EPISODE 631

[INTRODUCTION]

[0:00:00.3] JM: Git is a distributed file system for version control. Git is extremely reliable, fast and secure and this is owing to the fact that Git is one of the oldest pieces of open-source software, but even battle-tested software can have vulnerabilities. In this episode, we explore a subtle Git vulnerability that could have potentially led to Git users executing malicious scripts when they intended to simply pull a repository.

Today's guest, Edward Thompson is a Program Manager at Microsoft and he's also a maintainer of libgit2, which is a C implementation of Git. Edward writes about Git and he hosts the podcast All Things Git. He is passionate about Git development. This gave me deeper perspective on something that I just consider a tool, but the only reason that tool Git is so good, the only reason that it fades into the background for us is because there are people that are passionate enough to work on it on a regular basis.

Edward and I also spent some time talking about the vulnerabilities that can spread through shared code environments, particularly in the realm of Git NPM, the node package manager and PHP. We also touched on how development work flows around Git and Kubernetes are changing.

Full disclosure, Microsoft where Edward works is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:37.8] JM: You listen to this podcast to raise your skills. You're getting exposure to new technologies and becoming a better engineer because of it. Your job should reward you for being a constant learner, and Hired helps you find your dream job. Hired makes finding a new job easy. On Hired, companies request interviews from software engineers with upfront offers of salary and equity, so that you don't waste your time with a company that is not going to value your time.

Hired makes finding a job efficient and they work with more than 6,000 companies from startups to large public companies. Go to hired.com/sedaily and get $600 free if you find a job through Hired. Normally, you get $300 for finding a job through Hired, but if you use our link hired.com/sedaily, you get $600, plus you're supporting SE Daily. To get that $600 signing bonus upon finding a job, go to hired.com/sedaily.

Hired saves you time and it helps you find the job of your dreams. It's completely free. Also, if you're not looking for a job but you know someone who is, you can refer them to Hired and get a $1,337 bonus. You can go to hired.com/sedaily and click refer a friend.

Thanks to Hired for sponsoring Software Engineering Daily.

[INTERVIEW]

[0:03:20.2] JM: Edward Thompson, you are a Program Manager at Microsoft. Welcome to Software Engineering Daily.

[0:03:24.2] ET: Hi. Thanks for having me.

[0:03:25.5] JM: You're a maintainer of libgit2, which is a C implementation of Git, and you write, you podcast about Git, you have a podcast called All Things Git. Why are you so interested in Git?

[0:03:38.8] ET: That's a wonderful question and nobody's ever asked me that before, but I am. I am really interested in Git. I think it goes back, oh, geez. I don't even know how many years now. Let's call it 15 to make me sound younger than I actually am. I was working for a small company in central Illinois called SourceGear. Jeff, do you remember Visual SourceSafe?

[0:04:02.5] JM: I don't know what that is.

[0:04:04.0] ET: Oh, good. You dodged a bullet in your software development career. Visual SourceSafe was Microsoft's first stab at building a version control system. It was not good. They acquired it from this company and it was called SourceSafe back in the day, and it was a

command-line tool. It was revolutionary for what it was. Microsoft bought it and turned it into Visual SourceSafe and then let it languish. It wasn't great, and one of the things that it really didn't do was client-server. That's important, right? The way it communicated was over file shares and it was not good.

This company called SourceGear in central Illinois realized this and built a TCP/IP layer on top of it. That was my introduction to version control. I worked at SourceGear. I worked on a product called source off-site that enhanced Microsoft's Visual SourceSafe, and that was my first introduction to version control, and I've been stuck with it ever since. Now it's of course Git, because Git is the dominant version control system.

[0:05:09.4] JM: People do use Git for source code version control; that's the main use case of Git. What I wonder is if you look at Git from a different angle, it's a big decentralized file system, which is a bigger idea. You see people like – I don't know if you've seen the IPFS project, but they take a lot of inspiration from Git and use that inspiration to build a decentralized internet protocol. For somebody like you who spends so much time in Git, do you think of it as just a source control system? Is that enough to keep you excited about it, or do you have a more expansive view of what Git could eventually become?

[0:05:54.3] ET: I think of it as a version control system. I look at everything through a version control landscape, but that's just because I've been doing it for so long. What you say is exactly right. It is a big decentralized file system with the ability to branch and merge. IPFS is a really interesting example. I was actually talking to someone who wanted to build basically a Git semantics, I guess if you will, on top of IPFS. I think that that could be a really fascinating development.

Yeah, I think that a lot of people might look at Git as a file system and I hope that they do. I hope that people take it in a bunch of different directions. I think that it's really exciting, but not me. I'm just focused on developer productivity really.

[0:06:39.6] JM: Which still is a Greenfield opportunity. There's still a lot – long ways to go in terms of making developers more productive, even just in terms of how they use Git.

[0:06:49.8] ET: Absolutely. Absolutely. It's shocking. We've been working with Git at Microsoft for oh, I don't know, five or six years, not very long in the grand scheme of things. Even in that time, I thought that we would be a lot farther along, I guess, or we would have built a lot more Git tooling than we had as an entire industry, not just Microsoft. There's still so much we can do to make people's lives easier. You're absolutely right, there's a ton of opportunities.

[0:07:19.2] JM: Like what? Tell me something about the low-hanging fruit.

[0:07:21.5] ET: Oh, making it easier to use. Straight up, I'll just be honest with you. I think Git on the command line is very powerful, incredibly powerful, but with that power comes a lot of challenge if you're just getting started with it, right? There's this thing called HEAD and it's in all caps and it's special. Maybe it's not always in all caps depending on your operating system. I think that there's a lot of maybe pitfalls that people getting started with Git can fall into really easily if they don't take a step back and understand the system.

A lot of people will say that you really need to understand how Git works at a low level to be able to use it. I think that that's actually true and I think that it's disappointing that it's true. I think that we could do a better job of getting people started easier.

[0:08:10.4] JM: Yeah, and literally I think every job I had before I started the podcast, there was some point in my job where I got into a state in Git that I did not know how to like retrieve the work that I had obliterated somehow, and I had to get some senior engineer to help me reestablish what was going on in my local workspace. Maybe I'm just uniquely bad at Git, but I feel like I'm not. I feel like other people have – a lot of other people have endured through that.

[0:08:40.6] ET: You are not unique. That is sad. I think that we can do a lot to make people's lives easier. Even little stuff. I wrote this little app called Git Recover. What happens is you at one point run Giy Add on a file, and now for whatever reason that file is now gone, right? Maybe you made some changes in Git Add again, so how can you recover that? It's undelete for your Git repository. It'll just go through and see what has been put into your Git repository at some point in time and it will offer you to recover them.

As long as you've run Git Add, you can get that data back, because whenever you do that, it goes right into the object database. It may not be referenced anymore. You may not see it. It may not be obvious as to how to get to it, but you can get it back. I think stuff like that is low-hanging fruit.

[0:09:26.8] JM: I've used Git since college. I've used it the same basic way I think for most of my career, just using Git Add and check out and clone and those basic operations. Has Git itself evolved much in the last six, or seven, or eight years, or have there's been low-level efficiency improvements and security patches like we're going to talk about today? How has it been evolving?

[0:09:57.1] ET: I think from a user perspective, it evolves very quietly. Because Git was built originally by Linus Torvalds and Linus built the Linux kernel, of course. If you've ever seen any of Linus's real rants on the Linux kernel mailing list, it's always about breaking the kernel's contract with the user space programs.

The idea is that those APIs have to be completely stable. I think that get picked up that idea and carries it forward. I think that that's really nice. I think that that's important, so that if you write an application, let's say Visual Studio 20, the most recent version of Visual Studio now talks to Git, the command line application. It can basically talk to a number of versions of Git, because those command line contracts don't change. They've picked up that same ideology. It's not always obvious from an end-user perspective, especially if you are set in your ways as far as a workflow goes and I am too. You may not notice that Git is really changing all that much, but actually under the hood it's changing a lot.

We've just introduced a new on-the-wire protocol, which we'll make more efficient. Microsoft has actually been introducing a lot of improvements on I guess speed ups, if you will, for handling large repositories, because we've got a ton of really big repositories within Microsoft. Then you'll see other changes going on for more efficiency and just usability improvements. Again, if you just run the same things, you might not even notice them.

[0:11:32.9] JM: The efficiency of cloning a large repository, I could see that becoming increasingly important, especially with people just spinning up and tearing down environments,

laissez faire, deploying Kubernetes cluster and then wanting to clone all the infrastructure on to that cluster and then spin it up and then do another one. I mean, I can imagine the frequency with which people are cloning repositories increasing. With that frequency increasing, there's more demand on the on-the-wire protocol to get faster.

[0:12:10.5] ET: Yeah. The big improvements in the new on-the-wire protocol are really around when you have a lot of branches. When you run Git fetch, you get information about all the server's branches, right? These new protocol improvements really shave the amount of information that's transferred there and the handshaking there. Interestingly, when you run Git clone, we could do better compression or this and that, but the biggest change that we've made to improving clone is something called bitmaps. There's a really awesome Github engineering blog post about how this works. It used to be that you would clone the Linux kernel from Github and basically you would just see this message on the console that just said counting objects.

What Git was doing was trying to figure out what it needed to give you over the network. It would basically just walk the whole history, like if you were on Git log, it was doing that. On a big repository, a biggish repository like the Linux kernel, that takes forever. There were all these cool server-side fixes that they made to be able to compute that really quickly. That is one of the hidden things is all these – the hosting providers making these improvements. Github has done a lot of that.

We've done a lot of that at Microsoft, because we've started hosting the windows source tree, which is the biggest Git repository on the planet. We've had to make some really interesting improvements there. There was actually just a really cool blog post. It was Yesterday that it came out about some of the stuff that we've done, that we've actually moved in to Git; it's available in the most recent version of Git to speed that up. There's a lot of cool stuff going on in a lot of different places around performance.

[0:13:51.1] JM: Long story short, Git is evolving quite rapidly, or there's a lot being done to it.

[0:13:56.5] ET: Absolutely.

[0:13:57.6] JM: But it's quiet. We've done some previous shows on Git and we've covered some discussions at the basic commands. I want to get your definitions of some of these basic commands to set us up to talk about the vulnerability that we're going to get to. Not just that, but also I think this is hopefully going to get a deeper understanding for how some of these commands work. Let's talk just Git clone, right? I think most of people who are listening to this have probably used Git clone, but they may not know exactly what's going on under the hood. When I clone a repository, what's happening on my client? What's happening on the server that I'm cloning from?

[0:14:39.0] ET: Sure. When you run Git clone, I hope most people have, your client contacts the server and says, “Hi, I'd like a copy of this repository.” The server says, “Great.” It looks at its copy of the repository. When I say repository, I actually mean the history of your commits, the commits themselves and the files that have been checked in from the very first version to the very oldest version. If you have ever looked inside the folder that you're working in, your “working folder,” we often think of that as the Git repository. From a get developer point of view, we think of there's a folder in there called .git and that's really the important bits of your Git repository. That's where Git stores all that information, all the history, all the files that you've ever checked in are in there.

On a Git hosting provider like Github, or VSTS, or whoever, they store the contents of that .git folder on their servers. They don't check it out, so it's not like if you've got helloworld.c, you won't see that on the servers. You'll just see the contents of that .git folder. When you run Git clone on your client, the server will send down all of the information that is in that .git folder. It'll send down all of the objects, so that commits and the files that you've committed, and then the trees they're called, which make up the directory structure of every commit. Those get put into what's called a pack file and sent down to your client. It's basically like a big zip file is the easiest way to think of it. That gets sent down to your client.

Then the other thing that gets sent down is the information about the branches that are on the server. That gets stored on your client as well. Those go into the .git folder. Then Git checks out the most recent version of your default branch. It’s usually master. That's Git clone in a nutshell. The idea is that it gets a full copy of the repository off of the server and puts it on your machine and then checks it out.

[SPONSOR MESSAGE]

[0:16:52.5] JM: This episode of Software Engineering Daily is sponsored by Datadog. With automated monitoring, distributed tracing and logging, Datadog provides deep end-to-end visibility into the health and performance of modern applications. Build rich dashboards, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt at softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:17:39.6] JM: The .git is a directory, right? If I clone a repository, the remote repository that's hosted somewhere on the internet that repository has a folder that is the .git directory, which contains the information about the repository. Then within that directory, there's a lot of information. It tells you the schema of everything that is in this repository. It includes the information about the different modules and the different branches and all the different things that are in this repository. It also contains a config file. What is in that config file?

[0:18:18.7] ET: Yeah. The config file is something that just gets generated when you create a new repository. It's a special file, so when you run Git clone, you don't get the server's config file. You get your own config file generated when you run either Git init, or Git clone, or whatever when you when you create a repository locally. That's your local configuration data. It's important that that stays local, because it contains stuff like aliases. You could run commands based on the things that go into that config file. It needs to stay local. You wouldn't want to get some information from the server and maybe run a Git command and all of a sudden it's running something else. That would be terrible.

There's some cached information in there about maybe the way your computer works. Are you on a case insensitive file system? Get to text that when you create a repository. It looks at your file system and it does some checks to see if your case sensitive or case insensitive, and then it'll cache that information in Git config, so that it doesn't have to look it up again. Some people

think that that's a configuration option that you can change. It's really not. Don't change it, but it also – your Git config also has things like your name, so when you run Git commit, your name and e-mail address are recorded in that commit. That information is stored in your Git config as well. It's definitely a per user file, or per repository.

[0:19:43.2] JM: It's per user, it is a client-side construct. When you pull, when you clone a repository, you don't expect to get a Git config file. One thing that a config file contains I believe is hooks. Git hook is part of your config file. What is a Git hook?

[0:20:04.0] ET: It's not exactly part of the config file, although it's easiest to think about it that way. Also inside .git you've got this directory called hooks. There's a bunch of when you run Git init, you'll actually see pre-commit.sample in that directory. The idea is that those are programs that will run at various points of the Git workflow. When you run Git commit, if it will look inside that hooks directory and see if there's a file called pre-commit. If it is, it will run it and you could set that up to validate the changes before the commit actually goes into your local repository.

You could run a linter for example. You could run static code analysis. You could do any number of things, because it's just a shell script, or it could even be an executable that runs as part of the workflow. Then you could use that to reject the commit if it's got badly form. Maybe it uses tabs, instead of spaces. Obviously one is superior, and so you could reject that based on the pre-commit hook. It's not exactly part of the config file, but it's just as important that that's local, because if you ran Git clone and all of a sudden you got a pre-commit hook from somebody and you ran – you ran Git commit, all of a sudden you're running code that you didn't write, that you hope is trustworthy and it might not be.

[0:21:30.2] JM: Can a hook do anything within my operating system? Is this like a bash script that's going to run under typical circumstances?

[0:21:37.7] ET: That's exactly right. I mean, it can't do anything. It won't run as root, but it can do anything that you can do.

[0:21:42.4] JM: I can do RM-RF?

[0:21:44.4] ET: Absolutely can.

[0:21:45.6] JM: Yeah. A Git config file has, well essentially, it can interface with hooks, or it can run scripts under certain circumstances, but this config file is defined locally. You've clearly outlined why it is important that this information is defined locally on a per user basis, because you wouldn't want to clone the script that could potentially do anything and accidentally execute them. Maybe just emphasize, before we get to this vulnerability, why is it so important that our Git config file and the hooks that are associated with it are defined on our local machine?

[0:22:28.1] ET: Right. Can you imagine running Git clone, github.com/ethompson/badrepository and all of a sudden a Bitcoin miner started on your computer? I mean, if I told you, “Hey, check out my cool new Github project,” and you cloned it and all of a sudden your computer was – all your files were encrypted and locked and ransomewared, that would be bad. It's not what you expect to happen.

It's really important that all that stuff is only created on your local repository. It doesn't get sent down as part of the Git clone command. The person who creates a repository and puts it on Github can't do any remote code execution on your machine. That's super important and that's why Git has designed the config and the hooks to be completely only ever written on your local machine.

[0:23:19.3] JM: Right. We don't typically want to think when we run Git clone, we don't want something to start running. We just expect, oh, we get some files and then we can do things with those files, like explore them, but not execute code.

[0:23:34.7] ET: Right. If you want to build it and run it, that's fine. That's up to you, but you get to review that code before you build it and run it. It'd be terrible if there was some problem and all of a sudden Git was able to do whatever it wanted.

[0:23:48.1] JM: Last month, there was a vulnerability discovered in Git. How was this vulnerability discovered?

[0:23:54.7] ET: This vulnerability was as best I can tell, and so I've actually interviewed the person that did this, but it was a security researcher. His name was Etienne Stalmans. Etienne

was basically looking around, I think he just started a new job, and he was cloning his Git repository and he realized that it had sub-modules in it. He said, “You know what I don't know anything about is Git sub-modules.”

He started poking around and realized really quickly that he could get Git into a situation where it was actually running code that he had written through a hook. He could put a hook into a Git repository and convinced one of his co-workers, or whomever to clone it and it would run whatever he wanted it to. I think he then realized that, “Well, I wonder if I can take advantage of this in a bug bounty program?” He realized that Github pages, I don't know if you use Github pages, it's that –

[0:24:52.2] JM: I do. I do.

[0:24:53.6] ET: You do, okay.

[0:24:53.6] JM: Yes.

[0:24:54.6] ET: Yeah, me too. It's awesome, right? I use it for my blog.

[0:24:56.6] JM: Fantastic.

[0:24:57.9] ET: Yeah. I push up some markdown and it gets rendered and spits out something on ethompson.github.io or whatever. The way it does that is by cloning my Git repository that I push out to Github, onto Github pages, onto their website, because it's a static site generator. It runs Git clone and then it runs Jekyll. It turns out that it runs Git clone --recursive, which is the sneaky key to this puzzle that he found with sub-modules. Since there's that bug in sub-modules, you can exploit it on Github pages.

He was able to create a repository that had hooks in it using this bug, using this vulnerability, push it up to Github and then Github pages generated a site for him by running Git clone on his sneaky repository, and he got access to Github pages. It was found actually through the Github bug bounty.

[0:25:54.1] JM: The difference here between sub-modules and the repository itself that you're cloning, why is that difference important? Can you define exactly where is it that he discovered the vulnerability?

[0:26:10.6] ET: I can. It's a little bit challenging. This is not a trivial vulnerability. When you run Git clone, you get a copy of a repository from a server. If there are sub-modules there, then you don't actually download those sub-modules. You have to –

[0:26:26.8] JM: By the way, sub-module, is that interchangeable with subdirectory?
[0:26:29.5] ET: No, it's not. Sub-modules are a specific part of Git that allow you to put a Git repository inside another Git repository. Sub-modules are one of the – I said earlier that Git has some challenging usability issues sometimes. Sub-modules are one of those things. Sub-modules are a tough concept and they don't – a lot of people want to use them to split up a repository, or something.

Sub-modules work really well if you have some code that is a source dependency that you need to take. Sub-modules don't work really well in in a lot of other scenarios. They're not super common and how software works. The software that you run all the time gets the most attention, software that you run less frequently gets a little less attention, and so there can be some bugs lurking and this was one of those places.

When you run Git clone, you get the repository that you ask to clone. If there are “sub-modules,” they won't be downloaded. When you run Git clone --recursive, what will happen is Git will clone the original, the parent repository and then it'll look for any sub-modules. You might say, I've got a sub-module. It's in directory foo in my working directory and it's at github.com/wherever. Then, so Git will “recursively” do that clone. It'll get the parent, it'll look for any sub-modules and then will clone those sub-module repositories into the parent repository. It'll basically set up this big structure containing all of the repositories that are sub-modules within the parent repository. Again, you're not going to get the config from those either, or the hooks from those either. At least you shouldn't. The vulnerability that Etienne found was here.

[0:28:17.2] JM: Can you give a description for how that vulnerability could end up executing malicious code?

[0:28:25.5] ET: Yeah. Again, this is – let me crack open the .git folder a little bit again. I apologize for the boring details of this, but when you have a Git sub-module, it will record that information in a file, in your .git directory with all that other configuration data called modules, right? That basically is a mapping of where the sub-modules live within the working directory and the URL that they were cloned from, because you need that information in order to basically build the structure of your Git repository when you check out, or when you switch branches, or anything like that.

It's basically just three simple things. There's the name of the sub-module. By default, that's the same name as the folder that it goes into on disk. It's just a key, right? It's not important. It's just a key. Then there's the place that it gets checked out into your parent sub-module. You could create a new folder called foo and it points to http://github.com/ethompsonfoo and that's no problem, so it would also have the name of foo. The thing is that when you look at this file, you think of it and you think, “Well, that name is not really important.” It just happens to be the same name as the folder that it goes into in the working directory.

It turns out though that that name is important. It's actually really important, because the name of the sub-module is actually used inside the .git directory to create the location where the sub-modules .git directory goes. This is getting a little weird, I know. If you have a repository, there's a .git directory. Inside that .git directory, when you have sub-modules, their .git directories go into the parent’s .git directory. They're not called .git anymore. They're given the name of the sub-module.

If you change the name to something else, Git would respect that. I don't know why you would ever do that. This is just a weird accident. You could change the name to from food to bar and all of a sudden, it's not going into foo, it's going into bar. You could change it to ../ and all of a sudden it wouldn't get written into the .git modules directory anymore. You could change it to ../../ and it wouldn't be getting written into the docket directory at all. You would actually be getting written to the .git directories parent directory, which happens to be your working directory.

You could actually run Git clone and it would read the sub-module information from the parent repository and end up writing data into the working directory. That seems weird. It would put the dot Git directory, like splat somewhere in the working directory, or even anywhere on your disk. That seems weird, but it doesn't seem like an exploit.

The thing is you could actually publish a Git repository that had the contents of a sub-module's Git repository in the working directory. This is getting a little bit weird, so if you ran Git clone without the recursive flag, what you would see is this folder on disk and it's called foo, and it's got all the contents of the .git directory in it, like there's a file called head, there's a directory called objects, there's a config file, all that stuff. You'd look at it and go, "Well, that's really quite pointless." If you ran Git clone --recursive and your Git sub-module configuration was set up with this name that points to that directory, what Git is going to do is it's going to clone the parent, it's going to look at the Git modules, it's going to say, "Okay, well I need to clone this sub-module."

It's going to look at the name and it's going to say, "Well, okay. I know where I'm going to put this on disk." Obviously, I don't have this Git repository yet and it's going to open it and it's going to say, "Oh, well actually I do. I don't need to clone it at all. I've got it." It will just merrily continue on and it'll go and check out the data from that Git repository that you've put on disk. When it's done checking that out, it's going to run any of the hooks like a post-checkout hook that you might have.

You can publish a repository that has another Git repository inside of it, that there's some module configuration points to and then you can put hooks inside that Git repository that are going to run. Just by running Git clone --recursive, it's going to actually open up that hook and run it.

[0:33:09.4] JM: To be clear, why is the sub-module involved in this vulnerability? Again, why can't you have potentially malicious Git hooks in the main repository? Why is that not a vulnerability?

[0:33:23.6] ET: There's no way to actually publish them. If you try to check in a file into the .git folder yourself, Git will tell you that it can't do it. If you clone a repository that somehow snuck it

in, you could turn all those checks off in Git and then create a new repository in Git add .git/hook/postcheckout and commit it and publish it on Github. First of all, Github is going to reject it, but you could publish it on your own hosting site that also has turned off all this validation.

Still when somebody goes to clone that, their client will also do the validation. There's multiple steps involved in making sure that .git folder is never actually checked out on disk, except in this vulnerability of course.

[0:34:06.2] JM: Right. The Git server program prevents you from, or at least the implementations of it if we talked about Github for example, they have hard-coded ways of preventing you from putting a Git hook, this ability to – basically this configurable code that runs upon checking out, or in the – while you're in the process of checking out repository, or cloning a repository. This is not a recursive relationship, so it doesn't necessarily apply to the sub-modules within your .git directory on that remote server.

[0:34:45.1] ET: That's right. At least it didn't. I mean, it –

[0:34:46.8] JM: It didn't.

[0:34:47.5] ET: It certainly does now.

[0:34:48.0] JM: It does now. Yeah.

[0:34:50.3] ET: Right. Yeah. When we see a security vulnerability in Git and at least if it's disclosed responsibly, there's a Git mailing list for security that the various maintainers of Git and other implementations of Git like libgit2 and jgit are on, and always the first thing we do is we patch the hosting providers, so that nobody could use us as a vector for distributing these evil Git repositories, we call them. The first thing we do is we patch that, so that you can't push a Git repository that has a hook, or a malicious bit of code in it.

[0:35:24.3] JM: That's how Microsoft responded to the event, the acute response.

[0:35:29.8] ET: That's right. That's right. Github patched, Microsoft patched for visual studio team services. I know that Git lab announced a patch. I assume bitbucket did too, but I didn't actually see an announcement. Sometimes they just do it quietly and they don't want to draw attention to the matter.

[0:35:44.0] JM: Sure. What about the more comprehensive response? How did the vulnerability patch – what was the process of the vulnerability patch making its way into the core Git libraries?

[0:35:57.3] ET: Again, this this discussion all goes on in the security mailing list. Basically, somebody proposes a patch. In this case, I think it was Jeff King, Jeff aka Peff. He does a lot of the security work on Git. He proposes a patch and then everybody tries to attack it basically, like what avenues did you not think about? What are the idiosyncrasies of various operating systems that I might be able to use to still execute this code path that you didn't think about? Is case sensitivity a problem on a file system? I think in this case it was.

There are all sorts of fun, little filesystem quirks from HFS+, to NTFS, to EXT3 that are just a little bit different. We all tear that down and then once we're reasonably happy with it, that's when we patch all our servers, and then we let it bake for a little while, make sure nobody has any last-minute ideas. Then we release the patch.

We try to do simultaneous releases, at least I do on the libgit2 project. What I like to see is when there's a security announcement, Git will get patched, Git for Windows will get patched, because that's actually a different project than Git. It's got some release activities that have to go on and those are always challenging. Libgit2, we like to simultaneously ship with Git when it comes to security releases, and we've got our own security process, so we announced to everybody who uses libgit2 to that they're going to need to update, because it's used in the servers like Github and Git lab and visual studio team services. It's used in a bunch of get clients. We want to make sure that everybody is updated and safe all at the same time.

[0:37:38.9] JM: What are some other major vulnerabilities of the past that have affected users within Git?

[0:37:47.0] ET: The first one, in fact, the one that led to this coordinated Git mailing list was back in 2014. It was the first way that anybody figured out a way to write into the .git directory in it. Thankfully, it was much, much easier. I know I went on for a long time setting up how the Git module problem worked, because it was like I said, pretty complex. This one was really easy. When you run Git add, or Git status, or whatever, the .git directory is totally ignored. If you run Git add .gitconfig, Git will reject it. No problem.

Like I said, Git was built by a guy named Linus Torvalds, and he built it to manage the Linux kernel. No surprise, a lot of the people building and maintaining Git have come out of that environment. Most of Git is written in C. The way it protected itself was StrComp. If you're on Mac OS, or Windows, you might notice a problem here and that is that StrComp is case-sensitive. If you say – well, let's make sure that nobody writes into the .git directory and you use StrComp to do it, you're not going to prevent somebody from writing into the .GIT directory.
It was really easy to add a malicious hook for example, and run Git add .GIT./hook/postcheckout. Then you could commit it, you could push it up to Github and then you could encourage anybody to download your really awesome program. As soon as you cloned it, it would run whatever they had added in there.

[SPONSOR MESSAGE]

[0:39:32.1] JM: Raygun provides full stack, error, crash and performance monitoring for tech teams. Whether you're a software engineer looking to diagnose and resolve issues with greater speed and accuracy, or you're a product manager drowning in bug reports, or you're just concerned you're losing customers to poor quality online experiences, Raygun can provide you with the answers.

Get full stack, error and performance monitoring in one place. The next time you're struggling to replicate errors and performance issues in your codebase, think of Raygun. Head over to softwareengineering daily.com/raygun. Get up and running within minutes and dramatically improve the online experiences of your users.

Thank you to Raygun for being a sponsor of Software Engineering Daily. If you want to support the show while also checking out Raygun, go to softwareengineeringdaily.com/raygun.

[INTERVIEW CONTINUED]

[0:40:36.4] JM: The environment of get security, how does that compare to these other environments where we find ourselves having some sense of blind trust over the code that we're pulling? Like NPM, for example. I don't know what half the stuff that I pull off of NPM does, but it seems to work right. I'm not exactly sure how or why it works correctly. With PHP modules that I import in WordPress, My luck has been a little bit worse. I woke up one morning and there were banner ads being served on Software Engineering Daily, and I was like, "What happened?"

Apparently, the code for some random plug-in I was using had changed, and all of a sudden I was serving ads for acai berries and reverse mortgages and, you know, why not? How does the security of these environments compare? Is it even comparable? Are there comparisons to be drawn?

[0:41:35.9] ET: I think there are absolutely comparisons to be drawn. I think you're right. It's all about trust. I don't know the NPM ecosystem as well as I know say the new Git ecosystem. Even there, I can add a new Git package to my dotnet app. All of a sudden I'm running code that somebody else wrote, and I'm probably not going to audit it. I'm going to trust it when people tell me that it works, that it in fact works. I'm probably not going to blindly trust some package on new Git that has had five installs and has no comments on it. I'll happily use note a time for, example, because I done a little bit of careful thinking. I think that's true on Git repositories too.

Now there's a huge problem if you decide to download a Git repository and build the code and run the code. To be completely honest with you, you expect to be able to Git clone something without all of a sudden running a Bitcoin miner, but it's pretty similar. If I came to you and said, "Hey, Jeff I'd really like you to run Git clone --recursive, this funny URL that you've never thought about." I think you might actually be a little skeptical about that. There's always a bit of social engineering involved, either to convince somebody to clone a repository that you've made maliciously, or to convince a perfectly legitimate repository that a lot of people use and clone, to get your malicious code in there.

If I send you a pull request, that's all of a sudden got a bash script in it, you're going to look at that and be like, “What's going on here?” We take these things super seriously in the Git community. To be completely honest with you, I hope anyway that most people's healthy skepticism will help them avoid any problems to begin with.

[0:43:26.4] JM: Now unfortunately, there was a time when you could trust reviews on the internet. There was a time when you could trust that if Edward Thompson from Microsoft sent me a suspicious-looking URL and told me to Git clone it, theoretically even then I would be like, “Yeah, why not?” I'm not worried about some Bitcoin miner appearing on my computer. Of all the vectors that you could attack, why would you attack programmers through Git? The number of programmers is increasing. The expectations for what we know about, like what we can – what programmers have been trained on in terms of security best practice, those are frankly changing and it's becoming more uneven. Are there some fundamental weaknesses here and things like that?

You said like, I'm not going to trust a new Git package with five installs. Well I mean, somebody could bluff those installs, right? Just spread out a bunch of DNS, different servers, servers in different places and install a bunch of packages and then use that to inflate the numbers. I don't know. Maybe I'm being paranoid, but it seems like an issue.

[0:44:36.8] ET: I think it's good to be paranoid when it comes to security. I'm one of those guys who carries a USB condom around and won't plug his phone in to charge on things. Yeah, I admit I'm a little crazy. No, I think it's healthy to be to be skeptical and to be paranoid. At Microsoft, we do a lot of security training. We have red teams that are constantly sending out little phishing exercises and making sure that nobody clicks on them. Of course, somebody always does.

I think that getting to programmers is a potentially very valuable target, right? Because you could get into their code, you could, I don't know, why not put a Bitcoin miner on their website, instead of Bitcoin mining their machine? I think that there are at least right now, there are so many easier hurdles to get into a developer's machine, or even into a production website than using Git. Our goal is to keep it that way.

[0:45:27.9] JM: Right. Okay. I want to zoom out a little bit on more optimistic discussions. Git in terms of a tool to make developers more productive, I don't know if you've heard this term Git ops. This is something that has coming out of the Kubernetes community. Have you heard of this term?

[0:45:47.1] ET: I have. I saw Kelsi Hightower tweeting about it.

[0:45:49.6] JM: Yeah. Do you know anything about that?

[0:45:52.0] ET: I know that the idea is basically it's a good name. Well it's a catchy name for infrastructures code setups, right, where you're checking in your infrastructure configuration, or your, I don't know, CI configuration, actually into your Git repository so that it's versioned alongside your code. I think that's a great idea. Are there subtleties to that that I'm missing?
[0:46:13.9] JM: I don't know. What I've seen is that Git ops is it defines Git as the source of truth for a cluster. I don't really understand how that's something new. I'm trying to understand what's new about Git ops. I guess, it's the continuous integration side of things?

[0:46:29.2] ET: To be completely honest with you, I think that it's just a catchy name. Maybe I'm missing something, but it seems like infrastructures code, you're defining your whatever, YAML and checking it into your Git repository. I think that's great. I think that's an absolute best practice. That stuff should be versioned right alongside your code. Because okay, let's look at CI. If you want to do a CI build you can't have a CI definition that's configured on the website, or something, because my master branch and my release two branch might be built differently. I might need to change the way things are getting built, maybe the infrastructure changes underneath me. I need my code to reflect that, so if I ever need to roll back, I can still build it. I don't have to go change some clicky boxes on a website. That totally makes sense to me that my build configuration YAML gets checked in. I think that's a great best practice. It's calling it Git ops that I don't love.

[0:47:29.3] JM: All right. Well I'll have talk to somebody else who's, I don't know, closer to the coining of that term. More broadly, how is the usage of Git in continuous integration workflows evolving? I imagine you thought a fair bit about that, because you work on VS team services.

[0:47:45.6] ET: It's insane. I think the biggest change that we've seen over the last couple years is just the wide scale adoption of Git. I saw some Stack Overflow post, the Stack Overflow survey, the most recent one, showed that 90% of professional software developers are using Git. 90% that's insane. It's driving everything.

At Microsoft were trying to move all of our teams into Git, because we've had a number of them throughout the years just spread out in different tools. We call them in-silos, which is the nice polite way to say disorganized. The Windows team use this tool called Source Depot that was – it's a Microsoft tool. It's only used internally. It's not something we've ever shipped or sold. Then we built team foundation version control after that, thinking that that would be the product that would replace Source Depot and that we would also sell on the market.

We did that. We built it, we sold it on the market and nobody within Microsoft uses it. At least, that's not true. I'm sorry. Plenty of people use it, but not the Windows or Office team. We didn't succeed in removing the Source Depot from the equation. Now we had two tools heavily used within Microsoft. Then some teams started using Git. Now we had three. We've got this goal within Microsoft to move everybody onto one engineering system, and that engineering system is visual studio team services and Git. That's been driving a lot of the work that we've done.

We've been trying to figure out how to scale Git to handle our repository like Windows that has 4,000 people working in it every day. It's been really crazy. That's been where a lot of the investment we've been making has gone. It's really cool to see Git being able to scale up to a several hundred gigabyte source tree.

[0:49:46.0] JM: Microsoft acquired Github recently. Obviously, since you've been in Git for a long time, I imagine this must be exciting for you. What are the opportunities between Microsoft and Github that you're excited about?

[0:49:58.4] ET: I'm really excited about it. I don't know if you know, I used to work at Github, so I was one of the people that brought Git into Microsoft. At some point, I had an opportunity to go work at Github and I took it, and that was really cool. Then I had an opportunity to come back to Microsoft as a program manager. I was really excited to take that. I do want to – as you can tell, I've gotten the briefing on public relations and legal stuff.

I just want to make one little point and that is that we haven't acquired Github yet. We've agreed to acquire Github. It's going through regulatory approval now. It's not actually done yet. Until it is done, we're still two separate companies, so I can't really work with them. I can just sit and be excited. I am really so excited, because I think that Github is an amazing product. It's an amazing company with amazing people in it. I'm just so excited to work with them again.

I think that – so if I can be brutally honest, when I look at visual studio team services, for the last several years our design has not been amazing. We're not the best UI, we're not the best UX. Github has great design. It's an incredibly polished, incredibly lovely to use interface. We've been making great strides actually. We just announced that we're finally redoing our UI and our UX and it's available as a preview, but I'm so excited to see the sorts of things that we can do between the two teams to maybe think off each other.

I don't know. I don't know that this is something that we'll do, but Nat Friedman who's my corporate vice president now and once the acquisition is done, he'll be the CEO of Github. He had a Reddit AMA and he teased that we might be able to start using Github logins on some of the Microsoft developer tools. I think that would be amazing, because if I could just log in with Github, that would be great.

There's a couple of just subtle integration things. I think picking their brain as far as design is going to be amazing. I think that what we're doing, what we're launching soon in VSTS is going to be really killer, but I think having that infusion of the development skills between the VSTS team that's been doing all this new work and the Github team that's been doing all those – all their work for years is going to be really cool.

On the whole, I think Github is basically going to stay Github. I think we're just going to – I think that Microsoft as a whole is just going to be able to help them do better at that. The thing I'm most excited about is just them staying excellent.

[0:52:48.1] JM: Yeah, I'm looking forward to seeing whatever additive synergies there are between them. As a developer, I'm sure there's going to be really nice usability things that come out of it. I'm optimistic.

Okay. Well Edward, thanks for coming on Software Engineering Daily. It's been really great talking to you.

[0:53:03.4] ET: Oh, thank you. I love to crack open the Git repository, so I appreciate you letting me do it for a while.

[0:53:10.3] JM: Awesome. I recommend listeners to check out All Things Git, which is your podcast about Git. Actually that interview with Etienne who discovered the vulnerability, that was really great. Talk about the gift of child-like curiosity that struck him to find this vulnerability.

[0:53:29.0] ET: Yeah, it's amazing.

[0:53:29.9] JM: Cool. Okay, Edward. Well, I'll talk to you soon. Thank you.

[0:53:32.4] ET: Thank you.

[END OF INTERVIEW]

[0:53:35.6] JM: At Software Engineering Daily, we're always analyzing data to determine what our listeners care about. We actually have a lot of data, even though we're just a podcast. It always reminds me that organizations with much more engineering going on have an order of magnitude more data, than a podcast like Software Engineering Daily. That's why the job of data scientist is such a good job to get.

Flatiron School is training the next generation of data scientists and helping them land jobs. Flatiron School is an outcomes focused coding bootcamp that offers transformative education in-person and online. Flatiron School's data science program is a 15-week curriculum that mixes software engineering, statistical understanding and the ability to apply both skills in real-life scenarios.

All of the career-changing courses include money-back guarantees. If you don't get a job in six months, Flatiron school will refund your tuition and you can visit their website for details. As a

Software Engineering Daily listener, you can start learning for free at flatironschool.com/sedaily. You can get $500 off your first month of Flatiron School's online data science bootcamp, and you can get started with transforming your career towards data science. Go to flatironschool.com/sedaily and get $500 off your first month of their online data science course,

Thanks to Flatiron School for being a new sponsor of Software Engineering Daily.

[END]