EPISODE 1067

[INTRODUCTION]

[00:00:00] JM: Infrastructure as code tools are used to define the architecture of software systems. Common infrastructure as code tools include Terraform and AWS CloudFormation. When infrastructure is defined as code, we can use static analysis tools to analyze the code for configuration mistakes just as we could analyze a programming language with traditional static analysis tools. When a developer writes a program, that developer might use static analysis to parse a program for common mistakes; memory leaks, potential null pointers, security holes. The concept of static analysis can be extended to infrastructure as code as well allowing for the discovery of higher level problems such as insecure policies across cloud resources.

Guy Eisenkot is an engineer with Bridgecrew a company that makes static analysis tools for security and compliance. Guy joins the show to talk about cloud security and how static analysis can be used to improve the quality of infrastructure deployments.

[SPONSOR MESSAGE]

[00:01:08] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW]

[00:02:59] JM: Guy Eisenkot, welcome to the show.

[00:03:01] GE: Hi, Jeff. How are you?

[00:03:02] JM: Large enterprises have a dedicated security team. They also have dedicated software engineering teams. This has been the case for a long time. Overtime, there has been a shift in responsibility from the security teams doing security work. Do the software teams doing security work? Why have software teams increasingly had a responsibility to do security work?

[00:03:28] GE: Yeah, it's probably the first or probably the main understanding that me and my cofounders came to about two or three years ago when we started thinking about starting another startup. If I have to pinpoint one thing specifically, I think power to developers. I think we've seen it in our own engineering team how more and more product management decision and architecture decisions are just flowing downstream, and you have this amazing and very independent generation of developers who not only wants to build and design, but really wants to shape how things are going to get done forward, and I think that's been for our team and the teams that we've worked with, startups and enterprise, I think that was probably the main movement of shift we've seen in the past 12, 18 months.

[00:04:19] JM: Infrastructure as a code is a way of declaratively describing how your software architecture fits together. I feel like there is a connection between infrastructure as code and modern software engineering security practices. What's the connection between infrastructure as code and security?

[00:04:37] GE: They would, yeah. If we take a step back and maybe focus or zoon-in on cloud. In cloud, we define infrastructure as anything that can get changed or provision or control through a cloud provider API, and when we think about it, it's kind of crazy. Think of even the concept or a notion of a firewall, which is like a classic pillar or infrastructure, and like for the past 25 years, people have been selling it in boxes. You get it. You put it in your data center. You usually wire it. The person who knows the firewall best is the guy that dragged it into the data center.

When you think of a firewall in the cloud, like in Amazon, the security group. It's like six or seven lines of code. It's pretty ridiculous. You took this enormous idea of configuration, which was something that was very centralized and managed on almost a physical basis and you've turned into just another component within software. I think when I look at infrastructure as code and what it's doing to security in software development, I think about all these previous notions we've had of how or what infrastructure is and make these correlations into how they look like in the cloud.

Take databases as another example. In like our previous startup, we had like three different database systems and we thought we're like super advanced and crazy for using both of file system, one for machine learning and one for short-time retention and thinking we're reinventing the wheel.

Now with Amazon or Google Cloud, you have dozens of database services, all of them can be toggled on and off, provisioned, refactored, migrated with tens, even less than a hundred lines of code. We think when we think of security, we look at these pillars of infrastructure that are getting migrated into code and just really finding this role of what cloud infrastructure and what cloud infrastructure engineering has become.

[00:06:31] JM: Popular ways of defining infrastructure as code include Terraform and CloudFormation. Describe how these different tools for infrastructure as code declaratively managed? How do these tools work?

[00:06:46] GE: Yeah, sure. I think the best way to think of declarative is to think about an infrastructure or an architecture that you know what it's end state is going to be. The nice thing about declarative is you basically write a very simple language that defines objects that eventually get deployed just the way that you wrote them to be. Another thing that you can do with declarative, such as Terraform or CloudFormation, is to write it in steps and make sure that the next step or the next few steps are layering that initial plan that you have. This is extremely useful for writing in networking, for example. If you want to configure a new network, a subnet or a VPC in cloud infrastructure, it's really, really simple. Instead of remembering all the dotted lines and everything that gets connected together, to think of the end state, what's the application going to look like? What's its ingress or egress rules going to be? Once you start with that, you can basically build out any logic that makes it either more granular or scales it out and makes it more robust as you have that end state in mind.

When you have imperative, I think the most famous term to really have that picture this is basically thinking of recipes. Having a clear set of actions you'd like to perform and then basically give the system the ability to decide what are the best ways to orchestrate those steps going forward. This is super useful for people that are fleets of virtual machines, for example. Using imperative language really enables you to build them in a stateful manner that enables you to think of every step of the every way and everything that needs to get deployed every certain time.

[00:08:23] JM: How do people choose between Terraform and CloudFormation?

[00:08:29] GE: It's a good question. I think there are two considerations out there, and if you ask me, I think we're going to see a huge trend in the next coming months and seeing HashiCorp, which owns and maintains Terraform obviously is getting bigger and more noticeable in the landscape and there's more and more companies that want to take a piece of that and be part of that party.

Obviously, CloudFormation is Amazon's proprietary language for writing infrastructure as code. It actually has its own competitors internally for writing infrastructure. So you can use CloudFormation, but you can also use CDK, which is another way to write a more imperative style of infrastructure. But when you look at CloudFormation versus Terraform, I think the main difference is that you will eventually write CloudFormation only for your Amazon Web Services resources and you'll probably write Terraform for everything inside Amazon. They have great support of Amazon, but they also have this great ability to basically take any type of infrastructure, whether it's cloud infrastructure, but not only. So you can think of databases as a service, even GitHub. Your code repo can eventually get provisioned, changed, managed using that very robust, very flexible HashiCorp language that enables you to deploy resources at scale.

For our customers, Terraform has been a natural choice because it does give you the ability to use that same language and to continue using it even for your next cloud provider and for your next provisioning tools. But on the other hand, Amazon is super competitive and we're already seeing it ramp-up a lot of the features that Terraform was able to bring up, thanks to a huge community that's behind it. I expect to see some more competition between the two, but my personal opinion, more companies going to diverse their cloud portfolios, Terraform is eventually going to reign on top.

[00:10:20] JM: Have you heard of a company called Pulumi?

[00:10:24] GE: I have. Yeah.

[00:10:25] JM: Yeah, we just did a show with Pulumi a while ago, which is an interesting blend of classic programming languages and declarative infrastructure. I guess the goal is to give you the best of both worlds when it comes to the Puppet and Chef imperative world versus the Terraform declarative world. Do you see that as valuable or important?

[00:10:51] GE: I think so. If I have to categorize it in my head, and I'll be frank, I haven't used Pulumi. I have seen some of their – They have some very good documentation out there about what they're doing and always happy to see another loud voice for pro infrastructure as code. But think of all of these technologies eventually as additional abstractions that are supposed to help you simplify the very complex world of cloud native.

I think one customer of ours had told me, "Think of the hundreds of services and APIs I have to track on my existing services that I use on Amazon. Multiply that by my next cloud provider, whether it's Google Cloud or Azure," and basically Terraform has been a safe zone where they know we have to learn one language. Eventually it's a pretty similar abstraction on top of all three clouds, but I can definitely see why the complexity of multiple provisioning languages is probably a challenge that I can see it on the next seal. I think our current partners and customers were still climbing that uphill against working or getting a good or solid recognition of all the complexity they have with their existing cloud provider.

Bottom line, I see even in the near term, more and more technologies. I think I've even heard of another open source project just a few days back. Great ideas and great ways to create additional abstraction layers over the complexities of the cloud, and yeah, rooting for them. I think as we get more and more configuration as code, I think eventually developers are the ones that are benefitting, because we're giving them more power and the more centralized and independent teams are and how they build and how they think of software and the more they use infrastructure as code and the more we simplify infrastructure as code, the more robust, scalable and secure systems we'll see getting built.

[00:12:41] JM: What kinds of security mistakes get made when defining my infrastructure with these infrastructure as code tools?

[00:12:50] GE: Yeah. It's actually a great question. Actually one that I'm dealing with right now. Actually, the way first wave of research around the mistakes and errors around infrastructure as code are now getting published. There was a great research that the research arm from Palo Alto Networks released like a few months back which had like a great overview of the different types of configuration errors you can find in code. I think what they did is scanned public repositories and tried to identify within template files that can be found in these registries. What types of errors are building within them? Which is kind of a tricky analysis, because as we mentioned, once you have a declarative language, eventually you can wrap a configuration error with a protective configuration, additional protective layer that can basically make that previous configuration irrelevant, the same as vulnerabilities and exploits and what you can do to mitigate them.

Eventually I think it goes to four main areas. I think one main area that we see for configuration errors is around usage of default configurations. Cloud providers are very good at giving you this very broad canvas where you can build and apply default configurations to get things up and running very fast. Take machine learning as an example. In my pervious company, it took us two years to build a machine learning engine, and now with native tools like SageMaker and others, you can build the most robust Airlfow-based machine learning pipeline in minutes.

What we see is when users are using these default configurations and bringing it into their production environments, they forget that sometimes that these default configurations purpose was to get you up and running very fast and not necessarily to a point where you can basically build the most secure system.

I think one main area that we see is that people are just not very aware that default configurations are not very good. I think cloud providers understand that now more than ever and they're actually pushing some of those previously done mistakes, like leaving buckets public, or leaving encryption unused as things are getting, one, cheaper; and two, becoming default configurations. That's one bucket.

I think the other one is around making sure that everything gets logs and audited correctly. For me, or for us as security practitioners, this actually goes without saying, because we're used to building the software, the tracks, things that happen like OS telemetry, and networking telemetry, because we configure the firewalls in the old generation. But today when users are configuring their own infrastructure, if they don't or they're not bounded or restricted to use infrastructure that reports consistent telemetry, then I think they're just black holes. So suddenly you have ephemeral resources or different pockets within your public cloud that are just not logged and audited, regions that you have activity and you get build for them, but you're not really sure who's using them and why. That's like another mass, mass area where we're seeing misconfiguration, just not using some of those tools that are unfortunately not the most or the least pricey, but they're just – That bottom line, they're just that baseline that you have to have in order to know what's going on outside your spend on the cloud. Number two, just ensuring that everything gets logs and audited.

Number three is around compliance requirements. There's like two main ones that usually reoccur, which is the use of encryption and the proper management of users and credentials, and this is where it gets simple and complicated. It's simple in the way that it's easy to turn on encryption today probably now more than ever. On the other hand, more and more frameworks, like SOC2, NEXT, PCI are now requiring that you encrypt almost everything especially with data privacy becoming an area where more and more auditors are getting focused on.

It's easier on one hand. On the other hand, there are lots of services to encrypt. That's another bucket that we see a lot of misconfigurations in. The last one, it ties back to compliance, but it also ties back to everything, but it's the use of AWS IAM and any proprietary IAM, cloud native IAM for that matter. We have actually just introduced an open source tool that tries to help users clean up the mess for that, because we've seen so many environments that spun up in the last three, four, five years and they have these pockets of different generations of access controlling them, like two or three SAML providers, different users of like rows, permissions, inheriting rows and permissions. It's a crazy big roll, but lots of lots of misconfigurations in that area. Eventually, most of our users decide to attack it, because their compliance auditor found out that they're just not managing their IAM as they're supposed to.

[00:17:34] JM: Yeah. I mean, it sounds like it can be a nightmare. Even if you just talk about generations of developers coming through a company rather than generations of tools. You have these IAMs and IAMs groups that gets set up and it can be hard to keep track of which group has access to what and who is a part of what group. Tell me more about some of the mistakes that can be made – IAM by the way is identity and access management for those who don't know the acronym. But tell me more about some of the mistakes that can be made in policy management.

[00:18:14] GE: Sure, one of our favorite topics. Just maybe before that [inaudible 00:18:18]. Think of using IAM, and a lot of configuration for that matter, but IAM is like one of those that really sticks or nail it home, but think of it as like almost archeology. We've seen environments that are like 6, 7, 8-years-old, which are probably these are old environments in AWS terms, and you can really see the layers and layers of fossils that have come about, because users initially, maybe they use the internal IAM and then switched back to an external identity provider and then switched back. I think when you look at it, you have to understand that, like fashion, cloud developments has trends and has new services that are coming in all along and it seems like identity and access is the one, has been like a poster child for being one that's both super simple to set up in the sense that you can write or configure a new user fairly easily specially if you compare it to your active directory days where you had just one IT admin that was able to provision new roles in the system. Now, even the AWS managed policies give you quite vast permissions to create new roles and to edit permission documents and to – For almost every resource that you spin up.

On one hand, managing IAM in Amazon, for example, but it's the same for the other cloud providers. It's extremely easy in the sense that it's accessible. Developers have it in their hands today, but on the other hand, it's really hard to define what's the specific best practice, because it really goes down to what's your end result and what are you going to try to aspire to get.

We've seen a banking, an online banking company that had to go through their PCI audit that just decided to clear out everything that's not their existing SAML provider. They wanted to remove probably 90% of IAM configurations that were available just to be able to report to an auditor that the system is working with just a single provider not because it gives them the best speed or development base just because it's the only way to pass a PCI audit, to be able to define a very restricted set of users that have access to a single place.

I think where we've taken IAM as we've seen it, and you may understand, we've seen like probably over 100 AWS accounts so far with like every type of permutation of configurations you can think of. Fairly early in the process, we saw that one thing that you have to do with IAM is to codify it, and by codifying, it means unfortunately taking it out of AWS, out of that manual configuration console, which is actually very user-friendly, but very distractive in the sense that it really encourages people to build and write their own individual logics to how things should communicate and have access to each other and to ensure that everything gets managed as code.

We are, as you may understand, are firm believers in moving configurations that are manual into configurations that are automated. As I mentioned, we've just built out this tool which we've actually been using for the past month, a couple of months, to migrate the existing AWS AIM permissions into much a simpler, a model that basically, one, cleans all the unused users, groups, policy documents, permissions that are just not in use and we're using a great Amazon API called Access Advisor that gives us all the telemetry for free.

Once we get that all cleaned up, we basically migrate all those manual configuration into one neat Terraform file. The reason we think it's so cool is suddenly user management and access management isn't something that's a standalone configuration that you have to go and do every time you build a new project, but it's actually part of your code. Just like you build your networking configuration, as I mentioned, declarative infrastructure as code is great for that. You're just going to have to start working with your cloud provider and use their IAM provider and module pack to define those as code. If you're unfortunately using manual configurations, multiple providers, you can basically use that tool that I mentioned. That's called AirIAM. It's free open source Apache 2, and migrate all your manual configurations.

I think it's improved our way of tracking identity and access internally and also for our customers, and looking forward, I think more and more configurations are going to go through that transition, and I think Amazon is going to get it too and make much more of that IAM API set up get much more accessible through infrastructure as code, whether it's CloudFormation, CDK, landing zones, control tower, all those products that help you, Amazon products that help you set up and define policies, permissions beforehand. SAP is another one.

[SPONSOR MESSAGE]

[00:23:09] JM: If you listen to this show, you are probably a software engineer or a data scientist. If you want to develop skills to build machine learning models, check out Springboard. Springboard is an online education program that gives you hands-on experience with creating and deploying machine learning models into production, and every student who goes through Springboard is paired with a mentor, a machine learning expert who gives that student one-on-one mentorship support over video.

The Springboard program offers a job guarantee in its career tracks, meaning that you do not have to pay until you secure a job in machine learning. If you're curious about transitioning into machine learning, go to softwareengineeringdaily.com/springboard. Listeners can get $500 in scholarship if they use the code AI Springboard. This scholarship is for 20 students who enroll by going to softwareengineeringdaily.com/springboard and enter the code AI springboard. It takes about 10 minutes to apply. It's free and it's awarded on a first-come first-served basis. If you're interested in transitioning into machine learning, go to softwareengineeringdaily.com/springboard.

Anyone who is interested and likes the idea of building and deploying machine learning models, deep learning models, you might like Springboard. Go to softwareengineeringdaily.com/springboard, and thank you to Springboard for being a sponsor.

[INTERVIEW CONTINUED]

[00:24:45] JM: So you're saying that much of the way that policy is managed today is through a CLI tool or just going into the Amazon AWS dashboard, whereas in the future, maybe there should be higher level tools that allow you to reckon with these in more of an automated fashion.

[00:25:11] GE: Correct. Just to generalize, I think this goes for probably most of the first, second gen services that Amazon came out with. So think of S3 Buckets, EC2s, IAM, probably top or top three. I might be missing a few others. I think what happened with those more mature tools is that people have accumulated this long tail of configurations that now in 2019, 2020, when people are starting to look back and you've got some great open source and commercial tooling out there that helps you clean up the mess, you're suddenly seeing all that long tail of configurations you have dragged on.

I think open source community has been doing a great job in surfacing a lot of that bad practices around IAM hygiene and other areas. If I generalize it, I think what's happening even within the Amazon tool base is that they’re obviously understanding that people want to clean up a lot of that messed historically because they've seen that once you don't clean up that historically because they’ve seen that. Once you don’t clean up that historical mess and you keep those fossils buried inside your architectural layers, over time they can get exploited. This might've not happened two or three years, because – I don’t know, maybe it was like the grace of AWS at that point, but we’re seeing more and more of these resources getting attacked and targeted by both the research community that's looking for vulnerabilities, and that's great, but also for a much darker set of actors that just understands that these are APIs accessible to the Internet that you can exploit.

[00:26:45] JM: Now you've described policy management issues, and I eventually want to get to some of the tools that you're building, but I want to talk more about problems that occur in the creation of infrastructure as code configurations. Networking is another issue that can lead to problems in security if you configure a network incorrectly. What kinds of mistakes get made in network configuration in regards to infrastructure as code?

[00:27:18] GE: It's probably not as complex as it is for your IAM actually. Networking, as we see it, is eventually – You can almost think of it as two religions. You have people that come out of the identity space or have identity strong identity backgrounds and they decide to segregate and separate their environment and to decide how they're going to do multi-tenancy and other types of internal – Building those logical walls inside. You have this other religion of people that manage policies through networking.

In the cloud, and maybe I'm being too simplistic in this, but in the cloud, it really doesn't matter that much. Eventually in access group or a security group, which both get defined with a six or seven lines of code, they get exploited and manipulated and their managed policies look different, but eventually the work that goes into managing and provisioning them is pretty much the same in the amount of code you have to write.

Specifically, around networking, I think we've seen probably three main areas of problem, and this kind of talks to my previous points. One is, and this has been talked about quite a lot, is the use, again, of different configurations, epecifically Amazon's default VPC, which for some reason still enables you to create a new VPC that's wide open to the Internet. When you create a new VPC on Amazon, you eventually get yourself something that's supposed to be a virtually network, but eventually if it's widely open, for example, to the Internet or for like port 22 for SSH connections, it could be exploited immediately once it goes online.

We see users that understand that and actually Amazon now enables you to block that type of configuration and not use that default VPC that creates a default security group that are both inherently containing that configuration and mistaken policy. That's one. Second is around databases predominantly. Again, zooming on Amazon, you have your managed elastic search, your manage elastic ElastiCache database. You have S3 obviously, archiving databases, tons of ways to store data. You can use SQL relational, non-relational, Mongol. Lots and lots of these databases types. Where it gets complicated is that each and every one of these databases obviously rides on a unique protocol port and IP number and IP reg once defined.

What we've seen in our customer base, and this references mostly to companies that are in their fifth, sixth, seventh year to building on Amazon, pretty mature. We just see that these databases over time, various people touch them and use them and provision them. They just get left wide open. Think of a database that eventually serves a web application, but for some reason is accessible. Someone basically that scanning that range of addresses and looks for an Elasticsearch, for example, that has access to the Internet. Obviously, that’s misconfiguration or potential vulnerability and someone can exploit that.

I think one thing that we instruct most of our users at the first couple of days is basically to focus on all the communication back and forth from databases and to see what ports are open, how are security groups managed, because sometimes for various reasons eventually data needs to go somewhere. If it's to show something on a web application or to enrich a process that’s doing some complex recommendation engine, eventually data in a database doesn't serve anyone. That entire dynamic of more and more people touching data and moving data from place to place has historically gotten to a place where databases have become a place where we’ve seen more and more configuration errors and potential vulnerabilities.

I think if you look up Amazon Web Services and leaks for customers that are using Amazon Web Services, you will see that most of the data that was leaked was eventually coming from misconfigured databases, whether it's axis control over S3 or networking configuration on Elasticsearch that just enabled people to access them [inaudible 00:31:25] to do man in the middle, things that are slightly more complex. Note your default VPCs. Not your databases.

A third cluster is ephemeral resources. Think of everything that you need for a short period of time. I think the problem with that, it’s like it's a great way to save money, right? One thing that ephemeral resources came to the world, because you can basically spin up resources in the cloud for very short periods of time. Let them do what they need to do and shut them down.

Spot instances work this way and compute and short-term compute workers work this way and databases work this way. It's pretty awesome for development. We’ve seen a lot of configuration errors around that area, because what happens is that you can spin up ephemeral resources using AWS APIs, SDKs, even external services, like if you’re using Databricks, or a managed data pipeline. You can get them to spin up databases for you.

We've seen users that are defining that initial set of permissions and networking that's required in order to get these external as a service working for them, but forget that eventually they have a networking wrapper that gets open and remains unused. Think of a project or a data science project that you have that’s running databases on one hand, compute workers on the other, and those workers get launched up and down all the time. In the interim, which can sometimes be hours or days, you have these set of network configurations that’s actually unused. What happens is that sometimes these networking configurations get generated dynamically. So you get more and more of these networking configurations that are stale a lot of the time and are not being used, and overtime they become fossils and no one manages them. Obviously, if you don't properly manage those networking configurations, they become exploitable. If you don't clean them out, you won’t have – You’ll basically risk the possibility that someone will try to hack them.

[00:33:24] JM: Do these mistakes get made because people don't know what they should be doing or because they just like make a mistake, like a typo? Is a lack of knowledge or is it just like too much stuff in these configuration files such that they do something wrong, even though they know if they took a second look at that piece of code they would realize they're doing something wrong?

[00:33:53] GE: Yeah, great question. We haven't done the psychological research to really see where some of these are configuration come from. I think my experiences showed me that it's definitely not negligence or lack of willingness to write good infrastructure as it is just super-duper complex and manual. Obviously, not probably the most shiny part of the job. I don't I know a lot of developers. I don't do a lot of developers that like to write the unit tests that check out if their configuration is sounding the sense that it was configured based on the latest set of policies that someone applied in their infrastructure is code system, if you know what I mean.

I think what I've seen is that mostly the complexity, the amount of arguments, that amount of different elements you can insert and inject into resources has been probably the main contributor to misconfiguration errors being so common, and you have to connect that to where people are getting a lot of this configuration as code. There's a lot of different sources out there in the same way as like six or seven years ago, everybody just started scraping every popular open source and building their stack based on that great set of open source tools that helped us build applications like 2015, 2016, and that brought in tons of tons of vulnerabilities into our web applications. You have the exact same thing happening now with configuration. People are going to these public repositories which are eventually managed by good people from the community, from the cloud providers themselves. People are obviously contributing templates and modules of configurations, putting them out there so they help someone achieve a goal.

What they're not thinking about correctly, and I think that's probably the biggest disadvantage of building in the speed of infrastructure as code, is that as you write that declarative language, you're not constantly thinking, "Hey, is this going to – When I finish, is this going to be sound terms of the configurations and the different arguments that I've inserted?" Because, as I mentioned, you can write out firewall logic in six lines of code, like AWS security group, but if you don't really look at what's in those three lines of code and make sure that it doesn't only make your application work with two services that are indifferent VPCs or subnets, but it's also not exposing that compute ACID that you've just built to potential hazardous access, then whose fault is it? Who's accountable for it?

When we started to talk about that shift of responsibilities that started with security teams that were centralized and had the role and responsibility to make sure that people don't get phished online or that websites don't get harassed or tarnished, and now in the cloud, we're expecting them to be able to get inside each and every developer's mind and make sure that each and every configuration that they make is security-aware. I don't think it makes sense. It sounds to me like there should be another boundary between the developer, which we're asking from to write the best code they can to get the business goal as fast as possible. On the other hand, to have an internal force, hopefully it's like an automation, it doesn't have to be a person, that continues and makes sure that all this configuration that gets pushed into production is continuously getting checks for potential misconfiguration and errors, because eventually as more and more of these activities get crowd sourced, and when I mean crowd sourced, I mean they go back to developers when they inject one of these configure errors into code, the less experts we'll see and the less work we'll give security and DevOps folks, right?

[00:37:35] JM: All right. Well, this brings us to Chekhov, which is a project that your company, Bridgecrew created. Chekhov finds security and compliance misconfigurations. I can run this at CICD time, and as my code is getting ready to be deployed, I can check it for misconfigurations. What are the common security and compliance misconfigurations that I could find with Chekhov?

[00:38:07] GE: Great question. Chekhov is fun. I'll just give some background, but we started off like four months ago almost. After obviously – I'll go a little bit sidetrack, but I'll get back to your question in like 60 seconds. I promise. We saw that when you try to fix a lot of the configuration errors in runtime, I mean, specifically in AWS you're like using a Lambda function that would correct a configuration, what happens is that the nightly bid that contains that declarative Terraform plan is going to override that configuration fix that you made.

We understood that, specifically, the analysis of configuration and the correction of configuration has to happen much further upstream. My cofounder, Barak Schoster, which is you have to meet him some time. He's like a wonder kid. He's pretty amazing. He locked himself up this one weekend about 4, 5 months ago, and after two days where we haven't heard from him, he came out with Chekhov, which eventually it's a very simple Python tool that does study code analysis and there's like hundred others out there. But what we did with it is we just gave it the exact set of scanning capabilities that we saw Amazon and Google and Azure eventually requesting us as users to track obviously through a framework, like the CIS framework, which gives like the most foundational set of policies and checks people should monitor, and most cloud providers now give you that almost for free. You can toggle it on and use it for free, but no one was looking at the infrastructure as code.

He came up with it and we blitzed like two or three of our in-house developers writing up the content and making sure that it goes out there and really covers as many configuration types and configuration errors as we we've encountered and as close as possible to those frameworks. If someone really wants to adhere to that framework and that's how they're getting their NIST 800 or SOC 2 PCI DSS, have another layer of protection.

Eventually, Chekov looks at those same families that I mentioned before. Chekov looks obviously at the networking layer. So it looks at the security group and VPC configurations to track and see that you're using best practices and not exposing networking publicly to the Internet when you're not supposed to or you don't want to do it intentionally. It looks at databases, as I mentioned. It looks of databases for encryption on most of the popular databases. It looks at a lot of binary configurations. For example, it looks for all of the logging configurations that I also mentioned and makes sure that you have those toggled on.

For the past 2-1/2 month, we've put more and more emphasis on the compute. Added support for most of the EC2 management, obviously, but also for the managed Kubernetes capabilities now and just making sure more and more of those services are covered. Eventually, what we want to accomplish there is to have as much parity as we have with the policy, policy as a code obviously, which I don't think I've mentioned up until now, that we have defined in runtime.

Your cloud provider maybe helping you track or a third-party tool or an open-source tool is helping you track all the configurations within your cloud provider APIs and settings, but Chekov has been a great asset for us and for the community actually by being able to track those same sets of policies as the infrastructure is getting build in Terraform, CloudFormation, and very soon, a few other exciting languages, which we all know and love.

[SPONSOR MESSAGE]

[00:41:44] JM: Sponsoring today's podcast is Datadog, a cloud scale monitoring and analytics platform. Datadog integrates with more than 400 technologies, including Cloud Foundry, Docker, Kubernetes and Kafka, so that you can get deep visibility into every layer of your applications and infrastructure, in the cloud, on-premises, on containers or wherever those applications run.

With rich dashboards, machine learning powered alerts and distributed request tracing, Datadog helps teams resolve issues quickly and release new features faster. Start monitoring your dynamic cloud infrastructure today with a 14-day trial. Listeners of this podcast will also get a free T-shirt for trying Datadog. You can go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to get a free T-shirt for trying Datadog.

[INTERVIEW CONTINUED]

[00:42:46] JM: Chekhov has all these different policies that relate to different pieces of infrastructure, DynamoDB, and Fargate, and whatever else. Do these policies get crowd sourced? Are you writing most of them yourself? How do you get the necessary coverage of all the different infrastructure pieces that you could need some pre-baked set of security and compliance analysis over?

[00:43:26] GE: It’s probably the most encouraging topic, Jeff. Eventually, we’re a group of – Well, today, like 11 engineers strong that have like 200,000 other things that they want to do. We started off with the core CIS framework, which covers roughly 30 checks. It has like 50 something, but covers 30 checks that you can actually sample statically. Some of them require you to have a periodic analysis, a difference between time-to-time, like ensuring that key rotation happens every 30 days. That's something you're not going to be able to do with study code and analysis in Terraform. But everything that can get covered by the CIS framework, that's been our core focus initially.

As we saw, I think this was around January, as we saw more and more traction, and downloads, and stars, just issues that are getting open directly on GitHub, we saw there’s huge appetite for the community to chip-in and to help out with things that they're building that are not necessarily the things that we and our customers are building.

We have a great group out of a consulting company, very famous out of UK, which is building a lot of managed Kubernetes and Google Cloud, and they’ve decided to contribute some of their work back. We have an individual contributor from Europe whose like super keen on getting S3 configurations right. He took – In the framework, you have like three or four major policies that you want to look into. I think he had like six or seven additional ones. It’s been growing nicely, and I think it was in our initial objective initially to get the community to help us do this, but I think people saw the opportunity for a very simple solution that's written in Python, which I think is super important. Not enough infrastructure as code is done in Python these days.

People see Python, they see it's familiarity. They see how simple it is to write a policy, and people have been contributing. I think almost 40% of the content is now contributed. It's been growing nicely, and we're going to chip our part and continue to add more and more provisioning languages just to make sure that people can use that for their additional provisioning workspaces and to make sure that those get covered by a unified set of policies.

[00:45:46] JM: Do you have some particular areas of policy the you focused on and some particular pieces of infrastructure that you focused on? I mean, there are so many potential configurations that people could present to you. I ship my code to CICD and Chekhov runs over it. I got to imagine that there are some long-tail services and configuration mistakes that I could be making that would not be covered by Chekhov. How are you prioritizing what policies to get written?

[00:46:23] GE: Yeah. Eventually, it's a tradeoff between two main factors. One main factor is how common does a configuration get abused or get an error for? If you have some building blocks or some very popular modules out there that are actually cook containing most of the things that you need to get a proper resource in place. I think the – I will give just two example, but I think the module, the supported AWS modules for S3 and EC2 are great in the sense that they're springing up a very secure set of resources and all that ecosystem around both of these resources.

I'll give you the opposite example where we're just seeing a lot of configuration error, which is RDS for some reason. This is a combination of multiple arguments and attributes that you have when you configure RDS. People are also using RDS historically. Fixing RDS configuration is not that easy, because you have to migrate actually to a new RDS instance before you make the configuration fixes in Terraform, but very, very predominant.

We've actually focused on RDS and some additional popular database system, like Elasticsearch is another one of them, because we saw that those are just getting a lot of hits in terms of configuration failures and errors in our small customer base. Do you understand? One obviously is the amount of configuration errors. The second one is exploitability, and this is where it gets tricky. I think I mentioned also this already, but exploitability around configuration is probably not as mature as it is in other forms of programming languages and it's one of the challenges that our team has faced in the sense that you can basically wrap every configurations with so many layers that potentially can protect it where it gets to the point where you are asking yourself for families of configurations that are very deep inside the logic of the application whether it makes sense even to repair them or not. That's where we have the discretion to focus, for example, on resources that are facing the cloud. Think of the load balancer is I think a good example.

Your load balancer eventually is your frontend server that talks to the Internet and makes sure that all your requests are getting routed to the right places, but it has a like a very wide surface area in the sense that it can get attacked. If it gets exploited, it cannot lead people inside your organization. That's why we've doubled down on the elastic load balancer, ELB, the ALB, the previous generation as well just because it's a much more, just in a much more likely scenario going to get exploited as a source of configuration.

Also, it's another one of those resources that obviously got a few provisions and additions of additional features and APIs added to it and it also made some of the more granular configuration. I mean, getting it right slightly more complex. On the other, you have like a very good modules and templates for it, but when you think of something that has such a wide surface in terms of its touch with the public Internet, that's probably one set of configurations you really want to get right. I think for Chekhov, it was probably the right decision to focus on having like a very solid set of policies around it that really tried to nail the right configurations that you have to have on it.

[00:49:33] JM: Okay. If I run Chekhov during CICD, what would happen?

[00:49:41] GE: A great question. There is actually three ways you can run Chekhov. You can run it from your personal computer. You pip install Chekov. You point it to a folder, whether it's online, local that contains your configuration errors, and you'll basically get a report printed out of all the configuration errors first, and you can start knocking them off one-by-one. You get the annotation of where exactly the code, the configuration error was found? What policies in violation? Actually, we extract all the variables. If you're inheriting variables from other places within the directory, we'll actually point you to the variables which we used in order to scan the code, pretty cool.

One place, do it locally in your computer. Most of our users do that for the first time and then they see what big of the problem they have, companies that have like one Terraform that rules them all. Probably that's a good way to go do a one-time project. But probably if you're managing a team of like 5, 6, 7 engineers, bigger than that, you'll probably want to automate, and that's the two other methods. One way is to actually integrate to like a pre-commit hook. You can integrate this into your JetBrains ID, your GitHub ID. Anything that you use to write your code, and eventually it does the same logic of scan in a much shorter dimension, but in the sense, it captures the configurations before you push them into your – Or merge them into your codebase. That's probably a great place if you have like dedicated engineers that are writing infrastructure all the time. We've had a community contributed to plug-in to pre-commit, which I think is awesome. Just find those configuration errors as code is getting imported into the codebase and before it gets merged into a master branch.

The third place, and I think that's been probably the most popular use case, is to put it on a Jenkins server, a CircleCI server, and have it just run as one of your tests that are testing a branch once it's complete. We have a customer that ran locally saw that they have tens, I think even hundreds of users that are managing and changing Terraform code, and what they did basically is they integrated Chekhov into their [inaudible 00:51:46] system and what they did is that every branch, once it goes through the merge request, it goes through a set of checks obviously, unit tests and so forth, and one of them is Chekhov. If Chekhov fails at least one of the tests, the entire branch fails.

You can do this on probably any one of the popular continuous integration software and continues delivery software out there, but it's pretty cool. What happens is that the branch gets failed, the user, the developer, gets a notification that their branch had failed because of a Chekhov failure in identifying encrypted EBS volumes, for example, and they have to select between remediating, basically writing, looking it up on Terraform documentation. Seeing what's the argument? Why it's missing or an existing configuration is out of policy, or they can provide a suppression. They can provide an inline suppression, and once they do, actually, that CD systems picks it up and moves it to one of the code owners. Every time someone wants to skip, annotate a skip of a Chekov failure, it basically goes to the code owner's groups and they have to approve that request before it goes to another build process.

That's probably the more advanced way to use Chekhov. Yeah, I think it's been going fairly well for summer customers. We've seen like very, very big footprint customers. We know Amazon internally are using them. Even HashiCorp has tested it at some point. Salesforce have given it a try and given some good feedback Lots of good Google big companies have tried it and provided some valuable feedback and we're pretty happy to see it getting integrated and merged into more and more systems and in unique workflows.

[00:53:29] JM: Just to clarify, this is a static analysis tool for finding Ms. configurations. What's difficult about building a static analysis tool?

[00:53:42] GE: Just one thing, the content. Someone has to sit down, research or get the knowledge of what's a configuration. Create a signature or the rule and deploy it. I think we've had the maybe luck to say that we just – In the past year, we've done it in runtime, the same exact drill. We've written and built scans for Python-based scanners that have looked at Amazon telemetry and APIs and picked up on configurations and then we thought we saw there was no very mature equivalent in Python for build time, and that's been our mission.

Actually, writing the framework or doing the static analysis. I told you, it took Barak – Which is obviously a prodigy, but it took him two days. The challenge has been to write the content, and to write the right content. We tried to keep our policies in check and always to think if we would want those checks in CD system as well.

[00:54:36] JM: You are a part of Bridgecrew, which is a business. What is your platform do on top of the open source tooling?

[00:54:45] GE: Actually, we have a sound philosophy around it. Bridgecrew, just to give like the former background, it was founded in February 2019 between me and my two amazing cofounders, Idan Tendler, which is our CEO that sits in San Francisco, and Barak Schoster, which I mentioned is obviously like a computing science and open source prodigy, which I really – Probably the most smart person I met in my life, and it was founded like a year ago. Just raised – Just to announce, we raised in total $18 million. 14 million of those from a cool venture capital out of California called Battery Ventures, and obviously our seed funder is backed by NFX, which is another California and Israeli fund.

We believe in short that this entire layer of identifying the problems is something that should be free, and I think the community already understood this much better than us. There’s like tens, if not hundreds of tools out there that give great visibility to configuration and everything around your cloud infrastructure. We’re just tagging along for the ride and we’ve actually used and contributed to some of this amazing open source. Incorporated some of it back into our commercial application, but we believe that everybody, everybody should have the access to good visibility around their configuration and configuration errors. In that sense, open sourcing Chekhov was a no-brainer. We thought as part of this philosophy, we have to contribute this type of rational back, and we’re mostly focusing the business aspect on the fact that eventually you have tons and tons of the of great cloud security talent, but it's all focused in like three or four companies. They’re working probably for Netflix, or Airbnb, or Lyft, or Spotify, and there're tons of great developers out there that are not enjoying the privilege of either working with them, learning from them outside going to conferences, which are probably not going to happen in the next year. We wanted to almost democratize and take some of that knowledge around how to automate these processes and make them much more simpler for N-developers.

Some developers can download Chekhov and run them locally. Not all developers can incorporate them into their Jenkins pipeline, but very, very few developers actually know by heart all the different arguments and fixes you have to do for the different provisioning languages. The Bridgecrew platform, which actually has a community element to it that focuses entirely on the visibility aspect of it. As I mentioned, we want that area to be as accessible as possible and free, but the core platform, our pro plan if you will, is focused on making a lot of the corrections and the fixes accessible to a bigger crowd. Whereas our community plan, our open source is just giving you great visibility to your cloud infrastructure. If you want to fix things and do it like very, very fast, you should check out pro plan, which actually has like a 60-day free trial, which gives you the ability to look at it, and it contains both real-time and runtime fixes. You can invoke Lambda functions that encrypt all your S3 buckets with a single click or create your cloud trail auditing logs in two clicks and remove the unused IAM groups in like two clicks, and that's saving DevOps and security guys or gals tons and tons of time, but it's also exposing a lot of the fixes in build time.

If you see a bad configuration in Chekhov, you, as I mentioned, have to go and figure it out. You have to go to the Terraform documentation. The platform in our commercial offering contains a fix that you can eventually send out as a pull request and send it directly to an engineer and give them exactly a line of code that needs to be added exactly where it needs to be added. If it's to change an existing array or to change an existing argument or to add a new attribute, we basically provide that as a recommendation, and that's basically part of our core business to try to make some of that wealth of knowledge much more accessible to all developers and just to make people write more secure code eventually.

[00:58:39] JM: All right, Guy. Well, it's been great talking to you and thanks for coming on Software Engineering Daily.

[00:58:42] GE: Thank you, Jeff.

[END OF INTERVIEW]

[00:58:53] JM: When a large percentage of the population goes into quarantine, the dynamics of internet traffic change. Some companies need to scale down quickly in order to save money, while other companies like streamers and e-commerce retailers are scrambling to keep up with unprecedented demand. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible giving the same familiar SQL interface database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don't need to manage shards from your own client application, and because the data is distributed, you won't lose data if a machine or a data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds. Some of the world's largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily. That's cockroachlabs.com/sedaily.

[END]