Memory Vulnerabilities in Memory-safe Languages

Veit Heller
Information Security Meetup Berlin, August 2020

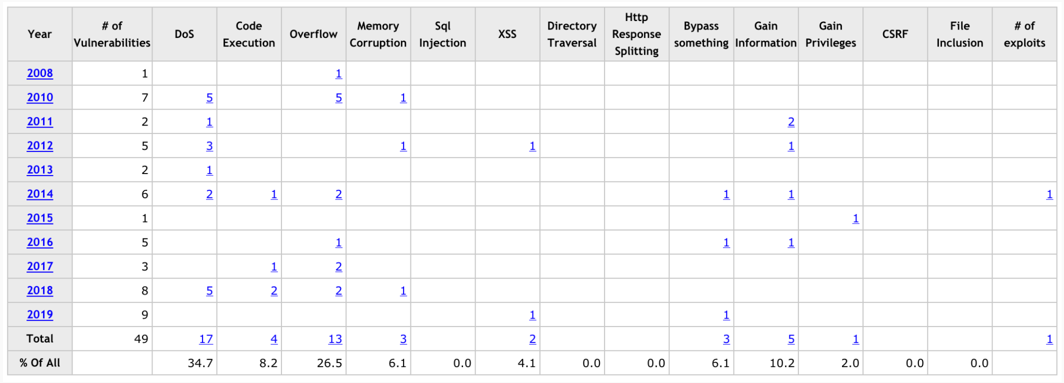
August 27, 2020

Scope

Scope

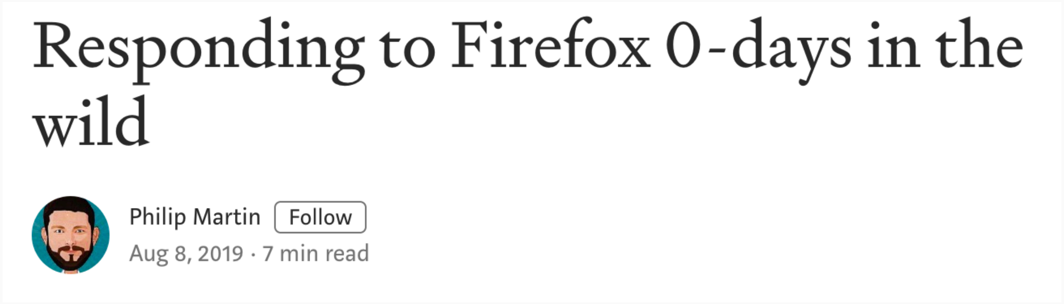
~~Compilers/Interpreters~~

Python



Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2008	1			1											
2010	7	5		5	1										
2011	2	1									2				
2012	5	3			1		1				1				
2013	2	1													
2014	6	2	1	2						1	1				1
2015	1											1			
2016	5			1						1	1				
2017	3		1	2											
2018	8	5	2	2	1										
2019	9						1			1					
Total	49	17	4	13	3		2			3	5	1			1
% Of All		34.7	8.2	26.5	6.1	0.0	4.1	0.0	0.0	6.1	10.2	2.0	0.0	0.0	

JavaScript

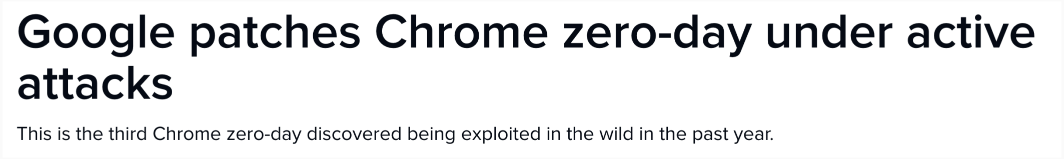


Responding to Firefox 0-days in the wild

Philip Martin

Aug 8, 2019 · 7 min read

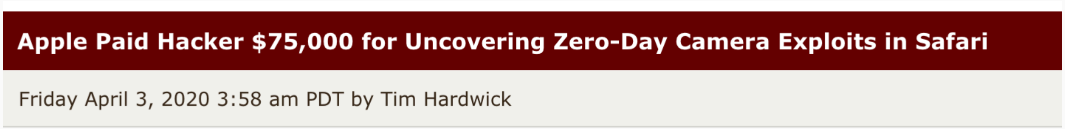
JavaScript…



Google patches Chrome zero-day under active attacks

This is the third Chrome zero-day discovered being exploited in the wild in the past year.

More JavaScript...


Apple Paid Hacker $75,000 for Uncovering Zero-Day Camera Exploits in Safari

Friday April 3, 2020 3:58 am PDT by Tim Hardwick

Scope

Runtimes

Scope

~~Bashing~~

Scope

~~Bashing~~

$\Rightarrow$ No Silver Bullets

Denial of Service (DoS)

Concurrency

Concurrency is getting easier to work with.

Concurrency

When something is easy to work with, we tend to shoot ourselves in the foot with it.

Concurrency

Let’s talk about channels.

Concurrency

If a value is written to a channel and never read, what happens to it?

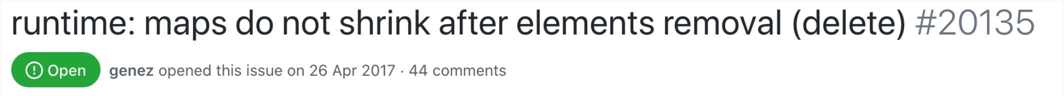
Concurrency

What happens if we wait to read and noone answers?

Concurrency

Always think about your processes' lifetimes.

Go issue 20135



runtime: maps do not shrink after elements removal (delete) #20135

Open genez opened this issue on 26 Apr 2017 · 44 comments

Go issue 20135

```
func main() {
  runtime.GC(); memUsage() // basically 0
  m := make(map[int]int)  // we start alloc'ing

  for i := 0; i < 100000; i++ { m[i] = i }

  runtime.GC() // nothing deleted, of course

  for i := 0; i < 100000; i++ { delete(m, i) }

  runtime.GC() // still nothing deleted!

  fmt.Println(m) // just to make sure GC is not too clever
}
```

Listing 1: Go sitting on your memory.

So?

- Memory bugs don't need to corrupt memory.

So?

- Memory bugs don't need to corrupt memory.
- Runtimes hide a lot from you (good and bad).

Haskell

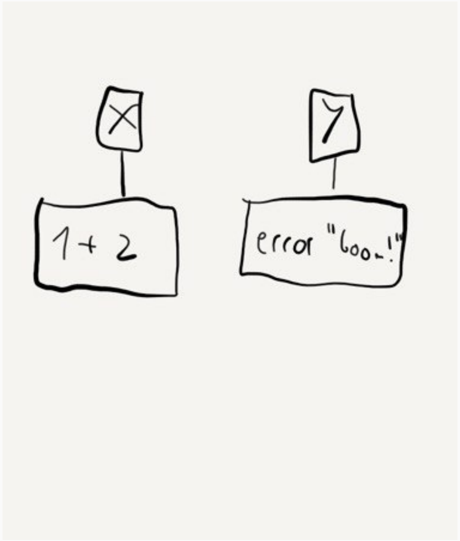
Haskell is lazy.

Haskell

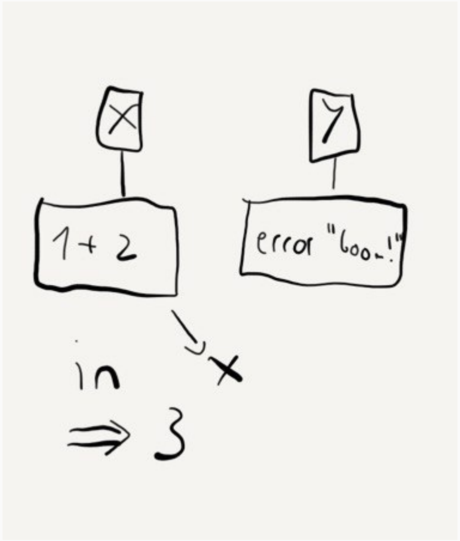
```haskell
let (x, y) = (1 + 2, error "boom!") in x -- => 3
```

Listing 2: Thunks in action.

Haskell



Haskell



Haskell

New vocabulary: space leaks.

Space Leaks

“Pinpointing spayce leaks is a skill that takes practice and perseverance. Better tools could significantly simplify the process.“
— Mitchell, Neil: Leaking Space. Eliminating memory hogs.

Space Leaks

“Using the benchmark I observed a space leak. But the program is huge, and manual code inspection usually needs a 10 line code fragment to have a change. So I started modifying the program to do less, and continued until the program did as little as it could, but still leaked space. After I fixed a space leak, I zoomed out and saw if the space leak persisted, and then had another go.“
— Mitchell, Neil: Fixing Space Leaks in Ghcide

So?

- Again: Runtimes hide a lot from you (good and bad).

So?

- Again: Runtimes hide a lot from you (good and bad).
- If your runtime is complex, it can feel like an adversary.

Memory Bugs

Rust

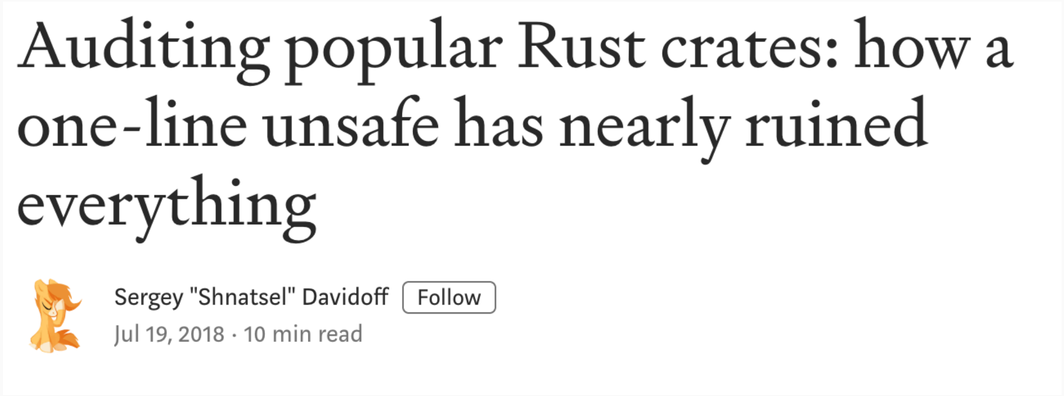
```rust
fn main() {
  unsafe fn dangerous<'a>() -> *const String {
    let tmp:String = "boom goes the dynamite!".to_string();
    &tmp
  }

  println!("{:?}", unsafe { dangerous().as_ref() })
}
```

Listing 3: `unsafe` considered... unsafe?

Rust

```
#![forbid(unsafe_code)]
```

Rust



Auditing popular Rust crates: how a one-line unsafe has nearly ruined everything

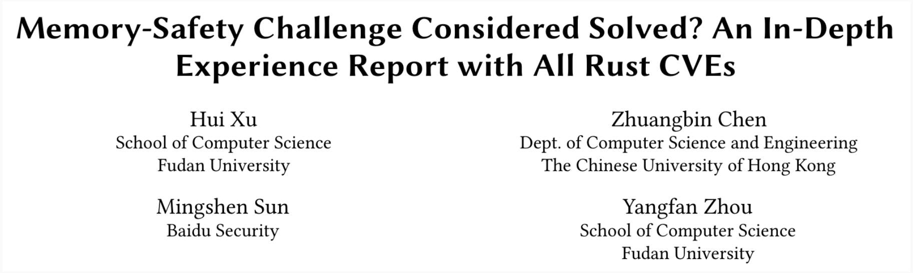
Sergey "Shnatsel" Davidoff

Jul 19, 2018 · 10 min read

Rust

“If you want to write DoS-critical code in Rust and use some existing libraries, you’re out of luck. Nobody cares about denial of service attacks. You can poke popular crates with a uzzer and get lots of those. When you report them, they do not get fixed.“
— Davidoff, Sergey (Shnatsel): How Rust’s standard library was vulnerable for years and nobody noticed

Rust



Memory-Safety Challenge Considered Solved? An In-Depth Experience Report with All Rust CVEs

Hui Xu
School of Computer Science
Fudan University

Zhuangbin Chen
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong

Mingshen Sun
Baidu Security

Yangfan Zhou
School of Computer Science
Fudan University

Rust

“Most importantly, we find while Rust successfully limits memory-safety risks to the realm of unsafe code, it also brings some side effects that cause new patterns of dangling-pointer issues. In particular, most of the use-after-free and double-free bugs are related to the automatic destruction mechanism associated with the ownership-based memory management scheme.“
— Xu, Hui et al.: Memory-Safety Challenge Considered Solved? An In-Depth Experience Report with All Rust CVEs

So?

- Don't use your language's escape hatches.

So?

- Don’t use your language’s escape hatches.
- Seriously.

So?

- Don't use your language's escape hatches.
- Seriously.
- Please.

So?

- Don’t use your language’s escape hatches.
- Seriously.
- Please.
- Or write proofs, but I know you won’t, so don’t.

Conclusions (and better vibes)

Conclusion

Everything sucks in its own way, and that’s alright.

Conclusion

Nothing will be perfectly secure. Make a better threat model.

References

- These slides: `https://github.com/hellerve/talks`
- Go bug 20135: `https://github.com/golang/go/issues/20135`
- Breaking Erlang Maps: `https://medium.com/@jlouis666/breaking-erlang-`
- RustBelt: `https://plv.mpi-sws.org/rustbelt`
- Space leak: A Haskell Sore Spot: `https://fremissant.net/leaky`
- Auditing popular Rust crates: how a one-line unsafe has nearly ruined everything: `https://medium.com/@shnatsel/auditing-popular-`
- Fixing Space Leaks in Ghcide: `https://neilmitchell.blogspot.com/2020/05/fixi`

References

- Apple Paid Hacker 75,000 for Uncovering Zero-Day Camera Exploits in Safari
  `https://www.macrumors.com/2020/04/03/apple-pai`
- Google patches Chrome zero-day under active attacks
  `https://www.zdnet.com/article/google-patches-c`
- Responding to Firefox 0-days in the wild
  `https://blog.coinbase.com/responding-to-firefo`
- Xu, Hui et al.: Memory-Safety Challenge Considered Solved? An In-Depth Experience Report with All Rust CVEs
- Kulal, Sumith et al.: Space leaks exploration in Haskell
- Mitchell, Neil: Leaking Space—Eliminating memory hogs

Thank you!

Questions?

Slides at `https://github.com/hellerve/talks`