**EPISODE 1532**

[INTRODUCTION]

**[00:00:00] ANNOUNCER:** This episode is hosted by Sean Falconer. Sean's been an academic founder and Googler. He has published works covering a wide range of topics from information visualization to quantum computing. Currently, Sean is Head of Developer Relations and Product Marketing at Skyflow and host of the podcast Partiality Redacted, a podcast about privacy and security engineering.

[INTERVIEW]

**[00:00:26] SF:** Varun, welcome to the show.

**[00:00:28] VB:** Thanks for having me, Sean.

**[00:00:29] SF:** Yeah, it's great to have you here. This is your first time on Software Daily. I think a natural place to start is to kick off with an introduction. Who are you? What's your background? And how did you end up where you are today?

**[00:00:43] VB:** Great question. I'm Varun Badhwar, Founder/CEO of Endor Labs. And we'll probably talk more about the subject. But Endor Labs has been around for a little over a year. Prior to this, I started two other companies. Most recently was a company called RedLock in the cloud security posture management space. We started the company in 2015. We were acquired by Palo Alto networks in 2018. And then I spent three years at Palo Alto building up what many might know of as Prisma Cloud today, which was a rebrand of RedLock plus expansion of that portfolio. I'm really focusing on cloud native application security.

My background, did my undergrad in computer science. Was not as smart of an engineer as many folks listening into this podcast might be. And found my calling in cyber security. Got into it pretty early. Spent about four years at salesforce.com starting when there were 1500 people, going to about 7000. Just amazing learnings around cloud before it became main stream and

just have been leveraging that knowledge and expertise to build companies and solve for hard problems for the industry.

**[00:01:52] SF:** Yeah, it must have been really interesting working in cyber security sort of in very early days of cloud. Just a topic that's I think has come up a couple times on the show in the past about there's lots of concerns, of course, from moving from an on-prem world to cloud world. And which is actually more secure and so forth? You must have solved or look at a lot of new types of problems when you were there at Salesforce.

**[00:02:14] VB:** Yeah. I mean, look, it was that fundamental problem was a disappearing perimeter. The second problem was really this notion of shared responsibility. Who is responsible for what components of the security stack, right? And then to this date, there's a lot of conversation on the subject. Oh, people talk about, "Okay. Well, Capital One got breached several years ago with AWS as well. How much of it was AWS's fault?" Not really. It was a configuration problem. Well, could AWS as a platform had made it easier to make sure that customers don't shoot themselves in the foot?

I did lots of learnings in kind of developing this. And I feel like even though we've been doing cloud security now for 12, 13 years as an industry, we're still in our second or third innings. We still, as an industry, have a long way to go. Most customers started with a lift and shift to the cloud and then realize the promise of the cloud isn't being met. And so, then they start kind of re-architecting their applications. And all of those things kind of have security implications, right?

When you go from long-lived servers that are traditional environment to now compute that can last several seconds and it's gone, I mean, how do you deal with security in such a femoral environment? I know this is not a cloud-specific topic. But we've had to think about a lot of things just from the ground up with a fresh perspective.

**[00:03:30] SF:** Yeah, I think we could easily spend an hour just talking about this topic. But maybe to bring things back on track for the conversation today, we're going to be talking about the software supply chain and the potential risks and challenges of using a large variety of open source libraries.

I feel like the concept of a software supply chain is something that has really sort of come onto my radar in the last year or so. I keep seeing talks at events. I saw some work coming to GitHub a couple of months ago. It seems like something that is suddenly very prevalent. Maybe I was just in the loop. I don't know. But starting with the basics, what is the software supply chain?

**[00:04:08] VB:** Yeah, Sean, great question, right? And we'll talk about why is this also becoming a forefront kind of conversation topic today. What is software supply chain? Look, up until like several years ago, a software engineer wrote 70%, 80%, 90% of the code themselves and there's very little borrowed code from other people, right? A lot of it was first-party code.

Today, the best engineers or the smartest engineers are the ones that start with saying, "What are the components that I can reuse that are already pre-existing, pre-built and available on the Internet?" Google is a developer's best friend. You've got Stack Overflow, and Reddit, and Hacker News and lots of ways to bring in information.

Today, statistics are that 80% to 90% of code in modern applications is code you don't write, but you borrow from various parts of the Internet. And obviously, GitHub has become a great source of that. There's 20 to 30 million different open source projects and packages out there that anybody you can use as a starting point.

When you build your house on borrowed components, and the analogy I'll draw here is a little bit like car, when Tesla's building a car, they're not building every component, every module, every chip, every part. They're assembling it.

And so, software engineering in 2022 and 2023 is becoming a lot more like software assembly. You take lots of components, you put them together and you then kind of tie it all with your first-party code to bring your application to life.

And, essentially, when we talk about software supply chain, we're now talking about all of these components that compose your application, that compose your infrastructure, that compose kind of this entire stack that you're ultimately running and delivering applications from.

Now, why is this something that we're all talking about today, right? The US government, I don't know if you've been following, but has now calling open source software security and national security issue. There's actually a secure open source act, a bill that was proposed in a bipartisan manner in the senate in September that's being pushed through. All of a sudden, this is the forefront. But why?

Well, two main incidents I'll call out. One was SolarWinds. This was about two years ago when you saw this breach of SolarWinds. Thousands of companies were using it in their IT stack. Their build systems were compromised. Their update that they then shipped to every customer was compromised and basically breached the environment for every customer using it.

Now, this was really the first time I would say software was used as a weapon against customers of that software, right? For no fault of mine, just because I use software from a trusted party and updated it, I was breached.

And up until that time, the worst people imagined could happen with software and code, the worry was, "Oh, is somebody going to steal my IP and walk away with a USB drive with my code and maybe share it with my competitors or whatever else?"

SolarWinds brought to the forefront this whole idea that your software can be used to attack your users, and your employees and whoever else is consuming that software. And that's a pretty scary proposition.

Now, exactly less than a year after that, we all dealt with Log4j. In fact, just about last week we called it the Log4j anniversary, because everybody's Christmas and New Year in software engineering, one way or the other, was ruined by Log4j last year. And this was a time where we realized that one component, very popular component, open source, having a problem for us trying to figure out spending thousands of hours to track where it is in my application? Who introduced it? What version are we using? Was extremely difficult. And so, I think all of these things together have really kind of brought this topic to the forefront around software supply chain.

**[00:08:05] SF:** Yeah, just going back to what you're talking about before about this idea of the way we built software today is bringing together essentially these different composable units. And that's another trend that I've seen over the last year, is this idea of like composable architecture, essentially. Now, software engineering, so much of it is let's take a bunch of existing components, bring them together to create something essentially new. And you're kind of like a lot of that work is like stitching those things together. And in a lot of ways it's a sort of – it's great because I think it's probably like speed the market is much, much faster because you can sort of borrow these different things.

But obviously, as you're talking about, there's also this unintended impact that could potentially happen from bringing in composable units from something that ends up being compromised. How do people – How can people kind of get a handle on this? What was the process or like the things that we learned from dealing with like SolarWinds or Log4j that people are potentially applying now?

**[00:09:06] VB:** Yeah, both SolarWinds and Log4j were a little kind of different in some ways. One was a commercial under MSA being used software, right? You had contractual obligations between vendor and customer. Log4j was actually much harder because it was just open source.

And so, let's going to talk about this and break this down a little bit. I think, foundationally, in our research what we have found is an average enterprise with about, call it 10,000 employees, 2000 to 3000 engineers, has about 44,000 open source components that the developers are directly bringing in. They intentionally go to GitHub. They download these projects.

Now, where the problem becomes more interesting, Sean, is for each one of those, on average, at least in the NPM ecosystem, it brings about 77 to 78 transitive dependencies that a developer is not paying attention to. This is just stuff, as I say, coming for free as gift to you. But that gift has a tax you have to pay.

Just take Log4j for example. A simple logging library has a 141 transitive dependencies that come with it. The hardest part in all of this, as developers are going in every single day and consuming all of this open source kind of code from the Internet, is we just don't know what the

sprawl looks like in our internal environment, like across multiple GitHub repositories, multiple application teams, multiple engineering teams. What are all the components? Who is bringing them in? From where? And how are we consuming all of this? The how is the harder part.

I'd say the industry has solved up to the point of scanning your manifest files, like if I'm XML and package.json files and figuring out, "Okay, here's the list of imported dependencies." But I may be importing Kubernetes, which has a hundred thousand lines of code. But I may be using only a thousand lines of those code and just calling one method one function. But I just don't have that level of visibility.

The biggest learning coming out of Log4j ways most people scramble. The most advanced engineering teams had war room setups, spreadsheets being shared saying, "Hey, Mr. Developer, please go check your code. If you're using Log4j, please put it in the spreadsheet. Tell me what version. How you're using it?" Et cetera. Et cetera. That step to thousands of hours, Sean.

And you wonder in 2022, 2023 when you have all these amazing observability tools across the stack, why don't we have observability deep enough in our dependency graph? That's the number one problem to answer your question.

The second part from there is, because when you add up these direct dependencies' times, all the transitive dependencies, an enterprise has hundreds of thousands of these different components. Now, how do I manage and measure risk around all of these? Well, I'll say the fundamental problem in the industry is we don't give developers visibility and telemetry to say, "Here's the health of this project, right?"

Today, as an engineer, you go to Google. You go find something on GitHub. You may look at their stars on the project. The Forex has a proxy to popularity. But you don't know who the humans are. You don't know what their motivations are. How good is the quality of their code? Did somebody actually analyze it? How good is their security posture? How disciplined are they about new commits? How disciplined are they about test code coverage? Do they actually run CI or not? These are all factors that play into your decision process as an engineer to say, "Hey, do I want to bring this code in? Is it sustainable? Is it high quality? And is it secure?" We don't

have good answers to that today. So, developers are flying blind as a result of that. That's another problem that needs to be solved.

And the last piece I'll say, as it relates to kind of all of these components coming in is how do you secure them at scale? Most engineers here have received reports from security teams for vulnerabilities. And I bet you, eight out of ten times, when you actually analyze your code, it turns out that it's what we would consider a false positive, or not applicable, or not exploitable in your environment.

Yet, the security teams in the industry today just don't do a good job of weeding that out. And the philosophy so far has been developers are guilty until proven innocent. And that needs to change. There is so much wastage of time. Because once you get that security vulnerability report, as an engineer, you're having to look at – spend hours looking at your code only to then go back to the security team and say, "Look, we have a false positive here and here's why." Our tools need to get better, Sean.

**[00:13:36] SF:** Mm-hmm. Yeah. And I imagine if, as an engineer, you're primarily seeing false positives from these different reports, you're going to eventually not take them as seriously. Maybe not dive into the details quite as much because you're like, "Well, the last 50 of these I've dealt with have all been false positive. I need to ship this feature next week. I need to focus on that." And that's kind of how a mistake could be made.

**[00:14:04] VB:** Yeah, 100%, right? And the other thing is like a lot of times security will say, "Well, look, I need a clean bill of health. Just go update the darn thing." I think what people don't appreciate, and this is why in engineering we have this terminology of dependency hell. It's a real challenge for engineers to have to go update dependencies, be concerned about what's going to break. Do they have enough test coverage? Do they accidentally take down another peers' application because they're pinned to a particular version of dependency? We just don't have visibility into all of this.

And there's no customer success team sitting behind all this open source software to help you. And so, I think as we think about moving forward, look, we all need to embrace open source software. We all need to do it the right way. And so, the philosophy that I always talk about is

you almost need a dependency life cycle management kind of thinking and a process that starts with what is the intake process when my developer came in this morning to work and decided to use a new kind of dependency? Well, how do I help them make the right selection call? Are they using the right version? Can this code be trusted? Is it good quality? Is it secure? Give them that metrics. Let them use it. Let them kind of deploy the product.

And then on an ongoing basis, measure and monitor the health of that project. Because, you know, what may be thriving today, six months from now, nine months from now, maybe an abandoned dependency, right? The maintainer may be gone. The GitHub repo is archived. But a developer just doesn't know. And unfortunately, you're going to find out the hard way when you have a bug and you're trying to get a fix and one's not available, right? I think we just need better visibility. And we need to pay attention to this problem.

**[00:15:45] SF:** Mm-hmm, yeah. I think that's a natural jumping off point to talk a little bit about in Endor Labs, which you're the CEO and founder of. What is Endor Labs? And you know how does it help developers essentially put guard rails around open source libraries?

**[00:16:00] VB:** Yeah, great question, Sean. Endor was really built firsthand because when we ran into SolarWinds, when I was running this 400-person engineering organization in Palo Alto networks, and when our board was asking us a lot of questions around our software supply chain, it was hard to answer this. It was hard to quantify the risk. It was hard to go figure out even what are the different components that compose our application? Who brought it in? How were they vetted? Or were they not vetted at all? And where is our risk?

And we did run a modern software composition analysis tool, a CA tool commonly known as, and it generated 68,000 alerts for my 400 engineers. And literally, my VP of engineering sat down with me and said, "Varun, do you want us to just stop feature development for six months? Because that's about how long it'll take out 400 people to go dedicate their time to go through these kind of list of alerts and figure out what to fix? How to fix? Etc." And we knew we didn't have those many security problems. Fundamentally, the whole approach was broken.

What Endor Labs started to do is really solve kind of this problem broadly around software supply chain security. But we said, "Let's start with the fundamental problem," which is when

80% to 90% of your code is code that you don't write, but borrow, A.K.A. dependencies, let's start with solving dependency management first."

And in dependency management we said we're going to build a platform and a capability that helps do three things. One, when developers are bringing in new things and considering what to use, let's give them the visibility and metrics basis of which they can decide if they want to still use this code and rely on it as part of their critical application. That's the selection process.

We then said security is completely broken. There's far too much noise killing developer productivity. And the fundamental problem there was everybody just relied on manifest files for scanning. The problem there, Sean, is you just don't know how your developers are using the code. Just because I am importing library doesn't mean I'm impacted by every vulnerability in that library. And that required a different approach technically to solve it.

The thing that we do very differently with security is we don't just rely on your manifest files. We actually are performing a very specific form of static analysis where we build call graphs of your code. And so, we can understand and trace exactly how your first party code calls your dependency. Which methods, functions and API calls it uses? What part of the code can be ignored? Because your developers aren't using it. What transitive dependencies as a result are being invoked? How are they being used? And so, the dependency graph is extremely detailed, which allows us to prioritize on vulnerabilities that are actually just reachable in your code. Not because there exists in a package version.

And through that, we can actually reduce about 80% of the vulnerabilities reported by existing security tools. That's a huge time saving. Because in our research, each one of those vulnerabilities takes about eight hours for a developer to investigate. You can just imagine, there's tens of thousands of hours of savings there. And the last piece we actually do is help our developers actually manage and maintain these dependencies to avoid the dependency health challenges.

So, examples of that. Well, your importing library, I see it in the manifest file, but your engineers refactored code and stopped using it. You could safely just remove that entire dependency and improve your build times, reduce your security blast radius.

There might be a package, and this just happened in the Go ecosystem last week. I can't remember the name, but very popular package. The maintainers decided to archive it rather than transferring over ownership to somebody. Well, how do I, an engineer, just busy on my application development know that, right? We can push that kind of information and reminders to developers and say, "Wait a minute. This is part of your critical application. It is no longer maintained. You need to road map a potential swap to go to an alternative dependency." Or, "Hey, you're going to need to update your library. But here are all the different paths through which it's accessed. Here are the different repos that have access to the same version. And so, how do you best update?"

We just try to simplify an engineer's life all the way from selection to security and maintenance. We have broader aspirations. But I'd say, first, we really want to nail down the dependency management problem.

One last thing I forgot to mention is the industry is talking a lot about this concept of software bill of materials, SBOMs. That to us is table stakes as part of this process. If you're teams, your management is asking you to produce SBOM artifacts for your application, we can help you automate that entire life cycle as well.

**[00:20:49] SF:** In terms of doing the static analysis, how does that work from if I want to apply this tool to my source code? Am I essentially giving access to my repo to Endor Labs to perform that static analysis?

**[00:21:04] VB:** That's exactly right, Sean. The way we integrate is let's say using GitHub Cloud or GitLab Cloud. We just have an app. You connect us to your Git platforms and we scan it. If your code is on-premise because you're using GitHub Enterprise server, or Bitbucket, or kind of one of the other ways, then the best way typically is customers will connect us into their CI process. There will be a step in CI pipelines. And every time there's a code commit, we would scan and analyze the code there after we do a one-time scanner for your existing repository.

In terms of how we surface all of the data and integrate into the user experience or developer experience, we have plugins for the IDEs. People can connect us. That's the earliest point of

feedback you would get. If you're doing an NPM install something, we would tell you right there and then, "Hey, Mr. developer, here's kind of the different problems you're likely to run into. And here's what your organizational policy may mandate."

And then in the CI process, companies can actually enforce that policy. Example being you can say I will not let you bring in a new dependency that's not even maintained or where the repository is archived. And so, we would do those checks and we could block you in the CI if you're doing that. And then there's a UI and an API available for the security orgs that want to see PAN repo visibility and kind of analytics and findings. They could do that through our UI or APIs as well.

**[00:22:29] SF:** I see. And then, let's say that someone – Let's take the Log4j vulnerability is an example. If I was using Endor Labs, how would that help me essentially address or manage the impact of something like Log4j?

**[00:22:46] VB:** Right. Great question. First and foremost, where people struggled with was figuring out which application, which repositories were using the affected version of Log4j. And the harder question was not only where we had directly imported it, but where had we imported something that was then transitively bringing in the impacted version of Log4j?

Let's start with a simple example. I have 90 different applications, and I need to quickly figure out how to do this. It took companies thousands of hours to manually review the code and manifest files to figure that out. We have that continuous inventory being maintained. It would take you five seconds to run a query that says, "Show me all repos where I have this version of Log4j." Step one.

Let's say I've gone from 90 repositories to 30 that have the affected version of Log4j. Well, my next question then is where am I actually using Log4j in a manner where I'm calling the vulnerable function? Right? Because you have to use a particular method call and function call in order to be exploitable. That is an even harder question to answer. And because of our static analysis and producing of call graphs, we have that information instantaneously available as well. So now you would say, "Okay, and this version been used, and it's actually reachable in my code." Great. Now I can get you that level of granularity.

The third piece is, okay, now I know where it is. Now I know where I need to fix it. The reason the industry statistics today are saying that about 40% of companies have still not fixed Log4j is because let's say you found the project, the source package where there's a problem, and you update it. And you're in the engineering platform team in a company with shared services. Let's say this is the auth module and 20 other teams use this auth module.

Just because you updated this package doesn't mean all those 20 teams have updated it. You now need to find out who those teams are that are using it and get them to make sure that they're updating it. And that visibility doesn't exist. I've talked to hundreds of different platform engineering teams. They just don't know. Who is using the components that I'm shipping? And how are they using it?

Because of our dependency graph, we actually have that information as well. We can say, "Okay, we found the source of the problem. We fixed it. Here's an updated package." Now, I know exactly which 20 teams to contact and say, "I'm watching you and you're still on the old version. That needs to be updated as well so we can kind of get all the way through the remediation." It's a very systemic way of solving this problem.

**[00:25:16] SF:** I work in data privacy in my day job, and I think like one of the big challenges historically with companies that are dealing with privacy challenges is that they essentially end up with a similar problem where their customer data is like all over the place. They have essentially a customer data sprawl. And in the problem that we're talking about, you essentially have this open source sprawl and all these transitive properties. Do you think that we need essentially some sort of shift as in way we kind of engineer products to rethink essentially this problem where I think, historically, if you're working on a particular project on a team in a large engineering organization, you're focused on solving that problem. And maybe you're just bringing in dependencies. But then they have essentially these large-scale impacts that could happen in the case of something like Log4j where people could be depending on you. And then even if you fix the problem, it hasn't necessarily fixed their problems and so on. And you end up with this huge sort of communication problem and a really hard problem to solve just of where things are and the logistics of getting everyone on board at the same time. Should we kind of be

rethinking maybe the way that we essentially depend on these types of softwares and architecture things?

**[00:26:34] VB:** Look, ideally, yes. Realistically, it's very hard to get there. You come from Google. You know this. Google has a monorepo environment. Literally, the core of Google applications are all shipped of the monorepo. If you want to bring in a new dependency, somebody actually has to take ownership of it within Google. It's vetted. It's reviewed. There's an army of people that kind of do fuzzing and all kinds of testing against the dependency. It's then introduced. Many times, it's forked if it's critical. And now, Google owns it, essentially, right? It manages and maintains it. And you can only ever have a single version of a dependency because it's in the monorepo. So, you can't end up with 35 different versions of the same popular package like Spring Boot.

Now, architecturally, from an engineering perspective, monorepo does bring a lot of benefits with dependency management. The reason I said realistically it may not be feasible is because companies that have made decisions are already there. Large enterprises I talk to have tens of thousands of code repositories. And guess what? They're not moving to a monorepo anytime soon. Nor do they have the army of people whose job it is to manage, review and maintain these dependencies. Question becomes how do you actually do this?

And actually, we've talked about this one concept. And you might see it on a website in some places. We talked about this concept called a virtual monorepo. Or we say what are the benefits of monorepo? Even if I'm using 100 different code repositories, I have a singular way to get visibility into the dependencies. Even though my team has no access to this other code repo, I at least have the inventory of their dependencies. I know what dependency versions they're on. Who's using it?

And so, one of the things we can do is help you consolidate versions versus having a sprawl. Because in your repo, you don't know what else is happening. So, you bring your best version of Spring Boot and everybody makes that decision.

With Endor Labs, we're trying to give you a virtual monorepo even though you'll never get to a monorepo. And we'll get you benefits where you can consolidate versions of dependencies. You

can have certain approved set of dependencies. Now, as it relates to the vetting process, that's where we're doing that work for you.

I know I've spent a lot of time talking about the call graphs and static analysis we do of your code repos. But I did talk about the analytics and the data we present on risk to the developers. The way we do that, Sean, is we are actually analyzing the – we're going from the package manager to the source code. And from the source code repositories in GitHub, we're analyzing all the behavior on GitHub. We're analyzing who are the humans behind these projects? We're analyzing how long have they been in the community? How many other projects do they support? Is it a corporate sponsored project or not? We're analyzing their metadata of their past releases. How many issues are being reported? How many issues are being fixed? How frequently are they being fixed? And so on and so forth.

There's a lot of machine learning and analytics that we're leveraging across large data sets to say, outside-in, how healthy is this community project and dependency? And that, again, in a Google way, is a lot of people, a lot of custom automation. But what we're trying to build at Endor Labs is really for the 99% of the enterprise that isn't Google and will never become Google.

**[00:29:49] SF:** Yeah, that's really interesting. I really like that idea of applying essentially modern AI, machine learning to automate a lot of that process of inspecting these repos and making sure that they're good from a security posture based on essentially the behavior that people are contributing. You can, I imagine, build up some model representation of what represents essentially like a healthy secure project versus something that's like insecure. And that prevents me from going and open sourcing something that maybe has malintent that is not going to look like essentially a healthy secure project.

**[00:30:26] VB:** Yeah. You know, Sean, it's also beyond just secure. It's quality of the code. Wo we actually download the code of all the versions where we're running lots of linters on that to see like are you following best practices? Do you have good task coverage? Do you actually have problematic APIs you're calling? We're trying to flag potentially API breaking changes. We're trying to make sure your application is, is performant and is high-quality when you bring in these dependencies.

**[00:30:52] SF:** Mm-hmm. Yeah, that's so super high value. What is, I guess, the process of building up those machine learning models? What are the inputs? And how did you go about actually building something like that?

**[00:31:07] VB:** Yeah, the input is really two set, right? There's metadata and then there's data by the code. The biggest inputs are typically we'll go after the repos and we'll get all of the information about the maintainers, the contributors. We'll look at the human graph, as I call it, right? What is a trust graph? Relationships? Locations? Other projects? Corporate sponsorships? Not sponsorships? Stuff like that.

Then we're looking at activity related metadata. Activity could be releases, pull requests, leveraging bots, issues being reported by the community, mean time to resolution. Lots of metrics there. And look, we're doing this for hundreds of thousands of projects from the outside. And so, we kind of get a mean basis of like, "Hey, here's the mean. Here's the median of where typically a lot of this falls. And wears things outlier?"

Then we're looking at certain behavior and pattern changes that are more anomalous to look. It's been a well-supported community. And all of a sudden, we're seeing a drop off on activity. What does that typically mean? We're looking at risky behaviors. Like, "Hey, we're not seeing code reviews on pull requests." Stuff like that. Or we're seeing a domain that's about to be expired on the email of the maintainer. Could that be a candidate for hijacking? The inputs are lots of different things about the GitHub project, community, code, activity, etc.

And then kind of coming out of this, and this kind of gets into a hard problem in cyber security and machine learning in general, is you just don't have a lot of labeled data, right? And that's a fundamental problem. I don't think that's a problem that just we have. It's been there for decades in cyber security. And so, I think there's always the input of like, "Hey, how do you start giving confidence levels and exposing certain threshold where your user can decide where they want to be in terms of accuracy, false positives, false negatives? What does the appetite to tune confidence levels of that information?" But, yeah, that's kind of how we're working on this problem.

**[00:33:09] SF:** How did you guys go about building the label data? Is that something that you're taking on internally at Endor Labs?

**[00:33:17] VB:** Yeah, pretty much. There is no better answer. I mean, either the customers provide that input into their product as they find some stuff that's problematic. Or you and your research team internally is continuing to find and label the data.

**[00:33:32] SF:** Great. You talked a little bit about sort of your thought process for the types of problems that you wanted to first address at Endor Labs. But can you talk a little bit about what the development process from an engineering perspective was of creating a project? Where did you begin? And how has essentially the product evolved since the start?

**[00:33:52] VB:** Yeah, it's a very interesting question. Because as we were assembling the team, there was really multiple facets of engineering talent that we needed to bring in. One was there's an internet scale problem building infrastructure and building the pipelines to scan millions of GitHub projects for every update that they ship and all of the metadata that I just described. So, you have a large data problem and building the pipelines to do this at scale and the infra to spawn off thousands and tens of thousands of containers to kind of do this in parallel.

There's a second problem, which is the static code analysis problem. It's never been done before. Most existing tools today just rely on manifest files to scan what dependencies you're using. But nobody has had the courage, let's put it that way, to go figure out all the way actually how am I using these dependencies? Just because they're imported is not enough information. I need to know how they're being used.

And so, for that, we need to go find the world-round experts that have been working on this problem. And it turns out a lot of them were in Academia. And so, our static analysis team or software analysis team, a lot of them are in Europe because that's just where a lot of the knowledge and expertise resides. And then there was a third set of kind of team we had to put together, which is the AI/ML analytics team, know which, again, we wanted prior experience in security. Because the labeled, unlabeled data is a real problem.

Those were kind of the three main teams we assembled, right? The large data and infra team, the software analysis team. Seven of our team members in engineering are actually PhDs in computer science or related fields. Really, just hard problem solving. And then the machine learning analytics team. And then, obviously, you have to round it up with a UI and other piece.

What we've ended up with, Sean, is today we have about 35 people in engineering. About 20 are in the US, many of which are in Palo Alto, but also remote US employees. We have about four or five in EMEA. And then the rest in India. And so, we're continuing to hire the best talent wherever we find it related to kind of domain-focused problem solving.

**[00:36:01] SF:** Mm-hmm. And from an organizational perspective, is a team that's focused to say on the machine learning problem, are they spread across these different time zones? Or where are the time zones handling specific problems?

**[00:36:16] VB:** Time zones aren't solving. I wish they were all solving for specific problems. It would make life from a management's perspective simpler. But they aren't. Our perspective has been let's find the best static analysis people in the world. Two of them end up being in the US. Four end up being in Europe. So be it.

And we try to do that, but we don't make that a barrier or a requirement in recruiting. It does mean obviously the teams post-Covid in most world. Organizations are becoming more globalized, and management and culture. Async communications are huge and giving people the opportunity using Slack pretty effectively, Zoom, recordings, opportunity for people to kind of comment, communicate. You can't just go into a conference room and whiteboard and make a decision right away.

**[00:37:05] SF:** Yeah, as an entrepreneur that built a company in a pre-pandemic era and then now doing it during the world that we live in, how has that experience been like? I imagine, there must be a lot of new challenges that you've had to sort of navigate as an entrepreneur to – it's great that it opens up your talent pool because you can hire essentially the best people in the world, as you mentioned. But you now have this challenge if you can't just go into a room and sort of whiteboard a solution.

**[00:37:34] VB:** Yeah. And look, Sean, I what I may say may be conflicted and people may have different points of view on it. But certainly, productivity is a challenge. Actually, I was reading over the weekend Mark Benioff posted on Slack at Salesforce saying, "Hey, we're noticing productivity for our new hires through pandemic has been nowhere near as high as productivity of the people that joined us pre-pandemic. And what can we do on onboarding?" And it's a real challenge.

And look, I will say, and people may not agree, but having people in a room when you're especially in a startup ideating, whiteboarding, not everything is perfect and process-driven, is more effective and more productive. And so, I think we have taken on the tax knowing that we're a post-Covid and post-pandemic company. We recognized that things may not move as fast as we would like them or have traditionally seen them if everybody was in an office.

That said, I would say people seem to be happier, right? And so, kind of for a long-term perspective, this is a marathon, not a race. Having the flexibility. People in Palo Alto show up three days a week. Not five. People who are remote are still welcomed. I think there's more of a happiness index oriented there.

I will say the camaraderie, the idea of, "Look, if everybody's here, we'll be having a holiday party right now." But it also doesn't feel right to have a holiday party for the people that are here but nothing for people that aren't. You have to be more mindful about being more inclusive. And how do you make sure that you don't end up in these silos of, "Hey, there's a culture for Palo Alto office. And then there's everybody else." I don't think that works.

We try really hard. I don't have the perfect answers. But I will say, while there's a lot of goodness that comes with kind of hybrid work, it does make it challenging. And productivity does get impacted.

**[00:39:20] SF:** Yeah, I think what a lot of companies saw at the beginning of the pandemic, they actually saw productivity go up because suddenly, especially if you're an engineer, you could have like more focused time. And no one was taking vacation as well was another factor that helped with productivity. But then I think over time, the productivity kind of went down.

I think one of the other challenges that I think that companies potentially face is that people really – especially in a startup, it's so mission-driven. You're there because you believe in the people, you believe in the founders. But in a remote world where it can feel more disconnected, you can end up in a situation where it feels like it's just another job. What's the difference between this job and some other job on basically just showing up, I'm writing code. And you lose a little bit of that sort of people connection and the passion. I think that's something that just takes more mindfulness and time to really build the culture that you want to build.

**[00:40:17] VB:** 100%, Sean. I think you nailed it. I think it's that mission and getting everybody a remote, secluded to kind of feel very connected to that mission. I think it's much easier being in-person do that. And the scary part is what we create here for these first 40 people is going to be what's going to scale to the next 400. And I think that's the thing that keeps me up at night, is how do I create that inclusive mission-driven culture but do it at a global scale?

**[00:40:43] SF:** Mm-hmm. Yeah, absolutely. Going back to the engineering at Endor Labs, what's the kind of most difficult engineering challenge that you've had to try to solve so far?

**[00:40:55] VB:** Boy! If I could use some interesting words, I'd say package managers like me with an NPM are piece of crap in many ways. And in the sense of just there's no standardization, right? Whether it's versioning, whether it's tracing back to source code, it's a mess, Sean, for lack of a better word. And honestly, the reason we get more and more committed to solving this problem is, every day, as we dig into how people create dependencies, manage dependencies, version dependencies, you really have an appreciation for this dependency health problem. But we get more committed to helping our customers with it.

There's just a lot of dirt in these language-specific ecosystems. The package managers, zero consistency across them. I think that's probably the hardest challenge is how do we go at scale machine-driven? Go find packages? Make sense out of packages? Make sense out of tracing back to their source code?

Just a simple problem. Sean, imagine this, right? From a security perspective and risk perspective, if you're bringing in something from a package manager a particular version and

there's no guarantee that that is what is in the source code of GitHub, that's a scary thought. And we haven't solved this problem today. People talk about providence and code signing. I mean, that's infancy. But I could be downloading something completely different than what appears to be in GitHub as a problem. That's one.

I think the other problem is from a customer perspective. How release engineering works for most companies? Everybody's different. How you're doing releasing, branching, tagging. Very, very different. And so, when we're building this user interface and developer experiences, what are we optimizing for, right? Everybody has a different release process. That's also another separate interesting engineering challenge.

I'd say we have committed to taking on this problem of dependency management. But it's a dirty problem. And that's why we think we're on to something. Because if we solve it well, we would provide a lot of relief to tons of engineers out there.

**[00:43:03] SF:** Yeah, absolutely. This is not a problem most people want to take on themselves, especially when they're focused on other things in their business. You mentioned you have to have expertise across machine learning, static analysis, infra, as well as your regular application development from like UI and so on. What are some of the tools and technologies that you're using behind the scenes that you can talk about?

**[00:43:31] VB:** I'll tell you, just our backend is mostly Go lang. We use React on our frontend. A lot of our early engineers came from Uber. And so, they had a strong preference. So, Bazel. Bazel is part of infra. There're a lot of other people that – There's a lot of hate for Bazel around here. But that's kind of what we use them for pipelines.

And then, what else? We're fully a GitHub shop internally for everything, from issues and ticketing. I will say we're reaching that point of questioning whether GitHub issues will continue to scale with kind of our needs of having these distributed teams and lots of different cycles. I mean, GitHub issues is simple but is not very customizable. And so, I think we're reaching some of the scale challenges there.

A lot of our engineering discussions, maybe too much happen on Slack, decisions and communications because of that async nature of the organization. Yeah, that's kind of for the most part what life looks like at Endor Labs.

One other interesting thing I'll tell you is, on hiring, my co-founder, Dimitri, and I made a commitment to ourselves saying the first 15 engineers we hire, there will be no more than two people from the same company. The easy answer for us could have been there's been hundreds of engineers that have worked for us in the past life. Let's just pick the first 20. We would be done in three months. And building product.

But it took us a solid six months to kind of breed the foundation. Because, again, when we're solving a problem around dependency management, release management, software development, every company has such a different perspective and culture around it that we wanted to bring in a lot of that into our initial team.

And the other reason is if you think about the product we're building, it's being built by our engineers for engineers. And I always tell them, unlike a traditional enterprise product where you have lots of product managers translating requirements from users to your engineers, here are our best product managers or our software developers. Because they have faced this problem firsthand in their day-to-day life. And they are best suited here. I have the ratio. Just to give you an appreciation. We have 35 software developers and one product manager in this company.

**[00:45:45] SF:** Yeah, that's great I really like your approach to sort of diversifying the team based on like engineering profile. Because I think you're exactly right. The way that different companies manage this shape the way engineers that come out of those companies think about these problems. And that can be substantially different if you work at Google, versus Uber, versus some other company.

What's next for Endor Labs?

**[00:46:13] VB:** We launched the company about two months ago out of stealth. And we're in a number of pilots with tech companies to very large enterprises, top 10 banks, top 10 insurers.

And right now, we're just focused on what classically people call getting through product market fit. And we believe we have a great product in our hands. And now we're getting it in the hands of our customers who've been gracious enough to give us lots of feedback in the process. But it's never done until the rubber meets the road and you can run the road test. We're looking forward and excited to get our first hundred customers onboard onto the platform.

**[00:46:51] SF:** Awesome. Anything else you'd like the audience to know?

**[00:46:56] VB:** Well, look, for those of you that have suffered from dependency health, relief is coming. We hope to solve this. We're early. If you're excited about what you've heard today, we'd love to hear from you. Check out endorlabs.com, our website. Reach out to us. We'd love to show you the product. Get feedback.

Like I said, this is all about for engineers by engineers. And we want to take away the tax that you have to pay when you use dependencies. It's easy to start using dependencies. It's very hard over time to keep them managed, up to date and maintaining. If you have good ideas in that and feedback for us, we'd love to hear and collaborate with you.

**[00:47:32] SF:** Great. Well, Varun, thanks so much for coming on the show and discussing Endor Labs and software supply chain. I really love tools that reduce friction, improve security, let developer teams basically focus on just doing the work. And I think you're solving a really big problem that no one really sort of has a handle on and I think has the potential for a major, major impact on the industry. Thanks for sharing your work that you're doing at Endor Labs and for coming on the show again. Cheers.

**[00:48:02] VB:** Yeah. Thanks, Sean. And happy holidays, everyone.

[END]