EPISODE 641

[INTRODUCTION]

[0:00:00.3] JM: Apple operating systems such as iOS are closed sourced. This closed sourced nature gives apple an extremely successful business model and a very different very software developer ecosystem than open source Linux-based systems. Since Linux is open source, the information on how to manipulate the system at a low-level is very public. The lack of information about low-level programming in Apple operating systems has led to a large community of jailbreaking, where people try to reverse engineer how the closed sourced systems function.

In today's episode, Max Bazaliy joins the show to describe how he reverse engineered an Apple Watch. It's a complex security challenge to jailbreak an Apple match, at Maxim describes in great detail. Max is a security researcher at Lookout, a mobile security company. So he is well-equipped to describe the internal security details of the Apple Watch.

Before we get started, I want to announce that we're hiring a creative operations lead. If you are an excellent communicator, please check out our job posting for a creative operations at softwareengineeringdaily.com/jobs. This is a great job for someone who just graduated a coding bootcamp or someone with a background in the arts who is making their way into technology. If you want to be creative and you want to learn about engineering, check it out at softwareengineeringdaily.com/jobs.

[SPONSOR MESSAGE]

[0:01:38.2] JM: Citus Data can scale your PostgreS database horizontally. For many of you, your PostgreS database is the heart of your application. You chose PostgreS because you trust it. After all, PostgreS is battle tested, trustworthy database software, but are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes dramatically speeding them up and giving you much lower latency.

Are you worried about hitting the limits of single node PostgreS and not being able to grow your app or having to spend your time on database infrastructure instead of creating new features for you application? Available as open source as a database as a service and as enterprise software, Citus makes it simple to shard PostgreS. Go to citusdata.com/sedaily to learn more about how Citus transforms PostgreS into a distributed database. That's citusdata.com/sedaily, citusdata.com/sedaily.

Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily.

Thank you to Citus Data for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:23.2] JM: Max Bizaliy, you're a security researcher at Lookout. Welcome to Software Engineering Daily.

[0:03:28.6] MB: Thank you. Yeah, I'm happy to have a chance to speak here today.

[0:03:33.3] JM: Yeah. So I saw your video at DEFCON, which was about jailbreaking an Apple Watch, and I want to talk through that as well as talk through jailbreaking and mobile security more broadly. Let's start with the topic of jailbreaking. What is jailbreaking?

[0:03:52.4] MB: So basically, jailbreaking is a process of hacking a device through device-specific restrictions. If we're talking about Apple Watch or any other Apple device, so it's a process to disable some of the software restrictions applied by Apple to run maybe unsigned code, or modify a system, to install maybe some new skins to a system and so on.

Usually, jailbreak process requires software vulnerabilities to use in a way to like escalate privileges and by pass some of the system security mechanisms. I would say jailbreaking is an interesting target for security researchers, because after jailbreaking, a security researcher gets a full attest of file system, which allows him to explore how the system works.

[0:04:42.7] JM: Why would you want to jailbreak an Apple Watch specifically?

[0:04:47.0] MB: Well, I would say that, first thing, I would like to prove that it's real, that it can happen. No one do it before publicly. Second thing, I would like to understand how the watch OS works. How the watch OS system works. What I can do with that? How it's similar to iOS and so on. It was like a fun project for me. Then it goes to something bigger like a jailbreak and so on.

[0:05:13.1] JM: An Apple Watch is a file system. It's an operating system. How is it different than hacking some other computer, like my desktop computer?

[0:05:22.9] MB: First of all, watch OS basically runs the same system as iPhones do. It's like slightly modified version of iOS and they share the same kernel. Watch OS share the same security restrictions as well. So the mail difference between hacking regular computer or a Mac is a mount of effort Apple put to protect their operating system. Which, I mean, you cannot easily run any applications that you want. All the code that is running the system should vetted by Apple or should be approved by Apple.

Even if you can run a code on a system, it will be pretty limited by application sandbox, which like isolate each application from each other to make spine or data leaking impossible. So it's like a pretty challenging task and it differ from, nowadays, operating system by just adding more security layers.

[0:06:21.0] JM: Now, if you could jailbreak an Apple Watch, what would you be able to run on it? What kind of software did you have in mind that you could potentially run on this watch if you were able to hack it open?

[0:06:32.9] MB: First of all, it was interesting for me from a security researcher point of view. I want to understand how the system work. Basically, after jailbreaking, I can run any open source code on it. This includes like servers, that includes proxy, that includes debuggers, that includes – I don't know, even like spyware and so on and so on.

For me, it was more like a reserved project that I just want to prove that I can get remote access to my watch and run any code I want on my watch. For example, like a server or so on.

[0:07:10.2] JM: Is there a large community of people online who jailbreak devices on a regular basis and share strategies around how to do jailbreaking?

[0:07:19.8] MB: Yeah, I would say the jailbreaking community is pretty strong, and I would say it's old, started from the very first days of iOS releases. Community was started around two weeks, which some modifications that you can apply to a system to change its behavior, for example, some cool animation, maybe some additional features in calls or SMS or so. It was basically built around the tweaks. Later on, security researchers start using jailbreaks as well as their way to find new security vulnerabilities in a system.

One of the things that the system is so closed, like only small part of a kernel is open sourced. Everything else is closed sourced, and to find security vulnerabilities, users need to get a tested system.

[0:08:11.0] JM: Device creators , like Apple, are they trying to prevent jailbreaking? Do they put mechanisms in place to try to discourage people from doing that or prevent people from doing that?

[0:08:21.7] MB: Oh, yeah, absolutely. Because that's their business and that the security for device, and the security on their customers. They do pretty much everything to prevent current jailbreaks and project from future jailbreaks by, I would say, applying the new security mitigations in software and as well as in hardware, which make all the future jailbreaks a way harder.

I would say that now amount of efforts that you need to put to make a jailbreak, for example, now and like four years earlier, it rise significantly. There is like much more things that you need whitepage now.

[0:09:01.0] JM: The strategy that we're going to go through in terms of how you jailbroke the Apple Watch, this is not something that you could have done remotely, right? This was something that you had to have access to the direct hardware in order to jailbreak it.

[0:09:13.7] MB: Well, that's a good question. I would say this kind of attack is possible remotely. The way why I did not use the remote approach is the diversion of watch OS that I was playing did not support the browser or like the web kit in that time. Other interfaces that allow any remote access was pretty limited.

In the newer version watch OS, it is possible to interact with a browser and that's pretty good attack record. Yeah, in my attack, I used application-based approach, which are basically installed on the applications to the Apple Watch that was like approved by Apple. This is how I start my attack.

[0:09:58.1] JM: The objective of your jailbreaking is to get into the kernel, the operating system kernel. Remind the audience what the operating system kernel is.

[0:10:08.2] MB: Basically, it's a core of a computer operating system and it gets a complete control of everything in a system. For security purpose, it's running in a separate address space in a private privileged environment. Basically, kernel do all the low level task in a system, like all the file system address, network address, memory management, process separation and so on and so on.

So each time the process trying to read the file, it goes to like reroute it to a kernel, and the kernel basically can make a decision to read this file or like to deny access to a file. So any low level operations, they route it in a kernel. Kernel provides some set of APIs that can be used by user mode processes to interact ways.

[0:10:55.3] JM: Your attack vector was an application extension. You considered several different attack vectors. Why did you go with the application extension? What is an application extension?

[0:11:07.9] MB: It's like a brief overview of possible attack vectors in that time that was available. So as we talk a little bit about remote attacks, the remote attack was an option or physical attack was an option. Physical attack basically require a physical test to a device and to where we can try to interact with the device using special diagnostic work that are found on a watch.

Problem with the physical approach is that we need to know very well how the boot process in device is working as well as know how the diagnostic part interact with the hack. Other problem is that Apple use their own formats as well as their own connectors type for almost everything, which means that I need to design my own connector to interact with a diagnostic port. So it was not a good option for me. That's why I switched to remote attacks. I found there are a few possibilities of that, like there is interaction between a phone and a watch. So there's maybe a good target in there.

Later on, I found that Apple support programs that are called application extensions to be running on a watch and that particular target. The application extension is basically a program that is bundled in iOS application, and most of the time it's used as a minion to iOS application, basically to show some notification on Apple watch or so. But it can be used to send data as well. It can contain some logic and send data back to a phone. For example, when you exercise in a gym, your watch can send data back to your phone, and phone going to make some decisions based on it.

Once the application extension, basically it's a program and it's running native code. As it runs in the native code, it's always good for an attacker that I can put my own code there and try to interact with the system. Other thing is as application extension is a part of iOS bundle package, it got signed by Apple, which mean I get Apple approval of running this application extension in the watch. So I don't need to care about code sign in and restrictions. This code is already approved by Apple.

[0:13:37.2] JM: The significance of code signing, can you explain that? What is code signing.

[0:13:42.2] MB: That's the security approach that uses in a modern operating system. The idea behind it that only signed code can be executed in a system, which mean only the code that

was checked for or approved by a vendor, in this case is Apple. For example, the app store, the way how the users download applications. Programmers upload their applications to Apple. Apple check that the application is not doing any malicious behavior or so on. Then say like, "Okay, it's okay to publish it in App Store." They basically sign this application with Apple certificate. That's the basic idea behind it.

All the Apple operating system, they now enforce that code that's want to be run in a system, that it should be code signed. It should be code signed and it should be code signed by apple.

[SPONSOR MESSAGE]

[0:14:43.1] JM: The Casper Mattress was designed by an in-house team of engineers that spent thousands of hours developing the mattress. As a software engineer, you know what kind of development and dedication it takes to build a great product. The result is an exceptional product. When you put in the amount of work and effort that went into the Casper Mattress, you get something that you'd use and recommend to your friends, and you deserve an exceptional night's rest yourself so that you can continue building great software.

Casper combines supportive memory foams for a sleep surface that's got just the right sink and just the right bounce. Plus, it's breathable design sleeps cool to help you regulate your temperature through the night. Stay cool, people. Stay cool.

Buying a Casper Mattress is completely risk free. Casper offers free delivery and free returns with a 100 night home trial. If you don't love it, they'll pick it up and give you a full refund. Like many of the software services that we have covered on Software Engineering Daily, they are great with refunds. Casper understands the importance of truly sleeping on a mattress before you commit, especially considering that you're going to spend a third of your life on that mattress.

Amazon and Google reviews consistently rank Casper as a favorite mattress, so try it out. Get a good night's rest and up-vote it yourself today. There's a special offer to Software Engineering Daily listeners. Get $50 towards select mattress purchases by visiting casper.com/sedaily and using the code sedaily at checkout.

Terms and conditions do apply. You'll get the select mattress purchases if you go to casper.com/sedaily and enter the code sedaily at checkout.

Thank you, Casper.

[INTERVIEW CONTINUED]

[0:16:53.1] JM: Now, how can you get around that? Because if your code that is going to potentially get into the kernel, leak the kernel base, do a kernel dump. If your code is going to be doing that, it's not going to be signed. You're only going to be able to get signed if you interface with Apple, if you get the Apple code signing function done on your code. How do you get around the code signing?

[0:17:17.3] MB: Yeah. There are a few approaches. One of them is basically find software vulnerability to bypass code signing checks. It was used multiple times in jailbreaks. Other more easier approach is to use a development certificate to sign your code. Basically, development certificate is a certificate issued by apple. It's like personal certificate that's given to a developer to an organization. But it allows you to sign code with some limitation.

The code that you sign in, it can be run at only on a few devices that you basically specify into a developer portal. In order words, if I get like 10 test devices, I can specify the device serial numbers in a developer portal and Apple is going to grant me a certificate that I can use to sign a code into devices.

[0:18:12.3] JM: Okay. To go back to the objective, you needed to leak the kernel-base and do a kernel dump. What is a kernel dump? Why did you need to do these kernel-related operations?

[0:18:24.8] MB: Step back on how the modern operating system works and how the security mitigation works. The kernel is it's like so significant part of a system. It run all the low level operations as well it enforce all the security restrictions, like sandboxing, like code signing and so on and so on.

To be able to jailbreak a device, we need to get kernel modification. Somehow we need to escalate the privileges and by pass some of the security measures to write a kernel memory.

The problem here is that kernel address in a memory randomized each time. Each time the device is booted, it is loaded into a different address. It is one of the security mitigations called address space, layout randomization. Basically, it's a way how to prevent exploitation. In this case, kernel exploitation.

So in a way to write something to a kernel, we need to know where the kernel is located in the memory. That was my very first task, is I get the code execution with the help of application extension. Now I use one of the vulnerabilities to understand where the kernel is located in the memory.

[0:19:39.6] JM: To clarify where we are so far, you developed an application extension and that application extension allowed you direct access to the kernel wherever the kernel was in that memory space?

[0:19:52.6] MB: Not that easy unfortunately. This application extension element to run native code. I found one of the security vulnerabilities in the interface that interacted with the kernel, which allow me to run this code into sandboxed environment, which means this API call was not prevented by sandbox. But it was powerful enough and contain a software vulnerability that allow me to leak the address in a kernel memory. So I still don't have a test to a kernel memory, but I know where the kernel is located.

[0:20:30.6] JM: But because it's randomized every time upon startup – Oh, okay. So you can just run this application extension every time and then locate where the kernel is sitting in memory each time.

[0:20:42.4] MB: Yeah, exactly. So that was the point of this type of box called info-leaks, just because the kernel address will be randomized that each time a device reboots. We need to each time calculate the kernel address in a memory. For this purpose, usually the software vulnerabilities get used that's going to help to leak where the kernel is located.

[0:21:03.9] JM: After you discover where the kernel is located using that application extension and that interface vulnerability in the application extension, what can you do? What you do once you know where the kernel is in memory?

[0:21:17.1] MB: Yeah. So now I know where the kernel is in Memory, but I still don't have a test to read or write to a kernel memory. I just know where it's located. Towards stage two of my exploit, I know – Well, I found other vulnerability in a kernel, which allow me to read the kernel data. The problem was that for construct, the exploit for this vulnerability, I need to know some data in a kernel and it's like chicken and egg problem. To read the kernel, you need to know something that it is in a kernel and you cannot read it until you found a way to read a kernel memory.

So it was a problem. So the way how I want to exploit and read the kernel memory, it require me to know some of the information about the kernel. I will start looking for ways how I can leak some of the code from a kernel.

[0:22:18.3] JM: If you could dump the kernel, then you could understand what's going on in there, how it's laid out in memory. Then if you can reverse engineer how it's laid out in memory, then maybe you can potentially reverse engineer how to read and write to that space.

[0:22:34.0] MB: Yeah, exactly. So if I can dump a kernel, I can analyze it, I can understand some of the constants in a kernel and I can understand where the code is located and how I can reuse the code that is already in a kernel memory to do the stuff that I want. For example, a read anywhere or a write anywhere.

[0:22:56.8] JM: The kernel, the representation in memory is just a set of data structures, right? It's kernel structs. Just like anybody who's developed object oriented software, deals with object or structures, the operating system kernel has its own set of structures that are represented in-memory. Can you describe a little bit more how the object system of a kernel exists in memory and can you look at it? Is it in a way that you can actually recognize from human readable format?

[0:23:32.0] MB: The kernel basically contains a code section, which is like the compiled code and it contains data section, where the old variables are stored, variables that can be modified during the code execution. There is basically the kernel memory called like a heap. The heap is where the kernel allocate the structures or the kernel allocates some data that is used, for example, each the time process is run in a system. The kernel allocate some of the data on the kernel heap.

Before creating an exploit, I need to understand how the kernel works. The problem was that which iOS kernel was different from any kernels I've seen before, like iOS or Mac OS, and the layout of data layout of some of the structures was different that I've seen on iOS or Mac OS. Which mean that I need to play some kind of cat and mouse game and guess what kind of data is it, like allocate in the new memory chunk or run in the new process and check how the kernel memory was changed to understand where it's located and how it's distributed over the kernel memory.

[0:24:54.6] JM: This is called kernel symbolication, right?

[0:24:58.2] MB: Similar thing. One quick remark how we get the kernel memory. As I say, that I did not – I know where the kernel is located in a memory, but I still don't have a powerful, that powerful, called a primitive, that powerful way to read the kernel memory. To read it, I need to know some part of a kernel. I use one of the tricks that I found a way to crash a system. I found a way to reboot a device and crash a kernel.

Luckily, when the kernel crash, it's saved the system state in called a crash dump. The crash dump contain the value of the CPU registers. Some of these CPU registers is what I need to leak four bytes of the kernel in a time. I skip crushing a kernel, I read the crash dumps, read four bytes and then repeat the process. It takes around like two weeks, but I get like around – I think it was 700 bytes of a kernel, which was enough for me to construct my exploit. I get enough data, click it by these crash dumps to construct the second stage of the exploit which allow me to easily read anywhere in a kernel. So I don't have this problem anymore.

[0:26:14.4] JM: So just to be clear, you hack together some series of scripts. They were just going to run repeatedly to load up your Apple Watch, locate the kernel in memory and then

crash it to emit – What did you say? Four bytes per crash, and then you run that for two weeks, which gave you enough of the kernel's state. Although, those were – Each time was a different state? I guess you had to be rebooting it under the exact same circumstances so that you have a somewhat consistent picture of what it looks like.

[0:26:47.4] MB: Here's how the processes look like. Unfortunately, cannot directly install applications on a watch. I basically construct my application extension that the first step of doing it is looking where the kernel is located in a memory. Then it is trying to crash a kernel by a specific address. Now I know where the kernel is located, and I need to leak the values of some of the memory address. I put this memory address in one function, and this function will interpret this address incorrectly and basically crash a kernel. But that leaked four bytes at a time.

So the process looks like you install application on a phone. The phone synchronize it with a watch. Watch get crashed. You wait until watch waked up. Basically, reboot a system. Then you keep nagging and keep waiting until the watch will synchronize the crash dump with the phone. Then you need to jailbreak a phone and copy this crash dump to your Mac. Then change address, like plus four bytes and repeat all these thing. That's why it takes so long.

Some of the steps I can automate, like recompile an application each time and so on. But some of the steps, I didn't find a good way to automate it. For example, I need to wait until the Apple Watch get rebooted and until it make a connection between a phone and a watch to download the crash dumps back. Sometimes it was unpredictable. I mean, it sometimes take two minutes until the watch will be rebooted. Sometimes it takes more. Yeah, it was challenging and that's why it takes so long. But finally, after weeks of this process, I get what I want.

[0:28:40.7] JM: Which was?

[0:28:42.1] MB: Which was 700 bytes of a kernel, and this allowed me to construct the next step of the exploit, which will be used in a same application extension. But instead of crashing a kernel, now it can use internal constants that I just leaked to read anywhere from a kernel memory. Anywhere, basically like write anywhere. It was basically pre-requisitioned to complete kernel memory read and write. This allowed me to read now the whole kernel, which in that time

was around – I think it was 16 megabytes of data in watch OS, which means if I was dumping in via four bytes, it can take just ages to dump 16 megabytes.

[0:29:26.4] JM: I see. You just needed to figure out enough information to be able to do a read over the kernel itself. Describe – How do you extrapolate 700 bytes to understanding how the address space is laid out and being able to read it?

[0:29:44.2] MB: Yeah, that's a good question. I found a vulnerability in a kernel, in a code doing serialization, which mean, for example, you can send dictionary with an object, like strings, like arrays, like Boleans. During the serialization object, the kernel can go to different execution flow. For sure, not checked by initial developers.

So long story short, I found a way, if I construct the string object and if I know where the kernel is located in a memory, I can say that the string start address is a kernel start address. Basically, try to read the whole kernel as a one, one, one long sting. That was my idea.

To achieve that, I need some of the constants from a kernel to basically construct this like C++ based string code. So to give you more context, these objects, they exist in a kernel space. From a user space, you can describe which objects you want to allocate in a kernel. You can describe that you can allocate in a kernel space, you want to allocate a string object. Here was the properties of this object. The starting address is this and the length of the object is this and so on.

So by constructing an object that points to the beginning of the kernel, and if we know the raw kernel size, we can try to reach the kernel to user space.

[SPONSOR MESSAGE]

[0:31:29.0] JM: Nobody becomes a developer to solve bugs. We like to develop software because we like to be creative. We like to build new things, but debugging is an unavoidable part of most developers' lives. So you might as well do it as best as you can. You might as well debug as efficiently as you can. Now you can drastically cut the time that it takes you to debug.

Rookout rapid production debugging allows developers to track down issues in production without any additional coding. Any redeployment, you don't have to restart your app. Classic debuggers can be difficult to set up, and with the debugger, you often aren't testing the code in a production environment. You're testing it on your own machine or in a staging server.

Rookout lets you debug issues as they are occurring in production. Rookout is modern debugging. You can insert Rookout non-breaking breakpoints to immediately collect any piece of data from your live code and pipeline it anywhere. Even if you never thought about it before or you didn't create instrumentation to collect it, you can insert these nonbreaking breakpoints on the fly.

Go to rook out.com/sedaily to start a free trial and see how Rookout works. See how much debugging time you can save with this futuristic debugging tool. Rookout integrates with modern tools like Slack, Datadog, Sentry and New Relic.

Try the debugger of the future, try Rookout at @rookout.com/sedaily. That's R-O-O-K-O-U-T.com/sedaily. Thanks to Rookout for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:33:34.0] JM: Being able to read it, does that mean you also know how to write to the kernel or – This is all about wanting to set up read and write primitives, because if you can read and write to the kernel, you basically have full control over the system. You can do anything. So once you had figured out the mapping to be able to read the kernel, does that also give you the ability to write to it?

[0:33:56.9] MB: No, but it depends on the vulnerability you get. My vulnerability was pretty powerful, which allowed me to execute in a code in a kernel space. Again, by using this fake string object. But if I can execute a code, I can execute some parts of the kernel. For example, like some assembly instructions that allow me to read or even like write to a kernel memory.

So answering your question, it depends on how powerful your vulnerability is. Mine was pretty powerful, which allowed me to read and write and even execute just in one vulnerability. Some

of the vulnerabilities, they're limited that you can only read. For example, when I was like crashing a kernel, it allowed me like only leak or read four bytes in a time, but not to write. In this case, you need additional vulnerability, this allowed you to write.

[0:34:53.3] JM: With the Apple Watch, you're typically in a sandbox mode. The developer is in a sandbox mode. There is a limited set of things that you can do. Are we beyond the point where you're constrained by the sandbox, or does the sandbox still limit you once you have this read/write primitive idea setup?

[0:35:14.3] MB: Yeah, sandbox is pretty powerful mechanism to prevent exploitation. Luckily, the software vulnerability that I found, it was not restricted by sandbox, which mean from my application extension, I can run this powerful API that can interact with a kernel, and it was not protected by sandbox. This kind of vulnerabilities, they're pretty rare, but they exist.

In other case, if I don't have this powerful vulnerability, I need additional vulnerability to get out of sandbox to find a way how to, for example, interact with APIs that I need or basically escalate my privileges to not be enforced by sandbox restrictions.

[0:36:01.2] JM: Once you have read/write capability in the kernel, what does that let you do?

[0:36:07.6] MB: Yeah, basically, as I say, most of the security restrictions, they're located in a kernel. Like code sign in enforcement, the sandbox itself is just a driver in a kernel. Other things that prevent us to, for example, expound or like launch the new processes in a system or so. They're all, all the strings are in a kernel.

When I get the read/write primitives to a kernel memory, I can do hot batching on a fly. Basically, modifying kernel code when the kernel is running. This is what I've done. So I know where the kernel is located in a memory. I read the whole kernel to a user mode so I can disassemble it. I can find the functions in a kernel that I need to modify. I start to do my batches in a memory just by changing the assembly instructions, basically.

[0:37:02.7] JM: So you set up read/write primitives. You disable security protections on the watch, and then you can set up SSH. If you get SSH set up, you've effectively jailbroken the watch. Why is that? Why was the goal to set up an SSH connection? Why is that as good as having totally jailbroken and having control over the whole device?

[0:37:25.1] MB: Yeah, basically, SSH is a part of a jailbreak process. So after you disable the security features of a system, like by patching a kernel, you need to provide some easy interface to interact with a system, because installing application extension each time to run your code is maybe not the best thing. SSH looks like a pretty good candidate that you can just make a connection directly to your watch and upload any file you want, any executable you want, run it and so on. Basically, you get an interactive console with your watch. Yeah, with full attest the file system, with full attest the memory and so on. You can do many things.

[0:38:08.2] JM: What frictions did you encounter when you were trying to create that SSH connection?

[0:38:13.6] MB: That was a tricky part. First of all, I was following the same steps as I was doing on iOS before by disabling the restrictions in a kernel. I was surprised that it was not enough. So in watch OS, it works much more restrictions in a kernel that prevents you to run any kind of SSH client on a watch, which makes sense, because Apple design its watch to not interact with network. Only use the iPhones for this purpose. Yeah, it required me additional patches in a kernel to even run the SSH connection.

Later on, I found that there is no good connection point to the watch, as watch is interacting only with iPhone and it is interacting only with a Bluetooth. It was a challenge how I can run my SSH connection over to the Bluetooth after looking for ways what we can do with that.

[0:39:12.2] JM: Yeah. So you needed to create a Bluetooth connection with the Apple watch using an iPhone in order to get this SSH connection running. That's something I didn't completely understand. Can you explain that in more detail?

[0:39:25.9] MB: Sure. The watch interacts with outward only by using an iPhone. Watch cannot interact with any other device, except the iPhone that is pair a device, which mean all the

information that watch retrieve, for example, even like application extension, it synchronize it from a phone using the Bluetooth connection. Which mean if I want to, for example, directly connect from my Mac to a watch, there is no easy way to do it. So I need to somehow connect to my phone. From a phone, I can connect to a watch. Again, because, phone using as a gateway to a watch.

[0:40:05.4] JM: Got it. I think we're at the point where you have achieved the jailbreak. So did you actually do anything? Do you had the watch jailbroken or mission accomplished?

[0:40:15.8] MB: Yeah, the first thing I just run the SSH connection and like proved that it works, that I can run any unsigned code on this tiny device. I start recompiling the open source software for this watch OS specific architecture as it is running on its own chip and you cannot just reuse the software. You need to recompile it for a watch, but doable.

I start recompiling the common line tools, like – I don't know, like [inaudible 0:40:47.3] and like LS, like move, like copy and these kinds of things to give some of the capabilities to newly jailbroken Apple watch system. I've recompiled some of the Python interpreters. So you can run Python on it. You can run servers on it on a watch. You can even connect – Remove some of the limitations in a kernel. If you connect your watch to a WiFi, you can even talk with [inaudible 0:41:18.4]. Basically, you can browse websites using your Apple Watch after jailbreak.

Of course, you can apply any modifications to a system, apply your own – I don't know, watch screens, change the UI and so on and so on. For me, it was that I can run any code on it. Basically, anything I want.

[0:41:42.0] JM: This talk too place at DEFCON, which is a security conference. What was the reception to the talk?

[0:41:51.1] MB: I would say it was pretty good, as it was the first public Apple Watch jailbreak. Previously, nobody take that seriously. How much information Apple watch know about the user. For example, all the messages, all the emails, all the GPS coordinates, all photos, music, health information and so on. So there's a lot of privacy on a watch.

Yeah, as soon as jailbreak get achieved, rise more questions about how protect our data. Yeah, even hacking this small device still can be pretty dangerous in a case of like data leak and user leaking. Yeah, I get a lot of questions from security researchers. How I achieved this? What was the challenges? I made a pretty detailed presentation about the vulnerabilities that I use and how I exploit them.

I would say it was a good starting point for a future watch OS research for security researches. So they can now understand how the system is working. What is the limitations? What are the security restrictions? What is the possible attack vectors? I would say for future, it's like more helpful to find more vulnerabilities in the system, report them and make the software more secure.

[0:43:17.1] JM: You work at Lookout. Lookout is a company that build mobile security products for organizations. What do you do at Lookout? What are the core Lookout products?

[0:43:30.4] MB: We do a set of products starting from a mobile protection when it's the software that checks for possible malware on your device. That check for possible vulnerabilities in your device. It is checking is the network secure or insecure. It is checking, for example, if you're trying to open the phishing site and so on.

My role in Lookout is a research. So I'm looking for new types of attacks on mobile phones. I'm looking for new types of attacks to bypass the security mitigations and sometimes reverse engineering how the system works to find a way how we can interact with the system. I would say, like broad set of tasks that I'm doing.

[0:44:14.0] JM: I think a lot of people have a perception that the threat vectors in an organization, particularly like social engineering related vectors, which I know are a lot of things that come into an organization these days. I think people have a perception that if they open these social engineering links and thinks on their mobile device, it's not as big of an issue. The real concern is people opening things on their laptop or on their desktop device. How much truth is there to that idea? How vulnerable in your typical enterprise are mobile devices to the typical attack vectors?

[0:44:55.5] MB: Well, I would say that mobile device, they're under the hood is pretty much the same operating systems. They're much more protected than the desktops. Still, it's operating system with its weak spots and limitations. I would say with the rise of bring your own device, I think it's much more workers use their mobile device to use their work email, to install some sensitive information.

I would say it's not going to replace the desktops, but it's pretty good additional attack vector, because you store the same documents on a mobile device. As soon as one mobile device will be comprised, attacker can get attest to internal network of an organization or attest to internal emails and so on. It's harder doing it on desktop. But still, that's attack vector.

[0:45:50.1] JM: Okay. Well, Max, it's been really great talking to you about jailbreaking Apple Watch, and I can see from the perspective of somebody that does mobile security for a living, why this would be something that's on your mind. I mean, an Apple Watch is just another mobile device and just another computer as you've explained in circuitous detail, and you've got to be aware of the security vectors that device could be subject to. If nothing else, then you probably learned about some vulnerabilities in the Apple Watch that Lookout might have to secure against in the future.

[0:46:28.8] MB: Yeah. For our listeners, if they want to go deeper into vulnerabilities, I think we can provide the link to a whitepaper with a very detailed explanation how they can reproduce this kind of attack on their watch.

[0:46:42.4] JM: Sure. Happy to do that. Okay. Well, Max, thanks for coming on Software Engineering Daily. It's been great talking to you.

[0:46:46.6] MB: Thank you, Jeffrey. It was a pretty good talk. Thank you.

[0:46:49.9] JM: Okay. Thanks.

[END OF INTERVIEW]

[0:46:53.1] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

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

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

[END]