EPISODE 1051

[INTRODUCTION]

[00:00:00] JM: Zoom video chat has become an indispensable part of our lives. In a crowded market of video conferencing apps, Zoom managed to build a product that performs better than the competition. Scaling with high-quality to hundreds of meeting participants and millions of concurrent users. Zoom's rapid growth in user adaption came from its focus on user experience and video call quality.

This focus on product quality came at some cost to security. As our entire digital world has moved on to Zoom, the engineering community has been scrutinizing Zoom more closely and discovered several places where the security practices of Zoom are lacking.

Patrick Wardle is an engineer with a strong understanding of Apple products. He recently wrote about several vulnerabilities he discovered on Zoom and he joins the show to talk about the security of large client-side Mac applications as well as the specific vulnerabilities of Zoom.

Before we start, I want to mention FindCollabs, the hackathon product that I've been building. FindCollabs is free for schools and nonprofits, and if you're looking to run a hackathon for your product or your company, you can check it out as well at findcollabs.com.

[SPONSOR MESSAGE]

[00:01:23] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and drop-downs, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers

can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW]

[00:03:00] JM: Patrick Wardle, welcome to the show.

[00:03:01] PW: Thanks, Jeff. Stoked to be on talking already with you about I guess breaking Zoom.

[00:03:06] JM: Indeed. Most of the episodes that we do on this show are about building web services that you access through the browsers. Developers are using cloud APIs and open source software, and this episode is going to do some coverage of a heavy client-side application on Mac, on a proprietary operating system. In many ways, this is going to diverge from a lot of the shows that I do.

On Mac, Zoom interfaces with the closed APIs of the Apple ecosystem. How does building on top of a closed ecosystem like Mac differ from building on an open source ecosystem like Linux?

[00:03:49] PW: Yeah, that's an excellent question, Jeff. I think kind of as you mentioned, we're going to be focusing more on a native client, and I think some of the issues that we'll see and we'll talk about are a result of that designed decision. I think you alluded as developers perhaps interface with proprietary APIs, a lot of the bugs or flaws that I found were related to the utilization of insecure APIs, perhaps not understanding the security implications of interfacing with those functions, those interfaces. That maybe because, again, they're proprietary closed sourced and perhaps not as well-documented as they could be.

I think there's definitely some risk from a security point of view when you're developing native clients if you don't fully understand the APIs. Also, as a developer, perhaps you're also not reading the back limitation as close as you could be, because to give Apple credit, they do provide some warnings against using or rather not using, not recommending a variety of these application programming interfaces.

[00:04:53] JM: The vulnerabilities that we're going to talk about, from what I can tell, they sidestep some of the norms, some of the best practices that Apple puts in place to maintain a secure application ecosystem. But the reason that Zoom did that is because, well, some people might say that Apple has an unfair advantage with their own platform. They preinstall FaceTime, and maybe it's only fair that Zoom uses some of these APIs in ways that maybe they're a little fast and loose with security, but they actually give the user a faster install experience. They may be give a user an experience with fewer security popups. Is there anything wrong with that?

[00:05:45] PW: That's an excellent question, and I think the answer to that is, as a user, what's more important? Usability or security? I think that's exactly the path Zoom took, where they focused on a very seamless experience, less button clicks, less authentication prompts, etc., etc. That made for a very user-friendly product, which became incredibly popular. But unfortunately, that was often at the cost of security.

I'm glad we're kind of mentioning the Apple component here, because as Apple locks down Mac OS more and more, a lot of times they're almost putting third-party developers at a disadvantage, because now, for example, if you're creating a security tool, the user has to just jump through all these hoops and open these various dialogues and manually approach certain applications, and it's just honestly a wreck, because a lot of times users will skip things and then the tool doesn't work and then the users will blame the application developers. Wherein reality, there's some responsibility. I would argue that Apple has kind of pushed some rather Draconian security constrictions.

I don't know what the answer is, because again, it's always a balance between usability and security. From a purely security point of view, everything Apple is doing, it's generally great, recommended. But that does impact the user experience. So then you see third-party developers or companies, for example, Zoom, kind of trying to skirt some of these problematic

or less than ideal user experiences, which make the product more user-friendly, but then may introduce security risks or issues.

[00:07:22] JM: One early vulnerability in Zoom that was discovered back in July allowed the Mac Zoom client to enable the user's camera without permission. How was the Zoom client forcing a user's webcam to turn on in this vulnerability that was discovered back in July?

[00:07:45] PW: Yeah. My understanding was if the user had Zoom installed, there was kind of this like Zoom webserver that would get installed in the background listening for requests. I believe it was designed or the idea was for it to only talk to the application. The security researcher found that a malicious website could send a request to that web service that was running. I believe what happen would be that web service would then proxy the request to the Zoom application or Zoom plugin that was installed, which had been preauthorized by the user to access the webcam.

The issue was it should not have been allowed to accept unauthorized remote request to turn on the webcam. I think that's the example of Zoom designing a product that's very user-friendly, but not putting security and privacy at the forefront or not really designing those, baking in those in from the ground-up. Because obviously, malicious websites should not be able to talk to local web service that's running in the background that allows you to arbitrarily turn on the webcam and spy on the user. Basically, proxying off the installed Zoom client.

[00:08:51] JM: What's a better version of that implementation. In this vulnerability, Zoom had installed a webserver on the user's computer and the problem was that that webserver was just more accessible, or what was wrong with that? What was the bad practice here?

[00:09:07] PW: Yeah. My understanding is this component that was listening for connections. Again, I believe it was designed to just listen or proxy perhaps a local request between the application or maybe from the browser to the client. Clearly though, they did not understand the attach scenario or fully lock it down, authenticate perhaps who is making the request. Again, my understanding was the design to only be used between the various Zoom components. There should have been some authentication of who's trying to talk to it and not let a random website kind of make this request.

It was kind of interesting, because once this came out, Zoom recommended people update. I believe they've removed the webserver component because there're better IPC mechanisms that you can utilize. But I think there was a flaw in the uninstaller where actually web service would still hang around even if you uninstall the Zoom client.

Apple actually pushed out an update via their malware removal tool, something called MRT, where the signature to detect and remove that web service component. Apple basically made the decision that that put enough Mac users at risk that they were going to use their, essentially, AV-like Daemond, and just remove that. Again, that brings an interesting point going back to the proprietary nature of Mac OS. I guess in Cupertino can arbitrarily remove software on your system. That's an interesting thing to consider.

[00:10:36] JM: As an example for how the Apple ecosystem works, can you talk more about that malware removal tool? This piece of software that sits in Mac OS?

[00:10:46] PW: Yeah. There's actually two pieces of AV-like tools or utilities that run and/or baked.

[00:10:53] JM: AV meaning antivirus.

[00:10:54] PW: Yes, antivirus, that are built-in or baked into Mac OS that Apple really doesn't like to talk about, because in a way it's acknowledging that Macs get viruses. Of course, all security researchers know this to be true, but the Apple market and department doesn't really like to highlight the fact.

The first one is the one we just mentioned. It's called MRT or the malware removal tool. It's a daemon that runs in the background. Occasionally, Apple will push out updates. It's kind of interesting, because the updates, these are new signatures to detect new malware or adware, or perhaps invulnerable Zoom clients that Apple has deemed necessary to be removed, right? That's the R in MRT. These signatures are actually compiled into the daemon. When Apple wants to update signature database, they actually have to push out an entirely new compiled daemon, maybe not the most extensible design.

Runs in the background though automatically, and if it finds any files or processes that match signatures, it will kill the process and remove them. We see that it's being used to remove backdoors, adware, or in some cases, legitimate third-party software that Apple has deemed to be a risk or put users at risk.

The other piece of antivirus-like technology on Mac OS is called XProtect. XProtect is more similar to a traditional antivirus kind of on-demand standard. It has a separate signature database that Apple updates every few weeks. What it does is when the user downloads a file from the internet and double clicks it, the system will intercept that request and scan that file to see if it matches any of XProtect's signatures. If it does, the user will be shown a popup saying, "Hey, this file contains malware," and Apple will block that.

That's good, because we still have unfortunately a rather large number of Mac users who indiscriminately download random applications or pirated their files or applications that may contain malware. Then when they go to click them, they're basically infecting themselves. Apple has designed some mechanisms to kind of jump in front of the user. Scan those files first to make sure that they're not about to infect themselves with known malware.

[00:13:00] JM: When Apple deployed this malware or it used this malware removal tool to take out this webserver component. I realized, this wasn't your security research. We'll get to your security research in a moment. In this instance of using the malware removal tool, did Apple break the functionality of Zoom?

[00:13:25] PW: That was definitely a risk, but my understanding is there were some – I would imagine, some extra checks to see that the rest of Zoom was either uninstalled or a newer version that did not have that dependency was running on the user system.

I think Apple who do take security and privacy very important generally put user experience first. I can't imagine they would have pushed out an update that would have broke existing Zoom clients. I didn't hear any discussion or users complaining about that. I would have to guess, have to assume that Apple did that in a way that wouldn't break existing Zoom functionality, because yes, even though that webserver component exposed a remotely exploitable

vulnerability. I think it would have been more problematic for Apple to indiscriminately break a large number of Zoom clients especially because this update was pushed out with no user interaction or no user acceptance or confirmation, which again I think is a very interesting takeaway from all of that.

[00:14:22] JM: In this context, how does Apple know what to change in Zoom? How does Apple know what Zoom component to uninstall? I assume Zoom is a proprietary binary. How does Apple figure out what to alter?

[00:14:42] PW: Yeah, that's an excellent question and also I think dovetails with how does Apple ultimately make that decision even once they figure out that component. I think their understanding was largely gleaned from the security reports, the research that the external security researcher published. I would like to imagine, Zoom also talked to Apple or vice versa, Apple reached out to Zoom. But Apple does have a very talented security team that likely could replicate what the external researcher report and then understand that, A, when Zoom uninstalls itself, it was leaving behind this webserver component. Again, it was running as a separate process, or when Zoom updated itself, I think the dependency on that service was no longer there.

I would imagine that what Apple did is internally ran through all these scenarios first before pushing this out. Apple has a pretty good track record not breaking things when they push out updates. I mean, there have been some missteps. But overall, they do a pretty good job. I think that's, again, because they prioritize not breaking anything above security and privacy, which I think is – Which is a good thing especially in this scenario. I would imagine that they ran through the various scenarios, "Okay, old version of Zoom is uninstalled and there's webservers. It's still running. Can we safely remove this?" or "Perhaps a new version of Zoom is installed. Can we detect that and then know that we can delete that?"

Perhaps if they detect that an old version was running and still had that dependency on that local web service component, they wouldn't perhaps maybe remove it in that specific instance. I think they probably did a lot of QA testing to make sure that it's rather controversial move, in some sense, arbitrarily removing software from user systems around the globe wasn't at least going to break anything.

[SPONSOR MESSAGE]

[00:16:32] JM: DNS allows users to navigate to your web endpoints, and whether they're there hitting your app from their mobile phone or accessing your website from their browser, DNS is critical infrastructure for any piece of software.

F5 Cloud Services builds fast, reliable load-balancing and DNS services. For more than 20 years, F5 has been building load-balancing infrastructure, and today, F5 Cloud Services provides global DNS infrastructure for lightning fast access around the world. If you're looking for a scalable, high-quality DNS provider, visit f5.com/sedaily and get a free trial of F5 Cloud Services. Geolocation-based routing, health checking, DDoS protection and the stability and reliability of F5 Networks. Go to f5.com/sedaily for a free trial of F5 Cloud Services; fast, scalable DNS and load-balancing infrastructure. That's f5.com/sedaily.

[INTERVIEW CONTINUED]

[00:17:41] JM: Another issue that has been widely discussed is the aggressive Zoom Mac OS installer, and you've written about this. The Zoom Mac OS installer performs its install job without the user clicking install or at least it used to. This was another interesting example of Zoom sidestepping the norms that Apple wants on its client developer ecosystem. I kind of hate the installation experience on a Mac. There're a lot of things to double click. I know it's a lot better than the installation experience I had when I was like installing software from CDs in 1995. But it's still like a lot of clicks compared to just like going to a website and that website like loading the software that I want to use. Why is this a problem? I mean, Zoom sidestepping the typical MacOS installation process. What's wrong with that?

[00:18:48] PW: Yeah. That's an interesting discussion, because I don't want to say that was taking out of context per se, but Zoom I think did get a little extra flat for that, but maybe was slightly misplaced. What happened was you have an installer package. It's a PKG. It's kind of how packages are generally distributed on Mac OS.

Normally what happens is you double click that PKG and it opens in Applesinstaller.app. There's always then a prompt basically saying, "Do you want to allow this process to run?" I forgot the exact verbiage. If the user clicks yes, there are some preinstall scripts that get run if the package contains those scripts. Then there is a request for the installation to continue.

What Zoom basically did is just packaged all their install logic in the preinstall scripts. There is still going to be a prompt that the user has to click, but it sidesteps or avoids the second popup that has the install button that the users click.

From a security point of view, I don't necessarily see a problem here because the user still has to manually click and at least click through one security prompt anyways. Other software developers do this too, WebEx. Another video chat conferencing application. It uses the same technique. I think where Zoom kind of got a bad rap is malware also uses this technique, because malware generally says, "We want to make our install experience as easy for our users as well."

Whether Zoom should get in trouble for this or people get all spun up on that, I mean, I think it's an interesting observation. Yes, it kind of sidesteps the status quo of having two popups, but the preinstall script is there for a reason and expects commands to be executed. So the fact that Zoom put the majority of their – Or their entire install instructions in there, I really don't think that's the end of the world. My personal opinion. Because again, users are still having to click through at least several prompts to get to that and agreeing to several popups. It reduces the total number of popups by one, but doesn't completely remove all of them. Again, it's not an exploit. It's not a vulnerability. It's just, in some sense, doing things a little more efficient that, yeah, maybe is not the exactly prescribed way. But better user experience, and I don't think it decreases or has any privacy or security issues.

[00:21:10] JM: That is the motivation for Zoom going around one of these dialog boxes that you would have to click through. It was simply a user experience or a thing where if you're at Zoom, you know that every additional dialog box that the user has to click through is an opportunity for drop off. So you just look for ways to alleviate that.

[00:21:30] PW: Yeah, 100%. That's exactly right. I mentioned, this is not an uncommon practice, both this practice and other similar practices. A while ago, we had Dropbox kind of sidestepping some Apple APIs which would generate authentication prompts and directly inserting themselves into a database. People got all spun up on that. But the reality is Mac OS and Apple had already – Or when Dropbox was run, Dropbox already had to ask the user for their credentials to install.

Then going through the API, again, displayed a secondary problem to ask the user for permission as well. Dropbox basically said, “Hey, we’re really authenticated. Why bother the user again? Let’s just directly talk to the database versus go through the API and avoid a secondary popup.” There is some precedence for this. Again, it falls somewhere in the gray area. But the motivation for these third-party application developers, these companies, is to increase the user experience largely because Apple adds more and more of these user prompts and you see users complaining about that on the internet.

When Catalina came out, someone posted a screenshot of when they logged in, there was 10 new prompts on their desktop for them to approve certain applications to be able to access the download folder in their documents, the desktop. In some ways, it's getting a little out of hand. Apple’s approach, it's almost like we’re having this Vista-like experience where you have to manually click approve on everything.

Again, a lot of these companies are trying to figure out ways to minimize the impact and user experience with no malice. Again, I think this was done 100% just to improve the user experience versus some shadier malicious activity.

[00:23:18] JM: Great. These are questionable Zoom security decisions that we have discussed thus far, these were discovered before you started investigating Zoom more seriously. When did you start examining Zoom yourself as a security researcher?

[00:23:35] PW: I believe it was last Monday. Earlier this week, yeah, days are all starting to blend together. But after seeing at least the tweet from my friend Felix about Zoom’s somewhat questionable installer practice from a security point of view, it just piqued my interest. Also, there had been some other issues where I had been pointing out that Zoom was sending information

to Facebook via this third-party SDK. Zoom fixed that and said, “Hey, we weren't aware of what this SDK was using.” Yeah, there was some plausible deniability.

But to me, it also showed that, “Okay, there's all these security and privacy issues and a lot of them are very low-hanging, let's say, or fairly obvious, right? They’re not super complex exploits and vulnerabilities. It does seem that Zoom has just not been written with privacy and security in mind. Where you talked about, they really focused on usability, which made for a very popular product. But I thought maybe there're some security issues especially in the Mac client, because a lot of times, developers are quickly putting together a Mac client.

I imagined, they had a good Windows client. The web client solved it and then Zoom says, “Hey, we need a native Mac client. Who knows how to write this?” It’s like someone’s like, “Oh, I’ve done a little bit of Mac programming.” “Okay, cool. Jump on Stack Overflow. Copy and paste cart.” I just had a feeling that maybe this product wasn’t designed, the Mac component, as optimally or securely as it could be.

I started looking in. Started at the installer, I immediately noticed that they – And Felix had pointed this out to, that perhaps there was an issue, because when they went to perform the elevated or privileged actions, which is fairly common, right? An installer needs to often put files and directories that the user may not have access to. They would do this via a very old and deprecated API. API is authenticate, execute with privileges.

Now the reason Zoom and in the past a lot of developers used this API is it's incredibly simple. Basically what you do is you give it a path to a binary or a script or a command that you want to be executed as root, and then behind-the-scenes, Apple takes care of all the authentication. Showing the access prompt, getting user credentials, validating them. If they’re validated and correct, Mac OS will then execute as root whatever the API was invoked with. One line of code from the developer, things just work.

The issue with this API is doesn't validate what it's about to execute as root. If there is a local attacker or a piece of underprivileged malware that's already on the system and it can detect that, for example, Zoom is performing install or an update, it can programmatically detect and wait for Zoom to execute the script, this command, as root and modify that. Inject some

commands, and Mac OS will naïvely blindly execute that because, again, it doesn't validate anything that goes on.

What Zoom was doing is they would ask the installer to run a script, and the script is called run with root, and you can see it in Zoom's installation package. All that was doing, it was moving the Zoom application into the application's directory because a standard user might not have access to arbitrarily put files in them.

The issue was the installer would place that run with root script in a user writable, world accessible tempt directory. We just mentioned the API doesn't validate what it's about to run as root. Just runs whatever the developer asks it to. What I did in my proof of concept was as Zoom was performing its install or an upgrade, I, with no special privileges, could add some commands to that run with root scripts. Again, it's just sitting in this world writable temp directory. Then once the user have authenticated, the system would run that script along with the malicious commands that I had injected into that. Those again would be executed as root.

So now I was very easily able to get root privileges on a system. That was kind of the first flaw I found. Again, rather common mistake that we in a lot of third-party APIs. The reason is it's very easy API to use. Apple though, to their credit, has largely warm developers. Don't use this API. It will throw a deprecation warning at compile time. Can go read the documentation, it says there's security issues with this, but the alternative, the more modern secure API is a bear to use. It like installs a background daemon. You can't pass command line arguments. You have to do it via XPC. The daemon can't remove itself. It's very secure, but it is incredibly difficult to use. A one line API versus like re-architecting your whole application, your whole installer experience, I can see why developers often still use the older version of the API.

[00:28:42] JM: Just make sure I understand this vulnerability correctly. If I'm installing Zoom, Zoom is going to use this authorization execute with privileges API, which puts a script into temp. Then that script in temp is executed as root, which will move the installed application into the applications directory to make it easier for the user to access. Then the vulnerability is the fact that some other program, even one without root access could append to that file in temp. Then that file in temp would then execute under root. Is that what it is?

[00:29:26] PW: Yeah, it's exactly right. When Zoom invokes this API, they give it a path to this temporary – This script in a temp directory. If they would've executed a binary that was owned with root permissions or something perhaps on a read-only DMG or in the package, a local attacker may not have been able to modify that.

What I'm saying is the API, though it's inherently insecure, can be invoked securely. The problem is Zoom was using it in a way where it would execute, as you noted, a script running out of the temp directory as root, and the script was accessible by anybody. Any other program, a local attacker, could, as we mentioned, inject malicious commands in that then exactly would be run as root. Kind of analyzing logic flaw, right? It's not something that's going to crash. It's not a buffer overflow. It's going to work consistently and reliably.

[00:30:20] JM: I'm sure this is my naïveté as anybody who don't knows anything about security, but if we're talking about some vulnerability that's going to be exposed to like some local attacker, couldn't a local attacker just execute that same blob of code? Couldn't the local attacker just execute a script that says authorization execute with privileges? Why not?

[00:30:42] PW: Yeah. We actually see malware do that. There's various levels of I would say stealth and believability. On one end, you have malware that gets on your system. It just arbitrarily invokes that same API, and there's a prompt on the desktop and some users are just like, "Oh! There's a new access prompt. I should put in my credentials."

[00:31:04] JM: Oh! Oh, okay. So then it would say like, "Awesome! Malware program has asked access to your system, or whatever."

[00:31:12] PW: Essentially. There are some ways where you can – This is again kind of the issue with Apple's authentication mechanism. You can control the icon and a lot of verbiage in that popup. You could, for example, run a – Once malware gets on the system, it could customize that popup say with an icon that belongs to Slack or Zoom or something that's installed on the user system and say, "Hey, there's an update. You need to execute." A suspicious user would probably at least say, "Why is this just randomly popping up?"

The next level, stealth and sophistication, is to basically utilize this kind of vulnerability where it's a flaw in a legitimate installer that user is going through and installing. There's really nothing amiss. The downside is you have to wait obviously until one of these installers or update occurs, but it does decrease the likelihood that a user will notice that they've just given this malware now root privileges.

Then if you go one step further, you have local privilege escalation vulnerabilities in the OS itself that malware can still slowly and surreptitiously trigger with no authentication popups or excuse. It's kind of the sliding scale of stealth and believability, and I think this kind of sits in the middle-ish. Yes, there are definitely other ways to perform a similar attack, and we have seen malware, unsophisticated malware, do exactly that in the past.

[00:32:39] JM: By the way, what was your process for figuring out this vulnerability? What files did you look at to understand this was going on?

[00:32:48] PW: Yeah. I've actually talked about this vulnerability a few times, the use of this API. What you can pretty much do is just run a process monitor that's going to show you what processes are being executed along with their command line arguments. Under the hood, this API performs kind of a lot of complex interactions. It talk to various daemons and XPC messages to do IPC, inter-process communications.

Ultimately what happens though, it spawns a setuid program called security_off trampoline, and it executes whatever was passed to that API. What you can do to detect this kind of vulnerability is simply look for that security-off trampoline process being executed and examine the first argument, because that's going to be the path to the binary or the command that will be run as root. You can then check the permissions of that file. If, for example, it's in a temp directory that unprivileged users can access, you've just found vulnerability.

Now if it's executing some Apple binary that lives on some read-only partition or is protected by system integrity protection, which is a mechanism in Mac OS that prevents code even running as root for modifying files. That means that instance of the invocation of the insecure API is not actually vulnerable, because they are executing something that you as an unprivileged attacker cannot modify.

But more often than not, it's something being run out of the downloads directory or a temp directory. Again, you can kind of surreptitiously hop in, inject some malicious commands or even replace the whole damn thing. Again, it will be run lovingly as root.

[00:34:29] JM: This term you mentioned there called off trampoline. I just like that term. Can you explain what that term is in more detail?

[00:34:37] PW: Sure. Behind-the-scenes, when a developer or an application invokes the authenticate execute with privileges API, once it's done some checks and got the user's credentials, it executes an Apple binary that is named security_off trampoline. This is a program, a binary, that has the setuid bit, which means it'll be executed with the permissions or privileges of whoever is invoking this.

In this case, it's invoked with root. This means whatever it executes is also going to run with root privileges. This kind of the last piece of the puzzle. Again, this is Apple's implementation of this API. What it does is it takes the string, the first parameter that was passed to the API by the developer and executes it now with privileges. The idea I guess is just it’s a trampoline as a proxy that is ultimately responsible for executing what the developer has requested with these elevated privileges.

Again, we can simply sniff or monitor for that process being invoked. Look at what it's about to execute. Then if we have the privileges as a non-privileged user to modify that, we’ve just found a local privilege escalation vulnerability.

[00:35:53] JM: I think here we’re seeing your level of experience in doing this kind of vulnerability searching or reverse engineering because you're aware of these tools that you can use to get visibility into what your programs are doing as you’re executing them. Also, you know, I guess you've just been trained from pattern matching what kinds of things are going to be signs that a program has done something that indicates it's got a security vulnerability.

[00:36:26] PW: Yeah, definitely. I always joke about this a little bit. I used to work at the NSA, the National Security Agency. I did some malware analysis there, but then I was also on the

offensive side of the house developing offensive cyber capability that the US government would then use in cyber operations. What I joke about is that experience kind of corrupted my mind I think in a positive way, but I'm always looking at things kind of from like how do I break this point of view?

I've been doing this for – Wow! Now probably 10, 15 years, and I also write a lot of security tools as well. As I go through these processes, I realize the potential missteps that developers could make coupled with my kind of gray hat hacker mindset that's continually in the back of my head whenever I install Zoom for legitimate purposes. I see that authentication pop that I'm like, "Hmm, I wonder how they are doing that."

The problem is, on Mac OS, a lot of times there's not a lot of good utilities. For example, the process monitor I described, there's not really good built-in capabilities for that. I actually went ahead and wrote a process monitor streets open source my website, and that's kind of an example of me also building tools to facilitate malware analysis, reverse engineering and vulnerability discovery.

Yeah, I definitely have a propensity for finding these bugs and definitely kind of a mindset whenever I see, in the scenario, an authentication popup on installer. I spend two or three minutes digging around and saying, "Oh! Vulnerable." Once you do this long enough, you start seeing patterns. This vulnerability, this insecure API unfortunately is fairly common. Once you start kind of looking or understanding of what to look for, it's fairly easy to find similar scenarios and reveal new security issues.

[SPONSOR MESSAGE]

[00:38:24] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves

the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:40:41] JM: Another vulnerability that we'll discuss that you found is Zoom's code injection for mic and camera access. If a user has given Zoom access to the mic and camera, it also opens that user up to malicious code being injected into the process space where that code can piggyback off of Zoom's mic and camera access. Does this mean that if I install Zoom and I give Zoom access to my camera and mic, any other program can also turn on my camera and mic?

[00:41:23] PW: Yes, exactly. In a nutshell, you captured it perfectly. This to me I think is a more problematic scenario. Probably one that has more likely security and privacy impacts in the local privilege vulnerability we just discussed. Taking a step back on recent versions of Mac OS, you're an arbitrary application. Even if you're Zoom, the first time you run and you try to access the mic or the web cam, Mac OS will intercept that request and block it displaying access prompts to the user saying, "Hey, Zoom would like to access the mic or web cam."

Now, in a situation like Zoom, the user is obviously going to click allow, because Zoom without mic and camera accesses, well, essentially useless. That's all well and good, right? Zoom should be afforded those security mechanisms or that access, and it's also good Apple has added this second layer of security. It's kind of connects back to what we talked about at the beginning where these are kind of these new popups that Apple has added. It can be a little annoying, but in the past we saw a lot of Mac malware that once it got access to a system, it would surreptitiously or very stealthfully turn on the mic, for example, to turn the system into a room capture audio device. You can imagine you'd hacked into a foreign government's computer system. The laptop is in a conference room, turn on that mic. You now have incredible access. This was something that advanced nation states were doing.

We also saw malware that would turn on the camera, the LED light would come on, but more sophisticated malware would wait until the user was not sitting in front of the computer, then turn on the camera perhaps to spy on them. There's some unfortunate cases of that actually happening as well.

Apple said, "Okay, we need to do something about this. We're going to protect the mic and the web cam and any application that wants to access them. The user is going to have to explicitly agree. This is all well and good. Zoom now has access to the mic and the web cam. If we look at how Zoom is compiled, how it's built, it's compiled with something called the hardened runtime. The hardened runtime something that Apple has designed that applications can opt into at compile time and it basically tells the operating system to protect the application.

There's a variety of protections it affords, but in the context of this discussion, it protects the application from malicious code injection attacks. This means when the application compiled with the hardened runtime is executing, if another program on the system, perhaps a piece of malware tries to inject a malicious library or modify one of the libraries that the application has a dependency on, the system will detect this tampering and actually block or prevent that injection from happening.

A very great security mechanism that Apple has offered application developers and provides a very high- level of self-defense, which is good, because if there wasn't the case, we could very easily do things like inject malicious code into Zoom. Piggyback off its mic and web cam access.

The issue with Zoom was, though it was compiled with this hardened runtime. For some unknown reason, they add an exception that instructed the operating system to allow third-party libraries to be loaded into Zoom's trusted processes. As soon as I saw that, I said, "Well, Zoom has access to the mic and web cam. Can I create a malicious library? Replace one of Zoom's dependencies, one of its own libraries with my malicious library?" Then whenever the user starts Zoom, my malicious library will get loaded into the address space. Then from the operating system's point of view, when my malicious library tries to access either the mic for the web cam, the operating system will see that it's just Zoom, the Zoom process. Since Zoom has been given access to the mic and the web cam previously by the user, my malicious library will now be able to access that.

This is problematic, because it means malicious code. Any other program or malware that's on your system can inject into Zoom to record Zoom meetings, Zoom discussions. Worse than malware or that malicious program, can actually execute Zoom at arbitrary times in the background in an visible manner. Turn on the mic or the web cam. Again, Mac OS will see that request, but will allow it because Zoom has been given access previously by the user. Kind of an interesting flaw there that I think definitely has some security and privacy implications.

[00:46:01] JM: What has involved in injecting malicious code into a process space? What does that look like in practice?

[00:46:12] PW: Yeah, that's an excellent question, and there're a myriad of techniques. The technique I chose is a technique that I kind of have, let's say, pioneered at least on the Mac OS side. The idea is you find a library that the application has a dependency on. That means is going to try to load it every time it's run. What you do is you take that legitimate library, in this case it was a little SSL library, and you rename it.

Now if you try to run the application after you've renamed the library it has a dependency on, the application will crash because it can no longer find the library that it requires to execute. What

you then do is you create a malicious library and you put it in the original location with the original name. Now when you run, because again in this scenario Zoom had turned off the library validation that the hardened runtime affords, Zoom will naïvely and blindly load your library instead of the original one.

Now there's one more piece of the puzzle. You've just replaced a legitimate library. That legitimate library had a dependency or the application had a dependency because it expected that library to do something. For example, we replace an SSL library. When Zoom goes and tries to make some SSL connections, the application is going to crash or quit because our malicious library obviously doesn't export or provide that functionality.

The key to the whole puzzle is this idea of dylib, dynamic library proxying. What you can do is you can compile your malicious dynamic library with a forwarding directive that says, "Hey, anytime someone asks me for some expose capability, an exported API, symbol, anything a library would normally expose, I don't have that implementation, but I know who does."

What we can do now is we can create that forwarding directive and point it to the original SSL library that we've rename, essentially replaced. So that when Zoom makes a call to us, which it believes is the legitimate SSL library, we simply proxy that request to the original library, and that is all seamlessly and handled by both the linker and the logic at runtime. That's really awesome, because we don't have to actually implement any code, any functionality that we have replaced. We just create this forwarding directive. Both the libraries get loaded and anytime Zoom makes a call into the SSL library, that gets automatically forwarded and handled by the original library. We're stoked, right? Our malicious code is running Zoom's adware space. Wan access the mic and the web cam, but we have not broken any legitimate functionality.

Very stealthy, very powerful technique to get code into remote processes in a way that's very difficult to detect and also doesn't break any legitimate functionality, which is very important, because if we crash Zoom or cause hiccups, people would start digging and would uncover this attack.

[00:49:08] JM: What has been Zoom's response to this particular vulnerability?

[00:49:14] PW: We kind of picked on Zoom a little bit, right? We put it out there. Privacy and security track record with less than stellar. They really prioritize usability over privacy and security. However, Zoom's response to this was stellar. First and foremost, they fixed these vulnerabilities within a day, which that's incredible. I think that already speaks a lot to bear newfound commitment to security and privacy.

Beyond that, they spent some time really trying to learn from these lessons and traded a very transparent and well-thought through plan that moving forward, how can they address these laws and ensure that in the future such laws are hopefully not introduced into production code that's being shipped to millions of users around the world.

The first thing they said is we're going to do a feature freeze. Instead of focusing on new features, we're going to now focus on security and privacy. This is great, because originally they put a lot of the resources and efforts, if not all of their resources and efforts, into creating new features versus improving privacy and security.

We can't fault Zoom too much, because this is largely driven by customer demands. For example, if Zoom had spent six months designing a new version of Zoom and only improve the privacy and security, the average user, the average business probably wouldn't care. Whereas Zoom comes out new version and says, "Hey, we now support virtualized backgrounds." You can change your backdrop to a beach in Hawaii. Consumers and businesses are like, "I love that. I'm stoked. Give me emojis. Give me backgrounds," right?

The market was dictating what Zoom prioritized. Unfortunately now, there's kind of been this 180. I just said unfortunately. I should say fortunately. Where now security and privacy is paramount, and we have seen companies banning the use of Zoom, because they are worried about some of these security issues that have come up both mine and others. For example, Zoom was routing some calls through China accidentally, supposedly, but that doesn't give a lot of companies kind of this warm, fuzzy feeling.

Zoom has basically said, "Okay. Hey, we really realized we need to focus on security and privacy. So feature freeze. We're going to put all engineering resources on to security and privacy efforts and proving that. That's awesome." They were also improving their bug bounty

program to attract more security researchers to audit their code and be financially compensated for any issues they find. They're also bring in external companies to do audits and pen tests of their software. I believe they also created an advisory board bringing in C-level execs from other companies to help them with design and security and privacy decisions. I think the response is I would argue incredibly emotionally immature and really shows that moving forward, they understand the relevance and importance of security and privacy and are taking all the necessary steps to mitigate that.

Now, there is a little bit of tech debt that they're going to have to wade through. I would be unsurprised if new security issues are found. But they now have a really good track record of fixing those quickly, being very transparent about them and moving forward. That's a good thing. I think this isn't a really a win-win scenario, because yes this will ultimately benefit users and customers around the world who are now going to be running an application that's far more secure and private, which will turn into increased sales for Zoom and/or the maintaining of existing customers. I love when a situation turns out to be a win-win, where the users benefit and business and Zoom benefits as well, because I think then everyone is stoked.

[00:52:54] JM: Okay. Great. Well, let's begin to wrap up. I just love to know a little bit about what your day-to-day work consists of. Is this what your life is? Just looking at applications and looking for vulnerabilities like this?

[00:53:11] PW: Often, usually the first thing I do is check the surf. Luckily I live out in Maui, and it's actually funny because I could see the waves are breaking really now. As soon as we're done with this podcast, I'm going to grab my surfboard and get some vitamin D and some exercise. Luckily, it's a very socially distancing activity. Super stoked about that.

My day-to-day job is I work as a principal security researcher at Jamf. At Jamf, we're building a Mac security product for the enterprise. What we do is we analyze a lot of malware, exploits, vulnerabilities, and then we build detection mechanisms for this. One of the detection mechanisms we have is for monitoring for this, for example, this thing secure API being called, and this is something that we already had and already worked on because of my past research.

As I mentioned somewhat earlier, when I get an invite for Zoom meeting and I install Zoom in my system and I see that authentication prompt, because of my past research and also the tool development of the research I do at Jamf, I am kind of already looking for these issues. I said, “Okay, I’m going to take a break, because I’m coding, and dig into this application and see what it's doing.

It’s nice that Jamf is doing me a lot of flexibility to kind of go off on these small tangents. Similarly, if a new piece of malware is detected, spend some time tearing it apart. Figuring out is it going to bypass our existing detections? Is it using some new exploit? Is it leveraging some really creative way to access the mic and web cam? It’s kind of this continual cycle between doing offensive cybersecurity research. Looking at malware, looking for exploits and vulnerabilities and then taking all of that knowledge and feeding it back into our security product.

For me, that's great, because I really like finding bugs, analyzing new malware, right? That’s really kind of allows me to scratch that offensive cybersecurity itch. But then being able to feed it back to a defensive products I think is really powerful way that we can ensure, at least strive to ensure that users around the world are hopefully more secure, because at the end of the day, that’s something I'm really passionate about. When people are getting hacked and their system is infected, that can be problematic. I think not to go too much off on a tangent, but we talked a little bit about malware that could access the web cam. There was a case I worked on that I actually ended up collaborating with the FBI where an individual had created this Mac malware called fruit fly, and his goal was to infect Mac users and turn on the web cam to spy on them. He did this for over a decade undetected. Good news is he eventually got caught, but unfortunately a lot of the victims that he had been spying on were children. This is a scenario where there's real psychological damage and impact to the victims of this. From a security point of view, this malware wasn't very sophisticated and should have been detected very easily. If we can work and create security products that can hopefully protect users, then we’re stoked.

That’s kind of my 9-to-5 job. Then my 5 to 11 job is creating free open source security tools for end-users. I run a website called Objective-See. Objective-See, and we talk a little bit about like, for example, the process monitor. I’d say, “Hey, there’s not a good process monitor for Mac OS. Let's create one,” or “Hey, I want an application that tells me anytime someone accesses the mic or the web cam.” Even if someone has gained access to my system on using Zoom to spy

on me when I'm not around. when I come back to my computer, there should at least be a pop-up or an alert that, “Hey, just to let you know, Zoom or something else was using your mic or the web cam.”

I created a free application that does that and it just gives you an alert anytime someone accesses the mic and the web cam. Again, if someone had been running this on their system and this fruit fly malware that was designed to spy on people. When it got infected on the system, this application would've alerted users to that. It’s kind of a goal. Just designing free largely open source tools for end-users to prevent the attacks that I’m both coming up with and that I'm seeing in the wild. That’s something I’m them really passionate about as well. Hopefully make somewhat of a difference in the world and kind of a way giving back and baking a lot of my talents and experiences into free utilities and tools for Mac users.

[00:57:36] JM: Yeah. Very interesting work. Very useful, positive work, and makes me want to do more security shows, because this is great series of stories, great series of vulnerabilities. I think that is one thing that stands out about the security ecosystem, is there're just the compelling stories. Thanks for coming on the show, Patrick. It’s been great talking.

[00:57:58] PW: Yeah. Thanks, Jeff. I would be stoked anytime to come back and talk nerdy about other security topics.

[00:58:03] JM: Yeah, let's do it again. We’ll do it again sometime. That sounds great.

[00:58:06] PW: All right. Thanks, Jeff.

[END OF INTERVIEW]

[00:58:17] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra’s architecture and its benefits, and we’re happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It’s a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]