EPISODE 1397

[INTRODUCTION]

[00:00:00] KP: Everyone is becoming increasingly aware of supply chains for physical goods. Software has its own supply chain. A supply chain of open source solutions exists as does the demand for these solutions by industry. Both have surely grown, but it would be nice to have a way of measuring by how much. The State of Software Supply Chain 2021 is an annual publication now in its seventh year. It's released by Sonatype.

In this interview, I speak with their field CTO, Ilkka Turunen. We will review some of the highlights from the report including the state of open source and some particularly interesting statistics about supply chain attacks.

[INTERVIEW]

[00:00:39] KP: Ilkka, welcome to Software Engineering Daily.

[00:00:44] IT: Thanks for having me.

[00:00:44] KP: So, tell me a little bit about the recent release of the state of the software supply chain report.

[00:00:51] IT: So yeah, the state of the software supply chain report is something that we produce every year here at Sonatype. It's in its seventh annual edition now. And what it does, it looks at the supply of open source, the demand and various different aspects of the industry's usage of open source at large things like, how much are we consuming? What are the risks associated with it? How are things evolving over time? And also, it has an element of the maturity of the supply chain? How are we as developers actually consuming it? Are we doing the right things? Are we improving over time? And this year's edition is quite a large collection of various different. Really, I see it as kind of a collection of different sorts of papers of the software supply chain itself. It has some pretty interesting tidbits that come out into the industry every year.

[00:01:45] KP: And who's the report ideally for? What's your typical consumer?

[00:01:51] IT: Yeah, it's really aimed at both of developers, consuming open source, I think that just seeing the volume of data, that large that's available, is something that's really exciting for developers. It contains some pretty good tidbits around the best practices of using open source, as well as people in charge of DevSecOps initiatives in organizations, really, because it kind of gives a sense of the supply chain side of things. Really, anyone who's interested or a senior leader in software engineering at large.

[00:02:23] KP: Broadly speaking, I'm sure every company's a little different, but what is open source adoption look like in big enterprise?

[00:02:32] IT: Looks very, very prominent. In fact, when we look that just the amount of growth of open source in general, we sell almost 73% annual year over year increase. What's pretty interesting is, that increase happens across all lines of industry. It doesn't really matter if it's a small company or a large company. They pretty much adopt everything. One of the kind of findings that we saw not specifically mentioned in this year's reports, but the the one previous where we run a survey every year, to enterprises, and ask them various questions about their maturity of dealing with open source, we see that on average, about 85% to 90%, of a standard piece of software is actually now comprised of open source and other third-party components and not stuff that's been written in house.

[00:03:27] KP: So long ago, there was this idea, and maybe we blame companies like Microsoft and Oracle, a little bit of IBM services group that open source was amateur, it was not as good, it was buggy. You had to use a professional closed source commercial product. Was that still prominent in the seven years you guys have been doing the report? Or is that ancient history?

[00:03:49] IT: Well, it's really funny you mentioned Microsoft as that, because they're the open source company now. They're running one of the largest ecosystems of open source out there, which is npmjs, ultimately owned by them. I think that was the font that was used in the beginning times, right? It was a way of saying, "Hey, don't you want to have a phone number to complain to if something breaks, you're going to have to fix it yourself." But I think what that kind

of dilutes is that open source is one thing. It isn't just a piece of software, or it isn't just a line of code. It can be kind of everything, and everything in between. Open source, in the context of development today really relies on dependencies, packages that can be consumed by developers as a part of their development cycles. And that really has resulted in a place where it's allowed us as an industry to shortcut a ton of really, really small but time-consuming busy work. Things like creating certain types of convenience functions all the way up to scaffolding an entire web service.

I'd say that open source has moved on significantly from that, and you can see it in the adoption metrics. You can see in the just the growth or volume as well as in the attack scape, things like people leveraging open source, which all speaks to the fact that it's turned entirely around. I would say that, we wouldn't have the modern software industry, if it wasn't for open source kind of backing it up.

[00:05:24] KP: When you say open source is growing, that sounds intuitively correct to me. But is there a way we can quantifiably measure that?

[00:05:31] IT: Yeah, absolutely. So, one of the ways that we do that is we're actually custodian, so one of the largest open source ecosystems in the world is called Maven Central, or the central repository, as we like to call it. Then essentially, it's the default location of dependencies for any sort of popular Java dependency system. So, if you're building Gradle, or Maven, or you're doing things like Android software, chances are that the default location that your dependencies are downloaded, actually is Maven Central.

So, we can use these registries. So, in the case of Java Script, as an example, we can use npmjs as a facsimile, which, in the case of Python, we can use pypi.org. And in the case of dotnet, we can use nougat.org. And what we can do is we can measure, you know, how much open source is being downloaded, how much open source is being published, what sort of things are coming there, how many versions of open source. And when you look at that, one of the kind of interesting things that you notice is that the download volume, the average growth rate of download volume across all of those ecosystems, is about 77% year over year. For as far as I can remember, over the years, that's been the average. It's not really changed at all, which seems to imply that the pace of adoption absolutely isn't slowing down.

If anything, I think that number has ratcheted up in some ecosystems quite a lot over the last year or so. So, I think even though it's not a be all and end all, I think it's a very good similarly to kind of observe both the supply of open source as well as how are we consuming it as end users and developers. So, I think that what's eye opening to me is just the fact that that pace isn't slowing down at all, like it hasn't slowed down in the last seven years. It shows no signs of slowing down. In fact, this is speeding up if anything.

[00:07:30] KP: Yeah, I feel like if someone developed a software project today, and it didn't rely on some open source contribution, that would be the point of the project to do that as a demonstration. It's sort of ubiquitous in that way. That means we're all using software written by people we don't know. What sort of risks does that open us up to?

[00:07:48] IT: Well, that's a scary thought, because if 90% of your software is not written by yourself, there certainly are risks. And that's been one of the key focal points of our report, and one of the things that we discovered is that this supply chain of the external code has become an increasingly attractive target. Historically, when we started looking at this, really, the risk was that you'd adopt a piece of code that had some sort of security, vulnerability, some sort of bug or some sort of unknown hold that somebody else could exploit in order to hack you as they were running through a hack campaign.

For many years, that was the predominant risk that was put upon you. You put yourself at risk by forgetting that you've adopted that library, and it kind of just stayed there, you never really maintained it. And then somebody discovered it and, essentially run an exploit against it and got in. When you look at some recent events over the last five, six years ago, you can attribute many of the largest hacks or largest breaches to that sort of situation. Equifax, for example, it was an unpatched REST API.

But when we looked at more recent events, we've really started seeing a phenomenon trending towards actually pausing the well of open source itself, meaning things like legitimate open source being taken over by malicious actors, either by finding the creds to publish a malicious versions, or sometimes even just people pretending to be good Samaritans, donating code to

these open source projects, and then tricking the project maintainers once they get maintainer access and taking the projects over.

More recently, we've started seeing much more intricate campaigns such as what happened with SolarWinds, obviously, last year, where people are actually actively campaigning to take over not just the open source itself, but your own build environments and things like this. So, all of that seems to imply that there is risk, but the risk isn't necessarily that the code itself is bad. It's the channel that we built to adopt the external code, rather than anything else.

[00:10:02] KP: Yeah, I've always been a little curious about how some of these things run. I guess I've never done the research. But I've PIP installed or NPM installed plenty of things just blindly, without thinking, where is this coming from? Could there be a man in the middle attack? What's behind the scenes there? What sort of organizations exists to guide these things and manage them?

[00:10:23] IT: Yeah, I mean, that's the interesting, unseen world of the supply chain. So, there isn't one central organization. For example, PIP is, well, the PyPi, which is the Python package index is a volunteer organization called the Python Software Foundation, that actually funds to the outwork that goes into running it. Npmjs is owned by GitHub, which is ultimately owned by Microsoft, making them ironically, one of the largest open source companies in the world, just on the merit of that.

In Java, Maven Central is entirely operated by Sonatype, my employer, who are the custodians of that part of the ecosystem, as well. And I believe Microsoft also runs nougat.org, which is one of these key things. So, the ecosystems themselves, they've most often grown organically, just for the need for the build tool, to have some sort of default location, that's literally the invention story of most of these. And over time, each and every one of them in kind of their own unique ways, have grown communities around them, projects that start using it as a default publishing location, or people that don't code from there, that creates kind of a virtuous cycle. And they've kind of grown from these very humble and often very unplanned beginnings to be the sort of massive backbones are that now serve, millions upon millions upon millions of developers across the world on thousands of things. That, I think, is kind of the eye-opening moment that many of us have seen over the last few years, you know, for example, when the registry goes

down, or there's a DNS problem. All of a sudden, your builds can't get to the things. And in those cases, then, it's a lovely head scratching moment for whoever is in charge to debug it on call.

[00:12:19] KP: I'm curious, if you have any details on this from Maven. Do people squat names? When I start a company, should I go get my package name that matches my company right away?

[00:12:31] IT: Well, that's a good, good practice to do anyway. Obviously, register your namespace just like you would register your domain. Within Maven Central, specifically, in order for you to publish into Maven Central, you do have to sign up with an account. You have to prove ownership of the domain that you're trying to take over. So, for example, if you tried to publish artifacts into Google, you'd have to be able to prove that you've got the right keys to publish into that, you'd have to be able to prove your ownership of that. One of the things that Maven Central requires you to do is actually register a textfield as part of your DNS entries and prove that you have ownership of that domain that you're trying to publish under.

In other ecosystems, that's not the case. There are ecosystems that take a philosophical stance that namespace is free for all essentially, first come first served. And that's certainly bred some security situations. In fact, a form of attack that arose this year is called namespace confusion, which absolutely relies on that particular mode of attack. I'll discover your name, and I'll go and squat that name, and then publish some open source, or not so good open source underneath that name. That really is something that most organizations should be aware of.

So, as a rule of thumb, absolutely, you should be going and registering those names, regardless. In central and in some other ecosystems, we're in in safer place just because of this ownership verification. But that's, unfortunately, not a universal truth, everywhere.

[00:14:12] KP: Yeah, I appreciate that verification step. But all that confirms, is that I have control of a domain that doesn't confirm I'm not a bad actor. If I could just set up my library and then push a bunch of malware and stuff to it, how are the systems not totally polluted with spam and nonsense like that?

[00:14:31] IT: Well, the funny thing is, they are. For example, last year, kind of one of the cool things that we do in the report that we publish, there's a timeline of attacks and that timeline actually has writeups that we've done and I do recall a specific attack where somebody literally just took torrent names and published them as packages just to kind of – they were like movie X, Y and Z, HD quality, 1080p, as a package in PIP, just to kind of trick people into downloading things.

So, of course, domain ownership isn't the be all and end all of you being good or bad. But it does give you a certain amount of trust that if you're getting a com.google artifact, it actually comes from Google. Obviously, if somebody takes that over, that's a bigger, bigger problem and that's something that a marketplace can't necessarily control at all times. But it certainly is a step of I'm literally just rolling the dice, by downloading this random package with a random unverified namespace.

[00:15:33] KP: So, how should a responsible developer look at their package maintainer? It seems to me people kind of blindly trust, “Oh, it's in PIP. It's in Nougat. Let me just use it.” To what level of scrutiny should the average developer be looking at the things they install?

[00:15:48] IT: Well, the practice is absolutely, you should put some scrutiny on this. So, one of the things that we actually look at in this year's report is what are good projects? Are there any good measures for you to take in order for you to kind of come to a conclusion if this is a good project or a bad project? And we basically looked at a few different frameworks for it, things like an open SSL criticality framework, which is a series of criteria, which measures kind of a project's community, its usage and activity and those sorts of things.

There are other services, things like libraries.io, which kind of gives you a sense of project activity, there's a little bit of a formula there. And then finally, there's a very simple metric that we kind of tried to come up with ourselves to just make it a little bit easier. Part of the problem is, in order for you, as a developer, as a responsible developer, to make these calls, you need quite a wide range of information. You need to know, is the project active? Is this being maintained? How often do they publish new versions? You need to know what the licensing situation is. As a developer adopting open source, one of the kind of key problems that that's often overlooked is can I even adopt this piece of code? Will that make me have to publish some part of my code

under a different license if I do that? Are the licenses that I'm adopting even compatible with each other, which is, a question that not a lot of developers ever asked from themselves.

Also, you kind of have to have a security degree, if you were to do this absolutely right. Like you'd have to you know, does he have any known security vulnerabilities? Do the CVS that I see on this name actually applied to the component? And there's a whole host of kind of very subjective things. Like, is this a good version for me, given what I have in my current supply chain, right? Organizationally, like, should I go for this because it's convenient for me right here in this moment? Or should I go for something that other projects within my organizations are already using? Because it's kind of like a rebuilding the same thing, why should we use different tools to build the same thing, if it's two different teams?

So, there's a lot that goes into it. It's not a simple problem to solve. If there's one thing that we did discover that was kind of surprising for me, was that kind of a sense, a sense of what is a good piece of open source in general, seems to be what we call the meantime to upgrade number, which is a fairly simple metric. We kind of came up with it just to kind of make it a little bit easier for us to look at this ourselves. We just said, “Hey, what if we look at a project with a few set of dependencies? And just ask, how often do they upgrade those dependencies on average?” So, we'll take a look at, package A has a dependency B, and B has a new version out there. How quick is A? The root package to adopt that new change. And we look at that number for all of its dependencies, its own dependency tree. It turns out that that MTT range when you kind of crank that out, and you kind of kind of get the meantime out there, the lower the number, generally speaking, there's a strong correlation to that package having positive security outcomes as in it, having less security vulnerabilities, it upgrading the issues quicker. Also, that has implications on the quality of the software itself. Because then being more active means that the program is probably more maintained, that piece of software that you're adopting, has a backing framework around it.

It's pretty surprising how such a simple metric in the end can be quite a strong indicator even across these more complicated frameworks to tell you that that that's a good thing. So, as a diligent developer, really you should be asking all these sorts of questions all the time, for every piece of open source you're adopting. Practically, you probably need to start with doesn't have

any known issues right now? And also, how active is the project? How well do they maintain their own house and how often do they renovate their code as they do?

[00:20:02] KP: So, it makes sense to me that you could find a positive correlation between a project that has few vulnerabilities and how quickly they're upgrading when their dependencies upgrade. But also, seems like there must be a limit to that. If I'm going to the bleeding edge, might I be exposing myself to zero days or flaws introduced? Is there no limit to how quickly I should be upgrading?

[00:20:26] IT: Well, I think that's a really, really interesting question. And that's actually something that came out in the report as well through what we call a herd migration analysis. So, what we did was, because we kind of asked the same question, like, okay, maybe the old adage, you know, when you look at the meantime, to upgrade on its own, you very quickly go, "Well, it just means that by adopting an active project, and then always keeping updated, I should be fine, right?"

When you look at migrations, and you then start looking at this in practice, what starts appearing is a couple of different roles. So, you're absolutely right with if you stay on the bleeding edge, part of the problem is, if there's a new security vulnerability, there are no ways forward until the project reacts. If there is, more likely, you're going to have some sort of API deprecation issue with the package itself. I'm on this version now, upgrade to the latest version, it broke something, and there's no path for me to go forward. I have to complain to the project later to like, crank it out and get things done.

So, we looked at a few projects, and we looked at the total population of all migration activities, upgrade activities, essentially, that we saw. We kind of set out a few rules for the upgrade. We kind of said, "Okay, what we don't want to see is this sort of no path forward situation where you have no way to go. You're in a bad version, and you can't move from that version, because there's no forward step for you to go." We look at things like, just the very simple thing, like, it's a bad upgrade, if you go from a non-vulnerable version to a version with a known security vulnerability, which is something that happens quite often actually. Because people just have a lack of awareness and lack of exposure to what version have security vulnerabilities.

And then finally, we looked at things like things like, okay, well, if projects upgrade things quickly, what's kind of coming out there. And when you look at that, we have a very kind of interesting piece of data in the report that actually shows like the mass migration activity on a specific component. I believe it was our spring core. We looked at all these sorts of situations. Generally speaking, the more current that you're staying, the better it is. But it's actually not ideal to stay at the latest bleeding edge. Actually, the most ideal version, in most cases seem to be about two, maybe free versions, 2.5 versions behind the bleeding edge, because it was kind of the perfect balance of you still have a few steps to go paths forward, if you do run into a problem, but at the same time, you're not too far behind to not benefit from, kind of all the cool innovation that kind of happens there. Which is actually frankly, kind of surprising. And then on the other hand, kind of not, because it kind of gets you to an old adage of like, well just, keep your stuff updated, and you should probably be fine.

[00:23:25] KP: Very active projects, will have a latest and an experimental branch and things like that, or as you say, you know, the last couple versions are still maintained. Can I generally trust that? I know that I'm getting myself into something, if I clone an experimental branch. Is latest, always best?

[00:23:44] IT: Well, latest probably isn't always best. So, as we kind of dove into these migrations, and we started looking at, okay, if it lands on a version with a known security vulnerability, all these situations, as I mentioned before, or it has some breaking changes, we kind of looked at all these migrations, and one of the things that we saw was actually the sort of rules that started coming out. So, we kind of split them between objectively bad choices to make and subjectively bad choices to make. And one of the things that we noticed is generally speaking, going into an experimental branch is a bad choice. Because often experimental branches get pulled back into the mainline, they run out of steam, they don't get maintained, it's very easy to get stuck in those versions. As in, the cost of upgrading away from them can become so costly, that many of us just don't want to do it. If you if you get entrenched in that version, it's very, very hard to get away from it.

The other problem there tends to be that you, with experimental appliances and things like this, you have very limited community support, especially once the eye of attention moves away from it. So, let's assume it's a good choice for you to do today. It might not be a very good choice

even after a year or so once the project has kind of adopted what's there. But your software might not be able to move to that mainland version, because maybe they they ran into some sort of API problem or something, and they and they had to like fundamentally change something in that branch. So, generally speaking, I'd say stick to the mainline releases, and generally speaking, stick a couple of versions behind, that's usually your best choice.

[00:25:29] KP: When I've looked into some of the attacks on repositories, they're to my eyes seem to be general attacks. They're whoever happens to have this installed, it's going to indiscriminately scan for email addresses or credit card numbers or something like that. I'm yet to see one that attacks a particular company or organization or anything like that. Do you think that's a viable attack path we're going to see?

[00:25:50] IT: Oh, no, that's a very, very common attack path already. In fact, that's one of the fastest growing modes of supply chain attack, when we look at this. So, in general, when we look that just the growth of attacks on the supply chain, really, the last 12 months have been a type of tremendous increase. I think we sold over 650% growth in just the volume of attacks, that was somewhere to the tune of about 16,000 recorded attacks and that was years today, not even that. So within there, there's many indiscriminate attacks, but one particular mode of attacking is something we call namespace scanning, also, namespace confusion, also known as dependency confusion as spent by the original author Alex Burson.

Essentially, what that is, is a way to target specific organizations by understanding what their packages or internal packages are called. So oftentimes, most organizations, the way they adopt open source is they'll use the same dependency manager for both their internal packages, which they'll store in some internal registry or artifact manager like Nexus, and then they'll also adopt an open source, right? That's that sort of 90/10 split that that occurs there. And what Alex discovered was that, what you could actually do is go out to these upstream registries, and register internal package names, because, like I said, many of these don't have this or namespace control, they don't require you to verify that you even own the domain.

So, what he found is he could deduce, for example, out of public GitHub entries or things like this, what internal package names organizations were using, then register that same package name in upstream registries, but publish them with an exceptionally high version number. And

once that happened, that package was published, he could then install backdoor exfiltrate, things like tokens or databases that were in the local system. And obviously, he was doing it as under an ethical research or contract. But you can very easily see that that mode of attack can be used for many things.

What happens is most of us will probably just say, download the latest version, right? Coming to our previous discussion about,what sort of version you should be on. The reality is most of us, when we look at our package JSONs or requirements txt, we tend to pin just download whatever's latest right now, because latest feels like it's the greatest. And Alex's mode of attack basically said, you're probably also doing that for your internal packages. So, if I register an internal package name in a public registry, and I publish it under version number 699, or something like that, your system is probably going to get tricked into downloading the latest, which is now there, because it's probably connected to both your internal registry and your open source registry.

So, doing that, it’s a bit incredibly low skilled attack. You don't you don't need any special knowledge other than the package names, or even a reasonable assumption of what those package names might be, and then you can have a specific payload. So that's definitely, definitely something that is the most fastest growing method of attack that we observe when we when we look at things. We regularly report to organizations about suspicious packages published under their namespace, which looks namespace related. And we've also even seen specific supply chain attacks targeting things like cryptocurrency applications. There was a very famous case of last year where somebody published, took over a piece of open source, they knew was used by a specific crypto wallet. What happened then was they got a little bit clever. They published an overtly malicious looking version in that legitimate dependency. Essentially, they took it over, published a couple of versions. The community went, “Hold on, something's wrong. There's clearly some very dodgy code here.” The community them quickly said, “Okay, the last known good version was version X, Y, and Z.” And what the attacker had actually done is in that particular version, they'd injected some malicious dependencies, which targeted a specific crypto wallet application, which now was in the process of upgrading to that version. And when that version was run on any client device, it stole all the funds from that application from that crypto wallet.

So, I think we estimate that they made out with over $4 million gotten gains out of that. So unfortunately, these tend to be very, very targeted nowadays. But because it's a supply chain, you can leverage it to have wide coverage, because these don't have to necessarily be very sophisticated attacks, because they rely on subverting trust that you have in these external pieces, but they can be very, very targeted as well, and that's certainly something that we're seeing quite a lot.

[00:30:56] KP: Yeah, I'm trying to imagine myself in the role of an IT professional and the head of security at like a medical records company, you've got to protect the data, and you're hearing about attacks like this. You're well aware of being targeted for ransomware attacks. My instinct might be, I need to lock down the developers, let me proxy all of these repositories, some heavy-handed approach.

[00:31:25] IT: Yeah, so like I said, these targeted attacks, they're more common than you think. We've seen things like crypto heists targeting specific crypto wallets. We've seen people targeting specific companies with these low skilled attacks that occur by just squatting your internal package names. And everything in between. You could argue that SolarWinds was a very targeted attack, as well as you know, another campaign that was levied against a company called Cold Cove earlier this year, as well. So unfortunately, they can be very targeted and that's really kind of the fastest growing method right now.

[00:31:59] KP: So, if I were the head of IT security, like a medical records company, I'd be terrified of all these kinds of attacks, and I'd be considering a heavy-handed approach, like, I'm going to proxy all of the repositories and every request comes through me, and I'm going to manage what the developers get back. Are there any approaches like that, that are viable? Or what do you see different large enterprises doing?

[00:32:24] IT: Yeah, I mean, that's actually not a bad practice. But it's less heavy handed than you think. So, I think in larger enterprises, when we're thinking about this, really, it's not a one stop shop thing. Auditing these things, isn't just a point activity. It's not really something that you can expect to do once and then kind of consider the open source good. If it has known security vulnerabilities that are discovered later, you might want to introduce auditing at all stages of your build pipeline, as an example, just to make sure that you're catching everything as they come,

because all of this is a function of time, as much as knowledge because we only have the knowledge that we do.

Generally speaking, there's actually a lot of advantages to just proxying your open source, using things like Nexus repo and Pandoc, just setting up proxy repo to your open source environments. Firstly, because it kind of builds a catalogue of what you're actually downloading to yourself, so you know, what's being put into your software, in case you need to figure it out later. And secondly, it allows you to do things like whilst we're downloading things, we can scan them on the fly. So, there are solutions, like our Nexus firewall, as an example, that kind of sit in between you and the open world. As your developers are requesting things, they can just be audited, on the fly, doesn't really have much of an impact on download times feels like the exact same experiences as just downloading directly would. But if it does have something that has like a 10 out of 10 security vulnerability, or really look sus, because it's been flagged for potentially suspicious activity, like a popular open source project with a new contributor with 0 commits before and a really dodgy email address, that's from throwaway service, that's usually a fairly suspect release.

Those sorts of things can be monitored for and and done for. So, if done right, that can be a fairly low touch way of doing quite a lot of this sort of activity upfront kind of avoiding the worst of the worst. Because software engineering, we've kind of got this saying, the fastest way of fixing a bug is the earlier find it, the faster and cheaper it is to fix. Well, the fastest way of dealing with security vulnerabilities just don't have it at all right. Don't get it in the first place. If you do have something, then kind of the earlier you can get to it within your own like development cycle, the faster and easier it is for you to rip it out or upgrade it or do something like that.

So, in enterprises, the organizations that do this right, kind of do have that sort of upfront scan, but it's a scan that just occurs as a part of the normal development workflow. They typically run these evaluations as a way of informing their developers known as a way of controlling their developers. I don't think there's any dev in the world that sets out to download stuff, right. That's, that's odd bad. I think it's just a lack of information. In larger enterprises, when we cross the balance between informing our developers when things are not good, and informing them, why they're not good, that can go a long way to making it seem like, “Well, that's just business as

usual. I'll just find something else. This version, wasn't it. Maybe I can do this recommended version over here instead."

So, that's definitely the case. On the on the flip side, if you go too heavy handed, I've definitely seen lots of cases where the CISOs have said, "You know what, you're not downloading any open source." And I can tell you, that's a pretty, pretty lonely path to tread for the CISO, because why do you hire developers? You hire them because they're smart. They'll find a way around the problem. Turn on the hotspot and download it at home or something. So, there's kind of no controlling it. It's just the part and parcel of how we work today. So, the best companies tend to be more informative than an enforcement driven. But there's a level of enforcement as well, especially in the beginning, that can sometimes be quite healthy, kind of avoiding worse outcomes down the line.

[00:36:41] KP: Well, if I understood correctly, it sounds like in the years you guys that Sonatype had been doing the state of the software supply chain report, this was the biggest growth of attacks you've seen, is that correct?

[00:36:52] IT: That's absolutely correct. Over the last few years, it's been over 650% year over year increase in just the volume of attacks that we've measured.

[00:37:03] KP: Do you have any explanation for that? So, why are we on this hockey stick trajectory?

[00:37:09] IT: Well, partially because it's a self-fulfilling prophecy, right? Because open source is very useful, we use a lot of open source. Because we use a lot of open source, it becomes an attractive target. I think the psychology is fairly simple. From an attacker's perspective, or as an adversary, if I say, "You know what, I want to hack this company." I've pretty got two paths, avenues – I mean, I've got several avenues to do that. But if I want to do it, the application route, I can either figure out how you're writing your own code, figure out the specific logic of all of your developers, the specific frameworks, do all the all the upfront work, or I can just find the headers that your website is throwing, find the top 10 most likely frameworks that you're using, and then find out all the known security vulnerabilities that affect that framework and try them all, see what sticks. It's not very hard to script that out and give it a go.

So, that's one aspect of it, because we're using a lot of open source, open source is a target of the attack. Outside of that, though, it also is just return on investment on the adversary side, It's easier for me to effect one code base, if I know that it has a downstream effect on potentially millions of consumers. One is a very one on one attack, the other one is a very one to many attack, and that can be desirable. So, that's why we've actually seen these attacks kind of move away from people targeting websites, adopting open source to actually targeting the upstream projects that they know that they can adopt. That's part of the reason.

Finally, it's simply just the emergence of more and more low skilled forms of attacks. A form of attack can simply be a registered customer, your target's package names in a registry, and then publish some very bad code, maybe somebody will get fooled, because there's no way for them to decipher if that's a legit thing or not. So, all of these contribute to it. And, you know, kind of another interesting thing that we actually did find in the report this year was that generally speaking, the more popular an open source project is, the more likely it is to have known and discovered vulnerabilities. So, it means that there's more eyeballs just all over the place, especially in the top 10 most adopted projects.

[00:39:32] KP: Well, I'm wondering if we can do a hypothetical. I'm imagining, maybe there's some very, very clever nefarious developer out there who has taken control of a popular project or maybe medium popular, and they've done something else clever, like found a way that only when it runs on an EC2 instance, will the code spin off a thread that uses 1% of the CPU to do cryptocurrency mining, and that they've just got this broad base a botnet that's doing a tiny amount of work and they've gotten it through open source. What are the odds that could be out there and undetected?

[00:40:07] IT: Very high. In fact, that's one of the very common form of attack to do that. So, we've seen things like a series of malware campaigns that prey on Discord installations. They look for discord, databases. If you have crypto in mind, actually much easier thing for you to do, not that I want to teach you how to run these campaigns. But a much easier thing and a fairly common thing to do is finding a medium popular project. And when I say medium popular, a couple of million downloads a week, which is a relatively small thing. There's plenty of smaller projects out there that actually come relatively popular, a single maintainer project or things like

that. What you do is, and what we've seen happen is these bit erlang strangers, good Samaritans, approaching these projects, and perhaps even for a while legitimately contributing to the project.

Once they get container or maintainer access, then taking those projects over and actually introducing a crypto miner, not even in the mainline code, but in the post install scripts. Many package managers have a mechanism for you to run some scripts to install any sort of payload, as soon as the package is installed. It's typically used to compile stuff on site.

So, what you just do is, mathematically speaking a couple of million downloads a week, you kind of know that that's going to happen, if they are on five seconds each in the system, let's assume they have perfect things, that's still quite a lot of currency that you get for not a lot of work. So actually, crypto mining campaigns are very, very attractive, because it's an easy monetization form. You don't have to exfiltrate any data, you just get money directly out of their CPU time, but we do see much more of these targeted campaigns as well. It really depends on what they want to get out of that attack, and the kind of scenario that you've just discussed, I wouldn't be surprised if there's something out there right now doing something like that.

[00:42:12] KP: Well, for whatever reason, I'm starting to in my mind analogize this problem to something like flu epidemics. So, we just live with influenza. We have for a while. We get annual shots and whatnot. We may be living for a long time with COVID. Will we always be living with these threats of attack on open source? Or will that be kind of some fix all that someone clever will come up with?

[00:42:36] IT: Well, I think, it's really important for me to say that this is a champagne problem of success, right? The only reason why we're having these discussions is because open source is very, very useful in most modern applications. It helps us deliver software faster. It helps us stand on the shoulders of giants. I certainly wouldn't be able to produce any sort of crypto library myself. I'll just something that's really, really useful in that sort of sense.

I think, in some ways, we wouldn't have the modern software economy that we have today, if we didn't have open source. So, in some ways, I think your analogy is absolutely right. It's just a part and parcel of the success of that. But that being said, you do have a certain amount of

hygiene. You do wash your hands. You do certainly take time to take precautions, especially nowadays, in public transit or things like that. So, similarly, from an engineering perspective, what that really means to me, is that we've got an engineering practice that we have to invent, as we're just doing it. In an ideal world, it'll become just a part of the little bit of hand washing that we do every day when we write our software, and that arose a lot of the situations.

But I don't think we're ever going to get completely relevant. I mean, that's the beauty of being an adversary over a defender. A defender has to find hundreds of scenarios to defend against. An attacker only needs to find one to exploit, right? So, the odds are always stacked against finding that and there are people with incentive to do that. They'll continue to find new and interesting ways.

[00:44:17] KP: Well, there's definitely some insights, trends and good advice in the report we didn't get to hear. For listeners that want the full report, where can they find it?

[00:44:25] IT: Yeah, just go to sonatype.com, you'll find a big link, sonatype.com/state-of-the-software-supply-chain-2021 or just Google State of the Software Supply Chain 2021 You'll find it. It's kind of a lovely website. You can scroll through the main insights and if you want to get the full report with all the scientific data, then you can get the PDF at the bottom of the page.

[00:44:47] KP: Ilkka, thank you so much for coming to Software Engineering Daily.

[00:44:49] IT: Thanks so much for having me. It’s been great.

[END]