Interviewer
What have been your motivations for learning Rust and choosing to use it?

Participant
I've been using Rust for a while, like I think this is like year six or seven. We all started Python and like a tiny bit of C that didn't really count. I guess I was curious and I just picked it up out of curiosity and then I continue using it because it kind of like the type safety, low level access, all that kind of stuff kind of lines up with my way of thinking and like the way I like to write code. So type safety, immutability, all that sort of stuff.

7:1 I guess I...
Low-Level Access
Memory Safety

Interviewer
How did you find those things that you value and see in Python prior to transitioning to Rust?

Participant
Ah, I mean, like I did say for like a semester at uni, but that like the semester, the uni course itself wasn't very well taught. So I didn't really like kind of forget, forgot all of that it was a but Python, when I was writing Python, I was doing like just very hacky scripts or like this, some of them had started getting bigger. But yeah, it was definitely once I picked up Rust that I learned or like a the language kind of pushes you towards doing writing better code. And you know, Python's obviously dynamically typed. So having a compiler is very nice. Yeah, I'm like, even today, I don't write C, but I read C and have to understand like how C works. And I still prefer Rust a lot more.

Interviewer
Gotcha. Okay. So what are you using unsafe Rust for?

Participant
We created a virtual machine for WebAssembly. We maintain library bindings for different libraries like this Python, there's, we've got a C API, we've got like a JavaScript API, you know, all these sorts of things. So I've doubled in those a bit. So that's FFI, that sort of thing.

Interviewer
Gotcha.

Participant
That FFI is the primary thing I do with unsafe Rust with. There's other ones about like, sometimes you have to implement like unsafe interfaces or like you might have to roll your own primitives or I don't know, something, I don't know, like say you have to do use futures and you have to write your own future and you have to use unsafe because of pinning or whatever or, you know, like, what else? Like I've written a couple of like allocators or sometimes you might need it for like a data structure you

7:2 There's oth...
Data Structur...and Op
7:3...
Custom Allocator

want to write, but I normally don't, you know, I don't bother writing data structures from scratch. I'll just use the library.

7:3... Preference for Safety

Interviewer
Gotcha. Okay. So I guess let's start with the, I have a bunch of FFI related questions. So let's start with those second two use cases that you mentioned to summarize it was implementing certain primitive types you mentioned. And then I'm blanking ...

Participant
Data structures, I probably wouldn't count that one.

Interviewer
Oh, okay. Gotcha.

Participant
Yep. I mean, yeah, like both of those use cases are kind of just ad hoc, like say I need to do something, say I, I don't know, say I need to write an allocator that wraps an allocator because I want to do statistics on it or like debugging purposes or I might need to implement an unsafe trait or it's just, yeah, it's kind of like it's not one project, it's just because some of the things that I do at work require like some parts are unsafe. And in personal projects, some things might be unsafe, like, but that's most of my personal projects is the unsafe and my personal projects is to do with FFI.

Interviewer
Gotcha, okay.

Participant
Yeah. Yeah, I wouldn't be able to pinpoint like one or two discrete areas.

Interviewer
Oh, okay. Gotcha. I guess more in terms of the documentation and the interfaces you're exposing to users then you mentioned earlier, like writing an unsafe interface or implementing a trait. Are you generally, like when you're writing unsafe code, is it always encapsulated in some sort of safe interface or are there cases where you have to expose some unsafe API to the user?

15

Participant
I try and hide it just for, like you try it just because that way it's that way I can control everything. I don't really want to have to trust because once you open it to the user, then you have to start worrying about a lot more stuff like, I don't know, but you have to rely on them to actually uphold your promise, like, yes, sure, you call it an unsafe trait. So in theory, the burden of proof is on them, but also like if they implement something

7:3 I try and hid... Engaging with...st Com
Exposing a Safe API

improperly, then they're going to create a ticket because they'll be like, oh, I use your library and it's a fault. And then, you know, so even though you say, yes, it's your fault, they still, like, you can't shift all of the burden onto the user. So it's easier to encapsulate everything.

7:3 I try a...
Engaging with...st Com
Exposing a Safe API

Interviewer
Gotcha. Okay. Yeah.

Participant
And a lot of the time you'll only expose, like, really low-level things as unsafe.

7:3...
Exposing an U...e API t

Interviewer
Hmm. Okay, So the things that you do expose as unsafe, are they documented in a way so that, like, if I'm calling some API or implementing some unsafe trait in one of your libraries, will I have a notion of like, okay, here are the preconditions I need to meet, here are the post-conditions that will happen, so I can, like, use that to guide what I'm doing?

Participant
It depends on the intended audience. So because a project can have multiple crates, because it has multiple components and you split things up. So there's kind of a distinction between, like, items that will be used by users directly and items that will be used by other crates within the project. So it's kind of like, you have to choose, you have to understand what your audience is. So if it's, if my audience is other people working on my projects, like my coworkers, you can kind of be slack there and just, like, leave it undocumented, because we know the code works. And you can talk to someone about it. And generally, you know, like, oh, you know, [name]'s the owner of that area of code. So if I have problems, I talk to them. So when the audience is, like, internal, then you can be lazy. When it's public, you'll want to have, you know, proper, you know, like, how you have the unsafe section of your documentation. And you'll say, like, you know, these are the assumptions, like, or whatever. It's used, assumptions are usually around lifetime of objects, like if you're using, or like, say I've got a pointer, how long does that pointer go into live for? Like is it okay for the implementation to stash away a copy of that pointer, or does it have to, you know, it's invalid by the end of the function call, or, you know, just random things like that. But yeah, usually, yeah, it's dependent on your audience.

7:34 So there's kind of a...
Audience-Dep...ocume
7:4 So when the audience is,...
Audience-Dep...ocu
Documented...ract

Interviewer
Okay, gotcha. Gotcha. So then in the screening survey, you, there are three types that you mentioned using in unsafe contexts that was Box, UnsafeCell, and then MaybeUninit. Can you start with Box, I guess, like, what are the situations where you would use a Box to contain some sort of, like, raw pointer type, or maybe, like, unwrap it to access the underlying allocation?

43 **Participant**
Oh, FFI. I need to allocate something. I do Box new, Box into raw, pass the pointer across, and then in, you know, you'll have like a destructor function in Box from raw, and drop it, easy and stuff like that. MaybeUninit is also like an FFI thing, like, you know, how C likes to use out pointers, so like, you'll do a MaybeUninit on the stack, and then you'll pass it into the function, it'll do the initialization, and then you do assume afterwards.

7:5 O... Allocation in Rust
Box<T>

44

45 **Interviewer**
Okay, gotcha. And then what about unsafe cell?

46

47 **Participant**
So there are a couple of, I haven't had to use it recently, but I know there have been times where I've had, like, FFI use cases where, say I've got a struct, like, say I'm trying to wrap a C API, and it might be mutating my, mutating, like, a struct or whatever. But because at C, you want all your methods to take a, not immutable, a shared reference to self, because pretty much all C functions make assumptions that it's shared pointers. So you want those semantics, but even though technically, yes, you're mutating, you don't want that unique access part of, you know, mutable reference. So that's where UnsafeCell can come in. It can let you preserve the C semantics of, you know, shared access, and, you know, still do mutations, all that kind of stuff.

7:7 But because at C,... Different FFI Memory
UnsafeCell<T>

48

49 **Interviewer**
Gotcha. Okay. And then, so I guess let's focus on the FFI stuff. Could you describe the types of projects that you're working on where you need the FFI?

50

51 **Participant**
Yeah. Like I mentioned…we have language bindings. So we have our, we have the Rust core [tool] crate, and then there's like a [tool] C API. And so obviously that's exposing a C API to external callers. And it's a fairly in-depth C API, because it has to feel that the WebAssembly spec for the C APIs.

52

53 **Interviewer**
Yeah, sure.

54

55 **Participant**
Then on the consuming side, you've got, you know, you have, you might have like a [tool] Python or whatever, and that needs to, you know, so that might be doing it from the other direction. There's like a, I'm calling into [tool], or something I do a lot in personal projects is, I might like wrap a C library. So that would mean I have to write a, you know, you write a sys crate to make sure things build and you get the symbols. Then you write like a more idiomatic wrapper on top of that. So you create like a idiomatic interface in Rust, and then I call in, call the functions from the sys crate.

7:8... Rust Bindings for a C

Interviewer
Gotcha. Gotcha. So when you're writing these bindings, let me back up just to summarize. So with [tool], it's like you have the central Rust library that has a C API, and then you're writing bindings to that C API and other languages so that you can, like it's sort of, it's unidirectional FFI use from other languages that are calling this Rust library, correct?

Participant
It's, you end up have, when you, it depends on the language, like some languages have like PyO3, which gives you nicer ways to do things, like, you know, wraps a lot of the unsafe, but you have like other languages where you have to write like going from Rust to the C API, and then from the other language down to the calling into the C API. So you have to, so it kind of sounds like it's unidirectional, but you have to do both sides of the bindings.

7:9 It's, you end up...
Directionality

Interviewer
Gotcha. Yeah.

Participant
No, yeah, if that makes sense.

Interviewer
Yeah. No, that totally makes sense. So with bindings, you mentioned you, so in addition to PyO3, you use CXX, and cbindgen, could you describe the usability of those tools and how you feel that they're helpful in your workflow?

Participant
I've dabbled with CXX. I like the way it does things.

...
CXX

Interviewer
Okay, gotcha. Gotcha.

Participant
Yeah. It works pretty well.

Interviewer
Gotcha.

Participant
Bindgen, yeah, it's kind of like the overall, you know, the tool works quite well. There are some, like, you do run into some build issues on like, you've got the decision, do I commit, do I generate my binding statically, and then publish those, or do I generate them in a build script? And

7:10 Bind...
Generation VS Validati

generating them in a build script is best because then you get bindings for exactly your architecture. So like, say the library, you're calling into changes API, depending on Windows or Mac or Linux or whatever, then, you know, you use, if you run Bindgen inside the build script, then you can, you know, you get, you get the correct symbols, the correct, the correct bindings for the architecture. But it's a real pain because you now have to add Bindgen as a dependency. And Bindgen pulls in LLVM, and LLVM is a gigantic pain as a dependency because you can have multiple incompatible versions of LLVM. I've had lots of situations where, like, you have two dependencies that pull in, say they both pull in Bindgen, Bindgen pulls in LLVM, LLVM pulls in llvm-sys, you can't have two incompatible versions of llvm-sys, otherwise you, cargo says, says no. And then you end up in like an impossible situation where you have to like, either fork an upstream project or make a pull request so that it's using the exact same version like for all dependencies, if that makes sense.

7:10 Bindgen, y...
Generation VS Validati
7:11 But it's a real pain because...
bindgen
Limitation of Bindir

Interviewer
Yeah. Yeah. No, that sounds quite challenging,

Participant
Especially as the number of copies of bindgen increases, depending on how many foreign bindings you use, right?

Interviewer
Gotcha.

Participant
And also, all of this adds to build times. So you, you really want it in your build script, but it's, it's a really painful, like a heavy dependency and it, yeah, and it's, it can be painful to, to use to manage versions. I can't remember what the cross compilation story is like, but knowing, like, native libraries, it's probably, actually no, it wouldn't matter. Yeah. I don't know. Cross compilation can sometimes be a pain, especially when native libraries are involved.

7:37 And also, all o...
bindgen
Limitation of Binding T

Interviewer
Okay. Yeah. And then, so do you invite these bindings manually then, or is it always sort of focused into your build system?

Participant
I would, no, I would not write it manually. Like, I wouldn't write, I wouldn't write it manually for anything other than like something trivial and hacky. So just like trying something out. And then once it's like, you know, bigger than like a couple of lines, like a couple of functions, you're not going to want to do that manually. So it's just better to, you know, I've got a header file, I just throw, throw the header file at bindgen, bindgen gives me some stuff. And then I can wrap that stuff.

7:12 I would, no, I...
Generation VS Validati

88

89 **Interviewer**
Gotcha. And then have there ever been times when the output from one of these tools then was something that needed to change? I guess what I'm thinking in particular is there are, like the Rust compiler has certain lints for types and foreign boundaries, where it'll give you a warning if you're using a type that, where the ABI isn't like specified. So people will ignore those lints and say this, like disable the improper type checking if there's a particular type that they, they know that they like, even if the spec isn't there, they, they know they need to use it. Like are there situations like that where you might need to, to get in and change things yourself for, or is it usually the output is fine?

90

91 **Participant**
No, I wouldn't. I don't think I've ever had to. And even then I would, I would be very, like if it was either myself or a co-worker that was having to go in and modify bindgen generated stuff, I would be asking questions because that's really bad from long-term maintainability perspective, because then header file updates, you regenerate your bindings, you've lost your changes. You can kind of automate that stuff by like, you can write tests that generate the code and then make sure they're up to date. And like it's called like a self modifying test, which is what Rust analyzed it. So there are techniques to, to keep your changes, but that's like, I wouldn't go down that path. That's, that's a bad idea. So I, if I had to have a situation like that, I would, I don't know, make a ticket to bindgen or, I don't know, I'd just figure it out.

7:13 No, I would... Generation VS Validati

7:38 S... Engaging with...st Com

92

93 **Interviewer**
That makes sense. Yeah. And then when you're working with these languages, so you're building bindings in both sides, how do you reconcile the differences between the memory model of Rust, which has these certain assumptions about uniqueness and mutability and then the memory model of like C and Python, which is quite different and much more relaxed?

94

95 **Participant**
It's really not an issue.

96

97 **Interviewer**
Oh, okay.

98

99 **Participant**
You just, I don't know, it's just like encapsulation, make sure you're, you write your APIs and like, it's as long as you keep everything high level, which you'll want to do anyway. So if, if it's very high level, it's a very easy to encapsulate those problems go away. Even for low level stuff, like if it's, if you've got a very chatty API that jumps between Rust and C a lot, things get a bit more difficult there. But usually you would hide that in like a single module that's, you know, just fall to the brim of unsafe. And then I

7:14 You j... Simple FFI

7:15 Even... Local, Minimal Uns

guess you'd kind of like, I don't know, I haven't seen it being too much of an issue. Like you've got MaybeUninit, you've got UnsafeCell, you know, you can, when in doubt, just throw another level of indirection at the problem and usually it goes away. And if you're, yeah, if you're getting to situations where you have to know the, like where the memory models are different enough that things are like, if you're getting to that point where it's a bit concerning, then you're, you probably need a really good reason to continue. Like, and, you know, you should be stepping back and being like, okay, why do I need to get down? Why do I have to worry about this? From, an engineering designs perspective.

7:16 L...
MaybeUninit<T>
Tacit Knowledge
UnsafeCell<T>

Interviewer
Okay. So what do you say that the types that you use in these bindings or, or that you pass into these bindings or reflecting these decisions? Like you're, you're not necessarily borrowing something and then passing it across the, the FFI?

Participant
What do you mean by borrowing something in passing?

Interviewer
Oh, I was in like passing in a rust reference to something that takes a C pointer.

Participant
I'm generally not a fan of, so you know how a function could take like a, say there's a C function that takes a pointer. I'm not normally a fan of rewriting it so that the rust version of it takes in like an Option reference because then C is thinking it's a pointer and then Rust is seeing it's an Option reference. And yes, those to a 100% ABI compatible. And I, like, you know, it's safe and correct to do that, that little transformation. But I feel like it's almost better to leave it as a pointer. Because then it's like everyone knows that this is like we're dealing with, you know, like we're dealing with pointers here. It's not just your run of the mill Option reference to like a whatever. So it's kind of like as a communication thing. I'm not normally a fan of like the people using the implicit cast from like reference to T to a pointer to T. That's right.

7:18 I'm generally not a f...
Option<T>
Simple FFI

Interviewer
Okay. So generally you'd want any, like it, you would not be a fan of cases where you're doing a lot of type conversion, even if it's along ABI compatible lines, because we feel that if you just have a more direct route from a rust raw pointer to a C route, like a C pointer, then that would potentially be more safe than if you're ...

Participant
I would say I want the function that the C library exposes should have the same like I should be able to look at the header file in C and then look at the the extern C block in rust and see that these two signatures are

7:19 I...
Simple FFI

identical.

Simple FFI

Interviewer
Gotcha. I see.

Participant
So yeah, so if that kind of makes sense, and that way you can like straight away I can see and I know that like I'm comfortable with doing that thing. Those like playing around with that sort of unsafe. I know that my coworkers are I can trust them. So I would prefer for them to just to not have the training wheels on and to do things to just be aware.

Interviewer
Yeah, that's right. Yeah, make sense. Yeah, no, that totally makes sense. Yeah. So then let me back up. There was one area of unsafe we didn't chat about to which is implementing allocators. Could you chat a bit more about when you need to do that?

Participant
You usually don't have to go too deep into it. It's more like you might want to reuse one or you might need to. There was one situation where in a previous company, I had created a web assembly program. And we were finding that it was running out of memory. And so and we're like this is really weird because you look at the code, it allocates a vector and then drops the vector. And then that's it. So like there was no memory leaks in that way. And so one of the things I did was I made an allocator which just wraps dlmalloc, which is the one we're using. I wrapped we alloc at the time. And then I was printing the address of the pointers and the size. And I was seeing that you know, the way we alloc allocates memory causes fragmentation, which then causes out of memory issues. So that was one time I had to implement an allocator. Other times is for like statistics. So I just want to be able to tell people, you know, you allocated, you know, the maximum memory you allocated was 10 megabytes or whatever. Usually, I wouldn't implement an allocator from scratch, just because it's kind of no point when there are loads of really good implementations. Like and, you know, especially those sorts of things I'm normally doing it for like work. So I'm not going to bother. Yeah, like I've read an allocator for like personal projects or for fun, but not for work.

7:20 And then t...
Custom Allocator

7:21 O...
Custom Allocator

Interviewer
Yeah. Gotcha. Gotcha. And then I guess from the perspective of the allocator, you mentioned like certain statistics that you'd like to check. Are there any other properties that you feel you would be able to have like checks or tracked if you had like a really smart allocator where you're doing a lot of profiling or or verification of memory? Like are there features that you feel like you could you could add to the stats tracking?

111

123 **Participant**
Stats tracking and like printing out thing allocations, like whenever an

Printf-Style Debuggin

allocation is done, that's for that's kind of like the printf style debugging. So it's quick and dirty. You know, it's only like 30 lines. Most of it can be generated by Rust analyzer anyway. If I needed to go any further, like if I had to troubleshoot like a fragmentation, like properly troubleshoot fragmentation or, you know, I wanted to find out where a particular thing was being allocated. I might start off with printf style debugging, like the, you know, the quick and dirty wrapper. But I would reach out for a proper tool, like I'd reach out for Valgrind, which can do, you know, can check for leaks and stuff. Or I might reach out for, I think it's that DHAT is a memory profiler. So I would reach out to, you know, tools that can do it for me. I'm not going to, yeah, not going to write on any more code than I need to. And plus they can do it better than I can.

7:22 S... Printf-Style Debugging

Interviewer
Gotcha. So there's like this boundary where you cross from the area of like, yeah, this is printf debugging to, okay, wait, I need something more sophisticated, but in those situations, you just prefer to use the standard tool that's already there for you.

Participant
Yeah. So pragmatic choice, like you're making a decision of how much energy do I want to put into this and what's the best use of my time. So like printf style debugging is easy and you can knock it out like that. But then if I wanted to like track my, okay, you know, is it that all my allocations of like, I don't know, a certain size are weird, you know, because allocators do bucketing or whatever. Like if I wanted to go into that detail, I wouldn't write something myself, I'd reuse an existing solution.

Interviewer
Gotcha. Okay. Yeah, that makes sense. So then I guess a broad question. Describe a bug that you face that involves unsafe rust in some way. So this could be like an FFI issue or just any case that you mentioned previously where unsafe was involved?

Participant
So my first job, I was working at a company that builds CNC machines for cutting foam. They build the controller for it. But they, and they also build a CADCAM system, a CADCAM package. So, you know, the idea is that you draw, you know, you'll draw, this is the shape I want to cut out of this big foam, chunk of foam, and then you tell the software, okay, go cut it. And then the software like, has a UI for talking to the controller and seeing like, okay, this is the like temperatures and whatever. And anyway, so that CADCAM package was written in Delphi, which I don't know if you've heard of it. It's very unpopular. It's not very popular. Like, and so as an experiment, I was working on, like, I wanted to get a particular new feature in. And I wanted to write it in Rust, because Rust was a lot nicer to use than Delphi. So I created a Rust library, which did this thing. It was for to do with pathfinding and nesting. So the idea is that, say, I've got a gigantic rectangle, which is my block of foam. How can I place my, like,

7:2... Manufacturing

say, I'm cutting up pillows. How do I place the pillows to optimally use the material? And I'd written a library for that in Rust. And Delphi was calling the Rust code. And I can't, it was, the root problem was that I had improperly defined the type on the Delphi side. So the equivalent of my extern C block, I'd written the function incorrectly. So, you know, Rust, the function might take like a, like an integer, an integer and a pointer. And on the Delphi side, I'd only said this function takes an integer and a pointer. So that means, you know, Delphi thinks there's two parameters. Rust thinks there's three parameters. So the Delphi, when it calls into the Rust, it only provides two parameters on, I think it would, you know, however it's called in, I think it's C, whatever, you know, anyway. So, you know, Delphi had passed in a certain number of parameters. Rust was expecting more. So the third parameter was uninitialized. And I was wondering why my code kept on segfaulting. And that pointer happened to contain like a, I think it contained like a logger or something. So it wasn't until, like, further down in my code that it started segfaulting because it would, you know, you wouldn't use your logger immediately. You'd use it like further down. And then with dereference pointer, this pointer, God knows where it came from. It's uninitialized memory or, you know, it's probably some other random place on the stack. And yeah. So that's one bug I found is that there was a mismatch between the two languages. And the function, the Rust function was expecting something that wasn't being provided. Yeah. And then the Rust function was doing, you know, it's just doing God knows what. And that required a lot of, I think I ended up stepping through it in a debugger, which is the only time I've ever had to use a debugger in Rust for FFI.

7:24 And I c...

FFI & Binding Bug

Mismatched B...r Decla

Interviewer
Gotcha. I was going to say that sounds like a, especially the fact that you don't use the logger until later, it sounds like it'd be very tricky to figure out.

Participant
Yeah. Yeah, it was a pain.

Interviewer
131 Especially since the solution was like a missing parameter too, that just seems like it'd be so frustrating.

138

Participant
139 Yeah. Yeah. Yeah. And it was just a case of the two were out of sync. And because I was writing the bindings on the Delphi side by hand, because bindgen can output, cbindgen can output a header file, which is fine, except Delphi doesn't use her C header files. So, so like there is a automated solution, but it only works for C, or things that can consume C, like Python C FFI. Yeah. Yeah. So that that was the root cause was that, you know, I changed the thing, things went, things, everything was being done manually. So, yeah, had a bad time.

7:25 And becau...

Generation VS Validati

140

Interviewer
Yeah. Gotcha. Yeah. Are there any other unsafe bugs that you can think of that are like a different characteristic?

Participant
I answered questions on the Rust user forums quite a bit. And I remember one of the things was they were trying the person was trying to copy a technique from that they'd normally use and see, which is where you take like a, I have a pointer to a bunch of bytes, and then I cast that to a book, like a pointer to a struct of whatever. So type punning style things.

Interviewer
Gotcha.

Participant
Yeah. And I remember, so the particular user was having issues with their code. Like, why can't I do this in Rust? And I think they ended up like trying to do type punning with when like a, one of the fields was a pointer itself, like a reference to a string or something like that. So, and you can imagine, I've just got a bunch of bytes, and then I've, I've cast it to a pointer to whatever. And if those bytes came from a file, sure, my struct will contain a field, you know, which contains a pointer, but that pointer points to garbage now, because, you know, it was written by a different process.

7:26 I've jus...
Incorrect Type Casting
Temporal Memory Bug

Interviewer
Gotcha.

Participant
So that, I guess that, yeah, that was a bug that another user had seen, and it was like incorrect use of type punning is probably how I put it. And I guess the best solution is don't do that. There are better ways. There are, you know, like you just use serde for serializing data, or you might write a, like a binary format, or something with a custom derived, like, I think there's a binread crate that will do custom derives for you.

Interviewer
Yeah. Yeah, that makes sense. So, and in this case, like, to clarify, was it the problem being the format of the pointer was incorrect, or just the fact pointer was being written to elsewhere?

Participant
So they, they had a, they'd read a file, and you get a Vec of U8. Yeah. Then they, you know, you can do dot as pointer, and that will give you a pointer to the start. They then cast that to a struct, you know, point it to a struct of whatever. And then one of the struct fields was, happened to be itself a pointer. So when they'd saved it, you can imagine, like, I save a

file is like a save as binary. And then I just want to read it back out. And what they've done is when it's the process that had saved it to a file, the data that was saved had appointed to stuff in its own memory, memory space, the process then goes away, or you're calling it, you're trying to read the file from a different process. And then the pointer that your file contains you know, points it. Yeah, essentially. So and yeah, like that's, and yeah, if that makes sense.

Interviewer
Yeah, totally. Yeah. Yeah. So they were, were they just assuming that the pointer was still usable?

Participant
I think they just didn't think about it.

Interviewer
Okay, gotcha.

Participant
Because of course, you know, it's unsafe. And like, it's perfectly fine to cast a pointer to bytes to a pointer to whatever, like Rust is perfectly happy to do that sort of type punning. But there's no, there's no, like, lints or anything, because that's, that's on you. And I mean, the best solution is the bytemuck crate is, because that will enforce it properly, like, it'll enforce that all of your fields are like binary compatible and so on. So there was a, yeah, if they hadn't tried to use that technique from C, which is just char pointer casting, if they tried to use something like the bytemuck crate, or which would do the same operation correctly, like in a type safe way. Or if they'd just use serde and like serialize to bin code or JSON or whatever, like they wouldn't have had the issue.

Interviewer
Yeah. Yeah. Actually, yeah, that makes sense. So then I guess with lints, the one bug finding tool you selected in the survey was Clippy. Have there been any particular lints that have been very helpful for you from that? Or has that just been like a generally useful tool?

Participant
Yeah, it's just generally useful. Yeah, like there are, I think there are some ones around unsafe. Generally, I don't know, I'm sure that they're very helpful. I don't know, I don't really run into the lints to, I don't trigger the lints too often. But I think it's just because like I write lints, the code likes, sorry, I write code that the lints like.

7:40 Gen... Clippy Doesn't Find Bu

157

171 **Interviewer**
Yeah, gotcha. And then have there been like obstacles that have prevented you from using other development tools with your code bases? Or has it just not been necessary?

Participant
So the type system is, oh, you know how like with Rust, if it compiles, it works? Like in the, in a previous job, where I was using C sharp, I'd literally use a debugger like two or three times a day. And I'd be constantly stepping through my code in C sharp, Visual Studio. And Rust, because I've probably had to use a debugger like in maybe once a year for Rust code. You don't know the tools. Like it's almost like you don't get enough practice with them. So like, you know, you always forget how GDB works. The integration with, so I think Rust Analyzer and VS Code let you use LLDB. And that's been nice a couple of times. Like to use a couple of times, but it can also be a bit, like sometimes it wasn't quite working for me. Or like, like to start stepping through the code, it would require, I think, VS Code, which is like this launch, like launch.js, whatever. So it's not, it wasn't plug and play. Like it wasn't just I click a button and then I jump into my code. It required a bit of setup beforehand.

7:27 So the typ...
If it compiles, it works

Interviewer
Gotcha. Gotcha.

Participant
Yeah. And because like not everyone has that setup, because like you barely ever have to use debuggers in Rust, like that can be annoying, because then you've got to look up the documentation or copy from someone else.

7:41 Yeah...
Engaging with...st Com

Interviewer
Yeah. Gotcha. Yeah. Okay. So then sort of final question here. Are there any problems that you've faced in writing unsafe that your development tools can't help you with? Or that you feel like there could be a better solution for?

Participant
I guess one I haven't mentioned is Miri. So Miri is awesome. There's one pain point. It's that it's not terribly useful when it comes to FFI, because it obviously can't interpret like C code. Like any other insurmountable things. They're not necessarily language things, but sometimes like when I've seen other people write code, they tend to try and like abstract things out. Like sometimes you try and you try and add abstraction layers so that you don't even have to write the unsafe code at the at the interface. So there is a crate called, I think it's like safer FFI, and which is really helpful because it gives you like derived macros. So I can write the safe code, the code safely, and then it will like wrap it in unsafe into argument transformation.

7:28 I gu...
Miri doesn't s...ort this

7:42 So t...
safer-ffi

Interviewer
Hmm, gotcha.

Participant

Tools like that are really helpful. But sometimes you can get to a point where you try and abstract things so much that now if there's any bug in the tool, like say the tool generates a slightly wrong thing, it's a gigantic pain to track down. And sometimes you, like when you're writing code, and you know, you're like, Oh, I do the same sort of logic three or four different times, like, you know, like if pointer is null return null, or, you know, something sort of things like that. People get tempted to abstract things out. And then the FFI code or the unsafe code itself, by introducing abstractions, becomes harder to read and reason about because now I don't just have all of my code in one function, like say it's 40 lines. Now I have to, it's now 10 lines or 15 lines, but I have to like manually inline all the functions that it calls to understand what's going on, if that makes sense. So yeah, yeah. It's like sometimes people can be too smart for their own good and that hurts. Like I've been, I've done that to myself a couple of times where you try and add abstractions, then it becomes impossible, like it becomes a pain to debug.

7:30 But sometimes you can get to... Difficulty with Generat

Interviewer

Gotcha. Gotcha. Yeah. No, that makes sense. Yeah. Any, any other problems that you can think of? With writing unsafe?

Participant

I can't think of any, I guess one problem would be like, there's a lack of, not writing unsafe, but the culture around unsafe is that everyone's like unsafe is the boogeyman. And it's like, you know, it's like people have either like the laissez-faire like, oh, I write, I've been writing C for 20 years, so I know what I'm doing. And then they proceed to do something stupid. Or you almost in reaction to that and to rust like desire for safe interfaces is that people are like, they try and shy away from it. And like they, any, any code that uses unsafe is like, like heavily, like unnecessarily, like criticized. So it's like this code, or yeah, I don't know how to, it's like it's unsafe can be demonized in a way. If that makes, I'm not sure if I'm quite make...

7:29 I can't think of any, I gu... Stigma Against Unsafe

Interviewer

No, no, you're, you're making perfect sense. Yeah, I'm totally following. Yeah. And I've, I think I've seen that that is as part of like the set of perspectives that people have on unsafe in prior interviews and engaging with the, with the community, like and on how that's, that's a one, one perception that people have. Yeah.

Participant

Yeah, you might remember.

Interviewer

Sorry, you go.

Participant

You might have heard about Actix web. So there's a Actix web a couple of years ago. So the original author of Actix web kind of did a couple of dodgy things with unsafe. And, and then people saw like people made tickets about it. And like the whole thing turned into a bit of a witch hunt. And so the original author of Actix web basically abandoned the whole community. So [name] has a really good blog post about it. I'll see if I can find it.

7:43 So there's...

Stigma Against Unsafe

Interviewer
Yeah, if you could get the link to that. This is sounding vaguely familiar, but I don't think I've seen the blog post. So yeah, that would be helpful.

Participant
Yeah. So here, I'll just paste it in chat. That's right. That was, I think that was a couple of months. Like there were, there were like two or three different waves of witch hunts against this project. And I think that was towards the end when the maintainer had just said, just given up. That's, that is a very interesting example of like what I mean when I say witch hunts. Yeah, there's, there's lots of stuff you can look into there.

Interviewer
Yeah. Yeah, I'll, I'll start there. Are there any other other examples that you can think of of this besides the Actix web situation?

Participant
That one's the easiest point to because [name] put it put together like a really nice summary of it. And he's, I think he was pretty objective. Like I remember reading it and feeling like that was a fair summary of events.

Interviewer
Gotcha, gotcha.

Participant
You see it on the user forums every now and then, like the kind of knee jack reaction. Or like someone's wanting to learn this stuff. So of course, you know, you're wanting to, you're trying to learn how, how do I write this unsafe interface? Or how do I like, I'm writing some code and I need to do like unsafe impulse send or unsafe impulse sync. And then, you know, because I'm, I'm trying to learn as part of that process. And of course, you get a right code to, to learn. And then people kind of jump on that and like, they can be very hard on them, if that makes sense.

7:31 You see it on the...

Stigma Against Unsafe

Interviewer
Gotcha.

Participant
Yeah. So there's not like one example I can point to, but like you see it

every now and then, it's usually like nowhere near as bad as Actix web, but it is a bit of a culture thing, just because sometimes people take the whole safety thing a bit too far. And don't realize that, you know, sometimes people are learning, sometimes it's for pragmatic reasons. You know, sometimes it's just because I write unsafe because, you know, I didn't know that was a better way or something like that. So there's a bit of a culture thing going on there.

7:32 but it is a...

Preference for Safety

Stigma Against Unsafe