Would Rust secure cURL?

16th Jan 2021

Rewriting programs in Rust has become a bit of a meme and one program that has been discussed a lot is cURL.

The first time people suggested rewriting cURL in Rust, the main author Daniel Stenberg wrote an article about why cURL is written in C and wouldn't be rewriten in Rust. It includes this section:

> **C is not the primary reason for our past vulnerabilities**
> There. The simple fact is that most of our past vulnerabilities happened because of logical mistakes in the code. Logical mistakes that aren't really language bound and they would not be fixed simply by changing language.
> Of course that leaves a share of problems that could've been avoided if we used another language. Buffer overflows, double frees and out of boundary reads etc, but the bulk of our security problems has not happened due to curl being written in C.

Three years later news arrived that some Rust code would be used in cURL, though only as an optional HTTP backend - it isn't a full rewrite. This news reignited the discussion (Reddit, Hacker News). It seems that some people are still under the impression that it is possible to write memory-safe C, and based on the above quote that *cURL is memory safe C!*

Is this true? Are the majority of cURL's security vulnerabilities logic mistakes?

It's easy to find out. The cURL authors have a great list of (known) cURL security vulnerabilities. If you skim it it becomes immediately obvious that no, cURL has plenty of memory safety bugs. Since there's a nice list with great descriptions of each bug it seems like a nice opportunity to measure how many bugs Rust would have prevented.

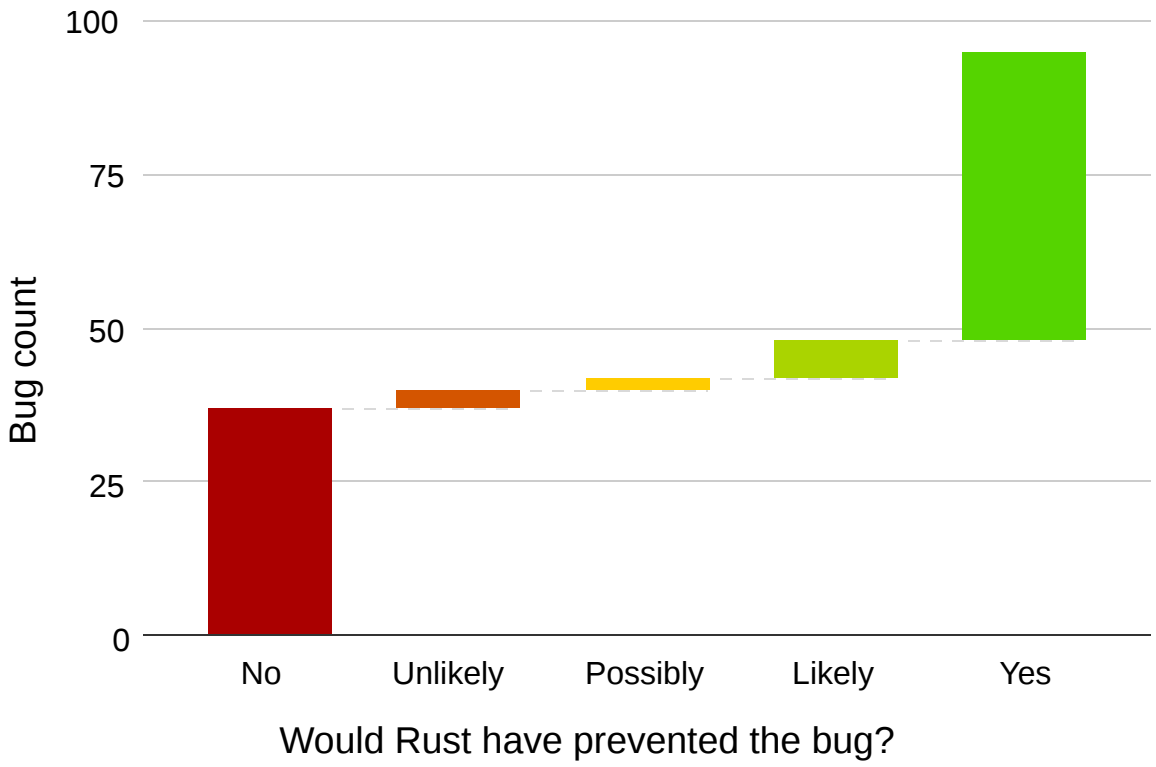
I think "how many historical bugs would this have prevented" is a really good way of judging a programming language or feature. For example this great study shows that using Typescript would have prevented approximately 15% of all bugs that you find in typical Javascript code. It's hard to argue against static types with evidence like that.

I went through the entire list of cURL security issues, and categorised all of the bugs, together with whether or not I think Rust would have prevented them. I did not look at the code for all of them (e.g. if it says “buffer overflow” then it’s pretty clear Rust would prevent it), so take these results with a small pinch of salt. Corrections welcome!

Results

There are 95 bugs. By my count **Rust would have prevented 53 of these**.

1. 47 are standard memory errors (overflows, use-after free, etc.). Rust would definitely prevent these. For comparison, Google found that 70% of Chrome’s high severity security bugs are memory errors.
2. 5 are integer overflows, which Rust does not prevent by default in release mode (though it can via an optional flag), but they lead to memory errors which it does prevent.
3. 1 was through misuse of `fgets()`. Rust does not stop you making difficult to use APIs, but it definitely reduces the chance, e.g. by warning you if you don’t use a `Result`. It’s hard to imagine this bug happening with Rust.



The remaining bugs are logic errors of some kind or another. There are definitely several of the sort “we should have checked thing, but didn’t” that Rust couldn’t help with. But there are also a decent number of other bugs that come from cURL doing ad-hoc inline character-by-character parsing of

just about everything, whereas in Rust you would probably use a library to fully parse things. I've generously counted these as `No` in my tally but I suspect they would be less likely with Rust.

Conclusion

It is safe to say that nobody can write memory-safe C, not even famous programmers that use all the tools. Here's Daniel in 2017:

> We keep scanning the curl code regularly with static code analyzers (we maintain a zero Coverity problems policy) and we run the test suite with valgrind and address sanitizers.

12 out of 15 of cURL's security issues since that statement have been memory errors (or integer overflows leading to memory errors).

Rust proponents may seem overly zealous and I think this has led to a minor backlash of people thinking "Rust can't be *that* great surely; these people must be confused zealots, like Trump supporters or Christians". But it's difficult to argue with numbers like these.

Some other observations

Some random things I noticed when reading the list.

- Some of these bugs are *really* complicated. It is not surprising at all that humans missed them. Only automated tools could detect or prevent these.
- A significant number of bugs (about 9) come from cURL trying to reuse connections and state that it shouldn't.
- The cURL descriptions of their security bugs are great.

The list

These are how I classified the bugs. If I've got something drastically wrong let me know.

#	Vulnerability	Classification	Rust prevention (0=no, 4=yes)
95	wrong connect-only connection	Logic, pointers	2
94	curl overwrite local file with -J	Logic	1
93	Partial password leak over DNS on HTTP redirect	Logic, quoting	1
92	FTP-KRB double-free	Memory	4
91	TFTP small blocksize heap buffer overflow	Memory	4
90	Windows OpenSSL engine code injection	Logic	0
89	TFTP receive buffer overflow	Memory	4
88	Integer overflows in curl_url_set	Integer overflow leading to memory	3
87	NTLM type-2 out-of-bounds buffer read	Memory	4
86	NTLMv2 type-3 header stack buffer overflow	Memory	4
85	SMTP end-of-response out-of-bounds read	Memory	4
84	warning message out-of-buffer read	Memory	4
83	use-after-free in handle close	Memory	4
82	SASL password overflow via integer overflow	Integer overflow leading to memory	3
81	NTLM password overflow via integer overflow	Integer overflow leading to memory	3
80	SMTP send heap buffer overflow	Memory	4
79	FTP shutdown response buffer overflow	Memory	4
78	RTSP bad headers buffer over-read	Memory	4
77	RTSP RTP buffer over-read	Memory	4

#	Vulnerability	Classification	Rust prevention (0=no, 4=yes)
76	LDAP NULL pointer dereference	Memory	4
75	FTP path trickery leads to NIL byte out of bounds write	Memory	4
74	HTTP authentication leak in redirects	Logic	0
73	HTTP/2 trailer out-of-bounds read	Memory	4
72	SSL out of buffer access	Memory	4
71	FTP wildcard out of bounds read	Memory	4
70	NTLM buffer overflow via integer overflow	Integer overflow leading to memory	3
69	IMAP FETCH response out of bounds read	Memory	4
68	FTP PWD response parser out of bounds read	Memory	4
67	URL globbing out of bounds read	Memory	4
66	TFTP sends more than buffer size	Memory	4
65	FILE buffer read out of bounds	Memory	4
64	URL file scheme drive letter buffer overflow	Memory	4
63	TLS session resumption client cert bypass (again)	Logic, reuse	0
62	–write-out out of buffer read	Memory	4
61	SSL_VERIFYSTATUS ignored	Logic	0
60	uninitialized random	Type error	4
59	printf floating point buffer overflow	Memory	4
58	Win CE schannel cert wildcard matches too much	Logic	0

#	Vulnerability	Classification	Rust prevention (0=no, 4=yes)
57	Win CE schannel cert name out of buffer read	Memory	4
56	cookie injection for other servers	Logic, difficult to use API	3
55	case insensitive password comparison	Logic, terrible function name	0
54	OOB write via unchecked multiplication	Memory	4
53	double-free in curl_maprintf	Memory	4
52	double-free in krb5 code	Memory	4
51	glob parser write/read out of bounds	Memory	4
50	curl_getdate read out of bounds	Memory	4
49	URL unescape heap overflow via integer truncation	Memory	4
48	Use-after-free via shared cookies	Memory	4
47	invalid URL parsing with '#'	Logic, used regex, now 2 problems	0
46	IDNA 2003 makes curl use wrong host	Logic, unicode insanity	1
45	curl escape and unescape integer overflows	Integer overflow leading to memory	3
44	Incorrect reuse of client certificates	Logic, reuse	0
43	TLS session resumption client cert bypass	Logic, reuse	0
42	Re-using connections with wrong client cert	Logic, reuse	0
41	use of connection struct after free	Memory	4
40	Windows DLL hijacking	Windows API nonsense	0
39	TLS certificate check bypass with mbedTLS/PolarSSL	Logic	0

#	Vulnerability	Classification	Rust prevention (0=no, 4=yes)
38	remote file name path traversal in curl tool for Windows	Logic, quoting	0
37	NTLM credentials not-checked for proxy connection re-use	Logic, reuse	0
36	SMB send off unrelated memory contents	Memory, but I think this is still reading from valid allocated memory, heartbleed style	0
35	lingering HTTP credentials in connection re-use	Logic, reuse	0
34	sensitive HTTP server headers also sent to proxies	Logic	0
33	host name out of boundary memory access	Memory	4
32	cookie parser out of boundary memory access	Memory	4
31	Negotiate not treated as connection-oriented	Logic	0
30	Re-using authenticated connection when unauthenticated	Logic, reuse	0
29	darwinssl certificate check bypass	Logic, reuse	0
28	URL request injection	Logic	0
27	duphandle read out of bounds	Memory	4
26	cookie leak for TLDs	Logic, parsing	0
25	cookie leak with IP address as domain	Logic	0
24	not verifying certs for TLS to IP address / Winssl	Logic	0
23	not verifying certs for TLS to IP address / Darwinssl	Logic	0

#	Vulnerability	Classification	Rust prevention (0=no, 4=yes)
22	IP address wildcard certificate validation	Logic	0
21	wrong re-use of connections	Logic	0
20	re-use of wrong HTTP NTLM connection	Logic, reuse	2
19	cert name check ignore GnuTLS	Logic	0
18	cert name check ignore OpenSSL	Logic	0
17	URL decode buffer boundary flaw	Memory	4
16	cookie domain tailmatch	Logic	0
15	SASL buffer overflow	Memory	4
14	SSL CBC IV vulnerability	Logic	0
13	URL sanitization vulnerability	Logic, parsing	0
12	inappropriate GSSAPI delegation	Logic	0
11	local file overwrite	Logic, parsing	0
10	data callback excessive length	Memory	4
9	embedded zero in cert name	Logic, null-terminated strings	4
8	Arbitrary File Access	Logic	0
7	GnuTLS insufficient cert verification	Logic	0
6	TFTP Packet Buffer Overflow	Memory	4
5	URL Buffer Overflow	Memory	4
4	NTLM Buffer Overflow	Memory	4
3	Authentication Buffer Overflows	Memory	4
2	Proxy Authentication Header Information Leakage	Logic	0
1	FTP Server Response Buffer Overflow	Memory	4