EPISODE 317

[INTRODUCTION]

[0:00:00.4] JM: Thursday, February 23rd was a big day in security news. Details were published about the Cloudbleed bug, which leaked tons of plaintext requests from across the internet into plain view. On the same day, the first collision attack against SHA-1 was demonstrated by researchers at Google, foretelling the demise of SHA-1 as a safe hashing function. What does this mean for the average engineer? What are the implications for regular internet users? How do these services; Cloudflare and SHA-1 — I realize SHA-1 is not technically a service — how do these things work?

Haseeb Qureshi interviews Max Burkhardt, a security researcher at Aribnb in this episode to get to the bottom of what exactly happened, what it means, and how it affects the security of web applications. Thanks to Hasseb for doing this episode on a whim because this was some breaking news that I think will be of great value to the Software Engineering Daily listeners. With that, I hope you enjoy this episode with Haseeb Qureshi and Max Burkhardt.

[SPONSOR MESSAGE]

[0:01:17.4] JM: Indeed Prime flips the typical model of job search and makes it easy to apply to multiple jobs and get multiple offers. Indeed Prime simplifies your job search and helps you land that ideal software engineering position. Candidates get immediate exposure to the best tech companies with just one simple application to Indeed Prime.

Companies on Indeed Prime’s exclusive platform will message candidates with salary and equity upfront. So if you’re an engineer, you just get messaged by these companies and the average software developer gets five employer contacts and an average salary offer of $125,000. If you’re an average software developer on this platform, you will get five contacts and that average salary offer of $125,000.

Indeed Prime is a 100% free for candidates. There are no strings attached, and you get a signing bonus when you’re hired. You get $2,000 to say thanks for using Indeed Prime, but if you are a Software Engineering Daily listener, you can sign up with indeed.com/sedaily, you can go to that URL, and you will get $5,000 instead. If you go to indeed.com/sedaily, it would support Software Engineering Daily and you would be able to be eligible for that $5,000 bonus instead of the normal $2,000 bonus on Indeed Prime.

Thanks to Indeed Prime for being a new sponsor of Software Engineering Daily and for representing a new way to get hired as an engineer and have a little more leverage, a little more optionality, and a little more ease of use.

[INTERVIEW]

[0:03:05.4] HQ: Hi. My guest today is Max Burkhardt. Max is a security engineer at Airbnb. Max, welcome to Software Engineering Daily.

[0:03:12.7] MB: Thank you. Pleased to be here.

[0:03:15.0] HQ: Oh! I'm really glad to have you. So I kind of got you on fairly short notice. I only told you about this a couple of days ago, but I want to talk to you — Right now we're talking basically the weakened after Cloudbleed, which has obviously been a pretty huge event in kinda security issues.

There are actually two major issues that I wanted to discuss today. I want to start by talking with Cloudbleed and I wanted to eventually get to talking about how Google created a SHA-1 collision. I want to dive into some of the technical aspects and kind of what they mean. But before we do that, let's start by scaring people and try to make this relevant so they understand exactly what went wrong and what they should be terrified.

Before we get into any of the details, I want to know why I should be scared that all these stuff happened.

[0:03:56.6] MB: Yeah, it's been a really exciting week for security news, but in a pretty dark way. There're been some big flaws in underlying internet architecture that have been discovered. Cloudbleed is really worrying because there's a bunch of pretty high profile sites, some of which handles sensitive information which could have had sensitive information leaked, and that includes passwords, or session tokens, or private communications. There's a possibility that there's a lot of sensitive information or credentials floating out there and it's unknown who has found it. That's really scary from the Cloudbleed issue.

On the SHA-1 side, a cryptographic hash function that's been used sort of in a bunch of protocols intended to be a secure hashing function has been shown not to be as — The first big collision was found, and this has been theorized about for a while, but now the writing is truly on the wall for that hash function. It is not secure.

[0:04:48.3] HQ: Okay. These are both two events with pretty big implications for how we think about security, and I think a lot of the assumptions that we make about security. Let's start by delving into what exactly happened with Cloudbleed.

For those who don't know, Cloudbleed was a bug found in Cloudflare, particularly their HTML parses which ended up having a bunch of effects of leaking a bunch of sensitive data. In order to understand what exactly happened, I think we need to start by — For those who don't know, can you explain what is Cloudflare. What do they do?

[0:05:17.2] MB: Yeah, Cloudflare is a content delivery network, or CDM. Basically, they proxy traffic for major websites and use a layer of frontend reverse proxy servers to help server content more efficiently. They have DDoS protection services and can really help reduce the load on a backend web service. They handle a lot of web traffic across the internet, and so flaws in that layer can be very wide reaching.

[0:05:44.6] HQ: Why do so many sites use something like Cloudflare. Why can't they just kind of do all these sort of stuff themselves?

[0:05:51.9] MB: Cloudflare is able to leverage a few things. They have technology which is specifically designed to cache stuff effectively. They have serves across the world, so they can usually ensure there's a local edge near to where a user is requesting the data. That's pretty helpful. It's an infrastructure that a company may not want to set up for themselves, so they do provide a good service there. They have also had a history of being very good on DDoS protection front. I think they're pretty popular for that reason as well.

[0:06:18.3] HQ: Okay. Pretty briefly, can you explain how does Cloudflare protect against DDoS attacks. How does it actually do that?

[0:06:23.6] MB: I don't know the internals of their functionality. First of all, they've got a lot of capacity, a lot more than a startup or a company might be able to have themselves. They also have a few ways to detect bot attacks, slow those bot attacks down, try and ensure there are really users that are sending traffic and sort of filter it so that the people who are legitimately using the site get through and then the attack doesn't.

[0:06:45.6] HQ: Got you. Got you. This is like a pretty critical piece of internet infrastructure.

[0:06:49.6] MB: Yeah.

[0:06:50.4] HQ: I don't know if there are a few other competitors, but Cloudflare has a big chunk of the Alexa Top 1000 Websites that they service. A large portion of the internet runs through Cloudflare. What are some of the major sites that you know that use Cloudflare?

[0:07:03.2] MB: Yeah, I was looking through kind of a list of the potential victims and some of the ones that stood out to me — And I'll mention briefly why I think these are important sort of in alphabetical order here. Authy which stores two-factor tokens for many users uses Cloudflare. Coinbase, one of the better Bitcoin exchanges also uses Cloudfare. Medium — Many company blogs are on there. Credential leaks could be really pragmatic in that side. OkCupid, which of course has a lot of private information that you don't necessarily want leaking. Uber, which there's a lot of potential of financial benefit to getting a lot of fake accounts or compromising a lot of accounts — And 1Password.

1Password was very concerning right away to a lot of people, because passwords vaults — If someone's master password has leaked, they could be really devastating. 1Password has come out and said that they have additional mitigations which make it, so this is not a huge issue for them. It's obviously something that's concerning. Just because you have one password doesn't necessarily mean that you need to panic right now.

[0:08:03.4] HQ: Right. Okay. The way that all of these was actually discovered, this was discovered fairly recently by Tavis Ormandy who works for Google's Project Zero. He sort of — Tell me as best as you can the story of how this was discovered, what happened, how long it took for all these information to become public. Can you sort of spin that yarn for me?

[0:08:20.8] MB: Yeah. It was really sort of a by change thing. This story is based off of Tavis' account in the bug report that he filed. Basically, Google does a lot of fuzzing projects. Fuzzing is where you basically throw random data at some sort of process and see if you can get it to crash or reveal any sort of parsing vulnerability. It's a really popular and effective way to find security vulnerabilities.

Google does a lot of these fuzzing, but for effective fuzzing you usually need a corpus of data to modify and tweak in order to use in your fuzzing attack. Tavis was working on looking at websites and making them into good test cases for Google's fuzzing objectives. In this process of taking webpages from around the internet and kinda looking at their code and cleaning them up and making sure there were good test cases, he started seeing really unusual stuff. This usually would manifest just things like blobs of binary data being returned in HTML pages, or things that looked like HTTP headers that were showing up. Pretty quickly, there appeared to be a big issue here.

Before long, Tavis as well as some other Google Project Zero folks were able to determine that this appeared to happen on certain Cloudflare customers when the page that was requested had some tag imbalances. Basically, HTML generally has a form that you have an open tag and a close tag. In some cases, when a tag was not closed properly, there was this weird stuff showing up. That's kind of how it's discovered. It was trying to make a corpus of data to use in a fuzzing project, and suddenly things were looking weird. Very quickly, Tavis was able to get in contact with Cloudflare in order to figure out what was going on.

[0:10:03.3] HQ: What is the standard procedure in that case? If you're a security researcher, you come across something that looks like a pretty grave bug. What are you supposed to do with that case?

[0:10:12.4] MB: Yeah, in this case, Tavis went to Twitter. A lot of the security communities on Twitter — Almost all of it, really. Tavis has a large Twitter following, so he was able to get in contact with senior security person at Cloudflare very, very quickly, which is great that that worked out in that case, but we don't all have the same Twitter following that Tavis does.

If you are trying to report what looks like a big security issue to a company, trying to go to bug bounty programs can be a good place to start. Those are generally monitored by security staff and perhaps who even get a bounty for reporting something. It's also sort of convention that the e-mail address is security@ and then the domain of the company generally would go to some sort of queue for the security team to look at. That's sort of been the convention, at least.

[0:10:54.6] HQ: Yeah. When Tavis discovered this though, he didn't come out and publicly declared that this was going on. He contacted Cloudflare directly and then waited a period of time before any of these was made public. Why did he do that and why is that — Is that generally the practice in security?

[0:11:09.1] MB: Yeah. This is a debate that has raged, really, since the beginning of the security industry, and it's called the responsible disclosure debate. The basic two sides are if you find a bug and you worked with the creator of the software to get it fixed, then you can probably have a patch out before the general public knows about it, and that's good. The other side says if you just release publicly, this applies the most pressure and allows people to immediately start acting to protect themselves and also really ensures that the company, or group that is producing the software really moves to solve this quickly as possible, because everyone knows they're vulnerable.

Google Project Zero has a policy for responsible disclosure. It's, I think, a pretty good one. Basically, they will contact companies or groups privately about issues they found and, basically, there's a series of deadlines by which a company has to either respond or get it fixed or give reasoning for not fixing it yet. If one of those deadlines has failed enough, there will be a public release.

In this case, I think it was about one week in between Cloudflare being contacted and them coming out with a public blog post. Though, I will note that Cloudflare was able to sort of stop the bleeding in most cases in about 47 minutes after hearing about this. They did move very quickly on mitigation.

[0:12:24.9] HQ: Right. When you say stop the bleeding, to be clear, what exactly did they do to stop the bleeding and what bleeding stopped?

[0:12:31.8] MB: Yeah. The issue as it turns out was that Cloudflare does some amount of HTTP or HTML rewriting on some of the content that pass through the servers. They use nginx as the reverse proxy layer, and they have some custom modules that if a customer has requested it, they can do some rewriting. This rewriting is things like changing email addresses to conceal them from the public internet or — There's a variety of things that Cloudflare can do, usually from a security perspective to try and automatically upgrade the security of a site.

This rewriting means that the Cloudflare nginx server has to parse the HTML being returned and change it before sending it on to the user. Cloudflare had recently introduced a new module for doing this rewriting and an interesting rather subtle sort of interaction between this new module and the old one made it so that when certain Cloudflare features were enabled, the rewriting module would read past the end of an internal nginx buffer when returning a response and could potentially produce output that is from some other requests, somewhere in the memory of that nginx process which could include HTTP request from any Cloudflare customer.

[0:13:46.2] HQ: So just to be clear — To be clear. These Cloudflare servers are serving requests from, let's say, 25% of the top 1000 Alexa sites. For some sites, they may enable Cloudflare to do certain HTML rewriting on the fly as certain pages. For those customers who enabled this, because of this bug in this HTML parsing, occasionally, this triggered some buffer overruns and read from other parts of the memory, which could have led to any request from any of the other requests that were being fed to that server. Is that correct?

[0:14:17.4] MB: Yeah, that's right. Just whether or not you have these features enabled doesn't change whether or not your stuff was leaked. It was just whether or not your site could be used as a vector for this.

When you asked me earlier, "What does stop the bleeding mean in this case?" When Cloudflare heard about this, I think they pretty quickly able to guess kind of what the issue was. They had recently enabled this new rewriting module, and so they basically flipped a kill switch and disabled it. That covered most of the likely cases where this could happen. I believe it's about four hours before they basically able to ship a code change that made it so that the vulnerable path was never followed regardless of configuration state. They did move very quickly on that and they were able to guess pretty quickly what the problem was.

That's sort of a critical thing in a lot of these security response stuff, is even before understanding the exact details of the overrun, which were quite technical and subtle in this case, they were able to kinda figure out why it might be happening and stopped it altogether before having to understand the exact details of the vulnerability.

[SPONSOR BREAK]

[0:15:27.1] JM: Your application sits on layers of dynamic infrastructure and supporting services. Datadog brings you visibility into every part of your infrastructure, plus, APM for monitoring your application's performance. Dashboarding, collaboration tools, and alerts let you develop your own workflow for observability and incident response. Datadog integrates seamlessly with all of your apps and systems; from Slack, to Amazon web services, so you can get visibility in minutes.

Go to softwareengineeringdaily.com/datadog to get started with Datadog and get a free t-shirt. With observability, distributed tracing, and customizable visualizations, Datadog is loved and trusted by thousands of enterprises including Salesforce, PagerDuty, and Zendesk.

If you haven't tried Datadog at your company or on your side project, go to softwareengineeringdaily.com/datadog to support Software Engineering Daily and get a free t-shirt. Our deepest thanks to Datadog for being a new sponsor of Software Engineering Daily, it is only with the help of sponsors like you that this show is successful. Thanks again.

[INTERVIEW CONTINUED]

[0:16:50.1] HQ: You explained very briefly that this was caused by a buffer overrun bug in the parse or code. Can you explain very briefly what is a buffer overrun and how it ended up happening these consequences?

[0:16:59.1] MB: Yeah. Buffer overruns are a pretty common issue in code that's written in C, or C++, older C++ that is. Basically, the issue is that in a language that does not automatically manage memory or automatically keep track of pointers for you, usually, when you're reading from memory, you started a particular address and then you read for some length of bytes. If you try and read past the end of that buffer, there's nothing that will actually stop you.

Basically, you're trying to create these control structures such that when you start reading an address, you'll only read as far as that buffer is allocating, and therefore you only get the data that you intended and not anything else that just be lying in memory at the time. However, going back to the top programming errors of all time, these sorts of things tend to be very vulnerable to things like one-off errors.

In this case, the whole thing could have been stopped had there been a greater than or equals comparator instead of an equals comparator on a particular line. Basically, there was a pointer that was pointing to where memory should be read from, where to start with. In certain cases, that pointer could get slightly more than the maximum value, but the comparator was only looking at whether or not that pointer was the maximum value. It didn't consider the possibility that it might be able to skip over.

When the C program started to read, it started to read from past the buffer and then continue as far as it thought was appropriate based on the type of routine. Basically — Suddenly, memory that was not intended for this use is being returned in this response and that really could be anything. It's very hard to predict what sort of things might be in a heap at any given time. The consequences of this bug are very wide reaching and scary. That's kind of the basics of how this occurred.

[0:18:48.2] HQ: Right. Let's say this bug is happening, it's basically randomly dumping memory from some arbitrary place in the heap. Of course, half the time that you're reading arbitrary data, the heap can be completely meaningless. It's not actually going to be some usefully parsable data from something. Some of the time, it is. Why wasn't this data from other request sent by, for example, Uber, or 1Password, or whatever, why was it not encrypted.

[0:19:11.7] MB: Cloudflare actually does their TLS termination at a different layer. The servers that did this HTML parsing and rewriting didn't actually touched TLS at all, and that's both a good thing and a bad thing. The good thing is that there is no chance that a customer's TLS certificate private key could be leaked in this, because these boxes never had it. They were looking at plaintext traffic. The bad thing is that, yes, the traffic was in plaintext at that point and so any requests that were in memory were just readable.

[0:19:41.0] HQ: Why is Cloudflare terminating the SSL, because, presumably, is SSL then again established between Cloudflare and the website serving the content?

[0:19:49.9] MB: Almost 100% of the time. That's how it should be setup. I assume that Cloudflare is doing that. Cloudflare will not send your traffic randomly over the internet unencrypted. Frequently, it is terminated out of the CDM layer for performance reasons, and that's just how a lot of CDMs are run.

[0:20:05.4] HQ: I see. Okay. To be clear, so we talked about how immediately once Cloudflare realized what was going on, they found the root cause, they shut it all off. What then did they have to do given that, now, they have discovered that all of these data is kind of floating out on the internet? What are the implications of that and what did they do for remediation?

[0:20:21.4] MB: Yeah, the really nasty thing here is that once you change this code, then, theoretically, you wouldn't be able to produce this bug anymore, or read any more request. The problem here was places where data that have been served in the past might still be accessible, and that really meant the search engine caches.

Search engines crawl websites. They keep the information that they see there in order to index it and maybe serve cache first in those pages later, et cetera. If a search crawler have hit any of these sites that did have the vulnerable configuration options and therefore were serving at random bits of memory, those might be saved in the cache. This is really scary, because not only is it pretty hard to detect what pages might be vulnerable to this, or what might indicate that this was happening. Also, there's so many search engines and there are so many people who are scraping the web all the time for storage.

Because the bug was discovered inside Google, Google started working on trying to purge this from caches as soon as possible, but it's very difficult to, because you just have HTML page and then there might be some weird data in that, but it's really hard to tell is if that's just the web page intentionally being weird, or if Cloudflare had accidently slipped memory in there.

I think that, basically, they took an approach of trying to count with a bunch of heuristics that seemed to really indicate that there was a leak and then purging things from caches when those heuristics fired. The sort of advantage there is that removing a cached copy of a page from Google's database is not that damaging of an action. It will get regenerated next time the crawler gets there. You aren't really just drawing anything permanently.

I think that Cloudflare has been working with other search engines to try and do the same thing for their cache, cache pages, but there're a lot of them these days and it's difficult to ensure this has been done properly. We also don't know how many people are just scraping the internet for other reasons that might have this sort of thing lying around.

[0:22:13.7] HQ: Right. Okay. This is obviously a big problem, because data is not just cache by search engines, but it's also cached by DNSs, it's cached by browsers, it's cached by routers. Now, from what I understand, these plaintext requests that were being sent over the internet are now cached all over the internet, not just by Google. Is that correct?

[0:22:31.7] MB: Yeah, that's right.

[0:22:32.8] HQ: Okay. Let me understand now. Given that we have a big problem now, we're trying to clean up all these data, try to scrub whatever it is that we can find, even though it's quite hard to find exactly where — Which and what pages were affected and how we can clean out these caches. Give me an idea of what exactly is the attack vector now. Let's say you're an attacker and you know all these data is somewhere out there. What can you actually do? What damage can you actually do with it?

[0:22:54.4] MB: Yeah, it's tough, because you have to actually find data that's useful to you, and it's sort of a saving grace and it's difficult to target this attack. The data that was being returned by Cloudflare is semi-random, it's just kind of whatever is on the heap at the time. You couldn't say, "I want to go find content from OkCupid and figure out exactly how to get that. It was just sort of probably evenly spread based on how much traffic was passing through. You just kinda have to try and collect as much data you can about the requests that were served by Cloudflare during this vulnerable time.

I should note, it's not like Cloudflare has been doing this forever, the earliest time it could have happened would be September 2016 when this started to be introduced and it would have happened at very low rates until, I think, February 13th, at which case some changes happened where it started happening a lot more.

Really, as an attacker, you'd have to try and find some big collection of cached pages and start crawling for that. When I heard about, I first thought of archived.org which I thought I might not be as quick to be able to purchase these things, because they're a volunteer project. They can't just assign a security team to go do it. I don't know how true that is. Hopefully they're on the job of trying to clear up this sensitive information.

[0:24:06.0] HQ: Right. You mentioned that as this bug was out in the wild, it seems like a very, very small percentage of the traffic from these affected sites was actually leaked, because this was a very low frequency bug and, of course, there was a spewing random memory, didn't necessarily spew any given request. Given that there was such a low rate of potential compromise. What should the potentially affected sites do about this? If you're Uber, or if you're OkCupid, what should you do?

[0:24:32.3] MB: Yeah, this is a tricky one. The percentage of requests that were affected is quite low. True. Cloudflare also processes a lot of traffic, and they gave the number of requests affected as a percentage of — Or percentage like that of any particular request, having a problem, and they process a lot of requests. I think the amount of data that's been leaked is actually quite large. This guess is based off of — Had some screenshots redacted that Tavis was posting in that bug thread. It looks like there was a lot of stuff that has gone out there.

I would advocate for sites that have been affected with this to reset sessions, at least. Basically, most websites don't remember your login, establish some sort of session token in your browser, or your phone, and that has some sort of lifetime on the server, expiring those and making for login again, I think would be a reasonable step. It's not the greatest UI. People don't love logging in, but it is a way to show your users that you want to be — That you're watching out for them, that you're trying to help ensure that their stuff isn't going to get compromised even though it's unlikely that it's been compromised, you're kinda watching out for them. I would advocate for session reset, at least.

Password resets are sort of the next step you could go from there. That's a pretty nasty one, because people really don't like changing their passwords. I'm not sure I would always advocate that. Depending on the sort of data you're processing, you might be able to make that call. Maybe if Authy determine that they had evidence that their data was being leaked, or passwords might being leaked, forcing a password reset there could be reasonable.

Again, this is all sort of a risk calculation, so it depends on what you're storing, how many sessions would be affected and sort of what a user would be required to do in order to get back and using your service. It is situational, but I would generally advocate for a session reset at least.

[0:26:13.9] HQ: Right. It's possible, of course, that if somebody was logging in, their password would have been in plaintext and then leaked, if that was one of the requests that was leaked. For most of their request, there is going to be — Basically, you can — If you take this person's session token, you can essentially impersonate them on most of these sites, right?

[0:26:29.2] MB: Mm-hmm.

[0:26:29.9] HQ: That makes sense intuitively. I want to ask you how you think Cloudflare handled the bug. It seems like a pretty big one. It seems like a pretty big scary event. For somebody who's a security engineer that's probably had gone through a lot of these sort of all of the fire alarms going off type of event. How do you think they did?

[0:26:46.2] MB: I think they did pretty well. They were able to triage the initial issue very quickly. They were responsive to Tavis, and they were able to stop new data from being released rather quickly. In Tavis' bug report, he expresses some frustration at the time it takes them to publish a public blog post and sort of the language they use. I see where he's coming from. He wanted it to be a little more scary, "Yes, this is indeed a scary bug." At the same time, I'm guessing that there wasn't an engineer who wrote most of that. These sorts of releases are very curated things by legal and public relations and some engineering, and so there's a lot of things to change in there.

I think they did a good job of responding to the incident. It sounds like really what caused this was them trying to migrate their parsing and rewriting infrastructures, something much more modern and probably — Hopefully, more secure. It's not like they were doing this, this happened because of extreme negligence on their part or anything. I think they handled it pretty well

[0:27:46.7] HQ: Do you think there will be any follow up for them after this bug sort of — The dust sort of settles?

[0:27:51.8] MB: It's really hard to say. Measuring security fallout is tough. It's certainly a bad press, and they were, in many ways, sort of angling for the security focused segment of the CDM market. That's kind of what they were target, and this is going to hurt their efforts a lot in that respect, because the subtleties of this bug may not be apparent to all purchases. They'll just hear that Cloudflare had a Cloudbleed and maybe it changed providers.

[0:28:16.9] HQ: Sure.

[0:28:17.7] MB: I think there might be some follow up there, but my hope is that the openness they showed in responding to it can help reassure people that they're going to be implanting procedures to make this never happens again, or something like that.

[0:28:31.1] HQ: Got you. Before we move on, I want to ask sort of one obviously very important question. If you're somebody who uses any of these services that potentially were affected by Cloudflare, what should you do?

[0:28:42.3] MB: I'd advocate changing your passwords. Just as a quick recap, the ones that I would be most concerned about are off a Coinbase medium, OkCupid, Uber, 1Password. It is a little bit of a pain, but it's just not worth the risk. Obviously, if you are using the same password on those sites as other things, it may be time to do larger rotations. This may be a good time to grab a password manager and start transitioning over to that if you haven't already. I can actually highly recommend 1Password, it's a fantastic password manager product.

[0:29:11.5] HQ: Great. What do you think — Trying to put some kind of a bow on this, what do you think are the lessons for the security community after seeing what happened with Cloudbleed?

[0:29:20.6] MB: Yeah, the response for a lot of people, and I agree with this, is this is just exhibit 5,3222, or whatever, of using C is dangerous. C has been with us a longtime, it's fast, and frequently used in this sort of like made to be real fast CDM type technology, but it's really not safe. The work that Mozilla is doing its Rust seems like it's providing a fantastic alternative to those sorts of system languages that still lets you code safely and eliminate this sort of bug class entirely. You won't hear about buffer overruns in a Java Program and it will be a lot harder to do it in a Rust program too.

I think we just need to continue that migration of taking things that were written in these old languages that aren't that safe and move them over to things that are much more modern. In web security, we talk a lot about trying to use frameworks to help ensure that writing bugs is really hard and covering these bug classes automatically to framework level, and languages can provide that in some cases. Theoretically, your language should be able to just sort of end the concept of buffer overruns entirely.

[0:30:24.6] HQ: Right. Right. That makes sense. One thing that also occurs to me is being one weird consequence of this particular bug, is that I feel like consumers are becoming more and more sensitive and almost more fatigued by all of these events that keep happening where their passwords get leaked here, this site gets attacked, that site gets attacked. I think sites are very sensitive now to the idea of like, "Hey, we're going to reset everybody's password," because people are afraid that users might think that it was their fault, or that they in fact were comprised, when some common piece of infrastructure was actually compromised.

I sense that that that's part of what makes this situation so tricky is that, for example, for Uber, a very, very small portion of all of their traffic was potentially getting leaked, but to reset all the passwords on Uber would make it seem like something really catastrophic had happened that was Uber's fault. Working as you do for a tech company and having the particular incentives, how do you see that sort of game being played, and do you think that's going to change at all?

[0:31:25.8] MB: Yeah, and I think this is — That discussion is happening at all these companies, "What is the risk that we think our users are under? At what point do we make that call of security above all things, or is this risk we can accept and keep user sessions alive?" There're a lot of things you can do to try and negate this sort of risk. First of all, there are many companies which have done really great things in kind of adding analysis to user sessions and user activities to figure out when something might be weird, like if a user has been compromised.

This can be a really powerful tool in the case that you might have some sort of session breach that isn't your fault. You can maybe tweak the parameters on those analysis and say, "Okay. Well, now that we know sessions might be at risk, let's be extra sensitive to sessions that are used in one geographic place one second and then something radically different a few seconds later. That is an indication of something a little suspicious and maybe we should ask that particular user to change their password or confirm something on their phone."

Those sorts of soft security features where it's not necessarily a hard requirement, but a way to detect that some sort of comprise may have occurred, can be really come in handy around these times. When building a product that has user logins, I think that something that people should think about implementing way sooner than many people do is the ability to kind of introspect session that exist and consider when one might be acting strangely.

[0:32:51.5] HQ: Right. Okay. Okay, let's transition from Cloudbleed to talking about the other big piece of news that came out last Thursday, which is Google's SHA-1 collision. Let's start with the absolute basics. I'm going to assume that maybe a listener might not have any understanding at all of what exactly is the big news here. Let's start from the simplest block. What is a cryptographic hash function to begin with?

[0:33:14.7] MB: A cryptograph hash function is a one way function that takes some arbitrary blob of data, some string of bytes and transforms it into a hash which is usually a pretty short string of bites that is unique to that blog, theoretically. The idea is that you cannot take a hashed version of data and turn it back into the original version. There are some other properties that cryptographic hash functions are supposed to hold and they can sort of come down to non-predictability. You shouldn't be able to easily fake hashes if you don't know what the original thing was, or you want to be able to say, "If I hash file A and something else hashes to file A, that's file A."

[0:33:57.3] HQ: Got you. It's like a unique signature that gets created.

[0:34:00.3] MB: That's the idea. Yes.

[0:34:01.4] HQ: Okay.

[0:34:01.5] MB: Now, if you think about it mathematically, if you're taking any string of bytes and reversing it to a fixed string of bytes that is pretty small, then theoretically, there must be more than one thing that can collapse to that shorter string of bytes. It's not infinite compression. The idea is that it's very hard to exploit that or find something that that sort of collision occurs.

[0:34:21.5] HQ: I see. Can you explain just very briefly, what's the different between hashing and encryption?

[0:34:27.6] MB: Yeah. Encryption is a two way function. When you take a file and encrypt it, the theory is that you can then take that encrypted version of the file and go back to the original if you have the key. If you have a hash, no one should be able to go back from the hash to the original file. A hash is really more about identification than protection. A hash will also irreversibly change the data that was in the original file. You wouldn't be able to understand anything about the content of that file based on its hash.

[0:34:55.1] HQ: Right. Okay. In cryptographic hash function, there's not just one. Obviously, there are a large number of hash functions. Why are there so many? How do I chose among cryptographic hash functions. Why doesn't one just clearly win?

[0:35:08.5] MB: There's a lot, and I think a lot of that is just because they get introduced overtime. We've been writing hash functions since before the 90s, and as computing power increases and as cryptographic research increases only to find they're bad, and then it's time to move to a new one. You kinda get this littered trail of broken hash functions in the wake of technological progress.

The tricky thing that makes hash function stay around is that maybe one gets popular for a few years, and so it gets implemented in all the libraries and frameworks and languages, and so it becomes very easy to use. Then, when a weakness or a break is discovered, suddenly it can be hard to get everything switched over.

You've probably heard of the MD5, which is a well-broken hash function. MD5 was introduced, I think, around 1991 and was totally broken around 2004, but it's still used in some places because everything implements it. It was very popular in the early 90s and maybe even early 2000s.

[SPONSOR BREAK]

[0:36:14.1] JM: Are you ready to build a stunning new website? With Wix.com, you can easily create a professional online presence for you and your clients. It's easy. Choose from hundreds of beautiful designer-made templates. Use the drag and drop editor to customize anything and everything. Add your text, images, videos and more.

[INTERVIEW CONTINUED]

[0:37:35.9] HQ: You say MD5 was totally broken. Can you explain what that — What does that mean? What can an attacker do if they know I'm using MD5 to hash something?

[0:37:43.9] MB: Yeah. I'll go quickly through the three qualities of cryptographic hash functions that are important. There's collision resistance, which is where you shouldn't be able to make two messages that hash to the same thing. That's a collision. That's what Google discovered in SHA-1 last week. There's second preimage resistance, which is where if you have a message that you're going to hash, it would be hard to find another message that hashes to the same thing. There's preimage resistance which has given a hash, it would be hard to find a message that computes to that hash.

Not all of these qualities have been broken for MD5, but many of them have, it's generally concerned to be totally unusable if you want it to be secure. SHA-1 just had its first collision, so Google is able to create a message and then create another message that hash to the same thing. The sort of note that's important to note about this is that Google controlled both messages. They didn't take something that's already been hashed with SHA-1, and then compute a new message that hash to the same thing.

[0:38:48.6] HQ: A second preimage attack is one where I have a document A that hashes to a value and I can create a document B and have them both hash to the same value. Google was not able to do that. That's not what they did.

[0:38:59.2] MB: Yeah, and those tend to be scary, because then you can take something like a certificate for a TLS connection that has already been generated and you might be able to create a new one that hash to the same thing. That's not what Google did, but the thing with Cryptographic hash functions is that as soon as the weaknesses start coming, they come fast. More difficult to — Discovered, and more ways to break them are found, and they degrade rather quickly as far as things are concerned. Basically, as soon as weaknesses start to appear — And weaknesses in SHA-1 started to appear on 2004, you want to start migrating off of them.

[0:39:33.3] HQ: SHA-1, I understand, was in the past, a fairly popular hashing function. I guess, probably still is. What are some of the applications that use SHA-1?

[0:39:42.2] MB: SHA-1 — Very popular indeed. Some of the things that have come up that people are concerned about — SHA-1, for a long time, was used as a way to fingerprint TLS certificates. Google has been leading the effort to deprecate SHA-1 on TLS certificates for some time. That's actually going pretty well, which is fantastic. There are actually many people in the industry who thought that Google was kind of overacting to SHA-1s weakening and Google just kinda showed them all that, "No. This is — A break was imminent. It's time to migrate."

TLS certificates used to be hashed for SHA-1. The CAs that abide by sort of the regulations for that industry are not allowed to issue these certs anymore.

[0:40:22.5] HQ: To be clear. Can you make it clear why it matters that they were using SHA-1? What is actually the vector of attack if CAs were still using SHA-1 and SHA-1 were totally broken?

[0:40:32.8] MB: I think this has happened once, I think, with an MD5, MD5 signed cert. Basically, an attack was able to create a certificate which they could present with a private they controlled that appeared to be the legitimate signed one of a website. This enables a full network interception person in the middle attack on TLS, because you can then present this fake certificate that you control the private key for, but because the thing that is signed is the hash of that certificate, it's possible to have this kind of duplication.

I believe this has been officially seemed to have been happened and used in malware once with an MD5 certificate. There is not an evidence that this has been done to a SHA-1 certificate yet if one was able to, and it seems like someone will be able to eventually. This would allow you to strip off the protections of TLS entirely.

[0:41:24.2] HQ: Well, okay. SHA-1 has already been deprecated for certificates.

[0:41:27.8] MB: Deprecated for CA certificates, yeah.

[0:41:29.8] HQ: Okay. What other applications use SHA-1?

[0:41:32.8] MB: Source control programs use a lot of it. Git uses SHA-1 to refer to objects and commits, and then if you're trying to use signed Git, where you have some sort of cryptographic proof of what changes occurred to a source tree, they're signing those hashes. You might be able to create two git repositories with the same hashes, but different contents. This is pretty concerning. SVN totally blows up if there's any sort of collision in its source tree. It appears that if you do put a colliding file like this in SVN, it will just kind of break the repository entirely.

Other popular applications of SHA-1 include things like BitTorrent, and then all sorts of things. SHA-1 is very common. It's been used for a long time. The reason it's called SHA-1 is that it was sort of the results of a standardizations process by, I believe, NIST, and kind of about this generic standard hashing algorithm name. They've since moved on to SHA-3, by the way.

[0:42:29.6] HQ: Okay. I know Git uses SHA-1, and Linus Torvald is the creator of Linux, has publicly claimed that Git is not vulnerable to any attacks based on SHA-1 collisions, or not any attacks, but with some caveats. Can you explain why that's the case?

[0:42:44.3] MB: Yeah. Linus said that Git is not vulnerable, because in addition to keeping track of the hash of objects, it also keeps track of their length. Because of how this collision attack works, it'd be really hard to make two things that both hash to the same thing and have the exact same length. His claim was that Git is not really vulnerable, because you wouldn't really be able to compute this collision in a way that Git would allow.

However, I was not a huge fan of his response, because he says it's not a huge problem, but this is something that I think Git should have sort of predicted. If the whole browser world and the CA world is moving off of SHA-1 on to something stronger, that's an incredibly molasses like industry. It's very hard to get them to move, but they were doing it, and they were doing it for these security reasons.

If the CA forum is kinda moving faster than you, I think that's something to be concerned about. I think Git should have tried to migrate to SHA-2, or SHA-3 a long time ago. Hopefully they take this opportunity to really make moves there now.

[0:43:45.6] HQ: Right. One argument I recall Linus making is that SHA-1 — Really, any hashing function that is using Git is not actually used for security so much as it's used to generate signatures and do D duplication and stuff like that. What do you think of that claim or that argument?

[0:44:01.0] MB: I think that he sort of undermines it himself in his own post. Two quote from it, he says, "So we do take advantage of some of the actual security features of a good cryptographic hash, and so breaking SHA-1 does have real downsides for us," because they're using crypto to sign trees based of SHA-1, and so the hash does end up being part of this chain of trust. For people who are expecting Git's signing features to really give them security, suddenly, there's this sort of like missing component of it, which is the security of the hash function underlying it is not good.

One thing that I should mention which is something that I have learned about in this whole incident, which was really cool, which is this concept of counter-crypt analysis. This is some research that's been going on to basically — Let's say you know a function has weakened, like we know SHA-1 is now. Can you detect if a collision attach has occurred on a particular file? It turns out yes. Google has released a tool where you can give it any sort of file and ask it, "Was this part of a collision attack?" You don't even need both files that were involved in the collision attack. You just need one of them.

What you can do is that you can analyze every file and try and see if there's someone who's trying to do some collision stuff and possibly mess with you. This is actually already been added to Google Drive and Gmail. If you try and mail someone a colliding file, I think — I'm not sure the exact thing here, but I think it will just reject it, because it knows that something is up.

Linus offered that. I think they've already build some patches that could allow you to run this counter-crypt analysis on all objects in the Git source tree, and therefore detect if someone is going to try and do some sort of collision attack against you.

[0:45:36.3] HQ: Okay. Given what Google was able to do, what sort of attack would actually be possible right now given this discovery?

[0:45:43.8] MB: Google created two PDF files that looked totally different, but have that same hash. Their sort of proposed theory was like you send someone a contract and ask them to sign it, and they sign it, and it's like a regular contract. Then, you produce this second contract that looks very bad for them that you claim they signed and then encore you, show that the SHA hash of your file is the same ,and therefore this person is bound to this new contract. I think this is a bit of a contrite example, 'cause I don't think they do a lot of SHA hashing inside courtrooms and also all you have to do is say, "Let's try SHA 256."

That was sort of that proposed one. If I were trying to attack this, it's interesting. I think the Git one is actually a pretty interesting approach, especially now that there're so many companies that are using Git as a source of truth for things like configuration, and security-critical code. Maybe you could create a commit batch, does something evil, but you can kind of slot it in there. I think that could be really interesting.

The BitTorrent one is also an interesting approach. I haven't seen a lot of people discussing this too much, but I think it might be possible to maybe poison; A, torrent, by basically starting to seed bad charts of a file that are basically the result of a SHA collision and basically making people's files break when they download them. I think that would be a pretty interesting attack, but it is expensive.

[0:47:10.0] HQ: Are those attacks only possible with a second preimage? How can you do it if you only have — You can generate two files that hash to the same value, but you can't decide what this hash will do.

[0:47:19.5] MB: Yeah, that's the thing, is you have to kind of make something that looks like it might be legit and then swapping your malicious one later. It only work in those sort of trust scenarios.

[0:47:27.9] HQ: Okay. You'd have to generate trust first for the file and then you could swap out a second file. Do you have arbitrary control over what's in that file, or is there some — Basically, do you have enter in a bunch of random garbage to make it hash to the same file?

[0:47:40.0] MB: There's going to be a lot of random garbage, but it turns out there's a lot of file formats that are quite forgiving of this. They pick PDF because there's a lot of places you can stick binary garbage in a PDF and it won't complain at all. PDF is a good example of that.

There are also a lot of things where — Types of files where you can have a lot of control of predicting the section of bytes without effecting its functionality. An executable binary is a perfect example. It's quite easy to create a zone in a binary that is never called into, or read from, or anything like that. It can just contain bytes. In the normal execution of the program, they don't do anything, but it could be affecting the hash.

[0:48:19.8] HQ: Right. Right. You mentioned earlier that this is a very expensive attack, potentially. Can you give me an idea? What do you mean by an expensive attack and how expensive exactly?

[0:48:28.4] MB: Yeah. Although Google is able to do a collision, it's not like they listed it with long division. They used a huge amount of computing power in order to generate the parameters that would be needed to do this.

I mentioned earlier that because your hashing something, that could be big into something small. There's always going to be a theoretical collision. Theoretically, if you just try any hashing off, you will eventually find the colliding file. The scale at which you'd have to try it enough is so large that it’s not really something that’s worth considering. We’re talking about millions and millions of years of computation in order to find something that just might collide.

Really, this is a way to efficiently create a collision, is what Google found. They used cloud compute resources that I think were from Google compute engine. Some people on the internet did some back of the napkin math on how much this would cost for EC2 using its g2.8xlarge instances. The number they came to was $560,000 if you’re using normal instances. You can get that down to about $110,000 if you’re using spot instances, which are the ones where you can’t use them during peak times and you’re kind of bidding on the price, but you can get them for a lot cheaper.

About $100,000 to create a colliding file. That’s not cheap. It’s not something that I’m just going to run off and do for fun. You could potentially imagine a case where someone who’s very advanced and well-funded might want to do this if they had a good enough reason.

If there is a way to use this against a TLS, I think $100,000 for TLS compromise, a particularly important connection might be valid. Again, this isn’t a second preimage attack. You’d have to do some tricky stuff.

[0:50:05.6] HQ: By tricky stuff, you mean actually getting two useful files that collide to the same value.

[0:50:09.3] MB: That’s right.

[0:50:10.0] HQ: Right. It sounds like from what I’m hearing from you that this is scary, but it doesn’t actually amount to any immediately practical targeted attacks that you can really do.

[0:50:20.0] MB: Yeah. I can’t really think of any. It’s just more a sign to the development and engineering community that any preconceptions they had about SHA-1 being safe are not correct.

As I mentioned, it has been weakened since about 2004. There has been an indication of it being pretty bad, but now there’s just more evidence of that.

[0:50:38.1] HQ: Right. Right. Is it true that every cryptographic hash function is subject to the same sort of decay overtime as computers get more powerful, or they're subject to more analysis, or have there been any cryptographic hash functions that have kind of just remained solid and not weakened overtime?

[0:50:53.4] MB: From what I know, everything has weakened, but I don't think that's necessarily a property that is inherent through them. Everything is sort of weakened overtime. SHA-2, which is sort of like used in a lot of place right now and considered secure. There was some paper that was starting to show some minor weakness in it. That was released around 2008. Don't panic about it, but that sort of thing is starting to show. SHA-3, which was standardized in 2012, has not yet been weakened as far as we know.

Computers will continue to get more powerful, but with the lengths of hashes used by things like SHA-2 and SHA-3, it's still not feasible to just guessing off things to find equilibrium. One thing that I should note is that even if quantum computers start working in the near term, there is not a known quantum algorithm that dramatically affects hash strength. Guessing a hash with a quantum computer is about as hard as it is with a traditional computer.

[0:51:47.0] HQ: Okay. It sounds like this is really the death now for SHA-1.

[0:51:50.6] MB: Definitely.

[0:51:51.0] HQ: How hard is it going to be to do all the work of refactoring code to get off of a weakened hash function and more on to something that's more robust?

[0:51:58.3] MB: It's hopefully not that hard. SHA-2 has been around for quite some time now and it should be pretty well supported by all the languages and frameworks out there. There are going to be changes you have to make. You have to kinda have a transition plans that you can still check data that is provided with SHA-1 and sort of a sunset process there. It shouldn't really be that hard. SHA-2 takes up slightly more space than SHA-1, but not that much more space. There will be changes, but it's generally something that should not be unbelievably hard to do.

[0:52:27.7] HQ: Got you. Got you. Okay, great. To wrap up the entirety of our discussion, which has been really, really fantastic. Thank you so much for sharing all your knowledge with me by the way. What was the response from Airbnb to seeing all these security issues? I know we don't use Cloudflare. Do we use SHA-1 for anything?

[0:52:44.2] MB: Yeah. I'm not going to say that we don't use SHA-1 anywhere, it does hop up from time to time, but it's not something we put a lot of infrastructure on. Everything that has been written recently should be using SHA-2. As a member of the security team, I try and make sure that it happens with this like sort of death knowledge you said. We can really start cracking down internally on the existence of SHA-1, but it's not something that we've pinned a lot of security on. It should be a pretty easy transition for us.

As you said, yeah, we didn't use Cloudflare, so got lucky on that one. Not affected.

[0:53:17.1] HQ: Sure. Sure. Just in terms of how these recent events make you reflect on internet security on the whole, it seems to be kind of scary that there's one provider that could have one bug that potentially compromises a very large portion of the internet. Do you have any thoughts on that? Do you have any insight that you could share with me?

[0:53:36.8] MB: Yeah, it is really scary. You hope at some point that the rate of big compromises or leaks slows down, and that really doesn't appear to have happened. It would be hard to cut the time. At least, for the last 5 or 10 years, we've been seeing increasingly large breaches and security compromises and all those sort of thing as more people start doing research and more people discovered these problems. That's boring, 'cause you're like, "Is there ever going to be a software. Are we ever going to have a secure internet?" It's always going to be hard.

Right now, the advantage is definitely on the attacker's side. Attackers theoretically have infinite time to find a bug, and that bug eventually will be something pretty severe. That is worrying, but overtime we do pick up knowledge that we can use to great effect, and we figure out techniques that really stop a lot of attacks.

One thing that i would like to point to that has been really powerful for security in the last few years is the spread of two-factor authentication, particularly on internal management systems.

Two-factor, or multi-factor is a really strong security guarantee that can help keep a lot of things safe even if there are weaknesses or faults discovery.

In general, you always want to be designing these things so that one thing can fail and you always have more protections. That's something that wasn't really done 10, or 15 years ago. It was sort of that hard shell, soft center style for networks, or security systems, and that's really changing. It's all about internal gating. It's all about monitoring things and knowing that stuff is going to fail, but being able to respond effectively and also contain the problem to something that you can manage.

That's something that we've learned over these years, is that approach that can work and it's the approach that is effective and something you want to keep doing a lot of so that hopefully when these things come down to the future, we can easily identify what it could have affected and contain the breach.

[0:55:33.0] HQ: Right. Max, this has been a fascinating and terrifying discussion. I wanted to thank you for taking the time to chat with me and chat with our listeners.

[0:55:40.9] MB: Yeah, thank you so much for coming up with these questions. It's been an interesting week, and I love discussing with you.

[0:55:45.7] HQ: Awesome.

[END OF INTERVIEW]

[0:55:53.4] JM: A few things before we go. If you like the music on Software Engineering Daily, you might like most recent album called Commodity, which features a lot of the songs that air on this podcast. My artist's name is The Prion on Spotify, Apple Music, and Amazon.

Also, Software Engineering Daily is having our second meet up, March 9th at Galvanize. You can find details on our meet up page. Finally, we are about to post our second listener survey, which is available on softwareengineeringdaily.com. This is your opportunity to have your voice heard and let us know how we can improve. This data is super valuable to us and we look at every single response, so please take the listener survey at softwareengineeringdaily.com.

Thanks for listening to all these announcements. We'll see you next time on Software Engineering Daily.

[END]