EPISODE 820

[INTRODUCTION]

[0:00:00.3] JM: The Linux operating system includes user space and kernel space. In user space, the user can create and interact with a variety of applications directly. In kernel space, the Linux kernel provides a stable environment in which device drivers interact with hardware and manage low-level resources.

A Linux container is a virtualized environment that runs within user space. To perform an operation, a process in a container in user space makes a syscall, which is a system call into kernel space. This allows the container to have access to resources like memory and disk. Kernel space must be kept secure to ensure the operating system’s integrity. Linux includes hundreds of syscalls. Each syscall represents an interface between the user space and the kernel space. Security vulnerabilities can emerge from this wide attack surface of different syscalls and most applications only need a small number of syscalls to provide their required functionality.

gVisor is a project to restrict the number of syscalls that the kernel and user space need to communicate. gVisor is a runtime layer between the user space container and the kernel space. gVisor reduces the number of syscalls that can be made into kernel space. The security properties of gVisor make it an exciting project today, but it is the portability features of gVisor that hint at a huge future opportunity.

By inserting an interpreter interface between containers and the Linux kernel, gVisor presents the container world with an opportunity to run on operating systems other than Linux. There are many reasons why it might be appealing to run containers on an operating system other than Linux. Linux was built many years ago before the explosion of small devices and smartphones and IoT hubs and voice assistance and smart cars. To be more speculative, Google is working on a secretive new operating system called Fuchsia, gVisor could be a layer that allows workloads to be ported from Linux servers to Fuchsia servers.

Yoshi Tamura is a Product Manager at Google with a background in containers and virtualization and he joins the show to talk about gVisor and the different kinds of virtualization. Some recent updates from Software Engineering Daily land, the FindCollabs first hackathon has ended. Congratulations to Arhythm, kit space and rivaly for winning first, second and third place respectively. They won $4,000, $1,000 and a set of SE Daily hoodies respectively. The most valuable feedback award and most helpful community member award both went to Vince Montgomery, who will receive the SE Daily towel, as well as the SE Daily old school bucket hat.

The links for all these things are in the show notes, along with few other updates. FindCollabs, my company is hiring a React developer. You can find that link in the show notes and a new version of Software Daily, which is our app and ad-free subscription service is in production at softwaredaily.com. That's an open source project and you can contribute to it as well if you're interested. Pod Sheets is our open source set of tools for managing podcasts and podcast businesses. We should have a stable version hopefully up in a couple weeks, where you can host your podcast if you're interested. If you want to contribute to it, you can of course also click into the show notes.

Let's get on with the show.

[SPONSOR MESSAGE]

[0:03:57.4] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point, whether you're building an app for millions of users, or just figuring out a side business.

As the most popular non-relational database MongoDB is intuitive and incredibly easy for development teams to use. Now with MongoDB Atlas, you can take advantage of MongoDB's flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security and high availability and data recovery and monitoring and elastic scaling.

Try MongoDB Atlas today for free, by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas, as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se. Thank you to MongoDB for being a sponsor.

[INTERVIEW]

[0:05:19.3] JM: Yoshi Tamura, thanks for coming on Software Engineering Daily. It's great to have you.

[0:05:22.4] YT: Thank you, Jeff. Great to meet you and thank you for having me today.

[0:05:25.5] JM: I'm excited to talk to you about different facets of virtualization. I'd like to start on the topic of virtualizing at the hypervisor level versus virtualizing at the container level, which is not exactly virtualization but is quite similar. How would you distinguish between those two areas?

[0:05:45.9] YT: It's a very good question. Those are two different technologies as you might already know. Virtualization and especially in virtual machines, as the name states, it provides machine level abstraction. If you wanted to chop up a resource, a resource into multiple machines, we exactly have virtual machines.

On the other hand, containers are virtualizing the operating system view more in the process side. You can imagine having a container sharing the operation resources on virtualizing operating system resource as if the container owns everything, but in reality it's actually shared by the host operating system.

[0:06:25.0] JM: Containers can run on bare metal, or containers can run on top of virtual machines, how would you contrast those two operating environments?

[0:06:35.1] YT: Continue to run containers, because they are basically a process right? Even if you run directly on bare metal, you still need to do the operating assistance. Somebody needs to do that operating system work. Fundamentally at this point, there are basically two approaches; one is that you run a kernel inside a virtual machine and then run on top of – on top of that, you run the containers. That's one way of doing.

The other way of doing, namely what gVisor actually does, is that they take the system calls, intercept that and do the emulation outside of that sandbox, or the context of container and then try – get the help of the host operating system to get back to the container. There are two different yeah, architectures as far as I know.

[0:07:23.4] JM: How do the performance characteristics of those two different environments compare to one another?

[0:07:28.2] YT: It really depends on the workloads. For example, if you have CP and memory oriented workloads, then even if you take the path of it's like gVisor which emulates the system call, then the overhead will be very, very limited. For example, this morning at 9:00 a.m. we had a session about GKE sandbox. There, I actually have slides about the performance. It's not on bare metal, it's more on GKE. We did the comparison between [inaudible 0:07:56.1], the native container without sandbox, versus GKE sandbox and performance.

For CPU and memory range workloads, the gap was only around 2% to 3%. We actually shared the stage with my launch customer from Descartes Lab, Tim. He was also showing that he used the GKE, or he's already using GKE sandbox in production, because he runs a SaaS. For that SaaS environment, they do you need – they need to isolate between tenants.

His customers workloads are more CPM memory-oriented workloads, thus he was super happy about the a additional defense and isolation provided over the cycles that he need to give to GKE sandbox.

[0:08:41.2] JM: Within Google core infrastructure itself, not talking about Google Cloud, as I understand most of the workloads do not use a VM. It is containers running directly on bare metal. Why is that the case?

[0:08:58.8] YT: What do you just mentioned, I don't have any data, or a good way to confirm. However, I will just say containers and container is really good at resource efficiency. It really leads to the [inaudible 0:09:13.7] too, but when you actually share a large machine or a large cluster by virtualization, what people forget about is that once you actually go down the path of virtual machine, not virtualization, virtual machine, then it's a machine. Machines don't usually change their memory sizes here and there. It's actually mostly fixed exactly.

If you actually start chopping up those resources based on virtual machine, for example, you need to pin down that memory for a gigabyte, even if you get smaller, 1 gigabyte, fine 12, 256, you need to stay there. On the other hand if it's in a host operating system in the contingent process, you can allocate a memory or page later. If it's not used – For example, in many cases like SaaS and a PaaS, right, a lot of sites or programs may not even run at all, but it needs to respond when the customers, or their users access that site.

You don't necessary want to – the customer may not want to pay for that, the service operator doesn't want to take the costs either, in that situation, what's going to happen? If you have a processor container, you can easily swap it out. If it's a virtual machine, it have to stay there.

[0:10:34.2] JM: We'll get into that – what you described is an allusion to the cold start problem. We can explore a little bit later on. You're talking about dynamic scale up and scale down of a single container. Could you describe an application where you would want to dynamically scale up or scale down a single container and how that scale up or scale down would work in practice?

[0:11:02.3] YT: That's a very good question. Let me think about it. There are so many workflows that will fit into that category. One could be a web application, I think. Even a web app, a simple web application that could represent, let's forget about scaling now, because that's one another way of doing to end up in a traffic. Let's say, we just happen to have a very small website and it's running inside a container. Now initially, you may actually just allocate small resources, right? Once you get more traffic and requests, or the web application need more data, then you want it to expand.

The tricky part that right now what's happening is that it's really hard to predict how much resource that would a process or a site need. Then if it's not needed anymore when the peak goes out, can it scale back? That is I think the characteristic of the workload. At your point, if you have workloads that are facing with customers or to the world that's hard to predict, if you have a very specific assumption about the resource consumption. For example, when you run on batch workloads, you already have some understanding about how much resource it could use. If you're actually facing unknown – when the business is changing, then within this container, you may want to go up and down flexibly to optimize the resource consumption.

[0:12:32.5] JM: We're talking here about virtualization on a server side device. I am accessing a Google server from my phone and I want to request my Gmail application, whether I'm looking at my inbox or I'm sending an e-mail, I'm hitting a virtualized instance of Gmail. It's probably running on a container, that container maybe it's in a VM, but sounds like it's probably on bare metal and some interesting architecture that we'll get into. Do we want virtualization on client devices also, or do we just want to talk about virtualization here on the server side?

[0:13:16.7] YT: It's a very good question. As far as I can tell from what's going on in the gVisor side and open source community, there are folks from ARM, did the presentation very, very recently. I think on even in clients, I don't actually know too much about that, but ARM apparently, not on – their server side ARM as well, but their client side, there's so many of the ARM-based chips. I would imagine that virtualization, or virtualization-based isolation will be interesting area in the coming years as well.

[0:13:50.2] JM: You work on, or you have worked on Android on Chrome, right?

[0:13:54.3] YT: Exactly. Go on.

[0:13:56.7] JM: I mean, that's a virtualization on a client device. I've got a Chromebook. I'm very happy that Android applications can run on it.

[0:14:05.5] YT: Thank you. I really worked on that. Yes.

[0:14:07.2] JM: I'm sure you did. Tell me about that. I mean, why do you say – given that note, why do you say that this will be interesting in the coming years? It sounds like it's already been interesting.

[0:14:18.9] YT: Oh, that's a great point. Now I don't work on – I'm not with the Chrome, so I don't want to talk on behalf of them. However, as there are some talks from Google I/O back in I think was 2016, when we're thinking about bringing the Android apps to Chrome OS and Chromebook, we thought about, "Okay, maybe we can use containers to bring those workflows to Chrome OS." The container was a good way to manage and also isolate the workloads and our initial motivation behind the scene of why the container started getting used in the context and our apps in Chrome OS.

[SPONSOR MESSAGE]

[0:15:08.5] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[INTERVIEW CONTINUED]

[0:17:20.4] JM: How does the challenge of running containers on a client device compared to those of running containers on server-side infrastructure?

[0:17:30.7] YT: Technically, I don't believe there are significant difference. If there's anything that is much more sensitive, I think the graphical part, or multimedia stuff, which will require a lot more access to the hardware could be different. I won't necessarily call it a challenge, but in case of server side as well, right? You only have workloads and they don't necessarily have to access those, right? Like me multimedia start. There are some differences like that. As a consequence, I will not be surprised. There are different approaches and architectures to accommodate that that's a requirement. I think that is the in a big picture that could be different.

[0:18:15.8] JM: If in some world where, let's imagine all the different application icons on my smart phone, for example. If any one of those – I mean, those don't run in containers today right? Or do they?

[0:18:27.1] YT: Which one are you talking –

[0:18:27.9] JM: Let's say I'm talking about Android.

[0:18:29.6] YT: Android. I don't believe so.

[0:18:31.9] JM: I don't think so. Let's imagine they did. If they did, we would have to figure out how to share media files between them. That's what you're saying.

[0:18:40.2] YT: Yeah, correct.

[0:18:41.5] JM: Containers, do they not have good strategies for sharing resources today?

[0:18:45.8] YT: Oh, sorry. Then I think the media point that I mentioned was more like seeing a graphics, right? Rendering. On the server side, you don't actually have that gaming, right?

[0:18:56.7] JM: Oh, I see what you're saying.

[0:18:57.8] YT: You don't need that – you may not necessary have that requirement. On the other hand, client side, is all about what you show up in the display, for example, or audio,

Bluetooth. Those are the content – or Wi-Fi, right? All those context, actually I'm saying that could make a difference when it comes down to the solution for a similar problem. Yes.
[0:19:18.9] JM: Right. Certainly doable, but hasn't really been implemented quite yet, which is why you're saying this will be an interesting area to explore in the future.

[0:19:26.0] YT: I think the Android apps on Chrome OS has one answer for sure. What I mean by the coming years is that for example, gVisor is still we have official products, like cloud run, which got announced in day one this time. Now we have GKE sandbox, which use gVisor to provide sandbox in GKE, Google Kubernetes Engine. A lot of things are happening server-side.

We don't actually want to have a product, or community people who say, "Yeah, we launched this product using gVisor." I think that's what I mean by from a context of gVisor and that type of isolation are still in the coming years, not necessary like I mentioned about other projects or products at this point.

[0:20:09.1] JM: Now as a smartphone user, or a client device user, you could even say, you are a consumer of technology. You also know deeply about virtualization. If I'm a user, why would this be appealing to me? Why would I want containers on a client device? Am I going to get performance benefits, or security benefits, maybe something with web assembly, I don't know. What am I getting?

[0:20:34.1] YT: I think the reason why containers on those client devices, I think from end-user perspective will not be visible. If anything, it will be more about how to deliver the application. For example on Android, there's Play Store. In Chrome OS and mostly on the web is the how the application, web application get to deploy to the client side.

Now when it comes down to other devices, we talked about – when we say client devices, like we talk about phone and laptop, but there are things like IoT devices, what are the application developer mechanism is going to be like? I asked this question to some people who are entrepreneur that – so would you want to use the existing client side programming model, or application builder model in that area? Exactly, your reaction. Exactly.

It's a different world, thus I think a lot of people think, “Can I just use containers, because it runs on server side? If it can only also run on the client side, it will make my life –” You still really have to deal with the difference between a server and a client. If I said like I talked about, like multimedia stuff on a client side, versus server side, which is much more a very simple, just CPU, in-memory and networking and just pretty much, right?

Just handling those differences and a for foreign factor is complicated enough. If there is a cost and demand that can I just use containers, test on a server side and then deploying the client side, or any other devices, right? Once run anywhere is always there.

[0:22:11.6] JM: I think what you're describing is, like I think about my experience as a mobile app developer. I've deployed to the Android App Store, I've deployed to the iOS App Store, not the most buttery smooth experience, right? There is a reason for that. It's because these are these low-friction, life-and-death applications that we’re downloading. If I download a smartphone app that breaks my phone, I might be in the middle of nowhere without my map all of a sudden. Oh, God. That's terrible. That's why we have this gating process with the App Stores.

That also speaks to the fact that why are we delivering applications this way? Why are we downloading applications in this way where it's all in this – I guess, I'm not an expert in Linux systems, but I guess it's all in the same user space basically. If we had containerization, we would have this natural application model enforced set of isolation boundaries, so that we could tell just by the rules of the game, this application cannot break out and ruin my day.

[0:23:23.3] YT: I think that's exactly – you’re spot-on. I think one of the biggest, largest benefit of the ecosystem, I'll call the application ecosystem from Play Store, in the iOS case the App Store, is definitely that is it secure to run this application on my phone? Can we make sure that that cannot get compromised my phone or any devices.

Therefore, I think or a long time, this running Docker or containers just pure, like bare containers on such sensitive devices. We're not a primetime for a while and we always thought that these application dealer mechanisms is there. I think there's a good reason for that. Now with more technologies like sandboxing, right? Now let's assume that just hypothetically, if it's just hypothetically, there is a technology perhaps and maybe based on gVisor or something else, it doesn't probably matter.

If there is a hypothetical technology on a form factor, right, on client side, which can run Docker in containers, and even if that was malicious, even that was a get compromised, if we can actually guarantee that if we shut down this container, we'll remove this container, if the device can get back to the safe state, that will be very appealing. Because all you have to do, because you know that even that a Docker container is compromised malicious, the threat will remain there. The threat will be contained. Before containers were not contained, therefore there is always a concern risk of that leaking outside, breaking out of container. We could contain, then that's why we'll remain there. Once we figure out, think we'll remove it, shut it down back to the fresh state. I think that is the appealing part.

[0:25:13.2] JM: I completely agree. Let's get to talking about gVisor. Explain what gVisor is.

[0:25:19.3] YT: gVisor is a technology that Google has been building for sandboxing. From a technical perspective, the most interesting part is that like I mentioned earlier, it intercepts a system calls. Then instead of, today containers are tied together. There's container and Linux kernel and talked directly. Instead of letting that happen, it intercepts the system calls, forward it to the user line kernel, user line kernel ready and go, instead of C.

First of all, the kernel in user line means is user line. It's not privilege. Then we use Go for memory safe and many other benefits that Go provides. That actually becomes a first isolation boundary and as a second layer of isolation. Even if an attacker can compromise that user line kernel, even gVisor may not be bug free. Even if attacker was able to compromise that kernel, there is additional second filter and namespace that will function as additional layer security.

When we say gVisor, the key part here is tap to system call between – sitting between container and the host kernel, insert two layers of isolation; one is sentry called user line kernel called sentry ready and go. Then the additional second filter namespace which will restrict the sentry's behavior is self.

[0:26:39.1] JM: What are the use cases of gVisor?

[0:26:41.7] YT: gVisor has been used inside Google and now coming out to Google call products. The most that there are two main users or use cases. One is that first, you wanted to have a defense in depth to your container regardless of use cases. Again, if users are aware the container may not be able to contain, then they will be able to use gVisor for additional defense in depth for their containers.

Now, the much more intuitive of a use cases is like SaaS and multi-tenancy. For example, I think we started this conversation from cloud run, where cloud run accepts various code coming from users, or even plugins, then the service provider has to run those programs in multi-tenant fashion by nature. Now again, if it's using container as a back-end, then as we talked already, it may not be able to contain in that situation, the multi-tenant situation, you do want isolate between those tenant workloads. That is one of the most intuitive use case of gVisor.

[0:27:58.8] JM: Here's a quote, “gVisor implements Linux by way of Linux.” What does that mean?

[0:28:05.1] YT: Who actually said that?

[0:28:07.0] JM: Your website. gVisor website.

[0:28:08.6] YT: Okay. Sorry, okay.

[0:28:10.8] JM: I should’ve cited it. I guess, the github.

[0:28:13.3] YT: It’s a github. I think what it means by that is it's still a Linux. The username kernel emulates Linux. The key part is that it doesn't deny the existence, or the value of Linux at all. It actually just intercepts a system call from the container and do a lot of work as much as possible. It will work cooperatively with the hosts on this kernel to offer the services that containers need.

[0:28:44.4] JM: Does it require that host to be a Linux kernel?

[0:28:49.0] YT: At this point, yes. At this point, I will say yes. Technically, the architecture, the gVisor's architecture where you're thinking, should be more general. We just don't have that implementation plan or anything, but theoretically it shouldn't be restricted to Linux on its own.

[0:29:05.6] JM: Let's say you could choose from a more diverse palette of host operating systems for your gVisor container to run on top of. Why would that be useful?

[0:29:18.0] YT: Other than Linux, you're saying?

[0:29:19.4] JM: Other than Linux.

[0:29:20.9] YT: That's interesting question.

[0:29:22.2] JM: Would there be another operating system you would want these containers to run on?

[0:29:26.7] YT: Hypothetically, okay that's a very good question. Even though we say Linux, there are various different flavor Linux as you already may know, type of the kernel versions, type of distribution. It's probably fair to say that having gVisor running on more platform, why there's Linux or anything else would help share this experience of isolating container in a stronger fashion; sandbox, the containers do contain can be spread to more environments.

Through that possibility, we may see more use cases that we even thought about – again, gVisor started from the Google's infrastructure. Now there's cloud and we're excited about those, even people outside of Google can actually benefit from the value, benefit from the technology through cloud product. As gVisor is open source, it's pretty much up to the user's imagination and use cases to hoarder two more other operating system, other flavor of the distribution, which is already happened that aren't in the initiatives from ARM folks, right?

We didn't necessarily plan that ahead, but they came to us and say, want to have gVisor running on ARM and that's interesting. Go ahead, please. We'll help you. You can see all the conversations and the presentation in public. I think, those are the exciting things that we expect from the open source community.

[SPONSOR MESSAGE]

[0:31:04.7] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so you can monitor your entire container cluster in one place. Datadog's new live container view provides insights into your container's health, resource consumption and deployment in real-time.

Filter to a specific Docker image, or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure. Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shirt.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[0:32:02.8] JM: When you think about this emergent world of running everything inside of a container on top of some other medium, Linux is that medium today. Is Linux fundamentally the best set of primitives for running those kinds of workloads, or is there something inherent in Linux that is perhaps at such a low-level that maybe we need an entirely new runtime environment altogether?

[0:32:36.3] YT: That's a very good question. I think through that once in a while. First of all, Linux has a lot of responsibility, right? It runs on various form factors, it has large set of devices that can run and it has a lot of functionality as well. I think, really want to make sure that whatever we do in gVisor is not about denying that path. It's just that it has a lot of responsibility, it has a lot of use cases, it's serving a lot of people even today.

Now given that situation, as a software engineer as everybody know, a lot of features as a consequence complexity come into play. Now in that situation, gVisor is a new approach. Really, we just call it a new approach. It's just one approach, but we think it's an interesting approach to see okay, so from what we observe from inside Google, we build something like this for our containers, right? Within our containers.

It's somewhat working for us and it's working for our cloud products now. Now let's see how that would work in other cases, or in the world in general is a drain that we're just started last year when we open sourced. Then so to your point, I think we don't listen to how the answer yet, but we were excited to see how that's going to turn out.

[0:34:01.0] JM: Google has been working on an open source operating system called Fuchsia. What would be required to get gVisor containers running on Fuchsia?

[0:34:12.6] YT: I probably don't have any data point to answer that question unfortunately. My knowledge of Fuchsia is very limited. Unfortunately, I won't be able to answer. There's only one thing that's in public that could be interesting. gVisor has its own networking stack, which I didn't talk. Networking stack also written in Go, called net stack. It's a separate repository, so not many people notice that. If you go to the Fuchsia’s website, they say they use that net stack.

[0:34:42.8] JM: What functionality does that networking stack include?

[0:34:46.8] YT: It's basically a TCP/IP stack and written in user space again, written in Go. It's the whole networking stack that Linux usually in any operating system has.

[0:35:00.2] JM: Let's take that as an example. You got this rewrite of the networking stack in Linux. It was originally written in C, I presume, and you've rewritten it in Go. In that rewrite, not only do you get the memory improvements of go, I guess the memory allocation improvements basically?

[0:35:21.0] YT: The type safety.

[0:35:22.3] JM: Type safety. Okay.

[0:35:23.3] YT: Assuming we’re safe. No point, you know, you can't move the pointers.

[0:35:26.4] JM: Yeah. Can you also find other optimizations, or rearchitectures to look at when you're doing a rewrite like that? How deeply are you looking at it? Or is at that point like, okay, it's Linux, its battle-tested. Let's just try to rewrite it basically porting from one language to another?

[0:35:42.1] YT: There are a lot of details, I think. It requires a lot of expertise of first of all, understanding the Linux kernel stack, any stack tag let's say. It is a very large portion right, with a lot of optimization. Thus, just rewriting is probably not way to frame. It's definitely a rewriting, but you do need that expertise about what can be done, what are the features, what are Y, the code structure as such.

I think porting is really easy to say. However in reality, it will for all lot more – and I'm not – I've been away from the software engineer world for now since I became product manager. It's like five years. I cannot speak on behalf of them. However, we do improve net stack literally every day. We know through this feedback of Jiva's or an open source and also our production cloud products. We get feedback like hey, can we improve this? Working a work for temple network performance and then we embarked on it.

It is still initiative going on and I think it will continue why, because Linux kernel will also keep improving. think that is a a interesting part when it comes down to okay, so it's not porting, it's continuous effort of improving the foundation of the computing on different forms.

[0:37:05.5] JM: Let's zoom back out. We've gotten quite deep into the weeds here. Containers themselves are not a sandbox. Explain why that is.

[0:37:16.2] YT: The most important fact is that continuous interface with kernel directly, meaning that when there's a vulnerability inside kernel, there is a possibility that the attacker can exploit that and then break out a container by bypassing the mechanism, or escalating the privilege. I gave this talk this morning and it's a non-intuitive. In fact, I'll be really honest, a couple of years ago, I also thought container was indeed in a strong isolation boundary.

[0:37:46.4] JM: A couple hours ago I thought the same thing.

[0:37:49.2] YT: Now 2016, I think it was summer, the bug called dirty cow came out. That was leveraging the vulnerability of the bug inside the copyright, so that's cow.

[0:38:00.1] JM: I'm sorry, what was the name of the bug?

[0:38:01.2] YT: The dirty cow.
[0:38:02.1] JM: Dirty cow.

[0:38:03.9] YT: It was exploiting the race condition in the memory subsystem of Linux kernel and it was basically to raise by the attacker. If the attacker can exploit that bug, then it was able to escalate a privilege and take over the route. Those are the a case is just one example, but those are one of the reasons that there is a possibility that process or attacker inside a container may be able to break up through the large attack surface that Linux kernel has.

[0:38:36.5] JM: How does gVisor insulate a container from kernel space?

[0:38:42.2] YT: Just repeating, but in case of gVisor, like where a container and hosts are sitting next to each other, gVisor come into the middle with two layers; first, it will intercept the system calls and then forward it to the user line kernel, again ready and go. Now and after that when this user in kernel call sentry wanted to get help from this to get the help from the Linux kernel, then in between there's another layer called second and namespace, which will set apart this sentry and the host kernel host and next kernel. Those are two layers of isolation that would set, make detached container in the kernel sitting on the host.

[0:39:22.4] JM: There's these two terms, sentry and sec com, I believe you used.

[0:39:27.4] YT: Sentry is the user line kernel and sec com, S-E-C –

[0:39:31.7] JM: Sec com. Was that sec com?

[0:39:33.9] YT: It was originally called secure computing I think. For people, formerly they call it sec com, or sec com BPF together with the Berkeley packet filtering language, I will make much more flexible to define the second rules.

[0:39:48.3] JM: If I understand the model correctly, your application in the gVisor world, your application is running in this user line virtualized kernel that's written in Go and it's interfacing through sentry and sec com. Sec com. Comp?

[0:40:08.2] YT: Comp, exactly. Computing comp.

[0:40:10.2] JM: Secure computing. Okay. All right. Sec comp. I think I've seen that in a movie. You've got, okay the user line kernel, that's where your application is running in the virtualized Go user line kernel, it's communicating through sentry and sec comp and it's communicating with your host's user space area, which is communicating with the kernel space. Okay, that's right. Now wit that said, we can focus on the sentry and the sec comp area. What is going on in there?

[0:40:37.8] YT: You're talking about implementation?

[0:40:39.5] JM: Well, what is it doing?

[0:40:40.7] YT: Oh, I see.

[0:40:41.9] JM: Just higher level first. We'll get on implementation.

[0:40:44.9] YT: Okay. I'll just step back a little bit. In case of the traditional native container, the container work I said interface at host kernel. Basically, you can actually have more than 300 system calls from the application to the kernel that you can hit.

[0:41:01.3] JM: There are 300 – more than 300 unique system calls.

[0:41:05.1] YT: System calls and then –

[0:41:05.7] JM: What's example of a system call, so people are just more familiar.

[0:41:08.0] YT: Sure. When you want to open a file, a file is your photos or something, then you have open and you have read. When you want to write your tweet for some – let's say that tweet is actually sent, so yeah, you wanted to –

[0:41:22.4] JM: I didn't even see you on Twitter. I saw a bodybuilder by the same name, but I didn't see you.

[0:41:26.3] YT: Oh, really? That's strange. Okay. I'm not a famous person at Twitter at all. My colleagues are much more or less celebrities. Anyways, so there are a bunch of a system with just with a file operation, there's open and write and need. Those are the type of system calls, very simple one.

[0:41:42.0] JM: Okay. Right. Sorry. You got a system call.

[0:41:45.2] YT: Then instead of just letting the application to hit the 300th of those system calls and also the other surfaces like /profile system, etc. Instead of in case of gVisor incident led that, by having the sentry emulating the kernel behavior and then going through this second filter. The second filter actually reduces the interface of the system call from the order of that 100th, 300th to mostly 10s, like around 50 or 60-ish.

It reduces signal. The reason that we can reduce this, is because the sentry, the user line kernel does a lot of work and it only requires very limited number of system calls to when the sentry needs help from Linux, they will use this system call of course.

[0:42:39.9] JM: Interesting.

[0:42:40.7] YT: That's how the tech surface. I just mentioned, like I mentioned that they're two layers, right? The two layers too, but when you just forget about that there are two layers, what is the fundamental difference? 300 ways to actually attack kernel, there's a 60. 50 to 60.

[0:42:56.5] JM: Now one thing I'm inferring from what you just said is that there are a lot of system calls in Linux that are not that useful.

[0:43:04.8] YT: That's I think the challenging part, right? Again, it all comes back to the responsibility of the Linux kernel house. It has run, topple – first of all, it has to run a multiple device. It does run a multiple device in any more foreign factor. Once you actually have a software, I will just say like, I don't want to mislead that Linux is the only – a program. When a program has to run a multi-device, that's a big thing.

Now after that, once you actually expand the type of form factors, there are a lot of applications. There are a lot of application. For each application, you have system calls, different use cases. You need host support from your host kernel, or host kernel, or a program, then you put up more needs, thus the tech surface has to naturally grow.

[0:43:52.5] JM: What's an example of a rare – an esoteric system call? Do you know of any?

[0:43:59.6] YT: I think I cannot come up with a one. I think I don't have –

[0:44:03.6] JM: I mean, I can imagine if I've got – if I'm running Linux on a smart flashlight or something, maybe there's unique interfaces for battery or something, I don't know, power management perhaps.

[0:44:15.9] YT: Yeah. For example, okay is a little bit controversial to say so. When your program does not have to directly interact with the underlying devices, like I octo is a very strong system call. IO control. IO octo became later doing and a lot of things as far as the underlying kernel module can do. I wouldn't want to believe that random user on program makes that call all the time.

However, if we say because it's dangerous, we're going to get rid of it, is going to be impossible. Because there are a lot of the admin programs that still run – still on the user line, but need a lot of support from those devices. In that situation, that is the only way that you can communicate and get support.

[0:45:06.7] JM: This is a pretty brilliant design, because you're using Berkeley packet filtering, which I assume is super performance, right? I mean, this gives you security isolation, probably with very minimal latency increase. It allows you ultimately to focus on only the security vulnerabilities that could be associated with 10 system calls, or 20 system calls, whatever, 50. I don't know however many you got it down to.

[0:45:36.7] YT: I think it changes, but a 60 is probably the a good number to say. I think, we have a public documentation we're coming up, I think we're going to call the 60.

[0:45:45.9] JM: How does that improve the life of – because let's say you're able to deploy this to the entirety of Google Cloud for example. I'm a security researcher that works at Google cloud, why does this make my life easier?

[0:46:01.0] YT: A security person inside Google, is that –

[0:46:02.3] JM: Inside Google. Inside Google, yeah. Inside Google Cloud, let's say all the applications across a given domain that I'm working on have now gVisor that's standing in between the user space and the kernel. How does that improve my life.

[0:46:16.4] YT: I think one of the most important aspect of this gVisor is that it does not require any modification. For example, just to be clear, this problem is not new at all. Thus in the Linux kernel world, we mentioned second ring, that's exactly the way to filter the system cost, to build on its own sandbox is widely used.

Now on top of that, there are SC Linux and add warmer. The commander access control system, that will restrict what a process can, which resource a process can access. The challenge of these approaches is that it's very useful. It's definitely useful. If you write the policy properly, that is the most performant way of providing, adding defense and depth in the context in that kernel. On your other hand, the challenge is that how do you make those policies right? Who would define that hundreds of rules? Is it the provider side, or is it the user who have to maintain it? How do you make sure that it can be reviewed, or even maintained?

The constant desire that we've been seeing and we now we're seeing even the word of cloud is that I think we understand the security is really important, but security people have to do a lot of things. This is just only we're talking about sandbox in container. They have to think about authentication, key management, networking. There are a bunch of things that security people have to think through. As a consequence, there's really constant desire that at least per container, can we have something that we can just turn it on, just turn it on, no change into application, no management policy, just a clickable button or the flippable flag and then focus on other more important things.

I think that will make, at least hopefully it will free up people's time, the security people's time and then focus on the rest of the area of the entire end-to-end security. I think that is the way that I vision that how gVisor can be helpful inside Google, outside Google, anywhere.

[0:48:25.6] JM: Yoshi Tamura, thanks for coming on the show. Great talking.

[0:48:27.5] YT: Thank you very much, Jeff.

[END OF INTERVIEW]

[0:48:30.6] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly.

You can store and manage unlimited data. You can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your site's functionality using Wix code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix code's built-in database and IDE, you've got one-click deployment that instantly updates all the content on your site. Everything is SEO-friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There is plenty that you can do without writing a line of code, although of course, if you are a developer then you can do much more. You can explore all the resources on the Wix code site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix code experts.

Check it out for yourself at wix.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you could do with Wix code today.

[END]