EPISODE 1455

[INTRODUCTION]

[00:00:00] JM: The JavaScript supply chain includes numerous vulnerabilities due to its expansive nature, and the long dependency chains. Socket is a new security company that can protect your most critical applications from supply chain attacks. They're taking an entirely new approach to one of the hardest problems in security, and a stagnant part of the industry that has historically been obsessed with just reporting on known vulnerabilities. Feross Aboukhadijeh is the founder and CEO of Socket security, and he joins the show to talk about Socket's approach to detecting and blocking supply chain attacks.

We are looking for a salesperson. If you're interested in working in podcast advertising sales, shoot me an email, jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:00:42] JM: Feross, welcome back to the show.

[00:00:44] FA: Yeah, glad to be here.

[00:00:47] JM: You've been on previously, I think just one time, talking about – it was like a streaming video player basically like a – was it Web Torrent, is that the name?

[00:00:58] FA: Yeah, that's right.

[00:01:00] JM: So, you've had a long string of different open-source projects that you've worked on, and it seems like commercializability has not really been your focus in the past, but now you're actually working on an open-source project that does have commercializability. Can you just describe your creative process when it comes to what software projects you pick?

[00:01:25] FA: That's something I don't really understand myself, sometimes. I mean, I generally am attracted to projects that surprise people in some way, or make them surprised at what computers can do and that's kind of why I did Web Torrent. That idea that you could torrent in a browser and have a website, do direct peer-to-peer connections, is something that I think a lot of people didn't know, was possible. Of course, that was enabled by WebRTC. And that was a pretty new technology when I started Web Torrent. And then, I think there's, there's something kind of amazing about computers and what you can do with them. So, whenever I see kind of a way to like, do something surprising or delightful, I'm attracted to that. It's something that – I call it mad science. It's kind of like stuff that people don't think you should be able to do, that you can do and you can kind of blow people's minds. That's usually what I'm attracted to. I mean, I think it's fun to just surprise people.

[00:02:24] JM: Given your level of creativity and your adventurousness, I'm kind of surprised you haven't focused completely on crypto or Web 3.0, or whatever you would call it. Do you have any aversion there? Or you just like to focus on stuff that's closer to enterprise? Or what keeps you from going down that path?

[00:02:48] FA: Crypto is definitely interesting. I was pretty excited about Bitcoin when I first heard about it in, I don't know what the year was, back when the price was like 10 bucks, I think. I read the paper, I thought it was amazing. I thought it was really interesting. I didn't buy any at the time because I was like a student, and that's obviously one of my everlasting regrets. But yeah, then I started working on Web Torrent as a way to learn about peer-to-peer systems.

So I was really interested in the idea of like a peer-to-peer network. It's so cool that the idea that all the nodes come together to accomplish some tasks they couldn't do alone, and that you don't need to trust these people. They're just random computers out there. But you can work together with them to do something like downloading content, or in the case of Bitcoin, maintaining a shared ledger. It was definitely interesting to me. I went down the torrent route, because I thought BitTorrent was a really interesting protocol, and I saw the way to connect it to WebRTC. And it was like this thing that I didn't think anyone else quite saw it.

I had worked on this project before, where I'd been exposed to WebRTC. So it seemed like I could put the two together pretty easily and come up with something useful, and so that's kind of why I was attracted to it. And then kind of through the mid-2010s, there was all this attention that got attracted to crypto, and there was the ICO craze. And that was when you started to see a lot more scammers coming into the space and kind of capitalizing on the hype around crypto. During that time, I kind of felt, I don't know, I guess I was a little turned off by all that. I would say I made somewhat of a mistake in terms of writing – I kind of wrote off the whole thing somewhat and sort of said, “Why are people paying all this attention to this crypto stuff?” You're seeing crypto jammed into places that didn't make sense. People were just tacking blockchains onto systems that really didn't need blockchains, just so that they could get the hype of saying that their project was tangentially related to blockchains into crypto.

So, I think that turned off a lot of, not just me, but a lot of like other technologists who were like, “Why are we doing buzzword driven development? It doesn't make that much sense.” I became a little bit of a skeptic at that time. But the thing that I think was kind of the mistake was just because 90% of people working in a space are there to get rich quick, it doesn't mean that the other 10% aren't doing really good work and actually doing like fundamental computer science research that is actually creating new abstractions and new primitives that you can use to actually do interesting stuff.

That was still going on while all these scammers were they're doing their scamming, right? So, it was just like, they were drawing up all the attention. I was there. I was sitting there working on Web Torrent, which is like has nothing to do with blockchain. And there's this pressure like, well, if I just said that I added a blockchain into the web torrent or if I kind of made a token, maybe I could capitalize on this, and I just refuse to do that. As a purist like, no, there's no need for a blockchain here. It actually would make things worse for this particular problem. Yeah, I just sort of sat on the sidelines and just let that whole thing happen.

But I definitely, I'm not like a hater or anything like that. There was that period where there's just a lot of scamming going on, I guess, and it's turned off a lot of people and myself included. But back in like 2018, I went back to get my Master's Degree and at Stanford, they had this class called Cryptocurrencies and Blockchains. I actually took it and it's taught – Dan Boneh and David Mazieres, two really awesome computer science professors, and got me excited again, about crypto and like, what you can do with it. There's a lot of really cool stuff going on. So, I know probably a lot of your viewers are in the skeptic camp. And I would say like, if you feel skeptical about some of the stuff you see, maybe just go look at the actual kind of computer science end of things and the primitives that people are inventing and kind of what you can do with this stuff. It's just cool computer science, honestly. That was kind of what made me excited again about it.

So yeah, I kind of went through a journey there. Started excited, got uninterested and then got excited again, if that makes sense. I don't want to go into this, into the space at this time. I mean, I think the stuff I'm working on with Socket is definitely more interesting right now. But maybe in the future, I will do that.

[00:06:52] JM: Yes, well, let's get into that. So, you have been developing for the web for a really long time. And you know JavaScript as well as anybody. JavaScript lends itself to all kinds of security issues, and I think that's just endemic in the way the web works and the way JavaScript works. So, it's hard to avoid. When I think about a JavaScript supply chain, I think about callback hell, in a sense, and like dependency trees, just think about left pad, contaminating so many packages that were downstream from it, or situations that were downstream from it. So, dependency management is so important in JavaScript. Can you describe how you see the JavaScript supply chain?

[00:07:49] FA: Yeah, so the JavaScript community is incredibly creative, and generative. And part of that comes from the permissionless nature of NPM, where anyone can publish a package. And part of that just comes from JavaScript being a language that doesn't come with the batteries included. So you get all this like really creative, and this interesting stuff that the community comes up with. I think, part of the reason why JavaScript is the largest programming community, not to mention, obviously, that it's the only language you can use in web browsers. But it is a source of amazing creativity and there's a ton of stuff out there that you can use. It's awesome. I mean, it's the reason, open source and particularly, the NPM registry is the reason why you can build a very complicated app in a very short period of time with a very small team of people, in a way that you couldn't do in previous eras, more broadly, like sharing code and abstraction and encapsulation, and reusability, and all these things are why you don't need to be an expert in parsing some random format or all the different browser differences or how does the BitTorrent protocol work. You can just call a library function to do that stuff, and you can get really far with that.

That's why 90% of the lines of code in a given application are coming from open source and the last 10% at the top, that kind of icing on the cake comes from the actual code that's unique to your application. But really, we're all sitting or standing on the shoulders of giants, and relying on all this code under the surface that powers everything. I mean, not to mention in JavaScript, but even just think about the web servers we're using, right? Like NGINX, and Node,js, and all this stuff, all this complexities there under the hood, all the way down to Linux, and our apps are built on top of all this stuff. So, it's awesome. It's powerful. But then, with it comes this like downside of complexity, and also risk. When you rely on open source code, you're trusting that the maintainer of that code made sure that the code is safe and secure, does what it says on the box, and that going forward into the future that they haven't lost control of their account, that they picked a good password, that they're keeping track of their credentials, and they're not letting other people get access to these packages, so that they can be backdoored or have now been added to them or what have you.

That's kind of the downside of the dark side is now we have these dependency trees that have grown and grown to the point where Hello World application in React can be hundreds or sometimes even thousands of dependencies just to get to a Hello World on the screen, which is ridiculous in a way. That's why you get this sort of meme, I guess, of people making fun of JavaScript developers forgetting how to code and needing to install a package to do left pad, which is like one or two lines of code.

Now, I don't think that's entirely fair. Obviously, I think that that these packages exist for a reason, and a lot of the people who are complaining about left pad, for instance, wrote their own left pad implementations and wrote them in like comments on these news articles and on Hacker News, and their implementations actually had bugs. So even something as simple as left pad is actually was it was hard for people to get perfectly right in all the different cases.

Anyway, just the kind of bigger picture that I'm seeing is that nowadays, in 2022, if you look at what the sort of the ecosystem, we're seeing nearly weekly attacks against the open-source software supply chain, mainly coming from hijacked packages. We're seeing people hijacking packages, because maintainers are reusing passwords. We're seeing people get added as maintainers by just emailing people and saying, “Hey, I'd love to be a maintainer, you're not maintaining this anymore, can I help you?” And then people get access that way. There's also an instance just this last January, literally less than a month ago, of somebody, maintainer sabotaging their own code, which is that was a very shocking incident, where somebody basically just decided to add, basically, kind of a denial of service into their own project with a wild true loop that would cause an infinite loop and would print out all these random characters. That was an attack against colors.js and faker. The point is that this is a problem that's kind of getting worse over time and it's accelerating, it seems like. That's kind of the landscape that I see. Hopefully that answers your question.

[00:12:02] JM: Yeah, it does. And the vulnerabilities in JavaScript, there's a lot of ways you could catch them. I mean, one that comes to mind is Snyk. You could just use Snyk to assess your repositories. You've built socket.dev, which is a JavaScript supply chain management tool, what's novel about Socket? What does it help protect you from in a way that is not done by previous technologies?

[00:12:36] FA: The most important difference is that Snyk and other tools in this category are what we call like a vulnerability scanner. So, the primary way they work is they look up what packages you're using, and they compare that to a database of known vulnerabilities. What is a known vulnerability? It's basically, maintainers make mistakes, introduce security bugs in their packages, usually, almost always, this is an accident. So, the way that something is written might have some security issue, and a security researcher discovers this, and they report it to the maintainer. Once it's fixed, then something called a CVE is issued. So, a CVE is just kind of an identifier. It's a number that identifies this particular vulnerability. And it goes into a database that's actually maintained by the federal government in the US.

It's called the NVD, or the National Vulnerability Database. And that's basically what all these tools are doing is they're just looking at what version of a package are you using in your project, and then is that version known to be vulnerable, by comparing it to this database that says which versions are vulnerable? And then your solution here is if something is vulnerable, then usually the report will tell you like, you can update to such and such version, and then you won't have the vulnerability anymore.

These vulnerabilities are – it's obviously bad to have these in production and to have these in your code, and so you can use a tool like Snyk to kind of keep up to date and kind of update your dependencies. The thing, though, is that these are kind of noisy, and partially this is because companies like them have an incentive to kind of inflate the number of these vulnerabilities that they find. So you get a lot of stuff in there, that's relatively low impact.

My biggest pet peeve is regular expression, Denial of Service, is this vulnerability type that is basically, when you have a regular expression that's slow to run on particular inputs, and when you find one of these, sure, it could be bad to have that in production. But in general, you'll see a lot of these things in development tools like dev dependencies, which something like Webpack or something like Browserify, which is a development tool that's running on source code that you control, to basically build your project and a vulnerability which involves a bad input causing a regular expression to take a really long, long, long time to run isn't going to be true triggered by you compiling your own JavaScript code, it's because it's an input that you control.

So that's just totally noise, right? It's completely pointless to report on that. So, it's a little hard for these tools sometimes to like distinguish, like whether something is a real issue or not, and so you end up with like a case where it feels to most people like 90%, 95% of the time, they're just updating these things to make the alerts go away, and they're not actually making things meaningfully more secure. There was actually a meme about this in the community recently, where the NPM audit command, which uses a very similar process, it just looks up in a database to tell you if something has known vulnerabilities was being very, very noisy and kind of just spamming everybody with these warnings that really have nothing to do with actually making your app more secure.

Anyway, that's kind of what the existing tools focus on, and I don't want to disparage them or anything like that. Because I do think you don't want to – you generally want to have known vulnerabilities in your code. It's sometimes okay to ship these to production, if they're low impact, or they're just really hard to trigger or you don't have time to fix them right now, like a regular expression, denial of service would be totally fine to kind of address when you have the time to get to it. But I want to contrast known vulnerabilities with malware, because that's completely different.

So malware is when a bad guy intentionally introduces bad code into a project or into a package. And it's almost always introduced by an attacker. Although, like I said, we've seen an example like last month where the maintainer themselves introduced the malware. But in general, this is something that is really nasty code that you never would want to ship to production ever. In fact, you never would even want to run it on your own computer. Because if you do, then your computer is compromised. I'll give you just a very concrete example of malware.

In October, and in November of last year, in 2021, there were three packages compromised by an attacker. What happened was, this package called – I'll just focus on UAParserl.js as the main one because this package is a user agent parser. So, it takes a user agent string from a browser, and it parses it and tells you like what browser you're dealing with here. It's a really simple kind of idea of what it's supposed to do. It's a pure function. It doesn't do too much stuff. But this package is dependent upon by a lot of other people.

It has seven million downloads every week and it is dependent on by almost three million GitHub repositories. So almost three million separate GitHub projects are using this, even stuff like React Native depends on this. If you're building a native app for iOS or Android, it has a dependency on this UAParser.js project. So it's a really critical package. What happened was in on October 5th, 2021, there was a post on a notorious Russian hacking forum where someone was offering to sell a NPM account to someone who, whoever wanted to buy it. So, the post said, "I'm selling a development account for NPM.js for a package that has more than seven million installations every week, and a lot of dependency depending on it. There's no 2FGA on the account, I have the login and the password, and you can log in and then change the email address and take over the package." And they were offering it for $20,000.

Two weeks after that post, UAParser.js was – three malicious versions of it were published on October 22. If you look at what actually is in those packages, I'll tell you what happened is actually you can crack open the code and look at what exactly it did. The attacker published new versions that added a install script, which is for those who don't know, it's basically a way that you can tell NPM that whenever a package is installed on a developer's computer, that a special shell command should be run automatically. And usually, this is used to compile like a native dependency, some C code that is required for the dependency to work. But it's also the favorite technique by attackers.

So, 60% of malware on NPM uses install scripts as an attack mechanism. So this particular install script would execute, and it happened to download a Monero miner, which is a cryptocurrency that would basically just start mining this cryptocurrency on your computer for the attacker. And then separately, it also did an additional attack on Windows computers, where it would download a DLL file, and it would register it on the machine, and then that would steal all the passwords in about 100 different windows programs as well as all the passwords in the Windows Credential Manager, and it would send that off to the attackers IP address.

This is a crazy thing. I mean, it's like you're just using some open-source software, and then suddenly, you install it one day, or you update to a new version one day, and then now it's doing all this extra stuff, right? It's stealing your passwords and sending them off to the attacker. So that's malware, and that's very different than this whole known vulnerability thing and hopefully that distinction is clear. This is something where – this wasn't in any database, right? This was like a new version was published and for like, I think for this particular one, it was caught relatively quickly, it was on NPM for I think, four or five hours before it was taken down. Because this was a very noisy attack with a cryptocurrency miner using 100% CPU. That's a pretty obvious thing. Okay, why is my computer slow? You know why? What is using 100% CPU, this was not a very sneaky attack. So that's why it was caught really, relatively fast.

But during that period of time, during that four or five hours when it was available on NPM, anyone who installed this package or anything that uses this package, and remember, I emphasize it had seven million downloads per week. So, this was a very popular package, anyone who installed it was completely compromised and this wasn't in a database. This wasn't in a known vulnerability database, because it was new malware. This is kind of the problem with these existing tools and why I felt like we needed a different approach to detect this stuff. Hopefully that makes sense.

[00:20:53] JM: Yeah, that's a great outline. I'd like to get a sense for where the scanning should occur, like should it happen in continuous delivery processes, or just ad hoc security scans? Give me a sense of the usage of Socket.

[00:21:11] FA: So we think that before any package is installed, you should check to see what Socket knows about it. What Socket does is we're watching NPM, and eventually, we're going to expand to other programming languages, Python, and so on, and so forth, that also have the same problem, but just at a smaller scale. Because in NPM is just, all the problems come to JavaScript first, because of the scale. But yeah, we're scanning every package, and we're doing that sort of on our end, so we've built a pipeline that can watch NPM publishes in real time, and we run a suite of around 70 checks that we've written.

You can think of them kind of like Lint rules. So, we're looking for specific things in the packages, and most of them are based on the actual code of the package, although some of the things we look for are actually around the metadata of the package. So, if we see like a new maintainer was added recently, that's an interesting data point, or if we see that the package has a lot of GitHub issues that are never being addressed, then that also kind of factors in a little bit too, sort of our understanding of the quality of the package. But primarily, we're looking at the code, and we're looking for specific things that the code does.

We know if a package is going to read your files on your file system, where if it's going to run a shell command, or if it's going to talk to the network, or if it's going to run eval and evaluate a string as code. We know if a package has obfuscated code. So, we can detect that there's a blob of code that has really short variable names, it has a lot of entropy, and it's been to the bottom of a random file, right? That was actually an attack that happened to a package called Event Stream, where a big blob of obfuscated code was added to the bottom of one of the files, and that one took people around a week to find and it was – that's actually an interesting story. If we have time, I could tell the story of Event Stream and that attack.

But the point is, basically, we're scanning all this stuff. And we can find these kinds of six signals that indicate that something that this package is doing certain things, and we can tag the package with that information. And then what you really want to know is like, okay, we're using a package at our company, let's say, and we've been using it for a year. And for that whole year, it's behaved in a certain way, right? It's never talked to the network, it's never read files, it's just a function that you call, maybe it computes something for you, right? And then suddenly, there's a new version that comes out today. Now, the package is running a shell script, as soon as you install it, and now it's talking to a bunch of servers, and it's also reading some files. Also, let's say, it's reading your environment variables, too. So, it's looking for tokens and your environment variables.

Now, by itself, looking at environment variables isn't necessarily a sign of malware. I mean, a lot of good packages, normal packages do that as part of their normal functioning. But what is suspicious here is that this package never did that for years. And then suddenly, a new version came out that is suddenly doing that. So, it's worth a second look, at the very least. If someone on your team is trying to update to this new version, or if you're using a bot that tries to update your dependencies for you, I'm as guilty of this as anyone. But usually, we have Dependabot installed on our repo and we get these PRs all the time, from dependable that's trying to update us. I just look at it, and I usually just say, “Okay, the tests pass. The changelog says it's a bug fix or something.” And then I just say, “Okay, merge.”

Because there's so many of these, and you want to kind of stay up to date as much as possible. And so that's a perfect way for a supply chain attack to happen. What socket can do is it can come into a pull request like that and tell you, “Hey, this package’s behavior has changed significantly in this version. Here's what it now does.” And you can look at that and you can then decide if you want to update or if it merits a little bit further investigation and will link you directly to the lines of code where it does the things that we report on. So, we'll tell you, “It's talking to the network. Here's the line where it does.” And you can click that and you can then look and see why is it now talking to a server? What exactly is it doing?

So then, you can make a much more informed decision about whether to take this update. And of course, we could do the same thing with new packages, too. If you're installing a new package, you also would want to know what exactly does this do, and I think of it almost kind of like a smartphone app where you when you install a smartphone app. It doesn't just get access to your camera and to your contacts and to your microphone right away. It has to ask permission. We think it'll work the same way with dependencies where when you install the dependency, you should be told upfront, what behaviors it's going to use and what things about it are noteworthy. And then you should decide if you want to continue and proceed. That's kind of how we think about it.

[00:25:46] JM: Can you give me an example of a vulnerability, that Socket might discover on my infrastructure, and walk me through how socket actually is engineered to analyze the supply chain? I just like to dive a little bit deeper into the engineering. We did a show with Synk, recently, actually, about how they build their dependency data structures, their software supply chain dependency infrastructure. It was pretty interesting, and I just love to hear about your story and how it compares.

[00:26:26] FA: Yeah, so the biggest thing that we do that is probably different from what they do, although, I mean, maybe I should go listen to their episode and hear the inside scoop about how they do it. We need to analyze the contents of every package to figure out its behavior. Because that's how Socket works is we do look up the CVEs in the database like they do and that’s a pretty simple thing to do. But beyond that, we actually want to know what the code is doing. So, in order to do that, we actually have to look at the code in every package. Right now, we just do static analysis, which means that we don't actually execute the code, we just treat the code that we're analyzing as like an input to our own analyses’ procedures, and then we – it’s ESLint, you can kind of think of it like a linter. We look at the code and we also look at the metadata around the package and we analyze it. And then we look for, like I said, about a series of 70 issues. You can find the list on our website, if you go to socket.dev and the header, there's a link called issues, and you can see exactly which things we're looking for there.

We do that for all of the packages on NPM. So, the way we've architected it is because that's a lot of packages, there's 1.8 million of them on NPM, not to mention every version of every package. There's something like, I think, close to 10 million versions, if you count every version of every package that are out there, and so that's a lot of code to analyze. So, we've actually structured our data processing pipeline, so that we can lazily calculate. We can lazily analyze a package when somebody requests it, if we need to do that. But what we're doing is we're kind of going and we've preprocessed a bunch of the most popular packages, really, every package over a certain popularity threshold. I think, right now, we've processed everything that has more than a thousand weekly downloads, which is probably all the things that most people are using.

And then beyond that, whenever a new package is published, we also follow the NPM feed. So, we know exactly instantly when a package is published, and we can analyze it right away. And then, all this goes into our database, basically. All this information is publicly – so we put it all on our website right away, so you can look up any package, just type it into the search box on Socket, and you can see what we found for that package, and all that data is just there. It's open for anyone to look at and to use.

And then kind of the next thing is like, okay, then how do you make the data useful? Because I mean, you can go to Socket before you choose to use a dependency and get an idea about like, what it is going to do, and what red flags you might want to watch out for. But the thing is, if you're on a team of people, not everyone on the team is going to remember to do that. So, what you really want to do is kind of in the security industry, they call it shift left. It's basically this idea that you want to kind of push as much of this security work as early into the development process as possible, because it's lower cost. If you do this kind of stuff earlier, than if you try to catch it later. One way to do that is to add our GitHub bot to your organization, or to your repo.

If you do that, then we can watch the pull requests for any changes to your package.json file. And whenever a dependency is added, or a version is modified in that file, we can look up the change and see whether something has changed that is security relevant to you and notify you about it. A concrete example of this would be, there's this package called angular-calendar. So, it's called angular-calendar, and it's a calendar widget that just sort of shows a calendar on a web page, like a web component of a calendar. This package, one version out, suddenly that added a dependency. And now the package actually, when you install it, it runs a shell command. It reads files, it reads your environment verbals, and it talks to the network, all at install time. This is a real example.

And this angular-calendar package, it has like millions of downloads. If you were using this in your app, and you had Socket installed, then you would know before updating to this new version, Socket would tell you that this package’s behaviors have changed in a significant way, and you'd be able to click around and see exactly why is it doing these new things.

In this particular case, for this package, angular-calendar, I would say, it wasn't outright malicious, it was more, I would say it's more of a gray area, why they were doing this. There's kind of dependency they added a called Scarf, which actually has a great mission, which is to help maintainers learn about kind of who is using their open-source code so that they can find out which companies are using it, so they could ask for funding and sponsorship, that kind of thing. This maintainer of angular-calendar added Scarf to the project, which means that now it's giving them this analytics, this telemetry basically of who is using their open-source project.

I understand the kind of intention behind this, but I think a lot of companies have policies that don't allow this kind of telemetry to be gathered from their systems. So, I think this is the kind of thing where you'd want to know, in advance that this behavior has changed significantly in this way. And so, Socket can help – in this example, Socket would have told you that this package is now doing this thing, it's now gathering analytics about how it's being used. And then you can make an informed decision on your own. Maybe it's fine, maybe you don't care, maybe you do care. So, we can surface that to you.
That's how we intervene in CI.

But then also, we're interested in making a CLI as well, which we're working on right now. That part is not ready yet. But we want to have a CLI that lets you actually catch this even earlier. So, at the time when you're actually installing the package locally, when you run NPM install. That's something we're working on right now, that will be ready soon.

[00:32:01] JM: When you think about the full functionality suite that you could build, on top of the basic vulnerability search, what does that look like? What are the other areas of JavaScript security, that you would see yourself getting into?

[00:32:22] FA: One thing that excites me right now is this idea of reproducible builds. One thing we're seeing a lot of right now in the JavaScript world is adoption of TypeScript, and one of the things that makes difficult is – well, maybe let me back up for a minute and tell people about a little bit of background about one particular technique that a lot of malware likes to use. One of the worst things about NPM right now is that when you go to the NPM website and look up a package, you can't actually see the code that you're about to install at all. I mean, it's open source, but it's not actually open and easy to read. There's a tab that they have on their website that says “Explore”, and if you click it, it just says, “Coming Soon”, or it says like an error message. It just never worked correctly. This tab has been there for years, and it just never worked correctly.

You basically resort to clicking the GitHub link, and then going to GitHub and looking at the code there to get an idea. And again, like most people don't even bother to do that. But if you're trying to be security conscious, you're trying to carefully select your dependencies, then that is something that you might do. The problem with that is there's no guarantee that the code that's on GitHub is the same as the code that's on NPM. A lot of malware will sneak into an NPM package, and the attacker will make sure that the code that's on GitHub doesn't reflect that. It doesn't have the malware. So, if you're just relying on going to the GitHub page, and looking there, you're going to miss out on this malware.

So, one of the things we'd like to be able to do is to sort of say, “Well, whenever we see that the code on GitHub doesn't match the code that's on NPM. That's a pretty big red flag, that something fishy is going on.” But what makes this hard is as people adopt things like TypeScript, and honestly, even like, there's things like CoffeeScript that came before that, that had the same problem. It's that you have a source folder, written in TypeScript. But then, when a maintainer actually publishes this package to NPM, they have to first build the Typescript and produce a JavaScript artifact. And then that's what actually gets published to NPM. That means that basically, all TypeScript packages are going to have code that's different on GitHub than it is on NPM. So, it makes that signal a lot less useful to us, because we wouldn't want to flag when those are different if it's going to flag every single TypeScript project.

Anyway, one of the things that I'm really excited about pushing for, that I think would be great for security and would also just be great for just kind of like just software quality in general is a reproducible build. For those who don't know that idea of a reproducible build is just that you should be able to build a project and get the same artifact that comes out regardless of who builds it. And that way, if people are relying on this compiled artifact, whether it's a binary or whether it's a bit of compiled TypeScript code, they don't need to trust the person who compiled it. They don't need to trust that this person, this machine was clean, and it didn't have a backdoor on it. They don't need to trust that this person didn't modify the source first, before compiling it, so that they could sneak in something.

With reproducible build, anyone can go and take the source code, build it, and get the exact same bit for bit output as the maintainer would have gotten. And so, they can confirm that nothing sneaky was added to the result. If we could have – what we want to do was we want to get to a place where we can actually take a given package, we can run NPM, install, and then NPM build, and then look at the resulting output and compare that to what's on NPM. If it matches, then we can say, “Alright, even though the code is different on GitHub here, we have a really high degree of certainty that the source there, that the human readable source code that anyone can look at is actually reflecting what is published in NPM." That's just great for reliability and for just kind of cleanliness, in terms of how we write and how we consume software.

That's something I'd love to help push and I think that's like one thing where we can actually, we can give packages that do this, a nice little badge that says that they have reproducible builds. But yeah, there's so many other directions we could go with this. I mean, we have this pipeline that where we can analyze code, all code, all open-source code. So, there are so many more things that we want to look for, and we just keep thinking of more and more ideas, and we're making this pipeline more and more like enriched with different data, as we think of new things to look for.

One other one that we're looking for that's really, really powerful is detecting typo squatting. I don't know how much – yeah, I mean, maybe I should find that for people. But basically, typo squatting is when someone registers a package with his name that's very similar to a popular package. And they hope that some percentage of people are going to make a typo when they're installing it. And because of how NPM works, where it will automatically run the install scripts in the package. You can basically make a typo, like a one or two-letter typo in a package, and you'll end up installing something that has like, let's say a hundred downloads a month. So, no one uses this thing, right? So, you were about to install something that's extremely popular, you make a one letter mistake, and suddenly you're executing code from some random package that no one has ever really even looked at before, and that's really bad.

So that's an example of another thing where we can detect that really easily. I don't know why the existing tooling doesn't do it. It's something that – like the number one supply chain attack vector right now that we're seeing, it's just pervasive, and we found an instance of a very, very popular library using a typo squat, and we had them fix it, but they weren't aware that they had this in their dependency list. It was just shocking how common this is. So that's something that we can detect as well, and we can tell you on the GitHub pull request that the package that you're installing appears to be a typo and just get you to double check that you actually installed the right one.

There are so many things like this that you can do if you really start to think about like, what are common sense things that we should be doing that no one is doing for whatever reason? We can just do those things for people, and that’s what the Socket project is trying to do.

[00:38:36] JM: Do you take any inspiration from any of the other security companies that you've seen built up over the years?

[00:38:45] FA: The thing that I think is the most important when it comes to tools like this is that the signal to noise ratio needs to be really high. I think efforts like NPM audit, which started out off with the best of intentions, which is basically to give everybody access to this vulnerability database and let them know when they're installing stuff that has vulnerabilities. It is a great thing. I'm glad that this feature exists. But it's kind of become so noisy now that a lot of people just ignore it. So, I think that's something that's really been on all of our minds is how do we make the stuff that we're telling people actionable, and keep the amount of these things low so that people don't start ignoring it? It's hard because the incentive for security company is to make their tool catch as many, to say that they can catch as many things as possible. So, there's this constant incentive from companies in the space to basically have as many of these known vulnerabilities in their database as possible so that they can say they have the biggest database and they catch the most things.

But as they pushed to do that, you end up with like lower and lower impact vulnerabilities being added to the database. That means that in fact, whenever you see an alert, like more often than not, it's actually fine to ignore it, because it's not like a real problem. It's in some function that's not actually called by you, or it's in a developer dependency, so it's never actually going to run on an untrusted input from a user, or it's just like, pretty low impact, but it's rated wrong. So, it's rated as severe or high when it's actually a low impact vulnerability.

That's something that is a tough balance to strike. But I think with the stuff we're looking for, we're always asking ourselves, is this worth bothering people about? Is it worth telling users about it? So, I think, if you take the typo squatting example that I mentioned a minute ago, that's something where, “Yes, I guess somebody might really want to install a package that has a hundred downloads a week, maybe that's really what they intended to install. Maybe they wrote the package themselves, and so, they know that it's safe, and they're just the first person to install it." Maybe that happens sometimes. But in the vast, vast, vast majority of the time, a package like that, which is one or two letters off from a package with millions of downloads, the vast majority of time that's going to be a typo, we think it's worth surfacing that in a comment, at least, to make sure people take a second look at that. If we get it wrong, sometimes it's not the end of the world. It's just a GitHub comment that you can ignore.

That's like an example of like one where I think it's a really high signal to noise, and so it's a no brainer. I just think that's probably the number one thing that I think we take from other tools is it's important to make this stuff high signal to noise.

[00:41:40] JM: Well, as we begin to draw to a close, is there anything you'd like to add about Socket that we haven't covered?

[00:41:47] FA: Well, I guess I'll mention one kind of cool thing, one cool feature that we have on our website, in case people are interested. There's a page, if you go to our footer, if you go to socket.dev and you go to the footer, there's a link that says removed packages, and it's pretty interesting for people who've been listening so far, and they're kind of interested in what we've been talking about. And they're curious to see what actual malware looks like in these packages, this is a great resource.

So, what we're doing is, like I mentioned, we're kind of watching NPM, and anytime a package is published, we're automatically kind of downloading it and running our analysis on it. One of the kind of cool side effects of that is, obviously, when we find malware, we tell NPM about it, and we try to get it taken down. But also, when others find things and report it to NPM, it also gets – it could get taken down for many reasons, maybe NPM itself even finds it and takes it down. But what's interesting is that, then we can actually see that that's happened, and then we can flag that. If you go to this page on our site, we have a listing of all the packages that have been removed and you can click and see exactly like what code was in those packages. It's really informative and interesting to see kind of what malware is out there and what it does.

If you click around, you'll find a whole bunch of interesting stuff on there. One example is, there's a lot of packages that have names that have company names in the package name, and you'll see things like yahoo-react-input, or things like you know, wix-chat-backend. So, it's like a company name, dash, some component in their system. And you can click on that and see exactly what code is in there. But like the actual attack that's happened here, that's going on, and why this package was taken down is because that was a dependency confusion attack, which is an attack where basically, a company has an internal NPM registry that they use to publish private packages that they don't want to share with everybody. And what can happen is, if an attacker knows the name of one of those packages, they can go to the public NPM registry and publish a package with the same name. And then if the internal tools in the company don't install packages correctly, and they prioritize the public version, instead of the private version, then that's how an attacker can actually get code into the supply chain of his company. Basically, they confuse the tools and the tools install the public version instead of the private version of the package.

If you look at this remove packages list on socket.dev, you'll see just a ton of this kind of stuff where all these companies, these attackers are trying to basically trick companies into installing their code, and it's just pretty wild to look at. It's really interesting to see kind of the attack code and what it does and like, what exactly are they doing. You'll find some stuff where they're just clearly stealing the environment variables and sending them to some IP address. But you'll also find stuff where the entire file is just completely obfuscated, and it's a bunch of like hex numbers in an array, that gets eval’d or something like that. It's really – there’s really wild, wild stuff in there. I just think it's an interesting resource that people should check out if they're at all curious about what does an actual malware package look like.

[00:44:55] JM: Awesome. Well, Feross, it's been a real pleasure talking to you once again. It’s impressive to see you launch a company and I wish you the best of luck with it.

[00:45:04] FA: I appreciate it. That's really kind of you and it's awesome to be able to come on here and to share it with you and to be coming back to the show where I launched my first open-source project, Web Torrent, all those years ago. So, thanks for having me.

[00:45:05] JM: Cool. Thank you.

[END]