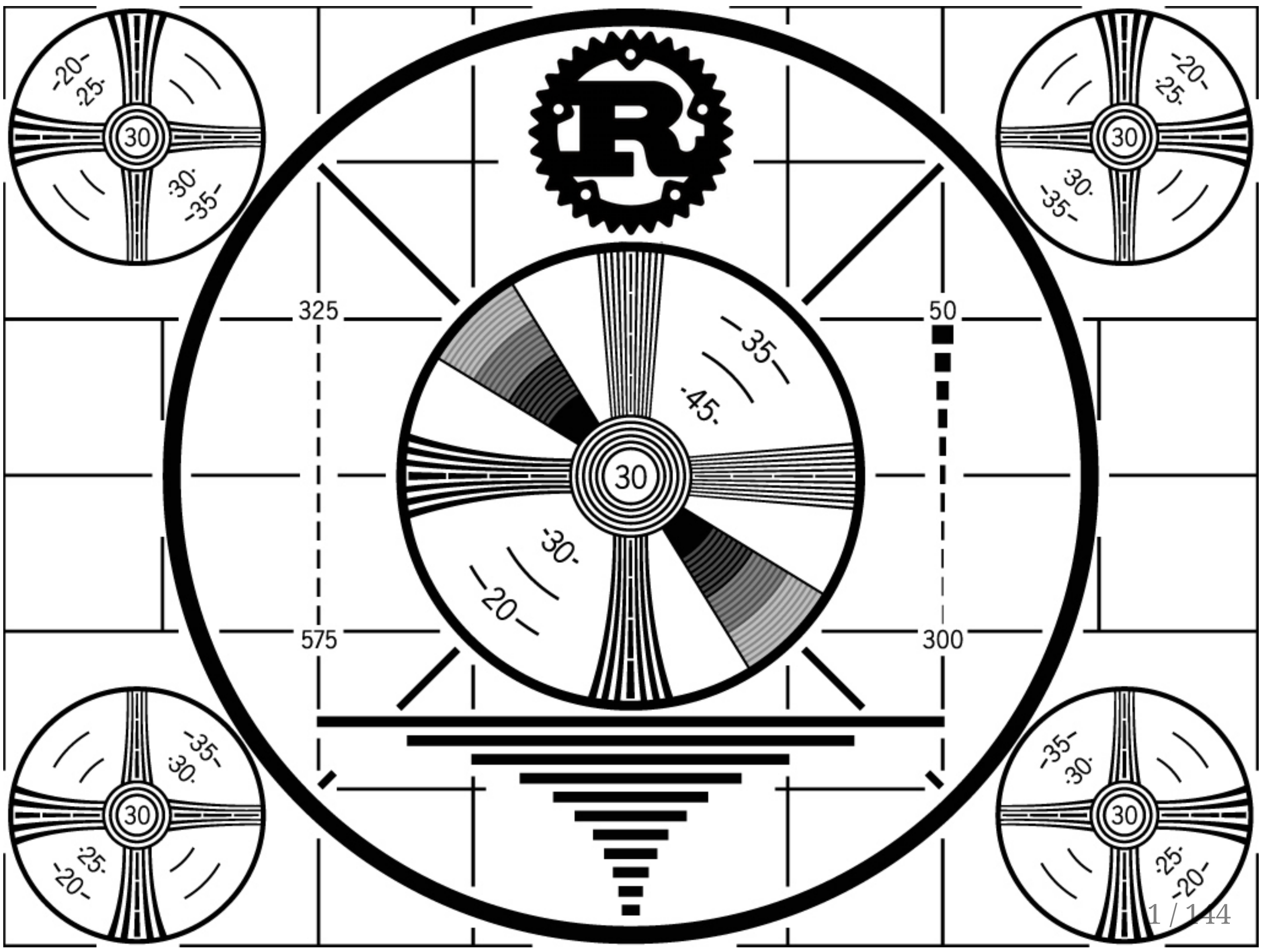
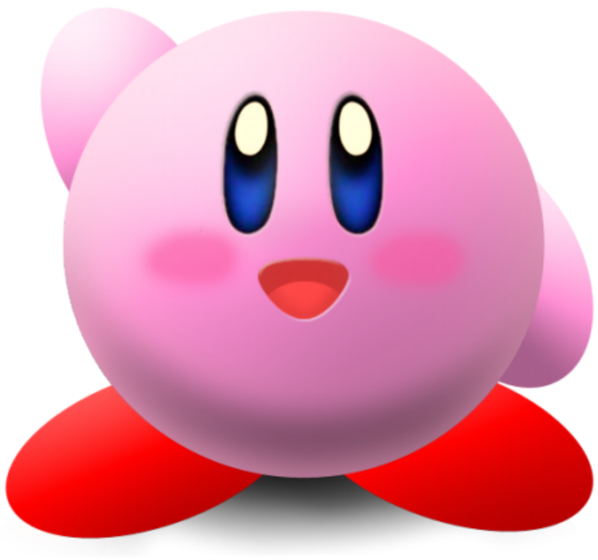
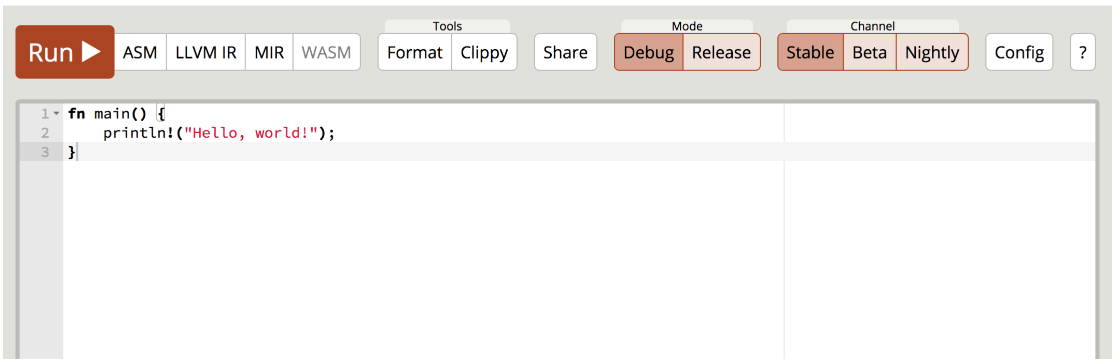

-20-
-25-
30
-30-
-35-
-45-
325
575
50
300

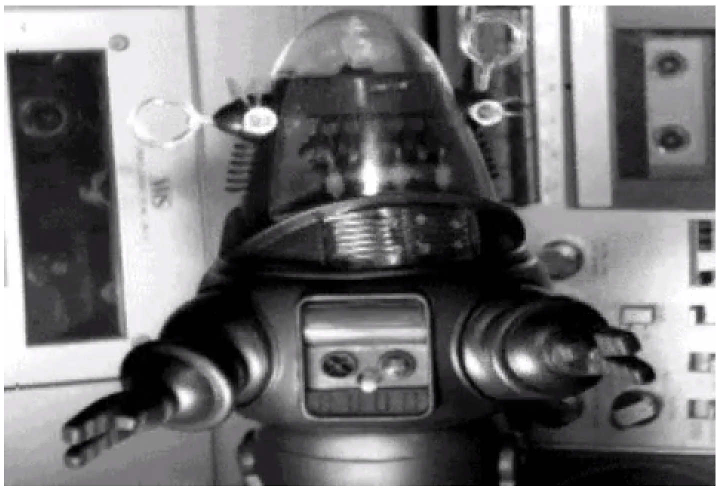
integer 32



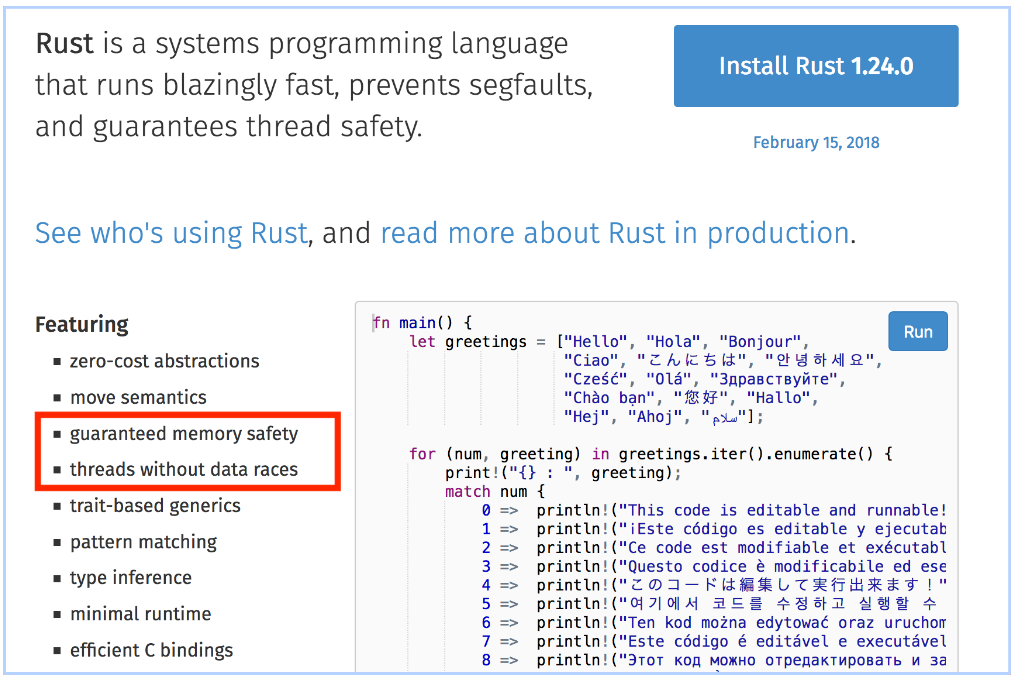


```rust
fn main() {
    println!("Hello, world!");
}
```

- Rust infrastructure team
- Working on a Rust video course for Manning
- A handful of crates
- Help out with AVR-Rust



- Examples of C and C++ code
- Does not mean C and C++ code is bad
- Does mean C or C++ programmers are bad
- Not a professional security researcher


Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

See who's using Rust, and read more about Rust in production.

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo",
                     "Hej", "Ahoj", "سلام"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnable!
            1 =>  println!("¡Este código es editable y ejecutab
            2 =>  println!("Ce code est modifiable et exécutabl
            3 =>  println!("Questo codice è modificabile ed ese
            4 =>  println!("このコードは編集して実行出来ます！"
            5 =>  println!("여기에서 코드를 수정하고 실행할 수
            6 =>  println!("Ten kod można edytować oraz urucho
            7 =>  println!("Este código é editável e executável
            8 =>  println!("Этот код можно отредактировать и за
```

This code isn't even possible in Rust.

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965. [...] This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Tony Hoare, 2009

Equivalent to

```
false
```

- Signed overflow
  - abort in debug
  - wrap in release
- Can't shift numbers by a negative amount
- Can't shift numbers by more than number of bits

```
```

```
43
```

- Variables changing unexpectedly => hard-to-track bugs
- Unchanging value can be optimizated
- Effect is different from other languages
    - Property of the variable, not the value

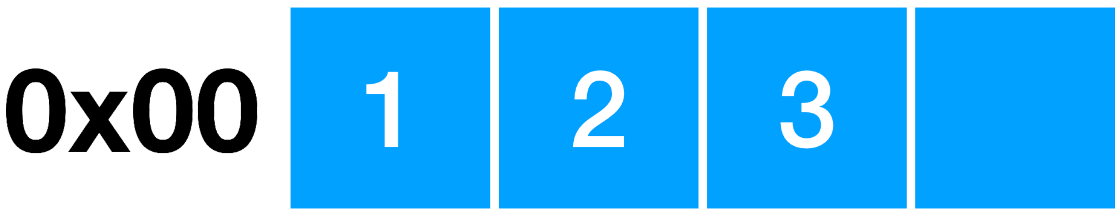
```
43
```

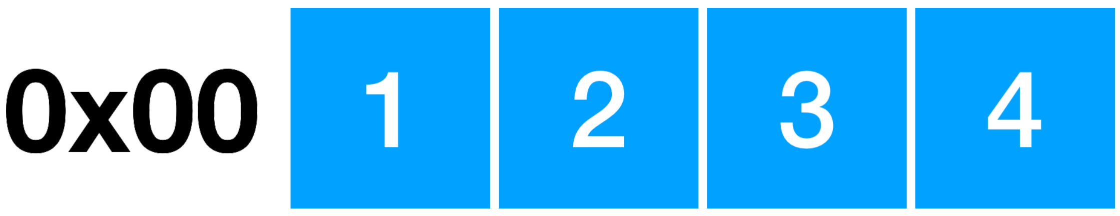
```
```

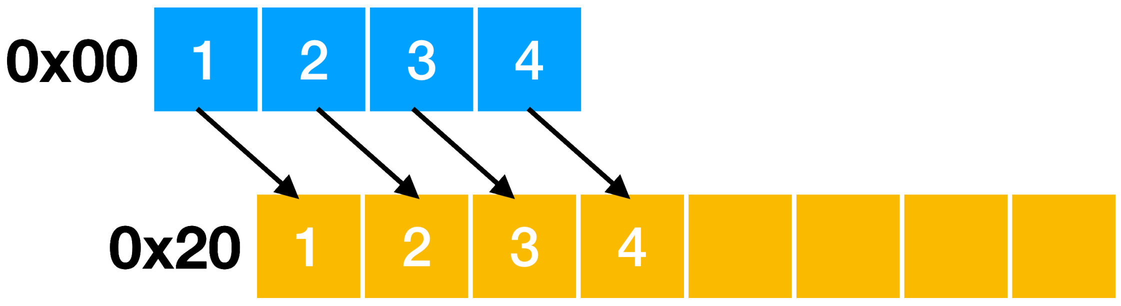
```
H�}�H��
```

```
0
1
2
```

```
1
2
3
2
4
6
```


0x00
1
2
3


0x00
1
2
3
4


0x00
1
2
3
4
0x20
1
2
3
4

```
[0, 1, 2, 2, 4, 6]
```

```
42
42
```

```
42
```

- Values are moved by default
- Also called "transferring ownership"
- More efficient
    - Old value doesn't need to be in a valid state
- Owner controls when resources cleaned up (RAII)

- Access value without transferring ownership
- Can be immutable or mutable
- There are rules. Only one of:
    - Many immutable borrows
    - One mutable borrow

```
42
42
```

```
43
43
```

```
100
```

- Metadata to relate input references to output references
    - How long the reference will remain valid
- Every reference has an associated lifetime
    - Not always relevant
    - Lifetime elision handles the common cases
- effectively means "forever"

```
100
```

> one of the best parts about stylo has been how much easier it has been to implement these style system optimizations that we need, because Rust
>
> can you imagine if we needed to implement this all in C++ in the timeframe we have
>
> yeah srsly
>
> heycam: it’s so rare that we get fuzz bugs in rust code
>
> heycam: considering all the complex stuff we’re doing
>
> heycam: think about how much time we could save if each one of those annoying compiler errors today was swapped for a fuzz bug tomorrow :-)
>
> heh
>
> you guys sound like an ad for Rust

```
1073741824
```

- "double-free" 42 CVE entries

- "double-free" 42 CVE entries
- "use-after-free" 321 CVE entries

- "double-free" 42 CVE entries
- "use-after-free" 321 CVE entries
- "uninitialized" 350 CVE entries

- "double-free" 42 CVE entries
- "use-after-free" 321 CVE entries
- "uninitialized" 350 CVE entries
- "null dereference" 1189 CVE entries

- "double-free" 42 CVE entries
- "use-after-free" 321 CVE entries
- "uninitialized" 350 CVE entries
- "null dereference" 1189 CVE entries
- "out-of-bounds" 1291 CVE entries

- "double-free" 36 bugs found

- "double-free" 36 bugs found
- "out-of-bounds" 84 bugs found

- "double-free" 36 bugs found
- "out-of-bounds" 84 bugs found
- "null dereference" 363 bugs found

- "double-free" 36 bugs found
- "out-of-bounds" 84 bugs found
- "null dereference" 363 bugs found
- "uninitialized" This result was limited to 500 bugs

- "double-free" 36 bugs found
- "out-of-bounds" 84 bugs found
- "null dereference" 363 bugs found
- "uninitialized" This result was limited to 500 bugs
- "use-after-free" This result was limited to 500 bugs

- : can be transferred between threads
- : references can be shared between threads

- : can be transferred between threads
- : references can be shared between threads

-
- (Atomics, Channels, Mutex)

- : can be transferred between threads
- : references can be shared between threads

-
- (Atomics, Channels, Mutex)

- Futures and async/await

Crates exist to make parallelization easy and safe.

> you can change a sequential iterator into a parallel iterator just by adding the crate, importing the trait and changing to . If it’s not thread-safe to do, then it won’t compile

Chris Morgan

- Integer overflow
- Deadlocks
- Leaks of memory and other resources
- Exiting without calling destructors

- The Rust compiler is conservative
- What we think:
  - Prevents code that would cause undefined behavior

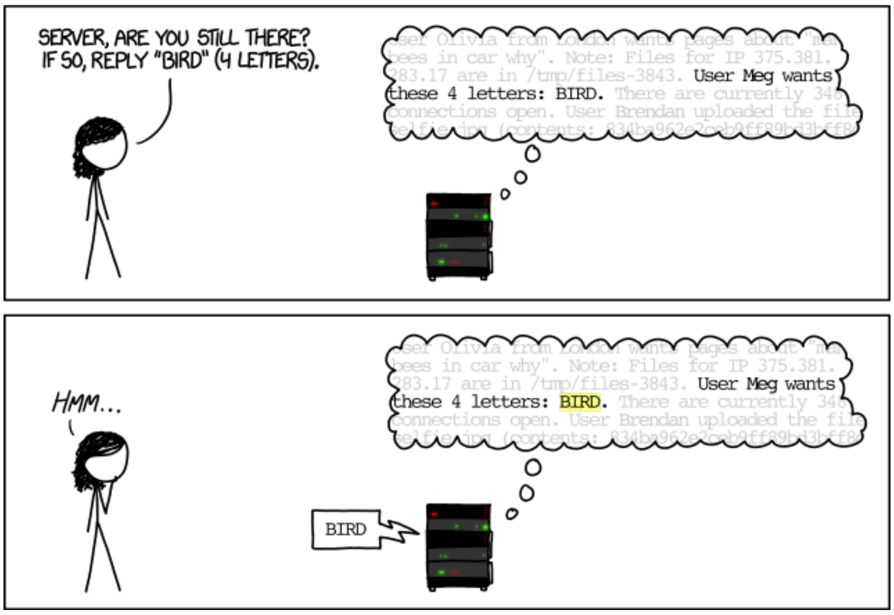
- The Rust compiler is conservative
- What we think:
  - Prevents code that would cause undefined behavior
- What it really is:
  - Prevents code it can't guarantee doesn't cause undefined behavior

- Dereference a raw pointer
- Call an unsafe function
- Implement an unsafe trait
- Read or write a mutable static variable
- Read a field of a union

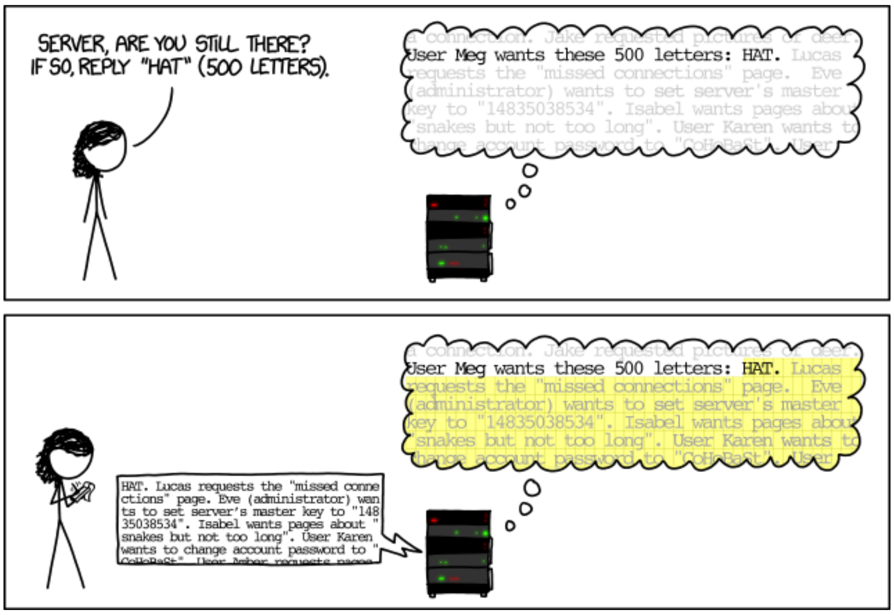
- References (/)
  - Are never
  - Always point to a valid value
  - Have lifetimes

- References (/)
    - Are never
    - Always point to a valid value
    - Have lifetimes
- Raw pointers (/)
    - Can be
    - Can point anywhere
    - Do not have lifetimes

- uses unsafe code internally
- You don't need unsafe code to use
- Type and module systems keep unsafe code contained


SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Meg wants these 4 letters: BIRD.
HMM...
BIRD

https://xkcd.com/1354/

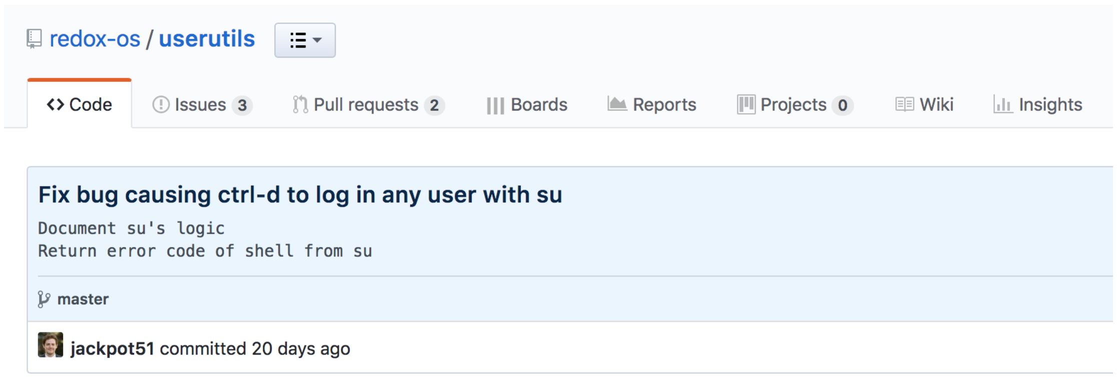

SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

https://xkcd.com/1354/

- Buffer overread
- Did not validate user input
- Reusing a buffer
- Wrote their own memory allocator

- Copy-pasting
- Poor alignment / curly braces
- makes it hard to follow code
- Lack of dead code warnings
- Intermingled error and success code

- Untrusted input
- Integer overflow
- Loose integer operations



Fix bug causing ctrl-d to log in any user with su

```
Document su's logic
Return error code of shell from su
```

master

jackpot51 committed 20 days ago

Redox OS

A Rust Operating System

http://www.redox-os.org info@redox-os.org

The best way to prevent these kinds of attacks is either to use a higher level language, which manages memory for you (albeit with less performance), or to be very, very, very, very careful when coding.

- https://xda-developers.com/a-demonstration-of-stagefright-like-mistakes/

- Reduces or eliminates entire classes of bugs
- Frequently at compile time
- Minimal to no change in performance
- Can performance
- Allows us to focus more on the logic of the problem

Rust lets us make new mistakes by preventing us from making the same old mistakes over and over.

Carol Nichols, Rust core team member

- Type system
- Safe defaults
- Error handling

integer 32