EPISODE 1433

[INTRODUCTION]

[00:00:01] JM: The software supply chain consists of packages, imports, dependencies, containers and API's. These different components each have unique security risks. To ensure the security of the software supply chain, many developers use tools to analyze and scan their infrastructure for vulnerabilities. Barak Schoster works at Bridgecrew, a DevSecOps cloud security platform. He joins the show to talk about the risks of the modern software supply chain and what his company does to alleviate it.

[INTERVIEW]

[00:00:29] JM: Barak, welcome back to the show.

[00:00:32] BS: Hey, Jeff, how are you?

[00:00:34] JM: Not bad. I want to talk about a few subjects today. We've talked about Bridgecrew in the past, which is the infrastructure security platform that you've built. I guess to start off, I'd like to get an understanding for how Bridgecrew has evolved since we last spoke. And maybe you could just touch on what your focus is as a developer security platform.

[00:01:01] BS: So since we last talked, Bridgecrew have created an extension to every step of the developer pipeline, from having an IDE extension, to Ci plugin, pull requests plugin and runtime, ability to scan infrastructure from code to cloud and identify misconfigs and briefs. And I think that over that past two years, the thing that we've learned, and all of the market have evolved, is that there are other risks in the software delivery framework. The market have seen attacks like the Codecov attack, Log4Shell attack, and other attacks on companies that really change a few weekends for a few a SREs out there. And we thought of how can we help those stories and security practitioners to know about those risks upfront.

So over time, Bridgecrew have added the ability to scan open source libraries for known vulnerabilities and open source images. And now, also scan the version control system

configuration and the CI/CD configuration. So I think that all of that is tied into how we define the supply chain of code. And I don't know how deep you want me to go with it. But there are many aspects of it that we discovered that we should inspect and fix in our code.

[00:02:34] JM: When you talk about the supply chain of code, give an overview for what that means to you.

[00:02:42] BS: So the term supply chain was kind on all manufacturers of cars at the beginning. In a car that's being built from an engine, a bunch of electricity products, some gas. And the thing that builds it is the pipeline to build a car and the people who is assembling of the different parts. And you can think about software as a similar piece. You have the CI pipeline. You have the full stack developers who takes different building blocks from the open source, whether it is Docker images, or open source packages. You have the infrastructure teams writing infrastructure code. You have the operation teams maintaining the runtime environment. And everybody are tasked with assembling the thing that we call an application software.

So when we talk about supply chain, we talk about all those different pieces and the security of each part. Let's take a few examples. The classic one that we've already talked about is security of the infrastructure code, which is the equivalent of your cloud security at runtime. Is your S3 bucket encrypted? Not public? Is your application standing behind a WAF, web application firewall, and is it configured correctly? Then the second piece is are you using a vulnerable Docker image for your containers? And what impact does the vulnerabilities within that Docker image has?

Some of those vulnerabilities are coming from open source packages. For example, in vulnerable NPM packages. A few months ago, we had an open source maintainer getting pissed on Fortune companies that are using Color.js. And he have decided that the model of open source and the compensation of open source maintainers is not fair in the world. And he decided to destruct some of the Color.js code base. And that led into destruction of the AWS CLI code base, because there was a missing locking piece between the imports of open source packages.

So open source packages is another vulnerability in yours supply chain, but also the configuration of the pipeline itself. So another attack that was interesting around June 2021 was an attack on Codecov, a code coverage software. And the TLDR of it is that the attackers have somewhat exploited the process of how Codecov is building Docker images. They used a script to modify the Docker image itself and used it to steal invariant variables from all of the CI/CD environments where Codecov was deployed. So Codecov is the most popular tool, or one of the most popular tools. For code coverage reports, you need to deploy it on a lot of your pipelines to get a sense of how good is your testing coverage. And it's very easy to install, very developer friendly.

And one of the images had a vulnerable piece that was injected from untrusted source that have leaked CI secrets, like your AWS access keys, your admin credentials, whatever it is, to an unknown, malicious actor. And one of the capabilities that we try to introduce also in our open source tooling in Chekov by Bridgecrew is the ability to verify that not only your open source packages are not vulnerable, and that your infrastructure as code is not misconfigured, but also identify if your version control system is defined correctly. For example, all of the commits are signed. You have a code review process with the mandatory approvers. And you have SSO. You know to identify the code contributors. And also, that your workflow does not use secrets in every context. And those secrets are only being used when necessary. So the goal is to have a full software delivery pipeline from open source packages, to infra, to the pipelines themselves. That is, you have trust that it is really hard to inject malicious actors into it.

[00:07:18] JM: So you outlined a lot of different attack vectors there. How do you insulate against the wide variety of attack vectors that could present themselves?

[00:07:31] BS: So the main risk to software delivery pipeline is people. And the main solution to those delivery pipelines is people. And the best thing that you can do is give all of those different engineers the power and the knowledge to understand that if they've done a mistake, to fix it very early on. So the thing that we highly recommend is to have something across the different delivery interfaces, whether it is your JetBrains ID, or VS Code, or your GitHub action, and have something to lint your code, whether these workflows files, or anything else, and understand if you're putting your company at risk. If, as an individual, you understand that very early on, you can save your company a lot of headache.

And the other way to do it is to introduce an investigation process where not as an individual who is coding a specific feature, but as maybe a security practitioner that overlooks the entire company's activity to have an investigation that will provide you the insights to inspect every step of this pipeline, from the open source capabilities, images, co-deliveries and version control system configuration.

[00:08:56] JM: Is there one particular domain that you see attacks occur in the most – Like, a category? Like, I don't know if it's through people pulling in bad dependencies in NPM, or using some insecure container image? Is there anything in particular that you see as the most vulnerable part of the of the software supply chain?

[00:09:26] BS: I think that's one of the common attacks is crypto mining. A lot of actors are trying to utilize different steps of your software delivery pipeline to inject a crypto miner. So one thing that was prevented about a year ago was the ability to run a GitHub action by any external contributor to a public repository, which led to running crypto miners on the expense of the repository owners. GitHub blocked that capability or made it harder. Not completely blocked. But people are still trying to find ways to do that.

Another way to do that is to inject a crypto minor to the init script of an NPM install or of a Python package install. So the setup phase can be vulnerable. And also injecting malicious code to Docker images that are not signed. So having unsigned and unverified workflows can lead to those activities. But the worst activities are people stealing code and secrets from your pipelines. And I think the most popular ones these crypto mining attempts on your pipeline. But the most risky ones are just data leakage of codes and secrets. Because if a bad actor heavier code and heavier admin credentials to deploy to production environment, they potentially put all of your customers if you're a vendor. Or if you're a service provider, it puts off your customers at risk and your customers' data.

I think that that's one of the thing that we've saw with an attack on a vendor named SolarWinds. So SolarWinds is a vendor on the application monitoring space that was attacked by a very sophisticated actor that have injected malicious code that can leak data from SolarWinds customers' base. So different ways of injecting bad code. Codecove have had bad code in their

build image. SolarWinds have had bad code in their application code. And all of us, most of the companies are using either open source tools or third-party providers to build their software. And almost every company is a software company nowadays.

I know that we use in Palo Alto Networks in more than 160 Different SaaS vendors, from CI system, version control system, to even our billing system. It's like everybody are using Salesforce. And everybody are using PagerDuty, DataDog. You have all of those third-party providers that makes you more productive in your day-to-day. And you should use them. But it does put some of your security at risk if those vendors are not having some best practices in place on their code supply chain.

[00:12:28] JM: So have there been any other instances of a SaaS provider getting a crypto jacking system inject? I mean, I can imagine, if something like Salesforce had a crypto jacker in it and it got deployed to every single Salesforce instance that people were running, that would be very pernicious. I don't think that's ever happened. But are there any other large-scale supply chain vulnerabilities that you can use as examples?

[00:13:04] BS: So Codecov was the first in 2021. Then there was SolarWinds. There was also an attack on **[inaudible 00:13:10]**. All those are SaaS tools, service providers. I think that the Codecov is the most interesting ones, because it could have been – Both Codecov SolarWinds could have been easily prevented by the best practices we should all use have – Make sure that you don't persist your Git secrets to any public repository. Like, scan your code, that it doesn't have any secrets before committing it. Make sure that you have code review process. And make sure that your images are signed.

And each one of those should be a relatively small effort from your DevOps team. And I know that every SaaS company have customers on their head, and they have production environments burning. And like it's hard to be a software engineering in the large-scale growth company that is a SaaS provider.

And having said that, to have the minimal best practices in place can be a good place to start with. And it's not that hard. It's not a lot of effort. Now, it's easy for me to judge in retrospect. And I don't fully blame Codecov, because there was not a huge awareness of the risks back then.

And I'm sure that everybody are doing mistakes. Every engineer is breaking production at some point. Every engineer is creating a hole in the security posture. But if you have the right tools, skills, education, team and culture, you can encourage conversation about it. And you can encourage the usage of open source tools and some minimal best practices to make sure that your code deliveries is as secured as you'd like it to be.

And I think that one of the efforts – Of the other efforts that we're seeing in the open source community is a project called Open SSF that was created originally by Google, and a new compliance framework named salsa SLSA, that tries to define what is the bare minimum of best practices you should have on your delivery pipeline. Having approvers, having signed commits, having signed images, not using plain text secrets, having branch protection rules, underlying first commits. It's a best practice that you'll have – Just to have better collaboration as a team. But you also need that to have a secure software delivery pipeline and making sure that nobody force pushes a malicious code.

So having those centralized set of rules really helps every security or every engineering team to be more mature. So as a company, we help our companies make sure that those rules exist. But there are other open source tools outside of art open source named Chekhov that are also helping to drive people into that space. And I think it's not a heavy lift. And you should always start doing that at some point.

[00:16:30] JM: Do you pay any attention to the cryptocurrency world outside of just the crypto jacking vulnerabilities? Have you looked at all into – I mean, this is obviously not exactly the domain of traditional infrastructure protection? But have you thought about crypto security, cryptocurrency security, at all as a domain to get into?

[00:16:55] BS: Maybe on my next venture. But no, I haven't done any research on secure wallets or secured transactions and fraud. It's an interesting space. I'm currently only trying to help people prevent crypto jacking attacks.

[00:17:12] JM: Gotcha. So when you're working on Bridgecrew and you're looking at the different domains of the software supply chain, you've got just running containers, container image security, GitHub security, and then you've got dependency injection issues. What are the

places and the tools you build into Bridgecrew to alleviate those different domains of supply chain insecurity?

[00:17:50] BS: I think that each of those categories that you've mentioned have their own supply chain risks. For example, in infrastructure as code, you can use an open source module that can run arbitrary code. In TerraForm, you have the data external block that you can run arbitrary code on. And on TerraForm, when you import a module, even if you choose a specific version of a module, it's not immutable. People can try to create another commit on the same version, and you will import a new model without knowing that that might have bad code in it.

And on package managers, like NPM, you can always try to import the latest code hoping it will be the most secure. But as the attack on Color.js that have occurred a couple of months ago, you might inject bad code from the latest version of this open source package. The same kind of cure on image, you can pull the latest and the latest can have a vulnerability.

There is also another attack that we haven't talked about, which is the Log4Shell effect. You can use an old version of logger. Like, like everybody's using a logger to log all of the incidents that is happening in the application. And this logger can lead to leakage of all your data to an external source. And what we've seen in a lot of buff logs is that, 24 hours after the attack was discovered on Log4Shell, is the amount of injection attempts to Log4Shell was skyrocketing in the entire Internet.

And the thing that you need to do is to really make sure that each of those mentioned domains is secured. Make sure your TerraForm code does not enable running arbitrary code and has secured configuration for all the cloud resources. Make sure all of your packages, files from different package managers, Maven, NPM, etc., one is using a lock file. And two, any locked package does not have at least a critical vulnerability. Three, make sure your CI pipeline runs in a segment environment with segmented network. Work against an immutable code repository like AWS CodeArtifact, or JFrog Artifactory so arbitrary code would not run software that will leak your code and secrets to an unknown IP. At worst, it will get into an internal IP, like your Artifactory.

And also inspect your runtime environment and be sure that those risks does not arrive to your productions. And you should act as if your CI/CD pipeline is the equivalent of your production. Because usually your CD pipeline will have admin access to deploy software. And multiple engineers will trigger it and will run the service account with admin access to deploy software. So you should scan and interact with each of those domains, each one of those with their own supply chain threat models.

[00:20:59] JM: How do you update your systems over time to be able to detect and mitigate new threats?

[00:21:10] BS: So we are lucky enough to have both a good research team internally and an amazing open source community. And luckily, we have more than 200 contributors who are not Palo Alto network employees. Just people out there from cool companies like Yelp, Airbnb, HashiCorp, AWS, Microsoft contributing content, which is new detection capabilities. Some of the ideas to secure the CI pipeline came from the cloud security forum. It's a cloud security community I highly recommend to participate in. I learned a lot from the practitioners there. And the idea to secure different steps of your workflows came from a discussion of pain points from practitioners in that domain.

So we have open source users who are helping us to increase our coverage and capabilities. We're active in community forums, and we have a research team that is researching always the next threats that we see in customers' environments.

[00:22:22] JM: When you look out at the market, there's a number of products or platforms that I see as similar to BridgeCrew. The most notable ones probably Snyk. Can you give me a sense of how you compare to Snyk or how you see them as a competitor or complimentary?

[00:22:46] BS: I think that each one of us have evolved from a different original space. Snyk have evolved from the application security space. And now it's driving into container security and infrastructure as code security. And Bridgecrew have evolved from infrastructure security both in code and runtime, and now evolving to image scanning, workflow scanning, and also application security. So each company has started in a different domain originally and have a different focus. And in the end of it, I think that all of those companies, Snyk and others

included, are essentially trying to secure the internet and make it a safer place for all of us. So I know some of the people at Snyk, and they're doing a blessed work. And I think that we're trying to do blessed work, too, in succeeding.

[00:23:38] JM: Can you tell me about the engineering of insulating against software supply chain issues? Like, give me a sense for the systems that you built? Programming language choices? Models for code scanning? Like, the architecture for how you solve these different problem domains? I'd like to just get into the engineering of the of how you solve various software supply chain issues.

[00:24:09] BS: So the basic of code security, generally, is to model code as graph objects, as nodal telegraph objects. The infrastructure as code base is a directed acyclic graph. It's a graph that maps the dependencies between resources. For example, easy tools to their security groups, with their firewalls and networking configuration. And you need to do this mapping to understand, "Hey, is this resource public? I have an EC2. Does the firewall expose it? And if the firewall exposes it, is the subnet exposing it too? And then, gateways, are they exposing it too? And what ports are they exposing it?

Now you need to embed into that graph model a question, right? This let's say that this asset is exposed. It's public. I have a public EC2. The configuration of the graph of all the connected resources, of networking and firewalls, have made it public. But is it public and vulnerable?

So to answer is it public and vulnerable, you need to embed another data structure, which is a tree of all of your open source dependencies that mapped into your application bill of material, or software bill of material. You take this software bill of material. You inject it into the graph. And you can answer, "Hey, my server is public and vulnerable." But then you want to ask, "Is it public and vulnerable and have access to customers' data?" So you can query your infrastructure as code directed acyclic graph and ask, "Does it have a network path to the database?"

And you can also scan your CI/CD workflows and check if you have plaintext secrets there, or you have unsecured delivery pipeline. You don't enforce reviewers, and people can inject bad code at any time into that server. So you have DAG for infrastructures code. You have three, four open source vulnerabilities, and also image vulnerabilities. You have a similar tree. And

now you inject to it another graph, which is mapping of the infrastructure deliveries to compute deliveries, applications, Dockers, images, and map them into the workflow that delivers them into production.

So the question that you can now ask is do I have a misconfigured infrastructure that is vulnerable and deployed via unsecured delivery pipeline? And to answer all of those, you need good coverage of those assets. And you also need coverage into the runtime environment. Because let's say that you have that vulnerability. You also want – In order to prioritize, you want to ask, “Where is it deployed to? Is it deployed to a dev environment or a testing environment? Okay, I have time. I don't need to fix it right now. I can deprioritize it for now because my dev environment does not have any sensitive data. Is it deployed to a production environment? Yes, it is. Alright. I should prioritize that because I put my company at risk.”

So the data model is a multi-layer graph, essentially. We built our own data structure to enable different queries of those layers. And behind the scene, it's a data access layer that combines data from RDS and a persistent graph object that will persist into S3 buckets. And we shard it between the different tenants of Bridgecrew. I think that's where a lot of the intellectual properties. And the other thing that we do is we build a graph of all of your runtime environment using cloud security posture management capabilities, where we scan your runtime environment, we build the graph, and we map that graph into your code using tracing capabilities.

[00:28:16] JM: So you said graph for container security? Is that right?

[00:28:21] BS: Graph is for infrastructure –

[00:28:23] JM: Graphs for infrastructure as code. Trees for container security.

[00:28:27] BS: Trees for a container. You need a tree, because let's say that you're using Spring, one of the most popular open source projects to bootstrap a Java application. Springs have another library that it depends on, which is Spring Log. And Spring Log is dependent in Log4J. When you want to ask, “Am I vulnerable to a Log4J CV?” called Log4Shell, you need to query the entire tree of dependencies of, “Hey, I’m dependent on Spring,” which is dependent

on Spring Log, which is dependent on Log4J, which is vulnerable. So you need a tree for that. And these trees composed also into your Docker image whenever you run Docker Build, and you bring in all of your open source packages into it.

[00:29:16] JM: Gotcha. And how big do those dependency data structures get? Do they become prohibitively large?

[00:29:25] BS: The largest that I've seen is a few millions of objects. It's because you have a production environment. And this production environment have millions of servers going to up and down. And all of those servers have application deployed on them. And all those applications have open source packages deployed on them. So it gets to be a few millions. The largest I've seen is almost 100 million entities. And we try – At the beginning, it was not an easy scale challenge to solve. But we managed to solve it using, so far, capabilities that are implemented in a serverless framework. And we also had a lot of work on sharding our data objects to enable fast queries on top of them.

[00:30:16] JM: So when you say sharding your data objects, those are the data objects that you're building for your users? Or those are the data structures that you're maintaining to understand the software supply chain on your side?

[00:30:36] BS: Both. So let's say that our user have decided to connect the repository and the user wants to – What are the best practices that they're using? And what are the bad practices they're using in code across all of the supply chain? So when you connect the repository, we build this graph. And the first thing that we did instead of having a full update of that graph on every commit that you're doing are only updating the deltas of your code change.

The second thing that we've done is we pre-calculated common queries. Like we pre-mapped what is a public object? What is an encrypted or unencrypted object? And what is an object that is traced from code through runtime? We know how to identify the runtime objects connected to it. And then we solve some of the scale issues by creating micro graphs. Some of the entities had a lot of edges. For example, a database can have a lot of entities communicating with it. A lambda function, or a container might have a lot of instances. So we managed to create micro graphs that can be queried when you investigate small portions of the graph for security risks.

And some of the user interface was built in a way that will enable you to paginate over the graph data structure in a way that is easy to search and explore the different vulnerabilities. So there has been a lot of engineering and product effort to analyze of the different investigation flows and that are common and all of the different data structures that should be used in that investigation workflow. It's also worth to mention that the data that we have in order to create those models is enormous. We are connected to a large number of repositories, which gave our data engineering team enough prior knowledge to play with some A-B testing. Can see the results of how the data structure would look like. And would we support that scale before exposing such capabilities to our customers?

[00:32:58] JM: Can you tell me more about the backing databases, or just in=memory structures you used for representing those graphs and trees for the dependencies?

[00:33:12] BS: So one of the most popular libraries for graph representation in Python is NetworkX is –Basically when you're exploring graph databases, you have three options. One is to try the AWS managed solution, AWS Neptune. And it was not a good fit for us. Maybe it is today. But at the beginning, it wasn't. It didn't support all the queries that we wanted to execute, and was not fast enough and did not hold the scale that we wanted in a reasonable pricing. It has changed.

So AWS team have done some good work, but it was not enough for us. The second option that you have is to use Spark with graphics. The third option is to use Neo4J, which has some licensing limitations or commercial aspects that you need to agree to. And the fourth option is to use a library like NetworkX to create those capabilities.

Now, Bridgecrew is an open source, open core company. Some of our graph logic is based on the open source project that we maintain, Checkov. And we wanted the capability to persist some of that graph logic into the open source community base. So we chose NetworkX. And on our commercial offering, we created a Lambda-based architecture that enabled scaling NetworkX to analyze with its graph of those different layers of supply chain. It's a purpose-built in-memory DB, if you'd like, for the purpose of supply chain security.

[00:34:58] JM: Got it? And I guess I'd like to get a little bit more context for how you're actually using these data structures. So if I understand correctly, you have your own data structures. And then you have the users' data structures, and you're doing some kind of diffing between them to understand the dependencies?

[00:35:18] BS: So when you mentioned user data structures, the thing that I mean by user data structure is we scan the user code, and we build a common data structure on top. So if you write TerraForm code, the data structure of TerraForm is directed acyclic graph. CloudFormation have a similar data structure, directed acyclic graph. Serverless have the same. And Kubernetes also have the same. You have dependencies between pods and services, and service meshes. And in TerraForm, you have dependencies between your EC2s, to your security groups, and to your EBS volumes. So you can ask a query, "Is my EC2 encrypted and not public?"

And then there is also a connection between your ECS Fargate, or Kube EKS, Kubernetes cluster, to your container images. All those are doing references in code in order to import an image. You're typing in your ECS task image=bytealpinepython3, or Alpine latest, or Ubuntu latest. And you might use also a custom image. Let's say, my app image and then a virtual number of it. So the connection between resources is mentioned in code. You have to describe it on the declarative language where you're writing your DevOps code. You can do hacky things like bash scripts. But over time, it will not hold, and then he will move into a declarative, or immutable, or imperative model. But you always define these dependencies somehow.

And if you haven't defined it in code, it's happening in runtime. Like if you'll go into your AWS console, you see what image is being used on the ECS task. So the graph model is being built once you connect either a code or a cloud environment. So the user data structure is – Or the user flow would be connected cloud and code repository. And we will build the data structure and the data model on top. And we will enable you to scan that data model with more than 1000 built-in best practices. And you can run your own query, your own graph query, to scan the different dependencies between resources.

[00:37:44] JM: Okay. Very cool. I'm glad we could cover some architecture there. So I guess to close off, since we've been talking about software supply chain most the time, do you have any

predictions about where the software supply chain vulnerabilities will come from in the near future?

[00:38:04] BS: I hope that not only the attacks that we hear about in the news will be the driver, but also, companies will want to increase their level of assurance using compliance frameworks like Salsa, And another driver can actually be the regulations around it. So one month ago, **[inaudible 00:38:25]** called Facebook, Google, Apple to talk about the open source security risk. And we see the US government being more active in conversations on open source security. So I do hope that it will come both from the regulations and from practitioners in a bottom up motion to make their pipeline more mature, repeatable, reproducible, and to be something that we can trust, because open source have done mostly good to the world, from Linux to the different packages of NPM. And we all wanted to keep doing that. And we're all enjoying that. And I think that it's more not a prediction. It's a hope that each maintainer on their own domain will work to make their popular open source package or tool more secure. And by that, we'll keep the Internet safe.

[00:39:24] JM: Cool. Well, Barak, it's been a real pleasure talking to you. And thanks for coming back on the show.

[00:39:30] BS: Thank you for having me. Always a pleasure, Jeffrey.

[END]