere cluster. They can manage it for you. They could stand up the machine for you, give you the connection information and you could be like, *All right, Ansible*. *This is my remote server*.

The problem you have though especially when you are starting out is side effects. When you are writing a role…Let's go back to the NGINX example. I am trying to write a role. I want to create an NGINX service running on a Linux box. I have to install NGINX, and I have to configure it to do what I want it to do. I want it to listen on port AD. I want it to listen on port 443. I wanted it to re-direct traffic between the two. I go out. I write my role. I push go.

Ansible connects to that remote Linux machine or that virtual machine on my laptop. It installs NGINX. It drops in the configuration clouds where it needs to go, and then it starts the NGINX service. Now, if you are great, and you are amazing at this, it comes up correctly the first time. Done. Move on with your life.

Suzanne: That is not me.

Matthew: Yes. So, most of the time, it doesn't. Most of the time you make a little mistake where you put a period where you didn't mean to in the configuration file or you screwed up, or you screwed up the syntax in some way, or you didn't install the right package. Because package names are different between different types of operating systems. So, what happens? All right, well, now, you got to debug. Why didn't NGINX come up? You log into that remote server or that virtual machine, and you try to figure it out. All right, so, you go into the configuration file, *Oh, there is a period*. I delete the period. Then I save the file. I restart the serviced. *All right, that is it*. You go back to your Ansible code and you find where your bug was, you fix it, and then you re-run your Ansible code. So what is going to happen? Well, it connects to that server. Remember, you already logged into that server to debug, and you already fixed the error. What Ansible is going to do is it is going to try to drop in the replacement configuration file, but it is going to notice that it doesn't have to, so it is not going to do it. It is going to move on. Then, it is not going to restart the service, because it doesn't have to. The service has already restarted. It is already working. Essentially, that second time you ran it after you debug your problem, it didn't do anything. If you start from a fresh server, is it going to work right the first time? You don't know. You're uncertain now. What you could do is you can log back into that server, and you can uninstall NGINX. You could delete the configuration file, and you could try to track down any other traces of things that were put in place, and then you could re-run the Ansible role. Now, of course, there are always little things that happened. In between there might be some manual stuff that you did that you forgot about when you were debugging. You are still not 100 percent certain that if you start from a fresh server that that role is going to work. The best thing for you to do would be to destroy that testing infrastructure and re-create it. Now, if you are talking about a virtual machine on your laptop, that might take a few minutes. You run a Vagrant command. You destroy it, or you click through the GUI virtual box, whatever you are using. Then, you re-create it, make sure you can connect to it, and then re-run your Ansible role. That is fine. A few minutes, it is not bad. I mean, it is a little manual, right? Because you got to run the commands. You could write a script to do-it-yourself, write a bash script or something that does it. It is not

terrible. If you are talking about remote infrastructure that your IT services team is handling for you, that is going to take longer. You probably have to put in a ticket or something. If you are handling it yourself, it is still going to take longer, a few minutes, and stuff like that. Especially if you are trying to develop this, and you are making mistakes a lot, it gets annoying very quickly. You could try to roll your own tool for handling it, or you could see if there was something out there for handling it for you, which is what Molecule is.

Suzanne: Where was Molecule in the '70s? That is what I want to know.

Matthew: Yes. Well, Molecule is a tool that is really just for Ansible. The idea behind Molecule is it handles the complete lifecycle of your Ansible development or testing environments. Then, it does a bunch of other stuff. Beyond just handling the infrastructure you can have it kickoff other tools. Things like linters or testing frameworks. It will pull in requirements for you. If you have like Galaxy requirements, like if you are mobilized on another role, it will pull those in for you. It also handles this idea of checking for idempotence. In the Ansible world, it is very important. You only want to change things if they need to be changed and you wanted to come up correctly.

Suzanne: I want to interrupt you for just a second because a lot of people that are not deep into this world do not understand what, I can never say it right, idempotence.

Matthew: Yes. So, yes, it is this concept that, no matter how many times you execute your Ansible role, it only changes things that it needs to change. It achieves the same result over and over and over again. That is the baseline definition of idempotence. Same thing, over and over and again.

Suzanne: It is a great word but you got to explain it the first time.

Matthew: It is a software engineering term. Of course, great word. Essentially, what that boils down to in the Ansible tool world is you run the role, and it does its stuff. Then you run the role again. If your role is idempotent, nothing should happen the second time. It should just power through. Every step, nothing should get changed because it got changed the first time. There might be some one-offs and stuff like that where you need that to not be the case. I have certainly run into those. In general, you don't want your Ansible role to make a bunch of changes. Then, when you re-run it, to make a whole bunch of changes again. You want it to make all the changes, it needs the first time it runs. Molecule helps you with that.

Molecule just does it. If something changed the second time around, it tells you, *All right. Well, your role is not idempotent*. Then you got to go back in and debug and figure out why. It is a very powerful tool and that is about diving too deep. What it is our next question?

Suzanne: Well, you go into a great deal of depth in the blog post. And so, I would say for people that have gotten to this point and go, *Oh, I need to understand all of this*, go to the blog post because you give some wonderful examples of things that you can do and what the ways in which Molecule can help you with your Ansible roles. What I also noted in the blog post that you also had a few cautions in terms of, *You are using Molecule but what are some of the things to be careful of if you're new to this particular tool set and using it?* Why don't you give us a couple of those?

Matthew: Right. So, when you first get started with Molecule, one of the provisioners that it can use is Docker, OK?

Suzanne: A very common provisioner.

Matthew: Yes, and the beauty behind it is I mentioned that if you were to stand up virtual machines on your laptop, it might take you minutes to destroy them and re-create them which isn't very long. With Docker, we are talking about seconds. It takes a second to destroy a Linux operating system and re-create it, right? Additionally with Docker, we can use Molecule to target a variety of different operating systems and operating system versions, right? We can create a matrix of different things that we want to support. Let's say, for example, back to the NGINX example, I want to write a role for NGINX that works across Debian, Red Hat, Red Hat 7, Red Hat 8, Debian 10, Debian 11. Well, in Molecule, that is simple. In my Molecule file, *I'll just add a bunch of different areas in there, call in my different Docker containers, and it will stand them up in parallel. It will run that role against them all, and then it will give you feedback on which ones worked and which ones didn't*. The gotchas though are containers are not full operating systems. They are containers, so they don't have everything that you need. Oftentimes, the base containers are missing, things like system D, which if we're talking about services in the Linux world, well, we need it. You get into this, *All right, I am starting with a new role, I have got to figure out what is my baseline Molecule configuration.* Luckily, there are a lot of people out there that have been doing this for a while. In the blog post, I call out a guy, his name is Jeff Geerling. He has written a book on Ansible. He has got a bunch of YouTube videos on it and has presented at Ansible fest and stuff. He has some great Ansible roles out there. Lots of times, when I first start writing a

new role, I go to one of these Ansible roles and I look at his Molecule configuration file and I look and I am like, *All right. This is how I want mine to be*. What he is relying on is he also maintains a set of Docker containers that have things like system D and it already have things like Ansible in already. So that you can kind of ignore a lot of the things about Docker containers not being full-blown operating systems. There are a few tweaks that you also need to maintain in the Molecule file. It is in that blog post as well. It is from Jeff Geerling's which is, you need to mount certain things inside the container. You need to make it basically so that the Docker container oftentimes has access to the Docker socket and stuff like that for it to work legitimately like a real operating system.

That is really like the main gotcha. You want use something like a container because we are talking seconds versus minutes here. It is crazy how fast your development cadence can be. When you just can destroy it and re-create it, and it takes almost no time to do. It is like, *Oh, I messed that up,all right. Goodbye*. *Now I am back*. The seconds versus minutes doesn't seem like a lot, but it kind of really adds up. You do want to use containers, and so you do need to try to avoid some of these gotchas. Yes, that is the main one. The major one is that one. There are some other ones that I mentioned in there but in general, it is just a different mindset. *I am not in a full host OS. I am in a container, so I have to be a little careful.*

Suzanne: OK. That is good advice. If I am a developer, and I want to start creating new roles in the Ansible sandbox, where would I begin? What resources and tools [are available]? You mentioned Jeff Geerling's work. What are some other things that are available to me that I should be looking at to have the best experience in learning this?

Matthew: Well, like you said, Jeff Geerling's GitHub repos are a really good starting point. There is also the Molecule blog post that we have out there. It gives you a good starting point on how to get started with the tool. In general, like the Red Hat Ansible documentation is actually very good. I find it extremely well-written. There are lots of good examples. The Ansible Galaxy is also a good point, a good place to start as well. What you tend to do when you start using Ansible is you find an author on Galaxy that writes roles that you think are well-written. You will look for something, and you will be like, *Oh, I know that name. I'll look at his role first*. Jeff Geerling is a good example that. All of his roles are pretty well-written. I found other ones out there where I recognize their picture or something like on the Ansible Galaxy, and I am like, *Oh, all right. I'll start there*. In general, the best way to start is to get your dev environment set up, which the blog post calls out how to do and

then to start messing around. The beauty of using something like Molecule is it is all local. It is all on my laptop. I don't have to rely on my IT services team to stand up infrastructure for me. Because we are relying on people that have done a lot of work for us, things like Jeff Geerling, or, if you want to look at my open-source GitHub repos for this work as well, which is on the SEI GitHub. You can just copy and paste our Molecule configs and start with them.

You can just be like, *All right, I got a start in config, I write Molecule tests, and I am off to the races*. That is the beauty of the tool. You can pick it up and move it around because it is just using Docker under the hub. It will run the same on your laptop as it does on mine. It is a great way of getting started, and it really does kind of help with the learning curve. I have always found Ansible in comparison to some of the other infrastructure-as-code tools to having like a shallower learning curve as it is. It is a lot easier to get started in. But this really lowers the barrier to entry a lot, using a tool like Molecule.

Suzanne: One of the things you mentioned in your post is that Ansible roles themselves can benefit from a DevSecOps approach. We have got a circular thing going on when we use Ansible in the service of a DevSecOps pipeline. Talk for a minute about how would I think about the DevSecOps approach to writing Ansible roles.

Matthew: Right. DevSecOps really is this idea of thinking about security throughout the entire DevOps pipeline. The DevOps pipeline is that infinity loop that you have probably all seen. If you haven't, the idea is that you start planning, you code, you build, you test, you release, you deploy, you operate, you monitor, and you are back at the beginning. That is the DevOps loop.

Security, the DevSecOps pipeline…There is a big security bubble around the whole thing where you are thinking about security throughout. With that being said, when we are designing our Ansible roles, and we want to think about it from that philosophy, we can think about adding security in early, during the planning stages and stuff like that. For example, back to the NGINX example. If I am running an NGINX server, and I want it to be more secure, I probably want to enable TLS. I want to enable encryption on that server. When you are designing your Ansible role, you put that during the planning phase and say, *We need to provide support for security from the beginning*. We need to provide support for allowing our users of this role to enable and configure TLS. You can put it in early like that in the planning stage. But then, additionally, I mentioned this, Molecule has the ability to call linters, or you can call linters yourself. There are linters out there for YAML

Lint, because Ansible code is written in YAML. I tend to use YAML Lint. That one is an early DevSecOps tool. It is this sort of static analyzer of your Ansible code. It really is more targeted towards giving you a common look and feel to your code. If you write this, and you run your YAML linter against that and somebody else writes it in your organization, another role, and they run the YAML Lint—if you were to look at the two YAML files side by side, you are going to have a common look to them. They are going to be formatted in similar roles and stuff like that. YAML Lint adds a little bit of that to it, which is good because we are all people, and we all need to be able to look at each other's code. It is helpful if everyone is using some standard format. That will add a little bit of security. It is not straight security targeted, but it usually falls under the static analyzer bucket. Additionally, there is another one called Ansible Lint, which tries to enforce some best practices for your Ansible code. So, YAML Lint is look and feel. Ansible Lint would be things like, *You shouldn't use the command module. You should try to use something like this*. It is a little bit different. It is a little bit higher level than just the look and feel. It is more like, *We suggest that you don't use these modules in this way because it is not best practice*. Whether or not it is a security issue or not, you know. Again it falls under that like static analyzer bucket thing. Finally, Molecule also has the capability of calling testing frameworks like I mentioned. I tend to use one called Testinfra, which is a Python library. What will happen during the test phase of my Molecule matrix, is it will hand it off to Testinfra, and I will write some Python-based tests in there to check things. I will check and make sure that the NGINX service is started. You could write tests in there to make sure that TLS is enabled and working. During the testing phase, you can add some security-related tests in to make sure that things are working there. Really with DevSecOps, back to what I said, you are trying to think about it from every step in the DevOps pipeline. Planning stages would be adding security in early on. Then the testing phase, right, is where we're calling like something like Testinfra and checking for it there. That is sort of how you would think about it from like a DevSecOps perspective.

Suzanne: OK. Great. Thank you, Matt. I want to know what you are working on now. You have written this blog post, you have sort of done a bunch of this work, what are you working on now that we are going to be able to bring you back to talk about in six months?

Matthew: Well, so I do a lot of work with containerization. I work with Kubernetes and Docker containers and stuff like that. I also do a lot of work with cloud infrastructure and automation. One of the things I have been tossing around for a while now is how do you work with a tool like Terraform

in a team setting. It is really easy to get up and running with Terraform as an individual, but there are a lot of considerations and even a lot of, not a lot, but there is some infrastructure that is required if you are going to start moving into running Terraform as a team. There are some like pre-baked stuff out there that you could utilize Terraform. HashiCorp has their own cloud like HCP thing offering now. But if I wanted to stand up infrastructure and start using Terraform the right way as a team with people like reviewing my Git pull requests, I am sorry, how do I do that? That is my next thing that I am looking at.

Suzanne: Excellent. Well, I look forward to having another conversation with you when you get a little farther with that work. I want to thank you very much for talking to us today. I know that it is difficult to converse about these kinds of things without having a code listing in your hand. I really thank you for having that conversation. I want to tell our listeners and our audience that we will include links to lots of stuff we have talked about, many resources. We will link to those in the transcript including Matt's blog post, of course, on this topic, which is one of our primary references. A reminder to our audience, finally, that our podcasts are available everywhere: SoundCloud, Stitcher, Apple Podcasts, Google, everybody in the world. Of course, if you want the video, you've got to go to the SEI's YouTube channel. We hope that some of you will do that as well. And I want to thank everyone again for joining us, especially Matt and everyone. Have a good rest of your day.

Matthew: Thanks for having me.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please do not hesitate to email us at info@sei.cmu.edu.