EPISODE 368

[INTRODUCTION]

[0:00:00.6] JM: Shopify is a company that helps customers build custom online storefronts. Shopify has built upon the same Ruby on Rails application since the founding of their business 12 years ago starting with Rails 0.5 and moving all the way to Rails 5. Mruby is a lightweight implementation of the Ruby Language. Shopify made the decision to use mruby to allow customers to create custom scripts that are run every time a customer adds items to their cart. However, since mruby was a language implementation that was not widely used, Shopify opted to post a bug bounty to the HackerOne bug bounty platform to find security vulnerabilities in their use of mruby.

What followed was a payout of over $500,000 as report after report flooded in of security vulnerabilities inside mruby itself. There were so many reports that Shopify made the decision to sandbox the mruby execution into separate processes and decreased the bounty awards by 90%.

In this episode, Jeremy Jung interviews Daniel Bovensiepen about mruby and the Shopify bug bounty program. I hope you enjoy the episode.

[SPONSOR MESSAGE]

[0:01:29.1] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today. Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job.

Maybe you want more flexibility, or more money, or remote work. Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly passionate about

technology. Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:59.1] JJ: Daniel Bovensiepen is a core committer on mruby since 2012. He designed and implemented the mruby package manager, test system, documentation generator and interactive mruby shell.

First, I'd like to talk a little bit about mruby. Ruby is a programming language that is most commonly associated with the web framework Ruby on Rails. There's a lot of very large companies these days such as GitHub, Twitter, and Stripe that really built their business on using Ruby as a web programming language. However, mruby targets a very different subset of the development community. What makes mruby a compelling target for software engineers?

[0:03:44.0] DB: Okay. First of all, yes, there are more than Ruby implementation. So when people like GitHub or actually also GitLab, using Ruby they most of the time use something called MRI or JRuby, which is both implementations, one in C and one in Java. They are very performant and very robust, but they both have a huge issue on the size and I do not want to call it clumsiness, but it's such a huge piece of software for both interpreters, which has grown over the years and it's very hard to actually small systems with these huge interpreters. If you have something like MRI, you already start with several megabytes of a web usage and a hard disc space and JRuby is even bigger.

On the other side, mruby gives you the possibility to build extremely small systems. We can actually build an mruby interpreter in sizes of maybe 40, 50 kilobytes. That's an order of magnitude smaller. Furthermore, this is actually possible because the mruby is extremely modular. So we build an interpreter which you can actually customize. Usually if you use a

programming language like Ruby or Java, you expect actually a specific standard library to be always available, and that's of course very helpful, but most of the time you actually do not need all features. Mruby gives you the chance to actually cut away all features you do not need.

For example, it might be that you actually do not want to have file system support or network stack. That seems for most software developers maybe a little bit strange, but if you look at micro-controllers for example, you do not have a file system, so you should not have file system stack inside of your Ruby interpreter. That is just a waste of space, and mruby gives you the possibility to actually do that. It's a flexible, very small Ruby implementation, which enables to build very small applications, which can run by itself but which can also very nicely integrated into other applications.

[0:05:45.2] JJ: You gave an example of installing an mruby application to, say, micro-controller, where you're very resource constrained. Can you give an example of somewhere that you would use mruby on a micro-controller?

[0:06:00.2] DB: Yes, one of the first examples where that actually worked was the so-called — Maybe the audience knows, Arduino platform, these very tiny hobbyist microcontrollers, usually, they have something called an Atmel, so an 8-bit microcontroller. They are pretty small and actually we cannot really port mruby on these ones because they're lacking some specific hardware elements.

In 2012, I think end of 2012, Arduino Foundation actually released something they called Arduino Due, and this is a microcontroller which has actually an arm processor. Actually, something like you have in your phone but much smaller, so called arm cortex m3. These microcontrollers, they are still pretty small, have a very low power consumption but we are actually able to put Ruby into these microcontrollers. We're talking about a microcontroller which has 196 kilobyte of RAM — No. 96 kilobyte of RAM and 196 kilobyte of flash actually, and mruby fits easily in that.

What I personally noticed that was actually building small little 3D printer some time ago and also some other projects based on it. That's more from an hobbyist point of view. As a company specially in Japan, they're actually using this kind of approach to build IoT devices. The last

couple of years, there's of course a lot of bullshitting about IoT, but there are some actually useful application. For example, smart power sockets measuring power, light bulbs which that can be remote controlled. These are applications where we fit very nicely in that. They're actually already in an implementation space on mruby for those.

[0:07:43.4] JJ: That's very interesting, because I've heard some people talk about using, say, JavaScript with microcontrollers, but I think in that context, I believe the JavaScript runtime is probably too large to actually run on the microcontroller and they're probably talking more about sending messages to it rather than actually running on a microcontroller. Does that sound —

[0:08:07.2] DB: That sounds reasonable. I know that for a long period, I think mainly was this Node.js. I'm not really a JavaScript expert, but now of course everybody knows Node.js. and just done that implementation was usually that people build small little microcontrollers and they still write that actually in C, but then have an HTTP API to the cloud, to a web server and they provide actually on the web server some JavaScript back-end where people can actually do the logic. The microcontroller will actually outsource the logic into the cloud. This was in the past.

But as a matter of fact, there are actually JavaScript implementations now which are similar to mruby and there's even something called MicroPython, which is actually a Python implementation, which has a similar approach as mruby. This is a market which grows quite dramatically, especially because most people do not want to develop C and we will probably talk a little bit later about some of the huge disadvantages of C, and microcontrollers are historically always been programmed in C or even an assembler. So it was either inwards more like C++ but it was still horrible. There's a lot huge community who wants actually to move to languages they know from the web environment. They feel more comfortable with it. There's a big move. In the past, I think most of the JavaScript was done server set, but it's also coming to microcontrollers slowly.

[0:09:37.2] JJ: Right. It seems like a lot of people who kind of grew up learning scripting languages like, say, Ruby, or Python or JavaScript. They're actually going to potentially be able to use those in the areas where, like you said, people used to use C. I think that's going to be a really powerful thing for people to get more people involved in working with microcontrollers or working with low resource devices, so I think that's really exciting.

[0:10:05.6] DB: Yes, I agree. Not to be not very fair about microcontrollers is maybe a little bit the direction which I'm very focused on. So that is one of the main reasons why I'm interested in mruby, but a huge use case of mruby is actually not only microcontrollers, but actually using mruby and embedding it in existing applications. There also the size plays a whole role. I think that's at the moment the bigger use case actually for mruby that people take mruby and put it into existing applications. A use case would be for example a game engine. You develop your game usually also on C++ for graphic performant stuff and then the storytelling for example is being done in a scripting language. In the past, often, lua was used for that and mruby also started as one competitor to lua to replace lua instead of this game scripting engine.

[0:11:01.8] JJ: You mentioned that the typical usage of mruby, or maybe the reason that it was created was actually reached only to embed and to, say, C programs, like you said with game engines and embedding into another application. What is different about mruby where it's able to do this where the traditional Ruby implementation is not?

[0:11:25.5] DB: Okay. First and foremost reason is definitely again the size, especially of embedding something into existing application, the RAM. If you, for example, spawn several instances of [inaudible 0**:**11**:**40.3] MRI and inside of an existing C application, which is possible. You can also use MRI to embed into a C application. You will consume a huge amount of memory. The use case for a lot of companies is that they actually want to spawn a lot of instances and give maybe each custom or each playing character in a game and own instance of mruby or Ruby. In this case, it makes a lot difference if you consume per instance 100 or 500 kilobytes of RAM or if you actually consume 10 or 20 megabyte of RAM instead of MRI. That's number one.

Number two is you can limit the functionality of mruby. When you give your users the possibility, for example, to edit code inside of your C applications, so you work maybe a C application and you give the user the possibility to actually script your program in Ruby, you do not want to give your user all the rights to the whole feature set of Ruby. Like for example to execute something on the shell or open and close files or delete a file system.

So a MRI essentially does not offer this functionality that you can remove these features. You could of course manipulate the source code, it's open source, but it's very cumbersome and it will break most of the time the interpreter. Mruby is designed to actually remove every single part individually. So you can create an mruby interpreter, which can only do, for example, class definition, arrays and strings, but it cannot do open files. It cannot do shell access. It cannot do network access. It's a little bit to the security of using Ruby in areas where usually would not like to give the user too much power.

[0:13:30.0] JJ: Okay. If I understand correctly, you can actually — In a way, you're making your own subset of mruby by being able to remove specific functions that you would deem unsafe to allow other users to use, and in that way you can kind of really control people's usage to make sure they only use it in the way that you intend. Is that correct?

[0:13:53.8] DB: Exactly. It's actually probably the best way of explaining it. You can make your own custom Ruby.

[0:14:01.0] JJ: You also mentioned that the RAM usage or regular Ruby is much higher than mruby. I've also noticed that Ruby in particular is challenging to bundle in a standalone fashion, and there's projects like traveling Ruby that include a precompiled Ruby interpreter. I've heard there are many issues with doing this dependency on C extensions and things like that.

[0:14:30.0] DB: Yes. One of the biggest thing is, first of all, that MRI works by default very badly on Windows. It works and we have actually several people who work very hard to make MRI properly run on Windows, but it is always some kind of pain. If you make these — I do not know how to call it, maybe bundled or in one click executable, where you do not have a script, like you usually have it in Ruby. You have text file, which execute [inaudible 0:14:58.6] about just one exe, executable which runs on Windows or which runs on Linux or Mac OS X. Then this is very cumbersome to do.

In mruby, we actually have a fairly complex cross-compiling infrastructure, which was developed sometime by me to actually support all different microcontrollers, but which now actually gives the benefit that we can also compile specific executables for Windows, Linux, Mac OS X from one source. You actually compile an individual executable, which contains everything you need

to run Ruby and not more. So you can actually create an executable, which is just half a megabyte and it contains not only the interpreter but also your source code, which you actually want to execute.

[SPONSOR MESSAGE]

[0:15:54.4] JM: Angular, React, View, Knockout. The forecast calls for a flurry of frameworks making it hard to decide which to use, or maybe you already have a preferred JavaScript framework, but you want to try out a new one. Wijmo and GrapeCity bring you the How to Choose the Best JavaScript Framework For Your Team e-book.

In this free eBook, you'll learn about JavaScript frameworks. You'll learn about software design patterns and you'll learn about the advantage of using frameworks with UI libraries, like Wijmo. You'll also learn about the basic history and the purposes of JavaScript's top frameworks. You'll also learn how to integrate a WIjmo UI control in pure JavaScript and in some of the top JavaScript frameworks that we've already were talked about, like Angular, React, View and Knockout.

Wijmo's spec method allows you to determine which framework is best suited to your project based on your priorities. Whatever those priorities and your selections are, you can learn how to migrate to a new framework in this e-book. Best of all, this e-book is free. You can download your copy today to help choose a framework for your work at softwareengineeringdaily.com/grapecity.

Thanks to GrapeCity for being a new sponsor Software Engineering Daily, and you can check out that e-book at softwareengineeringdaily.com/grapecity.

[INTERVIEW CONTINUED]

[0:17:36.9] JJ: Right. I've noticed that from what you're saying it seems like mruby is very suitable for providing command line applications because I've noticed that, for example, Heroku, their command line toolset, it used to be implemented in Ruby but they mentioned that they had

a lot of problems, like you said, on Windows, it's very challenging to deploy a Ruby binary. On other operating systems, you don't always know what version somebody has of Ruby.

[0:18:06.8] DB: Yes. There's this project, if I recall it right, mruby CLI I think? Which was done by one guy from Heroku and actually another mruby committer. They want actually to create these CLI applications in Ruby but immediately executable no matter where they are running, even if there's no Ruby interpreter available.

The one reason is for portability. Another reason was also for starting speed. MRI itself is actually not that slow to startup. As a matter of fact, JRuby, a lot of people complain always that JVM of course has some boot up time. As a matter fact, there's even a project where mruby is used to speed up the starting time of JRuby. There's one small project that made a small little CLI applications to a bootstrap JRuby which they did in the past in a different way in a much slower way.

[0:19:07.9] JJ: Interesting. It's almost like the different Ruby implementations are supporting one another in some way, which I find pretty fascinating.

[0:19:18.5] DB: Yeah. Definitely, the use case. Mruby will not replace MRI and also not JRuby and usually as a way. Everyone has its niche. Maybe I started a little bit off putting in the beginning that they are a little bit big and clumsy, but there's of course a reason. MRI, for example, is the only Ruby implementation from my perspective who can really do all of Ruby as I understand Ruby. JRuby is the closest competitor, which is maybe 99.99% compatible. There are some niches where it's not — I remember they work quite hard to actually get all the C-bindings running, but there are some cases which they are not 100% compatible. That's most of the time not an issue, but for some people that might be. On the other side, they're providing this high performance JVM which is maybe for some people surprising because most people associate Java with slowness. As a matter of fact, if you have the JVM running for someone, the git is actually not starting and speeds up JRuby quite dramatically. For high performance web applications, I hear that people at one point switch to JRuby to get some additional performance.

[0:20:33.5] JJ: People's perception of Java being slow I think maybe comes from a lot of Java desktop applications that in the past were —

[0:20:43.0] DB: That swing and —

[0:20:44.0] JJ: Yeah, that didn't look native or had a very awkward UI. There's also the Java applets that used to run in the browsers. I think, from a marketing perspective, they had some challenges there, but the JVM is actually — Most likely, it's the most advanced language VM I think that's out there.

[0:21:06.3] DB: Definitely, especially concerning the garbage collection. It's amazing what kind of algorithms they're providing which are actually switchable. Most people actually do not know that. Nothing bad about JVM. It's an amazing piece of software. I just do not like Java, but the JVM itself is an impressive piece of technology.

[0:21:26.8] JJ: Right. I think next what I'd like to move into is to talk a little bit about Shopify. Shopify, they make a product that allows users to create custom storefronts. What initially got me interested in this topic was their use of mruby and the bug bounty that came of that. First off, can you sort of explain, how did Shopify make use of mruby and what made it a good choice for them?

[0:21:57.8] DB: I'm not a Shopify user, but as I understand the Shopify concept is that they have this software as a service concept that you went shopping infrastructure at their place. You can maybe — If you like to sell t-shirts, then you can use their shopping infrastructure and they're handling all the complex stuff, like the merchant and the shopping carts and all these infrastructure.

Sometime ago, they provided a very interesting addition which is — I'm not sure how they call it, Shopify script, the engine or something like that, where they actually provide the possibility that you can add Ruby code to your shop and do some custom things. You cannot only click how your shop looks, but actually you can program how you shop reacts. One example there giving is that you can make some discounts, for example. If you put something in your shopping cart,

then they trigger some Ruby scripts, then you can do some own custom magic, like saying, if you shop two elements, then you get one for free or something like that.

They are actually using mruby, which I also learned quit late as a matter of fact. Yeah, they had at one point a feeling they should maybe have a look about the security of running Ruby code on their sellers, which is a good idea.

[0:23:16.5] JJ: Yeah. I think what happened next was fairly remarkable. They opened up a bug bounty for mruby through HackerOne. I guess one thing I would ask you is that were you surprised when they opened this bug bounty? Has there been any precedent for any corporate backing for mruby?

[0:23:36.7] DB: No, I would not call it corporate, but it was a government funding. The original mruby project started from — Don't let me lie, but I think the ministry of economy and trade industry of Japan. They sponsored the first activities to implement mruby. Next to that, there was no financial addition to mruby there. There were several companies, mainly from Japan, implementing things, but it was not like they're giving money all to find someone implementing something.

For mruby, it was quite new precedent that a company standout and say, "We are willing to put money in mruby if you help us improve it." That was quite — It's quite impressive. There's, of course, examples like that of other projects. For example, Linux kernel has big fashion developed in this way. For mruby, it's the first time.

[0:24:34.3] JJ: That's great, because in a way it almost legitimizes or kind of shows that companies are so interested in mruby that they're willing to invest into it. I think that's a really exciting vent for mruby.

[0:24:49.7] DB: I think what pushed actually mruby — It's my guess. I'm not part of Shopify, so I do not know the internal strategies. What makes for me sense to explain that actually was that Shopify is, to my knowledge, also developed in ruby itself, so I think even in a way its application. Shopify developed quite early Gem, the package manager from ruby is called Ruby Gem, and you can write small Gems to add functionality to MRI.

What they actually did was to build such a package to integrate mruby into MRI. Now it gets a little bit meter, because you can now run a ruby program. Instead of the Ruby program, you can execute Ruby, but the Ruby code you are executing inside of it would be executed by mruby. You have the protection and the limited function set of mruby. That makes a lot of sense because they are developing their whole shop infrastructure in standard Ruby and they're customized, they're giving a subset to Ruby to modify their own shop. The Ruby ecosystem helps itself to grow very neat.

At one point they apparently noticed, "Okay, we are now giving our users the possibility to put code into our system and execute it on our machines." Obviously, they've having a lot to lose. We are talking about a shopping system which handles money, which has to protect their customers and merchants. It was from their perspective probably quite reasonable to say, "Okay, we have to make a security audit." Apparently, the security audit went public from their perspective, because you could also imagined that they make that internally. They probably believe that it will be cheaper if they outsource it.

[0:26:45.5] JJ: In terms of the bug bounty, you mean.

[0:26:47.7] DB: Yes.

[0:26:49.2] JJ: Yeah. That actually takes us to HackerOne. HackerOne is a bug bounty platform that connects businesses with cyber security researchers. The question that I was going to ask is why would a company use HackerOne versus internal security reviews or security consultants?

[0:27:08.9] DB: I, of course, cannot speak for Shopify. I do not know the internal reasons. I can only guest. From the company I'm working for, we're also making security audits for safety relevant stuff and we also ask people. Our company is big enough. We have actually a specific department for that to come to us and look at the code. I believe it should always be outside from the development team who actually has developed, because these people will always look with fresh eyes on the infrastructure and will also try things developer would say, "No. You do not use my application this way."

I think developers testing their own code is a very bad idea, and the same goes for security checks in the end. The security issue is also just the software bug, sometimes with a little bit more press behind it, but in the end it's a software bug and it should be handled in the same way as software bugs are handled, testing it and if you want to find completely new concepts of working a software, it makes a lot of sense to look with new eyes on it.

There's a lot of security researchers actually out there, so I'm not really active in this scene, but in the past, apparently they were mostly paid maybe by finding bugs and selling them to some people who might use them to build mail worm. These days, actually a lot of companies actually provide money, which is great. These people can actually be legal and at the same time provide some beneficial servers. It seems to be a huge community who actually search bugs and gets a payment for that which is great.

[0:28:45.6] JJ: Right. Toby, the founder of Shopify, he made a post on — I believe it was Hacker News, where he said in a way — Shopify, because it's dealing with storefronts, security is really really important because they basically host the livelihoods of hundreds of thousands of other businesses. Part of their interest in HackerOne is actually they wanted to sort of spend as — Of course, it was to address the security vulnerabilities, but it was also to sort of be known as a company that will pay well and make it easy for you to work with them if you're in the white hack community. I think this was a way to, as you said, get them exposed to the much wider security community outside of their community.

[0:29:35.4] DB: It did. They definitely have proven that. Would we want to talk about the amounts of money they have paid out?

[0:29:41.2] JJ: Yeah, absolutely. Do you have the numbers?

[0:29:43.4] DB: Yeah, approximately. I think approximately the last value was around 518,000 U.S. dollar. More than half a million U.S. dollar, which is quite impressive.

[0:29:57.9] JJ: That's a lot of money.

[0:29:59.3] DB: Yeah, of course. It's amazing. If you think that a company is willing to spend that much money in a software project, it seems to be really valuable for them. Of course, there's some bad mouthing. Maybe we should also mention that short line, because that block entry I have written about I got actually a lot of bad feedback about that, but I think it's important to mention that they started — It was a bug bounty prices per bug from $1,000 for a very tiny bug which maybe crashes to VM but doesn't really put any security to the system. Up to $20,000 per bug, which actually comprises the security.

This is what I started was in November last year, 2016. I remember one month later they already have paid out several hundred thousands of U.S. dollar. Of course, people really founded a lot of bugs. Of course, for me, as someone who has worked on the software, a little bit tragical, but I'm happy that the bugs are found. It's on the other side also, a little bit depressive that there are so many bugs. After this one month stay, I said, "Okay, we are not actually sandboxing the whole system." What they did is that they put mruby into a sandbox and used some of these security and mechanisms and capability features of the Linux kernel to protect mruby a little bit more and capture most of the security bug and they also limited the amount of byte code and instructions you can actually execute. I remember also they limited runtime.

They did a lot of things actually to compensate for the amount of bugs, but even though that they then would use actually, I think, from 20,000 down to 10% for each bug. Instead of 20,000, you got them something like $2,000, which is still impressive to say the least for a security bug. They keep the HackerOne still open and there's actually one thing and nobody should forget even if a lot of complaints were going around that a lot of the security people do not believe in sandboxing. What I said, "Okay, this is not the right way of solving a problem." As a matter of fact, they keep the security bug bounty open. They're still paying out until today, so we still get new bug reports in that. We are still fixing them. They're still paying money out which has to be definitely appreciative from them. A very impressive contribution from them.

[0:32:30.4] JJ: Yeah, I think that's great because it sort of shows that they are committed to using mruby and they want to make sure that it continues to be an active platform and a secure platform. I think that's almost like a vote of confidence from them that they intend to continue using it.

[0:32:47.6] DB: One of the big advantages I hope now which comes out of that is of course that after Shopify has now invested so much and made mruby somehow more secure. I do not want to say that it's now secured. There's, for sure, still bugs which would be found and there will never be no bugs available in mruby. There will always be bugs, but it is certainly a secured data than it was half a year ago. I hope with this investment that also a lot of other companies are looking at it and saying, "Okay, I could use this embedded language, or this embedded language, but actually mruby has now invested so much and not time in actually concentrating off security fixing, that it's probably the best choice to actually use mruby because so many people have already looked at it."

[0:33:30.7] JJ: Right. Like you were sort of mentioning before, there were some people who were maybe upset when they saw the number of security issues that were reported. In a way, if security issues are not reported, it doesn't mean that they're not there.

[0:33:46.9] DB: Of course, not. It's just a personal thing. Every day when I see that there's another report. No. I'm very glad that they reported. You're absolutely right. We cannot close our eyes and just hope that nobody use our tooling and then no bugs are found. The bugs are there no matter what. It's great that someone found it and reported them.

[0:34:12.8] JJ: Right. I think this is not something that's limited to just mruby. I think this is something, and we'll maybe talk more a little bit about this later. In anytime there's a language that does not have a lot of corporate backing or a lot of users, let's say new languages like, say, Elixir or Crystal, things like that, that these are things that maybe people don't always think about that because people are not actively searching for security issues, there may be security issues in these other younger languages as well. I think this is good to build awareness of things like that.

[0:34:51.4] DB: Yes.

[0:34:53.0] JJ: Related to that, before the bug bounty was put into place, how often were mruby security issues reported by the community?

[0:35:01.3] DB: That's a tricky question because as a matter of fact I do not recall a single one who came and said, "There's a security bug. I can compromise something." They're aware of course. A lot of bugs were reported over the last five years and a lot of them were also some starting points for possible compromization or wide escalation of code injection.

As a matter of fact, nobody reported them as security bugs. People reported it as heap overflows, buffer overruns, but in the security context. Actively, nobody started to really look at security issues until Shopify started this activity. Now, if I recall right, so I also don't have the latest numbers in my head, but the last time I counted we had 200 reported security bugs. I think at the moment, there are two or three left open which were just recently reported, but all these 200 bugs were actually resolved and integrated.

[0:36:07.5] JJ: I think that kind of brings us to our next topic of what are sort of the most common vulnerabilities that are found as a result of the bug bounty and how are these vulnerabilities triggered?

[0:36:18.6] DB: Okay, maybe two things. I actually had to look because I was also very interested how much of my code was affected. I made a statistic and I think it was a little bit more a 90%. I actually made a count on my block of all security bugs were actually found in C code. The most often errors is malware management. That's the number one. That leads to different things like things like invalid point, heap overflows, null point, de references, memory corruptions, double freeze. All these standard bugs which happens because in a C requests the developer to handle the memory management.

I think less than 5% of the bugs were actually found in real Ruby code. That's quite impressive if you looked at approximately a little — Not 50, not too much away from 50% of all mruby code is actually written in Ruby itself. You have to not completely, but close to 1-1 ration from C to Ruby, but most errors where — So majority of the errors were found in C codes. A very clear indication for me that we're better optimized, get rid of the C code and try to rewrite as much as possible of mruby in Ruby itself. That's now less our target since day one.

[SPONSOR MESSAGE]

[INTERVIEW CONTINUED]

[0:39:22.0] JJ: This code that you would propose rewriting in Ruby, is this more things that are built into the standard library or —

[0:39:30.8] DB: Yes.

[0:39:31.8] JJ: Because I imagine — Okay. Because things like the virtual — Go ahead.

[0:39:35.6] DB: Sorry. One thing we of course have to remember as much as I like Ruby, but using Ruby is always defeating one purpose of mruby and it's increasing the interpreter size or the virtual machine size. The more Ruby code we write, the bigger the Ruby implementation will be. Of course, we can never be as efficient as pure C code. The [inaudible 0:39:58.5], the virtual machine, the garbage collector, all these things will stay as C code for these very reasons because we want to keep the code smaller.

There's actually a project called Rubino which targets to write a whole interpreter in Ruby itself, but this is not our target because it will increase the size. What we are mostly focusing of writing as many parts of the string library and even parts of the core library in Ruby. You actually can see that quite easily in the source code these days already that a lot of parts of the string libraries to arrays, to hash, most of them are actually implemented in Ruby.

[0:40:40.7] JJ: The vulnerability is that occur you're mentioning, it was things like null pointer, or heap use aperture freeze. These are all sort of related to manual memory management.

[0:40:52.9] DB: Yeah. First of all, yes. There were, of course, so I do not want to downplay some of the other bugs, there were. Of course, we logic ours. In Ruby, we cannot have this kind of bugs. There's no manual memory management, but still we can have bugs. Specifically, you can trick the interpreter by having very deep stacks that's — General issue was languages and there were several bugs reported off crashing Ruby on bringing to a whole state by making some construction which run forever. Some other examples where that the data structure, which is actually written in C could be shifted from some very smart allocations of Ruby objects.

I do not want to go into detail, but the data structure in Ruby is very simple. If you have a data structure and you have the data itself and then at one point, identification what kind of data is this. You have maybe string information, hello world, and then at one point in this data structure, there's an identification what class this data type is. There was actually one very impressive bug I have to say where someone noticed that — I think it came from a string, he started to build a string then he de-allocated it and allocated an array again and was that he could actually change the type of the data structure which can lead to very very scary effects.

These were actually bugs accessible via Ruby but the core issue was not memory management but logic errors. These bugs are also available. I do not want to blame C only. There were also just bugs where we did not consider also bugs like these by one errors that you have some loops that you have miscounted yourself into one element too long or one element too little.

[0:42:53.7] JJ: Right. The example you're giving, just so that I understand correctly, it would be as though you created a new object that you told Ruby was a string and you were somehow able to convince Ruby that the object at that memory location wasn't really a string. It was

actually some other data type, and so that would kind of create non-deterministic behavior within the runtime.

[0:43:17.9] DB: It will always be deterministic, but not in the way that it is expected.

[0:43:22.3] JJ: Right. Unexpected behavior.

[0:43:25.6] DB: You're absolutely right. This is a very funny one and I was really impressed when I saw it the first time. As I said, I'm not a security researcher. I know very little about this field and I was long time scratching my head, "How did they find something like that?" Again, I do not want to blame. I'm very happy that they found it, but I could not understand how they were able to find that because my understanding in the past was always as a security researcher you would may be look at source code and look at standard problems like memory allocations and then maybe figure edge cases in the source code and then poke this use case a little bit more.

This kind of bug, like I explained just now, and they had several of them, you could not find in that way. It was really mind-blowing to me. Till I noticed that they actually used these days very advanced fuzzing mechanisms. Most of these guys actually have some very smart fuzzing tooling chain and I actually played around with some of them. It's quite impressive what you can do these days automated. Actually, parts of the security community, even automated their own job. It's quite impressive.

[0:44:35.0] JJ: Can you elaborate a little bit on what you mean by fuzzing and how these security researchers are discovering these bugs without, I guess, actually reading the source code themselves.

[0:44:47.9] DB: Okay. Maybe I'll start shortly from the beginning. My understanding, and this is how we — There's some of our teams in the past was that we really use source code. You could on a long time use things like [inaudible 0:45:00.3] and identify some hotspots where it's worse looking at. In the end you would look actually at the source code, follow the structure and make some assumptions where the developer didn't really saw to the end and then try to poke that from the outside.

I think there are some strategies where people actually look permanently on new comments. They just subscribe to the Git repository and then they always look what kind of new comments are coming into the source code and then they have a short check to develop something stupid obviously here. That's a very obvious thing.

As a matter of fact, since — Yes, they went their way. Fuzzing, I think, is a term maybe known for several people, so the idea that you start and program and just feed it stuff, random stuff in the beginning. I tried that a long time ago and I was not really ever convinced at this concept. Makes lots of sense, it can find some very stupid bugs but it cannot really find complex things. As a matter of fact, I know some toolings. The famous one is this American Fuzzy Lop, which is fuzzer which you give some inputs, so you define some example Ruby code and then this fuzzer is feeding the software you want to test. One important thing is that you actually compile the source code you want to test with GCC add on from this fuzzer, and then this fuzzer is able to actually track the code pass. It can feed code and then it can look, "Okay. Where is my code pass going into the software?" Then it modifies the input and then it looks, "Can I come further? Can I reach new code pass with this modification? If no, then I try something new. If yes, then I start to mutate on this change," and that you can actually slowly find your way to corner cases, edge cases and software which are very rarely tested. This AFL, if you check actually their website, they have quite impressive weight of bugs they found. Essentially, you have to give them some reasonable inputs and then you would just give them a lot of CPU power.

Actually, there are some fancy optimization. Essentially, what you try to optimize is to execute. In our case, the mruby interpret is often as possible. I actually create for my own tests some small optimized interpreter, and with that I can — Something like 60,000, I remember, codes to the interpreter done per second and then it will — I think that one actually over Dragon Boat Festival found. It writes something like seven billion different inputs and found out of that approximately 2,000 ways of actually crashing the mruby interpreter.

[0:48:04.3] JJ: Oh, wow!

[0:48:05.6] DB: Out of that, then you have this number and I actually tried it because I wanted to see the differences between the mruby 1.2 version and one we're having now. The interesting

thing is the crashers, at the beginning, you are shocked immediately and say, “There’s still so much after all these improvement,” but then you actually look into it and then there are a lot of false positives. You actually find some ways of tweaking the mruby interpreter still today if you just take the standard configuration. There are just some things you cannot avoid, like endless loops if you do not limit interpreter. You can always, of course, create a denial of service. You have to take that always in consideration. Shopify did that, of course.

It’s a very impressive tooling, so I would suggest everybody who builds some kind of software which is actually as simple by any kind of unknown users to give it a try. That’s not a complex tooling. In the beginning, I was a little bit hesitant because we do not know anything about security research, but you can started with it. Of course, we’re getting a lot of value and you should know what you are doing. If you know the source code that helps already tremendously and having then this tooling, additionally, you can at least get some first information.

[0:49:19.1] JJ: Yeah, I think that’s fascinating. I think if I understand it correctly, you basically give — What was the name of the application you’re using?

[0:49:29.4] DB: It’s called AFL, American Fuzzy Lop. I think if you search for AFL, you usually find that AFL and fuzzing.

[0:49:37.9] JJ: If you use this AFL, you basically give it a sample application that takes inputs, and AFL can actually just send in as many types of inputs as it can just to sort of see where it can get inside your implementation.

[0:49:52.0] DB: Yes.

[0:49:52.8] JJ: Okay.

[0:49:53.9] DB: There’s this feeding that can be more and more complex. A lot of the security researchers, I looked at some of their GitHub repositories, they actually just used some random source code. I optimized the source code. I feed it with and then you can also actually teach AFL a little bit about keywords, so things like — I think an easy example would be SQL. They actually use AFL to check SQL Lite, and there you could ,for example, define specific keywords

which are used in SQL, like select, like insert, and then AFL can with that arrange the combinations in a little bit smarter way so it can essentially guess a little bit smarter.

Say, you can do of course with Ruby evens that Ruby syntaxes, I think, by far one of the complex from all programming languages. I actually use parser and J to extract all the keywords. With that, you can actually improve the performance of AFL to find better tooling. Even just starting randomly with that will already bring you quite far.

[0:50:59.4] JJ: I guess the dynamic language is like, say, Ruby, and Python, and JavaScript just because there are so many more valid inputs. There are so many more opportunities for AFL to find issues. Is that correct?

[0:51:15.4] DB: Yes. Even though that I would always claim that the complexity of Ruby specifically as extreme. A lot of people maybe know that Ruby's syntax can be quite awkward, but most of them have no idea how complex Ruby's syntax is actually. If you look at parser and J of Ruby, it's something like 6,000 lines of code. As a matter of fact, all Ruby implementations more or less based their interpreter, so they are parser on the same parser J which was written for MRI because nobody dares to re-implement just parser. There's this one thing what makes Ruby specifically very interesting target because there are so many possibilities and a lot of them might never be have thought by the developers.

[0:52:08.8] JJ: This parser, is it written in C or what language is it written?

[0:52:12.8] DB: Yes. It depends a little bit. Our one is of course based on Python together with C. You essentially generate the parser based on a pre-defined language. That's the normal way. that's not very specific to Ruby. Most programming languages have that. They are some other parser generators like ANTLR or Yacht. You have this structure, in our case, this famous parser J file and that contains, in our case, some C code and some declarations, how the SQL is used inside of the parsing, in the grammar. Then it actually generates C code to parse.

For example, JRuby interpreter would not generate C code, but JavaCode for example. Rubino infrastructure wouldn't generate actually Ruby code.

[0:53:09.0] JJ: I guess in terms of — Ruby was really built behind this concept of developer happiness and kind of a — In a way, it feels a little bit like writing pseudo code. That's the reason why there's a lot of developers who enjoy using Ruby. It sounds like in the efforts to make a language that's friendly for people to use, it ends up being almost a bit of a nightmare for the language interpreters. Is that fair to say?

[0:53:35.9] DB: That's absolutely fair to say. The target of Ruby is always to let the computer work harder than the developer. Some things people would say, "Why do you do that?" There's specific things like length and size. Quite often, you have for the same logic two different ways, or even more than two different ways of saying it. As an example is this famous if statement. In Ruby, you have also the unless statement. Instead of saying if not, you say unless, which the Python developers usually would put their hands on their heads and, "Why? It doesn't make sense. I just used not." It's a very different way of looking at problems. It fits personally in my mind very nice because I think I do not believe in one-size-fits-all, one right way. This is definitely how Ruby proceeds. Ruby gives you the freedom to write your code in a very polished way. There's even actually a so-called golf mode in MRI. Few people actually know that. Code golfing is sort of the art of writing very complex code which makes amazing things and Ruby has actually a golf mode to actually make even more complex code, which you cannot read anymore.

Actually, I think one of the MRI Ruby committer one of the strongest golf player, because he's using Ruby. On the other side, you can make very elaborate code like wave a very good example. You have to use — No, I'm sorry I'm not a wave developer but these have database table accessible to learn why and you essentially nearly write English language and you create these domain specific languages specifically for something.

Also one important — One famous thing is this aspect or this behavior driven development where you actually write also enclosed to English code to test and specify recode. Everybody independent if you can read Ruby or not, you will understand what is written because it is plain English. There's quite a flexibility, and I like that Ruby gives me both ways so I can start to hack away those prototype. Most of the timeI sort it away but I have then the possibility to iterate or what is called improve it something which is actually maintainable. That's always this claim that Ruby is not maintainable, but I hope that people slowly understand those projects like, GitHub,

GitLab, Taobao base camp. There are so many use cases where they have a huge code base and you can manage it. You just have to stop believing that the compiler will save you. you have to protect yourself with unit testing and behavior testing.

[0:56:30.2] JJ: I think that Ruby's syntax in the way that language has been designed, it's really lended itself to teaching people who are new to software development, because we see a lot of programming boot camps teaching Ruby. We see a lot of outreach programs that are trying to show people who are new to programming, get them some exposure to it. Often times, these programs use Ruby to demonstrate that, and I think it's a combination of the Ruby community itself.

Also, like you said, the way that Ruby allows you to think in different ways with something as trivial as, say, being able to say unless rather than just if. I think it's little touches like that that make it unique.

[0:57:19.6] DB: It's a stupid example. That's really one of the very small ones. The more fancier things is that you can actually look at your code in a procedural way or in a functional way. Lambda constructs a very powerful in Ruby. If you might as build to sync functional, you can actually come quite far. If unless is just something most people always mention, but it's just the tip of the iceberg.

[0:57:44.9] JJ: Right. I guess you would say in some ways, it's supports multiple sort of ways of looking at a problem, whether it's object oriented or functional or something else.

[0:57:57.6] DB: Even looking at one problem at the same time with different aspects. If I have data structures, of course I might prefer an object oriented way, but if I look at and I go at my problem in a bigger context, I want to solve that functional. If I want to write some glue code for some applications, which is something I do all the time, I might actually prefer procedural way because I just want to have a one way, a onetime use of a script and then sort it afterwards away. I do not want to be actually handling all these boilerplate which is usually necessary.

[0:58:34.0] JJ: It allows you to work at sort of whatever level of complexity that you decide to use for the given project.

Yeah, we've covered a lot, everything from mruby, to the bug bounty program, and Shopify, to the vulnerabilities that were found within mruby and kind of how those relate to other languages as well. Is there anything else that we missed that you'd like to mention?

[0:59:03.1] DB: Probably a lot, but I think we covered a lot. I think maybe some, not closing comments, but maybe as an outlook. The last version of Ruby was 1.2 already more than a year ago. We actually had the plan to release now the 1.3 version of mruby which has a lot of features actually included in this March. As a matter of fact, the Shopify bug bounty actually crossed our targets. They're a little bit because we said if they are now investing all their time, we are not going to release a new version which has obvious security bugs and then we immediately to put a new dot release. We postponed it for March to April and said, "Okay, when we have fixed all the Shopify bugs, then we will release."

Yeah, as I said, they were coming and coming and at the moment I think they are still I think three or four bugs open, our plan would be to very soon actually release 1.3 release of mruby which then has a new baseline and then we are starting on adding these keyword arguments to mruby. We'll work further on this ability. Of course, security bugs. If someone finds security bugs, I hope they're reporting them to us.

[1:00:19.8] JJ: Great. We'll all look forward to the next release of mruby, 1.3. I just want to thank you, Daniel, for taking the time to speak with me today and we look forward to seeing what you come up with next.

[1:00:33.5] DB: Okay. It was a pleasure. I hope it was interesting.

[1:00:35.5] JJ: Definitely. I think there's a lot of interesting material that our listeners are going to enjoy. Thanks again for coming on Software Engineering Daily.

[1:00:44.0] DB: Okay. Bye-bye.

[END OF INTERVIEW]

[1:00:48.3] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily.

Thanks again to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver this content to the listeners on a regular basis.

[END]