EPISODE 769

[INTRODUCTION]

[00:00:00] JM: A Kubernetes cluster presents multiple potential attack services. The cluster itself, a node running on that cluster, a pod running in the node, a container running in a pod. If you are managing your own Kubernetes cluster, you need to be aware of the security settings on your etcd, your API server and your container build pipeline.

Many of the security risks of a Kubernetes cluster can be avoided by using the default settings of Kubernetes, or by using a Kubernetes managed service from a cloud provider or an infrastructure company, but it is useful to know about the fundamentals of operating a secure cluster so that you can hopefully avoid falling victim to the most common vulnerabilities.

Liz Rice wrote the book *Kubernetes Security*, with coauthor, Michael Hausenblas, a previous guest on Software Engineering Daily. Liz works at Aqua Security, a company that develops security tools for cloud native and containerized applications. In today's show, Liz gives an overview of the security risks of a Kubernetes cluster and provides some best practices, including secret management, penetration testing and container lifecycle management.

[SPONSOR MESSAGE]

[00:01:20] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that's really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte. Byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW]

[00:03:09] JM: Liz Rice, welcome to Software Engineering Daily.

[00:03:11] LR: Thank you very much. Thanks for having me.

[00:03:13] JM: Listeners may have studied container security, and we've done a show about container security, but we have not covered Kubernetes security. When you compare the world of containers to the world of Kubernetes, how does the security story change?

[00:03:31] LR: Huge amounts of it is the same, because you are running containers in a Kubernetes environment. The things that you have to take into account in addition when you're running Kubernetes are really how do you make sure your Kubernetes setup itself is – Well, that your control plane for Kubernetes is set up security, that you've got permissions and access control set up security. But a lot of the actual runtime security around containers is really very much the same regardless or whether you're running under Kubernetes or just running a Vanilla Docker container straight out of the box.

[00:04:10] JM: What kinds of Kubernetes vulnerabilities does this present in contrast to the world with just containers?

[00:04:18] LR: Yes. So Kubernetes is a complex distributed system, and like any complex system, it's going to have flaws and that essentially means that there will be some vulnerabilities. Some of those vulnerabilities will be essentially software bugs. We saw one actually about a week before KubeCon in December. A pretty serious one revealed itself or was discovered in the software itself. But an awful lot of the potential ways that an attacker might compromise a Kubernetes cluster come down much more to do with human error or configuration error.

Over the last year or so, perhaps a little longer, we've seen a lot more focus from the Kubernetes community on making the default settings a bit more secure, moving more towards secure by default. For example, if you install a modern version of Kubernetes, you're going to have role base access control enabled by default, and that wasn't the case 18 months ago, let's say. So things are improving all the time, but it's still very possible to configure your cluster in a way that makes it disastrously accessible.

Another really great example was the case of Tesla, probably a year or so ago again, where they inadvertently exposed their Kubernetes dashboard to the internet. Really, it's a configuration error, it's human error. It's been made harder to make that kind of mistake now, but it's still possible.

[00:05:58] JM: The rise of configuration errors seems to be related to the fact that Kubernetes is this system that has a lot of declarative code and the configuration itself is declarative. Are there new kinds of paradigms for securing our systems that are new to people who are used to having more of an imperative programming system? Because I think in Kubernetes you're more describing the way that you want the system to be rather the kind of like actual programming your application where you've got this imperative code and you're saying, "Okay. First do this, then do this, then do this, then do this." Configuration is more like, "Here is the state of affairs."

[00:06:47] LR: That is a great question in terms of whether that approach, that taking that declarative approach has specifically different security impacts. I can't think of anything immediately that changes, because we do it that way, except for the fact that, yeah, things are going to change – They're not going to change, but they're going to be recreated for you without direct intervention.

So for example, if you have a deployment in Kubernetes, that deployment might know to keep a certain number of instances of a pod up and running, and you can delete those pods. Let's say you discover that those pods are vulnerable. If you delete those pods, you actually need to know that they will be recreated again automatically because the deployment exists. You need to update the deployment to a newer version of those pods. I suppose there are things that you need to know how Kubernetes is going to be affected by the changes in that declarative configuration that you're making. Maybe sometimes those aren't entirely obvious.

[00:07:56] JM: Kubernetes clusters have a variety of components. There's the cluster itself. There's the example of a node running on a cluster. You've got a pod running inside of a node. You've got a container running inside of a pod. Are all of these components potentially an attack surface?

[00:08:14] LR: Yes. Yes. You'd have to say that. I think in any system, any piece of software is a potential attack surface. One of the principles that we try to apply, well, in any system, but we can say particularly in a containerized environment, the less software you're running, the better. That could be inside your containers. You want to try to minimize the code that's contained within each container image that you're deploying, because if code isn't there, nobody can exploit a vulnerability in that code.

The same thing also applies to each host in the cluster. The ideal situation is that the host is only running sufficient software to run your Kubernetes cluster and maybe some supporting functions like logging, but you don't ideally want to have any other software installed on those machines exactly to reduce that attack surface.

[00:09:12] JM: Securing the node where the master Kubernetes API server is running is quite important. Could you explain why that is such an important task for Kubernetes operators?

[00:09:26] LR: It really comes down to the fact that if you have full control over the master mode, if you have API control over all of the – Or full access to the full API through that master node, you effectively have – It's like root on your entire cluster. Now, there are lots of things that you can do particularly with role base access control that I mentioned before where today's

defaults out of the box make it less likely that you're going to give away those root privileges and everything and everybody, but you do need to be careful about the access that you give away. Because as I say, it is effectively — People talk about Kubernetes as being a little bit like a distributed kernel. I think that's more of a metaphor than really a fact, but you can think of admin access to the Kubernetes API as equivalent to root access to a machine. That API access can give away everything.

[00:10:31] JM: What are some security defaults we can put in place to protect that Kubernetes master API server?

[00:10:40] LR: So a lot of the kind of really fundamental things are now set for you by default with the – Certainly, with the installers like you by DM. They'll do things like insure that you can't – Or if you access that API anonymously, you can only query for basically things like health. You set by default an anonymous user can't create or destroy pods or figure out what deployments you have running, figure out what name spaces you have.

[00:11:12] JM: Another important API or a critical API in Kubernetes is the Kublet API. So the Kublet is the component that runs on every node and keeps your pods running, starts and stops pods on that node. Anybody who has access to that Kublet API can effectively run software or inspect the existing running software or interfere with existing running software on that node.

Again, by default, you won't have anonymous access to that Kublet API, but you want to keep it that way. There are tools out there to help you assess that. Actually, I'm a maintainer of a project called Kube-Bench, which is an open source tool for running the Center for Internet Securities benchmarks on the configuration of your Kubernetes cluster. So there are benchmarks on the configuration of your Kubernetes clusters. So those benchmarks will basically check whether or not you have these configuration settings applied in a secure way.

You might not want to follow every single one of those guidelines. You might have reasons why in your environment you want to do things differently, but at least running those benchmarks against your cluster will give you a good idea if there's anything badly wrong with your configuration.

[00:12:36] LR: Let's say I've configured my cluster. I believe it to be secure, and I run Kubernetes benchmarking. What is it going to test? What are some of the elements of the Kubernetes benchmarking tool that you work on?

[00:12:51] JM: Yeah. So you can run Kube-Bench on [inaudible 00:12:54]. So you would run it on each node in your cluster. So there are different tests around on the master node and on the worker nodes, and they'll check the configuration that's been applied to the different Kubernetes components. So on the master node, it will be checking various settings on the API server and on your etcd components. On your worker nodes, it's going to be checking the configuration for Kublet, and all these various different components where they have configuration settings Kube-Bench is essentially checking whether they are set up appropriately. So you would run that on your cluster within each node. You can actually run that tool as a Kubernetes pod. So you can schedule that to run on each node in your cluster.

Another tool that I'm involved with takes a different approach, and this is something called Kube-Hunter. Again, it's another open source tool. It is basically penetration testing for Kubernetes clusters. So you can run this outside of your cluster and see what an attacker might see. It essentially tries to make network requests to the various different components within your cluster, and if it finds things that it can access, it will report those to you. You can also run it inside a pod in your cluster, and then that's giving you the perspective of if an attacker managed to compromise a pod, somehow they manage to get inside one of your pods, perhaps through a vulnerability in some application that you're running in a container. If they've got that access, what can they then do on your Kubernetes cluster? That's really interesting because of things like if you're a pod, you run as a service account. It's essentially like an identity that that pod is running under.

So if you compromise a pod, you can get hold of that service account credentials, and something like Kube-Hunter could then say, "Well, what can I do now I've got these credentials?" You could see what the kind of blast radius of that kind of attack might be. So it's pretty fun to play with.

[SPONSOR MESSAGE]

[00:15:14] JM: MongoDB is the most popular, non-relational database and it is very easy to use. Whether you work at a startup or a Fortune 100 company, chances are that some team or someone within your company is using MongoDB for work and personal projects.

Now with MongoDB Stitch, you can build secure and extend your MongoDB applications easily and reliably. MongoDB Stitch is a serverless platform from MongoDB. It allows you to build rich interactions with your database.

Use Stitch triggers to react to database changes and authentication events in real-time. Automatically sync data between documents held in MongoDB mobile or in the database backend. Use Stitch functions to run functions in the cloud.

To try it out yourself today, experiment with $10 in free credit towards MongoDB's cloud products; MongoDB Atlas, the hosted MongoDB database service, and Stitch. You can get these $10 in free credits by going to mongodb.com/sedaily. Try out the MongoDB platform by going to mongodb.com/sedaily. Get $10 in free credit. It's the perfect amount to get going with that side project that you've been meaning to build.

You can try out serverless with MongoDB. Serverless is an emergent pattern. It's something that you want to get acquainted with if you're a developer, and getting that $10 in free credit to have a serverless platform right next to your Mongo database is really a great place to start. So go to mongodb.com/sedaily. Claim that credit, and thanks to MongoDB for being a sponsor of Software Engineering Daily. I love Mongo DB. I use it in most of my projects. It's just the database that I want to use.

Thanks to MongoDB.

[INTERVIEW CONTINUED]

[00:17:22] JM: You’re giving us some perspective into the open source world of Kubernetes in your discussion of these tools that you work on. So you’ve talked about two tools, the Kub benchmark tool and the Kub-Hunter tool. The first one is the benchmarking. Just checking if

you've got your configuration settings in a same manner, and the other one is the pen testing tool. How have these open source tools changed overtime?

[00:17:54] LR: Yes. So with Kube-Bench, we very much want to reflect what the CIS benchmark specifies. We've taken the view that we're not trying to say what should the settings be. If we disagree with the spec, we should go and report that and raise issues with the CIS benchmark itself, but we will implement that. The CIS benchmark gets revised less frequently, let's say, than Kubernetes itself. So maybe every two or three Kubernetes releases we get an updated version of the CIS benchmark published. I'm actually also involved as an author on that particular benchmark. So I know the kind of processes that we're going through to get that spec released, and it's not something that we've been able to get out quite in step with the Kubernetes releases.

But what we've seen over the last year or so is some of those every release in Kubernetes will typically improve what the defaults are and the spec's recommendations will be the same, but quite often the defaults have improved. So you're more likely to – When you run the test, you're more likely to see the results you want to see just by running the out of the box defaults.

Kub-Hunter on the other hand, basically the sky is the limit in terms of the creativity, if you like that, and a hacker could apply to trying to breach a cluster. You could apply the same creativity to inventing tests to go into Kube-Hunter and try to exploit a cluster in different ways. Yeah, overtime it has grown. We've seen new tests being added and new tests being suggested, and long may that continue.

[00:19:46] JM: Do you know if people ever use the Kub-Hunter pen testing tool on clusters that they don't own to see if they can find a vulnerability and install cryptocurrency mining on those clusters? Not to give the listeners any ideas.

[00:20:04] LR: Yeah. So when we first released Kub-Hunter, we did kind of – We worried about that, but the truth is that if you're actually one of the bad guys, you've got tools anyways. There are lots of more general purpose exploit tools that can be used for good purposes and they can be used for bad purposes.

Obviously, we tell people you shouldn't be running this except against your own cluster. We can't police that unfortunately. But we took the view that this is a tool designed to help people who are operating Kubernetes clusters and aren't necessarily always security experts. It's a good idea to kind of democratize security by giving people tools that make it easier for them to see what potential holes there might be, because you can guarantee that even if the good guys don't know what the holes are, the bad guys know how to find them. They have these scripts that are trying to exploit any open port, any piece of software that they can get a grip into. You only have to standup a website and you'll see attacks. You'll see requests from IP addresses that you find surprising, and that's because people are scamming the internet looking for things they can steal essentially. So that's happening anyway. We may as well arm the good guys with some tools to help them defend themselves, and that's the view we took with Kube-Hunter.

[00:21:37] JM: To come back to configuration, etcd is a tool that is used for shared configuration in a Kubernetes cluster. What are the potential vulnerabilities that can come from an insecure etcd?

[00:21:53] LR: Now, I am not sure whether this is still the case in the latest release, but it certainly used to be case that etcd was not encrypted by default. Essentially, etcd is storing all of the state information for Kubernetes configuration, and it can also be the store for Kubernetes secrets. If doesn't have to be. You can set up other backend storage solutions, something like Hashicorp vault or a key management system. But out of the box, your Kubernetes secrets will get stored in etcd, and they get Base64 encoded, so they're not human readable.

But Base64 encoding is not the same as encryption. It doesn't require any kind of key, and it would basically mean anybody who can get on to that master node where you're – Well, any node that's got an etcd instance running on it, if they can get at that etcd API, they can extract all your secrets, which is a very bad idea, and there may be reasons why you have people who have writes to access –The internal people who makes legitimate sense to access some of your Kubernetes information, configuration information, but you don't have to give everybody the credentials to your password, your databases. You really want to minimize who you give credentials to and who has access to those credentials.

So etcd – Not encrypting your etcd cluster and using it for secrets has certainly been a fairly common scenario that we've seen people do and not really realize the risk that they're running. It's not like the worst thing in the world, because your etcd cluster is probably not accessible directly on the internet, but anybody who does have access to that cluster or who manages to get access to that cluster making it easy for them to get hold of those credentials is really opening the door wide open to giving them whatever confidential information you might have.

[00:24:08] JM: My cluster consists of containers. Container images are built from code and put into a registry, and then they're taken from the registry and built into containers. What are the security vulnerabilities in that lifecycle?

[00:24:23] LR: Cool! It's quite a long list. Yes. So when you build that image, you start from some source code and you're building your container image, you're probably pulling in some third-party dependencies. Maybe not all languages do, but most languages, most software out there is using some other third-party packages. You want to know that you're not building in known vulnerabilities as you build those container images.

So whenever I'm asked what's the first thing I should do to secure my secure container deployment, my go-to answer is use of vulnerability scanner. Make sure that your container images don't contain known high-severity vulnerabilities. There are plenty of vulnerability scanners out there. There are commercial solutions like the one that we have Aqua. There are free solutions. We actually have one called MicroScanner. There are open source solutions like Clair.

So there are lots of options out there and some of them are free. Some of them are open sources. No excuse not to do vulnerability scanning, and that is probably the single most – If you pick one thing, that's probably the most effective thing you can do to secure your containers.

Then you talked about putting them into a registry and then they're going to be pulled out of that registry at deploy time. A lot of, let's say, enterprise users will use a private registry to make it much, much harder to have the possibility of things like man in the middle attacks, or some

issues that means the software that you pull from that registry isn't exactly the same as the software that you push to that registry.

Another solution to that kind of thing is essentially fingerprinting, admission control, and this is actually powerful. We have in Aqua where we can fingerprint the image at the point where you're pushing it into the registry. Then when you pull it, if the fingerprint doesn't match, we don't allow it to be deployed. But making sure that that image that you run is exactly the image you expected to run. That's another important step in the security of that container.

Then I guess the last thing that I would talk about when we're thinking about the lifecycle of that container is runtime protection on that container. So this is kind of advantage that you get when you move to containers. You're probably talking about – Even if you're not talking about microservices, you're probably decomposing your software into smaller components than you would have run in a traditional monolithic environment.

Oftentimes, we are talking about microservices where we know exactly what that container is supposed to do. We can pretty easily learn or build a profile of what normal behavior for that container looks like.

So say for example if you're running an NGINX container, the container image – If you pull the image from a public repository, maybe you pull that image from Docker Hub, it's based on a Linux distribution. It's got lots of different executable in there, lots of those that are standard things. But the only functionality you need from that container image is the NGINX executable itself, and if you're running a container that's supposed to be running NGINX, if you ever saw it doing anything other than NGINX, if you ever saw it trying to execute a different binary, that could well be a sign of an attack. It's certainly a red flag, and that's what security solutions like Aqua are doing at runtime to help people detect and prevent attacks at runtime in containers.

[00:28:20] JM: There are also some open source projects around ensuring the integrity of container images. There's TUF and Notary. Describe what these projects do.

[00:28:31] LR: Yes. So TUF stands for the update framework, and Notary is an implementation of that framework. Yeah, it's about signing images essentially. Knowing that – Again, it comes to

this question of knowing that the image you're deploying is the software that you expect it to be and that it was signed by the provider that you expected it to be signed by, whether that's internal or some software provider, and also that it's the correct version of that software. So you want to know that a possible attack is that there's an old version of the piece of software that has an exploitable vulnerability in it.

There's a newer version in which that vulnerability is been patched. If an attacker were able to get you to deploy the old version when you thought you were deploying the new one, they would still have that exploit available to them. So one of the pieces of the TUF spec of what Notary implements is making sure that you are getting the latest version. It's not just that it's signed, but it's the latest version that was signed.

[00:29:44] JM: And there are these tools for detecting vulnerabilities in the containers, the container image scanner. Can you talk in more detail about what a container image scanner does?

[00:29:54] LR: Yeah. It's starts from the idea of the vulnerability database. There is I think called the NVD, the National Vulnerability Database, where every known or reported exploit is listed. They all have an identifier called a CVE, and you can – the NVD maps between these CVE identifiers and the package inversion in which that issue exists.

So let's say you are looking at a package like SSL and you could go to the NVD and lookup SSL and find which version of their package have which vulnerability, if there will be a set of vulnerabilities for every version. At the sort of basic level, all vulnerability scanners are looking at the packages inside an image and checking them against a vulnerability database starting from the NVD and saying, "Well, does this – For each of these packages, what are the vulnerabilities and how severe are they?"

Now, the problem is that the NVD only knows about the released version of a software package. It doesn't know about patches that might have been applied, and different Linux distributions will have different approaches to batching. So for example, if you're using a Debian flavor of Linux, it will – They tend to not want to take a new version of packages terribly frequently, but they'll

opt instead to patch or back apply that patch to the version of the software that they already have in their version.

So they have to fix for the vulnerability, but the NVD doesn't necessarily know about it. So the vulnerability scanner ideally wants to take into account not just the NVD, but it will say the information about patches that is made available by at least different distributions, and you get similar information from different software vendors and different languages and so on, where the NVD is a good sort of source of the raw information about what vulnerabilities are on which packages, but if you look at that and that alone, you'll have a tendency to see a lot of false positives. By taking into account more of these information about patches, you would hopefully see fewer false positives. I think if anybody ever says you're going to see no false positives, then I wouldn't believe them. I would be highly skeptical. But taking into account more sources of information will give you a better level of accuracy in those vulnerability reports.

It's quite a complicated process, and I think as a user, it's actually quite difficult, because if you try three different vulnerability scanners and they all give you three different answers to how vulnerable your image is, you have to have trust in the supplier of that scanner. How do you know as perhaps not necessarily a security expert, how do you know which of those reports is true? And that is basically just a matter of trusting your vendor.

[00:33:16] JM: Image scanning is an example of a security procedure that is often integrated into the CI/DC process. Describe in more detail how awareness of security effects our continuous integration pipeline.

[00:33:34] LR: So we quite often hear the term shifting left, and that's the idea of trying to apply security earlier in that pipeline. So if it's a left to right pipeline, we're shifting security left. A big part of that idea is really saying we used to have a much more kind of monolithic approach, where somebody would write software, developers would write software and then operations teams would figure out how that software is going to be deployed and security teams come along right at the end of the process to kind of figure out whether or not machines need to be patched and what firewall ports need to be opened and that kind of thing.

With this move to CI/CD and a move to devops or you could even say dev sec ops, we're talking about trying to move a lot of that thinking much earlier in the cycle, and where we can only much like trying to fix things earlier in the software development lifecycle. The earlier we can fix things, the quicker it is, the more quickly we're sort of iterating essentially the less expensive it is.

So rather than if we applied security right at the end, maybe we create a container image and it goes through testing and it goes through staging environment and what have you, and then at the point where we go to production, it turns out that that image has got some vulnerability or a based image that the security team aren't happy with for whatever reason.

[00:35:07] JM: The CI pipeline, anything else to say in that regard?

[00:35:10] LR: Yeah. I was talking more sort of generally about this idea of shifting left and trying to make it so that developers can take more responsibility for security issues by doing things like if we do a vulnerability scanner, part of the build process is part of the CI/CD process before they push the image to the registry. Then if there is a high-severity of vulnerability, they can fix that before it goes anywhere near getting deployed into production, and they can fix these straightaway and they know what it is that they have – Why do they need that third-party dependency. So that's kind of the benefit of doing that scanning earlier on in the process.

[SPONSOR MESSAGE]

[00:36:01] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real-time.

See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java. Start monitoring your microservices today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:36:56] JM: Let's talk about another best practice area. So one best practice around managing your containers and being security aware is building your container images to keep the container and its functionality small, and this gets into more of a software architecture question. But do you have any principles around keeping the container size down?

[00:37:24] LR: We talked a little bit about attack surface earlier on, and that aside from just making an image smaller and easier and quicker to pull, you're also reducing the attack surface if you can keep it small. So definitely something to try to do.

I think to some extent it depends on the language that you're working in. If you're using something like Go that can compile down to a standalone binary, then you can ship containers with just that binary in it. Maybe there's some supporting configuration files or something, but you don't need any other executables, and that's great. But you can't do the same thing if you're using – I don't know, a language like Node or Python, or any language that doesn't compile down to a standalone binary. You are going to have to ship some supporting packages.

Oftentimes the best way to do that is to ship a Linux base image that has then got those whatever the requirements are for your language kind of pre-installed, and then the dependencies, and then your software. Part of this can be – You have choices about what your base image Linux distribution is. So Alpine is a very popular choice in the world of containers for reducing the size of that image, because it's designed to be small.

I would say that not all enterprises are comfortable running with Alpine Linux. It's, I guess, less of a known entity, and you know what I was saying about vulnerability management and how it varies from distribution to distribution. It's quite interesting when we'd look at Alpine, because you can see in the sort of manifests for a package that Alpine distributes, you can see that they've applied – It's all open source. You can see that they have applied patches to address vulnerabilities, but you have to kind of do kind of quite a lot of research to check whether you agree with that patch. Probably 99 times out of a 100, they're great patches and perfectly

correct. But I think for that reason, some enterprise are nervous of using Alpine as that kind of base image.

Other choices, you can end up with a very bloated image because you chose a large base image distribution. So trying to – I would definitely say don't try to install extra software into that base image. Kind of classic thing that used to happen, I don't see this so much anymore, but you used to see people assuming that they're going to treat containers the same way they would treat a host and thinking that they're going to have to do things like SSH into a container, and you don't need to do that. We're going to treat these containers [inaudible 00:40:25]. You don't like somebody in that container, you don't change software in it. You destroy it, rebuild the image and redeploy a new version of that container.

So that means you don't need things like SSH installed into your base image. So you can definitely avoid putting in a lot of software that you might have otherwise had in the machine image that you used for a host.

[00:40:50] JM: Containers get networked together. They are communicating with each other. What vulnerabilities can come from insecure Kubernetes network policy?

[00:41:01] LR: Wow! So usually when we're talking about networking in a security context, the networking per se doesn't kind of create a vulnerability. It creates a way for somebody to get in or move around, particularly if they've already discovered a vulnerability. Supposed we've got a container that has got an exploit it. So you've been able to compromise that container, depending on what the network policies are, you may be able to access other containers. You may be able to access other host machines. You may or may not be able to kind of open access to the outside internet.

Now, ideally, you want every container to only be able to perform the kind of network open to destinations that it needs to. So we things like service meshes that will ensure that you've got encrypted like SSL connections between pairs of services so that nobody can intercept the traffic between them. So that services can only communicate with services that they're supposed to. Then that will mean that if you have got a compromised container it's harder for

the attacker to kind of use that to communicate. They can only communicate with the containers that that container is supposed to communicate with.

Similarly for sort of Kubernetes network policy, you can kind of – I'm not going to say it's a replacement for sort of firewall, but you can use it to add a layer of firewall-like protection to limit the kinds of network traffic that can flow, and that's a useful layer of defense. Probably it's really about limiting the last radius if an attack happens.

[00:42:53] JM: You touched on secret management a little bit earlier in our conversation about etcd. Every application has secrets to manage. Where are secrets stored on a Kubernetes cluster and what kinds of policy should I have around managing these secrets?

[00:43:13] LR: Right. So Kubernetes secrets, there is an object type in Kubernetes or resource type in Kubernetes called a secret. So they are treated as first-class citizens within Kubernetes these days. But it really is a matter of how you store that secret. So you can – By default, it will go into the etcd cluster. As we were discussing earlier, you'd want that etcd cluster to be encrypted if that's where you're storing your secrets and it won't be encrypted by default.

But you have other options as well. So you can plugin – The architecture allows for plugging in different storage solutions for those secrets. I know there's a plugin for Hashicorp vault. So that means that your secret values is – Imagine there are others I'm not sure exactly which ones are available today. But it means that your credentials, the actual secret values are going to be stored in some component that is designed for storing these things, encrypted and ensuring that they're accessible only by entities or people with the appropriate privileges. So that storage is really important.

At Aqua we do some interesting things around how you can inject secret – Essentially, we enable that backend integration with these different backend stores. Name a backend store and we very likely have an integration with it, and making sure that those secrets can then only be injected into the right containers at the right time.

Kubernetes actually supports a couple of different options for this. You can get the secret into the container, lay that as an environment variable or as a file that's mounted into the pod as you

start it, as it's started. It's generally considered better practice to use the file version. But if you're using environment variables, that is a – It's fairly common to see a piece of software that if it crashes or if a problem occurs, it's quite common to see the environment logged out. If that log contains your secrets in the clear, and so it'd be if they're an environment variable, then anybody who can see your logs can see your credentials. You probably didn't want that to be the case.

So for that reason, it's generally considered better practice to use the file approach, where your code is going to need to read the secrets out of a file that's been mounted into the container as it starts. But the file is actually a temporary file. So it's not actually written to disc, which is nice.

[00:46:04] JM: Whereas you hear about containers is that they are cattle and not pets. You should be able to lose any specific container and have it be replaced and not be concerned that this container went down and had some relevant state that you can't recover from losing. Should we take this to any particular extreme in the case of security? Should we be cycling out our containers on a regular basis to ensure that there is not some sort of attacker who has nestled into our container system and just sitting there doing stuff or watching our traffic? Should we by cycling our containers?

[00:46:54] LR: Yeah, it's a really good idea. There's actually a concept called reverse uptime, which is the idea that your container should not be allowed to leave beyond a certain number of probably days. But yes, just cycling them, and you could cycle them much more frequently. I once saw a demo. I think it was a DockerCon, where they were cycling every five minutes just for entertainment value I suppose at the conference.

But yes, absolutely. You should – It's good practice to cycle those containers. I mean, partly if you've got an attack in progress, as you say, anything that's untoward will be reset. That attacker's foothold is lost, and it can also be helping you with malicious issues. Maybe you've got a container with a small memory leak happening. Kill the container, that issue goes away. It's going to be constrained by – You've probably got C groups limiting the amount of memory it can use anyway, but it's another nice thing you can do.

You can also apply it – We've been talking about it at the container level or the pub level. You can also apply this cattle not pets principles to your nodes. Essentially just rebuilding and redeploying your nodes every so often. It's exactly the same principle. If somebody had got a foothold into that node, well, you'd blown it way. So they're going to have to start again from scratch.

But the other thing I really like about that approach is it's kind of a fire drill for what happens when your nodes just go down, which it's going to happen sometime. So why not make sure that everything acts the way you expect it to. Do the under control conditions and then if something goes wrong or somebody – Whatever happens in your cloud provider, that everything is set up to deal with that.

[00:48:56] JM: You work at Aqua Security, and one subject of the show that recurs is the question of build versus buy, and we've explored many of the fundamental principles of Kubernetes and container security so far. What kinds of tools would I potentially get from a vendor like Aqua Security?

[00:49:19] LR: So I think there is always that kind of tension between open source versus proprietary, and security is one of those things where enterprises have valuable data. They need to apply appropriate measures to protect that data, and they can either hire a whole load of people who are security experts or they can use tools to reduce the amount that needs to be done manually or the amount of code that they would need to write to do it themselves.

So if you come to a vendor like Aqua, we have things like the vulnerability scanning that we talked about before. We have image assurance, which is essentially making sure that the images you're running are the images that you expected to run, that they match precisely what you scanned. So they match the fingerprints that is the software you intended to deploy. Runtime protection, so making sure that containers behave the way they're expected to. Then if they start doing anything anomalous, you can either prevent that or get an alert on it.

That kind of thing, particularly the runtime security, there is an open source solution called Falco, which can alert you to issues, potential anomalies, but the prevention is not something

that I'm aware of and there are other commercial solutions, but I'm not aware of any open source solutions that provide that prevention side of things.

I think the other thing that you're doing when you work with a vendor, part of it is just the integrated story. You have everything told together in a nice dashboard so you can see what the security status of your deployment is. But you're working with a team that you hope – That you can call on to – We've not just throwing software over the wall and saying, "Get on with it," because oftentimes enterprises have complex security requirements or concerns and they can come and talk to our team, our solutions architects, who've worked with dozens of other enterprises and they've probably seen the same problems and the same concerns before. So they know what the best practices are, and I think that's an important advantage that you get. You're not just buying software. You're also getting the ability to leverage expertise that's been built up across working with dozens of other enterprises.

[00:51:51] JM: Okay. Well, Liz Rice, thank you for coming on Software Engineering Daily. It's been really fun talking to you and it's, of course, been a pleasure meeting you at KubeCon and having our various conversations. I hope to see you at the next one.

[00:52:05] LR: Likewise. Thank you so much for inviting me again, and yeah, pleasure to spend this time with you.

[END OF INTERVIEW]

[00:52:14] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[END]