EPISODE 1214

[INTRODUCTION]

[00:00:00] JM: Static analysis is a type of debugging that identifies defects without running the code. Static analysis tools can be especially useful for enforcing security policies by analyzing code for security vulnerabilities early in the development process allowing teams to rapidly address potential issues and conform to best practices. R2c has developed a fast open source static analysis tool called Semgrep. Semgrep provides syntax-aware code scanning and a database of thousands of community-defined rules to compare your code against. Semgrep also makes it easy for security engineers and developers to define custom rules to enforce their organization's policies. R2c's platform has been adopted by a lot of companies such as Dropbox and Snowflake and they are gaining a lot of traction.

Isaac Evans is the founder and CEO of r2c. Before founding r2c he was an entrepreneur in residence at Red Point and a computer scientist at the U.S. Department of Defense. Isaac joins the show today to talk about how r2c is helping teams improve cloud security and static analysis. How static analysis fits into CI/CD workflows and what to expect from r2c and the Semgrep project in the future.

[INTERVIEW]

[00:01:15] JM: Isaac, welcome to the show.

[00:01:17] IE: Hey, Jeff. It's great to be here.

[00:01:19] JM: I have done probably 15 shows over the years on static analysis tools. Why are there so many static analysis tools?

[00:01:30] IE: Well, you know, Jeff, I think it's something really deep about developer psychology actually. If I wake you up in the middle of the night and I'm like, "Hey, here's a list

of 20 bugs to go fix." You're going to be like, "Why did you wake me up with this list of bugs?" But if I show you a way that you can automate finding and fixing 20 bugs, even if it took you twice as long, you I think would be much more excited about it. So I feel like developers, as a group, we love automating, finding things, automating, fixing things and being able to do that in code. It's fun. It's addictive. So I think that's why you see this proliferation of just for many different languages and frameworks, people love writing these kinds of tools.

[00:02:14] JM: Maybe you could explain what a static analysis tool is, because it's probably a bigger category than I'm giving it credit for.

[00:02:22] IE: Yeah, for sure. There's, broadly speaking, two kinds of program analysis that you can do. So you've got static analysis, which generally speaking you feed in a source code or binary and the analysis does some kind of inspection on that source or binary and then gives you a list of results or findings. Dynamic analysis would be something that instruments the application while it's running. So let's imagine we wanted to say, "Hey, how many places in the code base call function foo static analysis?" Like the simplest thing we could do would just be to fire up a grep, grep for foo in the code base. Count how many results. Dynamic analysis, we might like basically put a break point inside foo, and every time it gets called look at the stack trace and over the course of the program running count how many distinct callers there were. So that's at like a basic level the difference between the two. And the tool that my company has been working on and that I'm a maintainer for focuses exclusively on the static analysis side.

[00:03:28] JM: And from your work on previous, or working with previous tools, like previous static analysis tools, was there some gap in the viability of those tools that you saw?

[00:03:41] IE: Yeah. The tool that we're working on, which is called Semgrep, is kind of a middle ground, I would say, between things like linters and grep and the really expensive commercial static analysis tools. So you've probably – Developers have created many different tools that are generally per language. So you've got like ESLint for JavaScript, Flake8 for Python, RuboCop for Ruby. There's plenty of great linter frameworks for Go lang. And on the

other end of the spectrum you've got really expensive, you're talking a million plus dollars a year commercial tools that are designed to scan for vulnerabilities. And what we felt like was that a lot of the really advanced analysis tech, program analysis technology, was kind of locked up in those commercial tools which were being unfortunately really used mainly for compliance purposes. And if there was a way to give an average developer access to those tools maybe by building one that was actually free and open source, a lot more people might become interested in the tooling.

And the other thing that we did, which I think is really cool, is we made it really easy to get started. So I can show you how to write essentially like a basic linter in about two minutes and it'll look very similar to code as opposed to being something that requires you to know a lot about program analysis and abstract syntax trees and things like that. So we've tried to make Semgrep actually feel just very similar to grep, but it's kind of like grep for code.

[00:05:24] JM: Can you give an example for how Semgrep works?

[00:05:28] IE: Yeah, absolutely. So let's say that we want to find a function named foo in our source code. Like we could grep for foo, but then that would find comments that have foo in it or strings that have foo in it. What we would search for with Semgrep is we'd install Semgrep and then we'd basically say Semgrep-e, and we'd put foo, left paren, right paren, and that will report every place that's actually a function call inside our program with no arguments to foo. So in a lot of cases the Semgrep queries are actually valid code themselves, which is really interesting.

Now if you want to make it a little bit more complex, let's imagine that we want to find, “Hey, where are all the places where foo is called with one as the first argument?” Obviously you could write a grep for that as well, but with Semgrep, again, just valid code. You're going to type foo left paren, one, right paren. And what's cool about that is you know imagine you had some variable elsewhere in the program like variable X and you defined X to equal one and it was a constant and then later on you called foo with X. Semgrep will still match that call with X because it understands that the value of X is one, and you kind of get all of that smartness

from the engine for free. All you have to do is specify, “Hey, I want to find all of the function calls with one as the first argument.” And really anything that's equivalent to that construct is going to be matched by this engine for a relatively simple query.

[00:07:04] JM: And when am I running this? Is this at build time?

[00:07:08] IE: Semgrep is fast enough that you can actually run it in the editor. This is one of the other unique things, is we've tried to just make it really fast. Not quite as fast as grep, but it's pretty close. So you can run it on like every time you save the file. And then you can also set it up in your like Git commit hooks. You can set it up in CI. You can set it up as like a very long later stage process. But really, and I think this is one of the things that's unique. We're trying to put it just like the linters as early as possible in the development flow. Whereas, traditionally, tools that have a big security focus don't get run in the editor. There's something that happens much later on.

[00:07:51] JM: So why is that useful to have static analysis tool that can run across such a wide variety of use cases?

[00:08:00] IE: The way I think about this is usually when you grab a static analysis tool, like somebody else wrote the rules for it, right? And so it's like, “Hey, I think that you should write your Python code this way.” And a lot of the time you know with a linter, it's focused on formatting, right? Like, “Oh, like it should be this much indentation, this little indentation,” and it's about consistency. Semgrep doesn't really care about the formatting. It cares much more about the sort of like security performance, quality issues inside the code.

So let's imagine that like you've got some project at your company that like you used to have this method named method one and now you want people to just start using method two. You've got like a new version of the API. So you could write a Semgrep that just looks for method one. And then one of the other kind of unique features that it has is you can use the same query syntax to specify an autofix. So it's very easy to write a rule that just says, “Hey,

anytime somebody types like our custom object.method1, make it go to .method2," and that can appear in the IDE as a suggestion.

And so what you've kind of done is you've taken this static analysis tool and you've customized it to be something that's actually very, very specific to your project or your broader team in a way that if you were just grabbing someone else's tool, it would never have that level of understanding that's tailored to what you're trying to do at your organization. And then you're just automating it essentially. Making it really easy to forward patch the code as it's being written.

[00:09:43] JM: I'm curious about the architecture of Semgrep, because you have a tool that you want to have for static analysis across all kinds of different languages, Go, and Ruby, and Typescript, and Java, and so on, but you want to minimize the amount of work you have to do in building static analysis tools for all those different languages. So what's your plan of attack?

[00:10:08] IE: Yeah. Well, our competitor, Grep, supports an infinite number of languages and is very, very fast. So we need a strong strategy to be able to compete with something like that. Semgrep is a tool originally known as Sgrep that was developed at Facebook. And at Facebook, a lot of the code that got written was for parsing. So that involves like, "Hey, like imagine we've got a PHP file. We need to parse that PHP file into a tree-like structure, specifically an abstract syntax tree, and then Semgrep can kind of work its magic." And under the hood what's happening is the query that you wrote which, again, probably looks very similar to the code. That gets parsed by the same parser. So now you have two trees. You have the tree you're trying to search and then you've got your search term as a tree. You're doing a tree matching problem over them.

It turns out that a lot of the complexity is involved with just supporting every weird PHP feature and like Lua feature and you know all these languages being able to properly parse them. And so we've actually re-architected the tool since it's Facebook days to use a GitHub project called Tree-sitter, which if you've seen any of the jump to definition or show references work on github.com, Tree-sitter is actually the same open source project underneath powering both

of those tools. And that's been amazing for us because we're up to – If you include the alpha level languages, 17 languages that are supported in Semgrep and four of those languages have actually come just from outside contributors who have been able to follow the same process that we're using. But it also puts us in a pretty cool position, because basically like if it's a semi-popular programming language, we'll probably support it, which is not something that a lot of static analysis tools do. Usually they just support one language for the open source versions.

[00:12:08] JM: So how does Semgrep fit into the workflow of a developer? Like I already don't really want to spend a bunch of time writing unit tests, but I have to spend time writing unit tests. When am I writing static analysis?

[00:12:29] IE: Yeah, great question. So there're a couple different use cases that we've seen people get a lot of mileage out of Semgrep for. So sometimes it's like, “Hey, like I really wish we had a custom lint for Go that could find this sort of thing, because we just had a bug and it's really clear looking at it right now, “Oh, yeah. Like when someone calls foo and then they don't call bar before they call this other thing, that's just like 99% of the time going to be a bug.” And with Semgrep you can go off and just write something that will check for that in like two minutes and you can put it in CI, you can put it in the editor, wherever. So that's kind of the most basic like, “Hey, like let's do some bug fixing here.”

Broadly speaking, like if you're kind of just like, “Hey, like this sounds cool, but I’m lazy.” We've got a bunch of rules over a thousand now that other people have written that we've got a website, semgrep.dev and, you can just go browse around and see these rules. And so it can be dumb things like, “Hey, like I was writing Python and I put like a Python debugger call in and I forgot to take it out.” So, “Hey, that should just be like a check when I do my Git commit or like before I pass my CI stage.” We didn't leave any debugger calls in there, or maybe console.log.

It can also get much more sophisticated. So we have a really amazing security team. And as a company – So Semgrep is an open source project. As a company, our focus is on using this to

prevent security vulnerabilities. And so if you go to those rule sets you'll find basically pre-packaged collections of rules that are looking for specific security issues across languages and frameworks.

[00:14:14] JM: Are there any economies of scale to lots of people adopting your tooling? Like are there language-specific anti-patterns that you can find using static analysis that people can share with one another and they can run general purpose static analysis rules across their code to find those kinds of common problems?

[00:14:37] IE: Yeah, definitely. So one example that I'll give you is say you're writing Python and you try to use a string as a Boolean. The behavior of this will actually vary depending on the specific string within its prefix. And so we actually had a bug that was related to this that we found in our own code base. And after we saw that we wrote a Semgrep rule for it and we kind of published it out to the rule registry. So now like the rule registry, there's a nice frontend for it on the Semgrep website, but it's just a GitHub repo at the end of the day. And in fact like all of the semgrep rules, you can run them on the command line of course. But if you want to kind of package it up and give it like an ID and a message, there's just a YAML syntax that's pretty easy to use.

We actually have like an interactive playground editor that if you don't even want to touch the YAML, you can just fill out some fields and it'll format it all for you. But we've gotten a lot of people from just like various companies who have been using Semgrep contributing their rules back to the global community. And actually we're kind of in a good problem to have position right now where we have more rules coming in than we know what to do with. And so we need better systems to help people discover, "Hey, like these are rules that could be really helpful for security, or for correctness, or like maybe you just want to make sure you don't have people catching base exception in a language like Python."

[00:16:13] JM: Do users replace their existing static analysis tools with Semgrep or do they use it in concert with their existing static analysis tools?

[00:16:29] IE: So it depends on the user. We've seen both combinations, I guess. I think the thing that really makes Semgrep unique is this ability to just come in and like in a few minutes you can learn everything about the syntax because it's mainly syntax you already know, because it's mostly just valid code. The rules are going to look very similar to your code. And so if you're just getting started, we'll see people kind of be like, “Hey!” Like, “Wow! I have a god mode function that makes any user into a super user. And so I just want to get like a Slack notification whenever somebody makes a PR that calls that function.” You can do that in a few minutes with Semgrep.

We also see people who are actually the sorts of people who maintain linters. Kind of using Semgrep as a prototyping tool to be like, “Hey, like maybe I would turn this into a more full-fledged lint later on, but I’ll first do it in Semgrep just because it's so fast and easy.” And our goal is that eventually the Semgrep engine becomes powerful enough that you don't have to go to like a linter framework tool. You can just express it all in the more simple language. But we've certainly seen people basically be like, “Hey –” And actually you know those collections of rules that I mentioned, we've ported a lot of tools, rules into the Semgrep language. So if you've used a tool like the ESLint security plug-in or Bandit or Gosec, we've actually written their equivalent rules in Semgrep. They're not necessarily better, but they tend to be a lot shorter and it shows us where the shortcomings of our tools are.

We've also had some of the other scanner tools or linters adopt Semgrep as an engine. So node.js scan which is one of the most popular for JavaScript and is currently the default JavaScript scanner in GitLab, if I remember correctly, actually just recently ported their entire core to use Semgrep as an underlying engine.

[00:18:29] JM: Can you describe in more detail how you see Semgrep being used as a modern security tool, a scalable security tool?

[00:18:42] IE: Definitely. So if you've ever interacted with a security tool and especially one that does static analysis, you're probably thinking, “Oh yeah, it has a bunch of false positives.” Like it's going to tell me that using the random function is a security risk because it's not

cryptographically secure random, and I'm out here just trying to pick which image to go first on my carousel, right? So there's oftentimes just a big mismatch between the model that those tools have. And it comes back to this kind of one size fits all approach, right? Where if I'm trying to ship a security scanning tool, I need to be maximally conservative about the kind of rules that I put in it. So I'm going to tell you everything could be a security risk.

And what we've seen people do with Semgrep is basically come in and be like, "Hey, actually, we kind of have a good idea of where our security risks might be, and it's probably going to be like this specific kind of issue inside this framework." And so either they can write those rules themselves or we help them write those rules and then package them up into a rule set. But really what we're thinking about when we see teams adopt it is it's much more about kind of reaching an understanding between the security team and the development team of hey, "Like, we're going to use uh you know React for the frontend." And like React by design is very difficult to have XSS vulnerabilities in. If you know about React, they renamed the JQuery in their HTML functionality. In React that's called dangerously set inner HTML because it's an easy source of XSS vulnerabilities.

And so you can just write a Semgrep rule that's like, "Hey, flag dangerously set inner HTML and then like tag the security team." If you're using the hosted version, it can just make a PR comment on GitHub or GitLab for you and then you can have a conversation about it. And then there's an easy – You can just put a comment to suppress it. But it's much more designed for teams that want to kind of reach an understanding between development and security about, "Hey, this is what we're actually trying to do," rather than, "Oh, we'd like an out of the box thing that's going to report every issue under the sun." And the security team will go through and triage thousands of these and kind of spam the developers with reports that might not actually be relevant.

In fact, on that topic, Jeff, we recently had a post go viral on Reddit and it was an issue that someone had created in our GitHub repository. And the issue was from a throwaway GitHub account. And they basically came in and said, "Hey, my manager is an asset who doesn't care about security. He's trying to make us like change the code into this bizarre construct

specifically to avoid what Semgrep is finding. Can you update the Semgrep rules to find this like alternate construct?

And so Semgrep is super easy to change rules. So like we had that fixed like almost immediately. But it really speaks to I think for a lot of developers there's this kind of dysfunctional relationship between the security team where the security team comes in and they don't really understand necessarily like how the code works or like where the security flaws really are. And at the same time there's a big push to get developers to take more responsibility into their own hands for security, but there's honestly not a lot of multilingual program analysis, static analysis tools that are FOSS software out there. In fact, we're the only multilingual FOSS tool that has this level of language support that I can think of.

So our goal is to really – I think I don't believe that developers have zero care about security. I think like people really want to do the right thing and security is ultimately another class of bugs, potentially a very important one. And so my hope is that you know we give people the kind of tools to figure out like, “Hey, maybe your security team doesn't know what the issues are, but you do,” and you can kind of set that up in a really easy to use way.

[00:23:03] JM: So you mentioned that Semgrep was based on this tool that came out of Facebook. Can you talk more about the history and I guess why that tool was originally developed at Facebook and how it evolved into what you've built at r2c?

[00:23:20] IE: Definitely. So actually you need to go back even further than Facebook for the origination of some of the ideas in this tool. The custom syntax that we use in Semgrep is inspired by a tool named Coccinelle, which actually did a whole bunch of refactoring in the Linux kernel for kind of like bugs. And. basically Coccinelle a way to say, “Hey, like whenever we see a construct like this, make it look like that.” And it was much more complicated than what Semgrep aims to do and it was specifically designed for you know giant C code bases where you have a lot of these problems.

But if I remember right, Coccinelle ended up making over 10,000 automatic patches to the Linux kernel, which is a very good track record for an automated tool. And one of the Coccinelle developers was the first program analysis hire at Facebook. And he ended up creating an adaption of the Coccinelle as part of a broader framework for program analysis called Sgrep. And I was not an early Facebook engineer, but my understanding was that a lot of people there were able to write Sgrep expressions to kind of look for, “Oh, hey. When you're following this function that's kind of unexpected, please reach out to the security team and make sure that you get the sign off from us.” It was used really in the same way that we're hoping other people will use it as a communication, conversation starter like, “Hey, like interesting. We didn't expect people to do that. Are you sure that's what you want? Maybe you’d think about this other thing instead.”

[00:25:00] JM: Can you tell me more about the founding story of r2c like why you decided to focus on this problem of all problems?

[00:25:10] IE: Yeah. My co-founders and I met as undergrads at MIT, and we didn't work together until a few years after graduation. I had been working in the government on a scholarship program. I had actually gotten to do some really cool stuff more in the reverse engineering space. My co-founders were working at Palantir. They had done a lot of consulting for kind of large fortune 500 companies that were having security breaches. And when we compared notes, we felt like the way that people thought about security at these companies as opposed to places like Facebook and Google was just very different.

And at Facebook and Google, they had tools like Sgrep to basically you know provide security advice. Not just kind of the spammy security advice, but really good high-quality security advice early to developers whether it's in CI or in the editor. And at the fortune 500s or the kind of just like more broad corporate world and government as well, security tools are just kind of tacked on at the end. It's like, “Hey, yeah. Now it's time to do the security scan. Oh! We found a thousand issues. Time to fix all the high severity issues if we have time and then just get the thing out the door.”

And not only is that less efficient. It ends up just being way more costly because you know that the application is certainly vulnerable to these like broad classes of defects. So imagine you're building like healthcare.gov. You know that it has XSS vulnerabilities and things like that and you end up buying even more products after you've done these security scan on the code to try to protect it in production. And what I think is really fascinating is that the big technology companies have figured out how to just broadly eliminate entire classes of vulnerability by changing the development process and using some lightweight static analysis tools to enforce it.

[00:27:15] JM: Yeah, that's really interesting. Do you have any other perspectives on the gaps in software development between government and industry?

[00:27:26] IE: Well, one of the things I'll say is that in the security world especially you've kind of got like the fortune 500s and then you've got the banking sector and then you've got the government and they all assume that everybody is doing a better job than they are. And the reality is that I think the only people who are really doing a great job are, broadly speaking, the Facebook, Google, Netflix, Amazon cruise of the world. And Facebook in particular, I actually don't think Facebook is even using Sgrep anymore. That was their original name for it. They have invested so heavily in hiring an army of very smart PHD or equivalent level program analysis people. They built a lot of tooling that is just deeply tuned to Facebook's infrastructure and needs. Same with Google.

On the other hand, there are a lot of ideas there that we think are really valuable that we're trying to put into the platform that we're building around Semgrep. So one that seems really obvious that none of the traditional security scanning tools that government, banks, broader industry have is the fixed rate. So like, "Hey, like if I interrupt you as a developer," and I'm like, "Hey, here's this thing, random. It's not cryptographically secure random. Did you know that?" What does the developer do? What is their reaction and what are the statistics and metrics on how much of the time is the developer like, "Well, thank you, tool. I did not know that. I'm going to accept the suggestion to use a cryptographically-secure random number source." And what percentage of the time is the developer saying, "Get this thing out of my way. Stop

failing my build. I'm very annoyed and I'm going to figure out how to ignore this thing and shut it up as fast as possible."

And so like Google's tooling most notably, like they've very publicly talked about like implementing this kind of feature. And we just shipped a version of that that we've been dog fooding ourselves and just in the past couple of days made publicly available, and it's been great. And like the random example is a perfect one because, yeah, like we had a zero percent fixed rate for all of the stuff related to randomness and the code base, which kind of makes sense because we're not developing cryptographic code. We're not really concerned about cryptographically-secure random number generators being used. Now there are some exceptions and there are some reasons that certain libraries and things would need that, but broadly speaking, we're worse off for notifying the developers about those sorts of issues.

[00:30:00] JM: Well, coming back to your business, I'd love to know about the go-to-market strategy for building a static analysis tool like Semgrep. How do you get people to pay for it? How do you price it? Just tell me about the go-to-market strategy for your company.

[00:30:20] IE: We're still early in the go-to-market strategy, Jeff. I would say that the folks at a16z have this great blog post called commercializing open source, which goes through the kind of stages that they see in this process. And I think where we're at right now is we have a good project community fit. So we have a lot of like great teams that have already adopted the open source Semgrip. They've set it up themselves. These are places like just fantastic developer talent from like Atlassian, to GitLab, to less known valley companies. But you've got like Dropbox is a big user of the open source as well. And our goal is really to figure out kind of, "Hey!" So like Semgrip has a lot of resonance for people who are tasked with automating code review, automating security stuff in code. What is the product that we build up on that that is kind of respectful of the open source approach that we've taken and provides value when you're thinking about it in the context of an organization?

So you start out with project community fit and then hopefully you move into a product market fit where like your early users are telling you, “Hey, this is something that I would like to buy that builds on top of this great engine that you all have built.”

And so I think we've talked about a couple things from that perspective. Semgrep is an LGPL project. It is not open core. So the whole thing is freely available under that license, which is awesome. There's a SaaS version of Semgrep, which also has like a very generous free tier. That gets you – And like the rules are just in a public GitHub repo as well. So you can run off with the open source version and the rules and have a great time without talking to us at all. But our goal is to build enough value into the SaaS version of it and actually things like fix rate, which I just mentioned, are a really good example of that. Those are things that, “Hey, you would need to have like a couple people on the team kind of invested in this workflow and like using it day-to-day.” And then you'd be faced with this problem of, “Wow! We have hundreds or thousands of repos. There are thousands of Semgrep rules.” Like what do we want to actually block the build on and what do we want to just kind of get notified on and how do we do that in a way that doesn't overwhelm us in the way that a traditional static analysis tool would do?”

So I would say we're still figuring it out, but we're trying to draw a good line between the sorts of things that make sense for an enterprise are the paid SaaS features and the sorts of things that make sense for an individual developer who's like, “Oh, I just want like a cool tool that I can run in the editor.” Those things are not monetized and they stay under the FOSS licenses.

[00:33:20] JM: Tell me about some of the most difficult technical challenges that you've had to solve in building your company thus far.

[00:33:29] IE: Well, I feel like we actually got lucky, because we were originally building something that was all about the same problem, trying to address the same problem, but it was much more from the approach of like using the traditional linter frameworks and things like that. And when Johan, who was the early fellow from Facebook joined our team, and he was actually basically semi-retired because it turns out when you're an early Facebook engineer

you can retire. And when he joined us he was like, “Hey, like I really like what you guys are trying to do. I really like your mission.” And then after a few weeks he was like, “Oh, did you know about this tool that I wrote when I was at Facebook?” And we were like, “No.” And so that was sort of the spark that like we got to leverage this amazing open source technology from Facebook with the original author of that tooling.

So a lot of our technical challenges were just kind of like wiped away because Script had been under development for like 10 years at that point. But after that, like there were a host of other challenges. So one of the biggest problems as I mentioned earlier is supporting many, many languages. And so transitioning from this world where we had kind of like parsers that were fully under our control to parsers that are also open source projects maintained by the community was quite difficult. And in fact like the Tree-sitter project of whom the primary maintainer is, if I remember correctly, Max Brunsfeld over at GitHub, is just beautifully designed and that it has this whole – like you specify essentially a declarative language for the parser that's like, “Hey, like here's how PHP looks. Here's how you would parse it.” But that's not an executable program. And then the Tree-sitter stuff takes that grammar and converts it actually into a very optimized C parser, which is super-fast, which lifts the kind of raw text into a tree structure.

And then what Semgrep has to do and the part that's the heavy lifting for us is we have to translate that tree structure into a generic tree structure. So like you've got your Python program, you've got your Ruby program, you've got your JavaScript program. We're translating all of those into this generic AST. But we're still trying to keep some of the like engine that has like features which might be really specific to the language. So just as an example, like imagine you're looking for a Python function that has like keyword arguments. So like the function is named foo and you're looking for like A equals one, B equals two as the keyword arguments.

Well, in Python, the order of the keyword arguments doesn't matter. So if you call foo with B equals two, A equals one, it's the same thing. And one of the cool things about Semgrep is that because it's not truly generic, it still has language-specific knowledge. Semgrep will actually

find both variations of those keyword arguments for you if you're using Python as a language. But in order to do that, that means that we can't just do this trivial translation from the parsed Python code into the super generic AST structure.

[00:36:48] JM: So do you think this will be the kind of company where the challenges are more around building the right product and getting the documentation correct and getting the go-to-market motion correct? Those are going to be the more challenging aspects rather than the technical challenges.

[00:37:11] IE: I guess I would say the technical challenges are really considerable. We just have an incredible team that has exceeded my expectations and beliefs about what's possible. So like the technical challenge that we're working on right now is auto fix where we're interested in saying, “Hey, imagine you found this function with these arguments. We want to use this declarative syntax to transform it into something totally different.” But if you have like comments in the middle of the code or like stuff like that, we don't want to just blow away like the structure around the code.

So getting that right uh is even more tricky because it involves kind of like a rendering back from that tree structure into the code. Since like the team who's working on that is so talented, frankly, yeah, I think my big questions are around the go-to-market. And really the fundamental question there is, “Hey, why have you know these other security tools, like the ones that are really expensive with all these advanced capabilities, why have developers never really picked them up?” When I was an undergrad graduating from MIT, I’d never heard of tools like Coverity. Why was that?” And like having now talked to a lot of people there, they never really kind of tried a bottoms-up structure where they were targeting, “Hey, like what if we could get the individual developer to like be excited about this and try it?” They were always like, “Hey, top down sales. Sell to the CTO. Sell to the chief security officer.”

And so for us to really succeed we have to build a tool that emphasizes putting the developer first. Developer first security is the thing that I think if we can pull that off and actually like build something on top of this open source project that gives developers a great experience, I don't

think there are a lot of tools out there in the security space unfortunately that are really thinking about, “Hey, how does it feel as a developer to get a thousand warnings of you're doing things wrong from a security perspective?” So that's where building that product on top of the technology is just really crucial.

[00:39:27] JM: Well, I’d love to get your perspective for what you're focused on right now within the company and how you see the next 6, 12, 18 months going.

[00:39:40] IE: So we just shipped this fixed rate feature and we're starting to look at the metrics on it. And what we're realizing is that we know a lot more about the like massive database of rules than anybody else does and we need to do a better job of sharing that knowledge and giving people rules that are really applicable, right?

So if you're writing a like Python web app with a JavaScript frontend, we shouldn't run PHP rules over your code base. That's obvious. But we should also suggest rules that are very specific. So like, “Hey, like it's not just Python. Are you using Django? Are you using Flask?” Like what are the security pitfalls of using one of those frameworks? Where are the knobs that somebody can go in and like if somebody makes this call and they're like, “Hey, yeah, like don't check this thing.” All of a sudden the security properties of that framework are super degraded.

And so the challenge for us and like what we're going to be focusing on is basically coming in and building on top of this amazing engine and this amazing collection of rules that are available in that GitHub repository. Intelligence that basically says, “Hey, because you are running Semgrep and you're checking for X, Y and Z, you're probably not going to have or you're just much less likely to have security problems that are related to this area. So maybe that's like XSS.” But like you're not doing anything to check for – Like you're using JWT, JSON web tokens as your authentication mechanism and you're not checking that people aren't using none as the cryptographic algorithm for verification, which would be like an obvious gap and you should know if somebody does that.

So building the kind of intelligence to say, “Hey, like what are the things I should be checking for?” And checking for them in a sensible way is basically the next step, I would say, for the hosted cloud side of the application.

[00:41:44] JM: Well, as we begin to wrap up, I’d love to get your perspective on the wider market. You spent some time like looking at different business ideas before you started r2c. And I just like to get your thoughts on the future or the present and where you see is the biggest opportunities in the world of software.

[00:42:07] IE: Well, I’ve got good news if you were a developer listening to this podcast, because I think that many, many opportunities have to do with making things better for the developer to avoid having to do kind of wasteful work later on. So, to me, like there are all these products in the security space that probably I had never even heard of before I started doing research. I’ll name one of them. Like web application firewalls have a bunch of different pieces of functionality. One of the things that they typically advertise is like, “Oh!” Like, “Hey, we'll protect you from SQL injection. We’ll protect you from cross-site scripting.” And it's really the wrong place to do that kind of protection. And when you look into it they may claim to be using things like AI and ML, but actually it's quite primitive. And I think that the market is going to realize this eventually and it's going to say like, “Hey, like Google doesn't buy products like that to protect themselves against these vulnerabilities. They do it by architecting the code in a way that leaves them invulnerable to these classes of attacks.”

And so my thinking is basically that I would definitely bet on application security and uh DevSecOps to use the buzzword as areas where like there's just so much that can be gained by really sharing security knowledge with developers and asking them to participate in the process as opposed to our current setup where like it's a tail-end thing and we buy a lot of products after the fact. So because of that, my prediction is that you know many categories of security tools that companies spend millions to tens, hundreds of millions of dollars on will eventually be obviated by just developers getting better training and better tooling for how they write the application in the first place. And of course there'll be a long tail of giant C code bases that don't have those properties. But this is tool true not just of you know like static

analysis tools like Semgrep, but also of frameworks. React was explicitly designed at Facebook with one of the goals being to prevent cross-site scripting, which is the most common vulnerability classification for web app. A language like Rust was explicitly designed to just render not relevant many of the memory corruption type bugs that have plagued languages like C and C++. So I'm very optimistic about the role that developers will shape in the landscape.

[00:44:50] JM: Awesome. Well, that sounds like a pretty good place to wrap up. Do you have anything else to add, Isaac?

[00:44:56] IE: I think the last thing would just be you know like, hey, this is a FOSS tool. We're very active on GitHub and we have a Slack. We would absolutely love feedback. You can go to semgrep.dev or just Google it. You don't have to install anything. There's an interactive playground that you can just use to kind of like see what queries match what. And if you're like, "Huh! I feel like that should match that thing." Send us a bug report. We'd love the feedback.

[00:45:25] JM: Cool. Well, thanks again. I appreciate you coming on.

[00:45:28] IE: Likewise, Jeff. Thanks so much for hosting me.

[END]