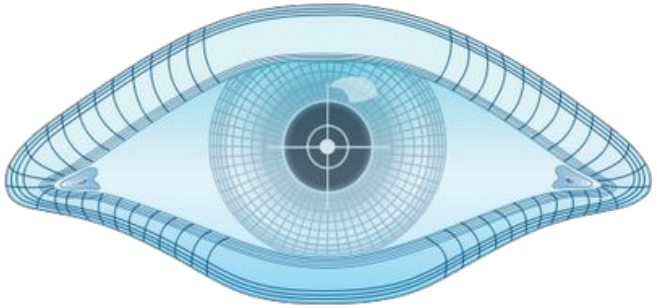
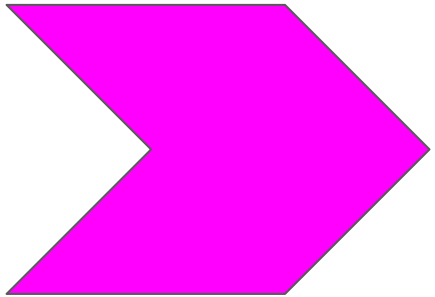
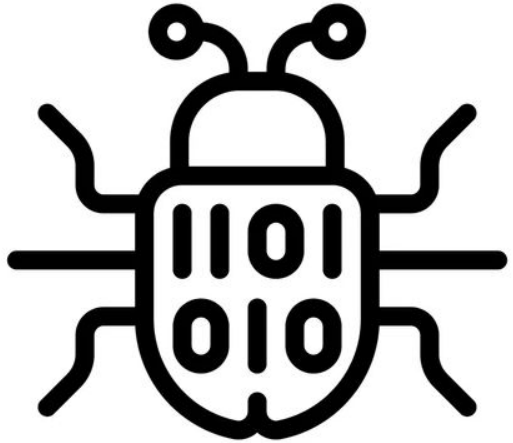
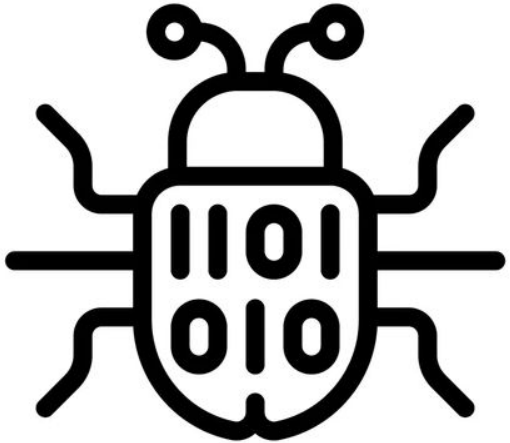
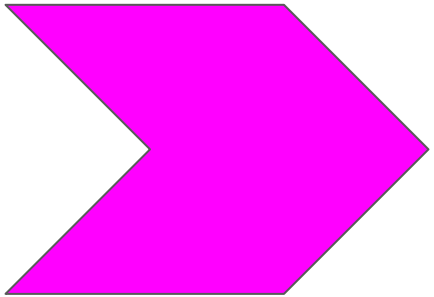

EXPLOSIVE OVERFLOW

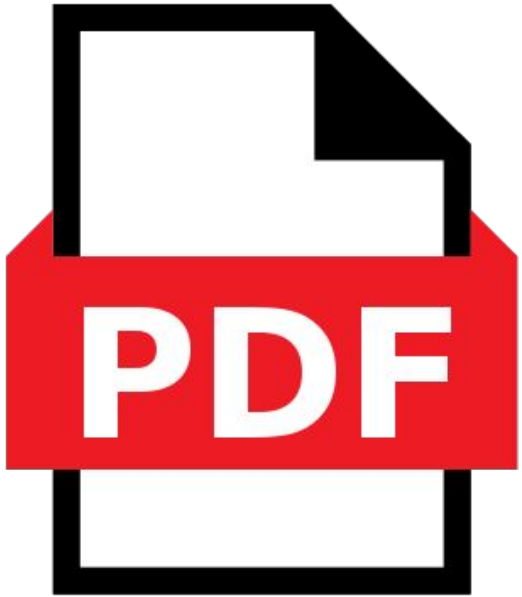




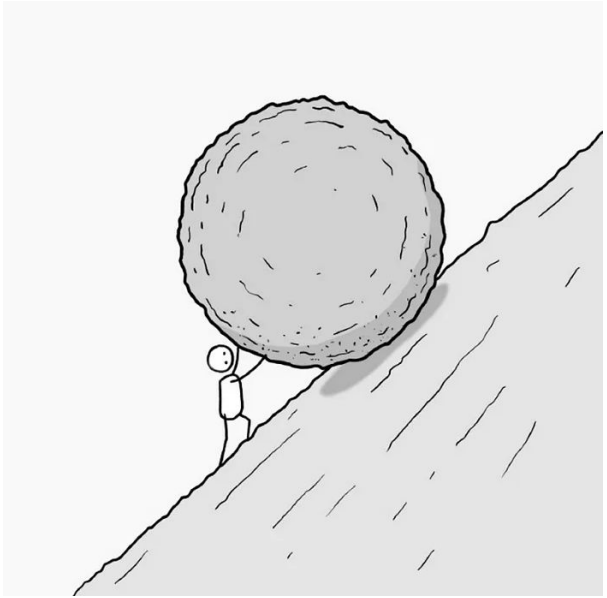




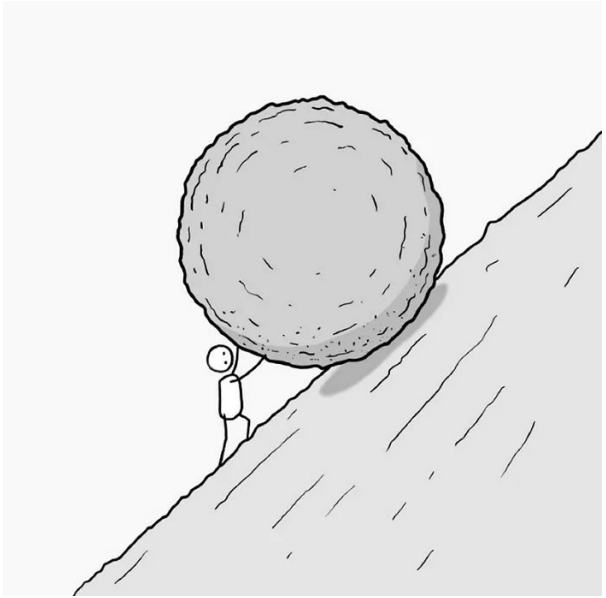



PDF

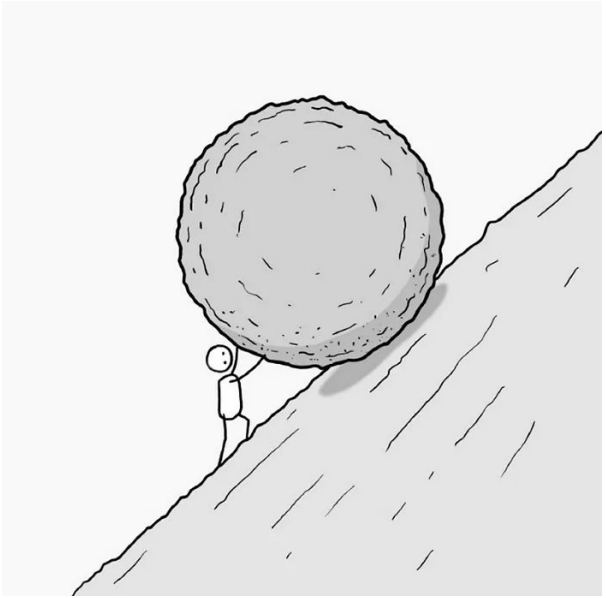
Year 1



Year 2



Year 3



GonzoHacker
@GonzoHacker

There is no evidence of intrusion as we're not very good about logging

12:56 AM · Apr 4, 2018 · Twitter Web Client

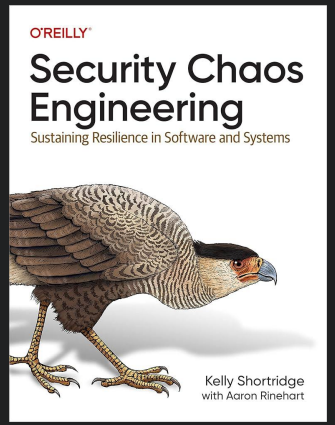
Software Security Assurance

In theory...

“Let’s make a product more secure…”

Security Chaos Engineering

- Security is a subset of resilience.
- Resilience is an *emergent property of the entire system* and cannot be measured by only analyzing components.
- To create and assess systems, we make mental models of it, cognitive representations of external reality.
- Given a complex system to assess, how do we gather requirements?



Standards & Guidelines

(shows intent)

Compatibility with Engineering Culture

- Usability
- Psychological safety
- Avoid adding friction
- Maintain pleasant culture
- Blameless

SecEng Toolset

- SAST: Static Application Security Testing
- SCA: Software Composition Analysis
- SBoM: Software Bill of Materials
- DAST: Dynamic AppSec Testing
- Secrets detection
- Pipeline scanning (incl. Cloud, IaC, Containers, etc)
- Threat modeling
- Secure coding trainings

February 26, 2024


let mut wtr = WriterBuilder::new()
.has_headers(false)
.quote(b'#')
.double_quote(true)
.from_writer(&mut result_data);
let _ = wtr.serialize(&record);
it's unhackable!
MADE
WITH
C++

BACK TO THE BUILDING BLOCKS:

A Path Toward Secure and Measurable Software

FEBRUARY 2024

THE WHITE HOUSE
WASHINGTON

“the current ecosystem does not sufficiently incentivize the investments required to secure the foundations of cyberspace”

1. Hardware
2. Formal Methods
3. Memory-Safe Programming Languages

1. Hardware
2. Formal Methods
3. Memory-Safe Programming Languages

1. Hardware
2. Formal Methods
3. Memory-Safe Programming Languages

Formal Methods

- Proving that a program behaves as intended.
- Create a mathematical model of the program.
- Some compilers and kernels have been formally verified.

Continuous Formal Verification of Amazon s2n

Andrey Chudnov[1], Nathan Collins[1], Byron Cook[3,4], Joey Dodds[1], Brian Huffman[1], Colm MacCárthaigh[3], Stephen Magill[1(✉)], Eric Mertens[1], Eric Mullen[2], Serdar Tasiran[3], Aaron Tomb[1], and Eddy Westbrook[1]

[1] Galois, Inc., Portland, USA
stephen@galois.com
[2] University of Washington, Seattle, USA
[3] Amazon Web Services, Seattle, USA
[4] University College London, London, UK

Abstract. We describe formal verification of s2n, the open source TLS implementation used in numerous Amazon services. A key aspect of this proof infrastructure is continuous checking, to ensure that properties remain proven during the lifetime of the software. At each change to the code, proofs are automatically re-established with little to no interaction from the developers. We describe the proof itself and the technical decisions that enabled integration into development.

Limits of Formal Verification

- Prove that the *binary* is a correct implementation of the specification.

- See Ken Thompson's "Reflections on Trusting Trust" (1984)

  "No amount of source-level verification or scrutiny will protect you from using untrusted code."

1. Hardware
2. Formal Methods
3. Memory-Safe Programming Languages


POLICE
THE C PROGRAMMING LANGUAGE
Assembly Language Step by Step
C++
BENJAMIN SMITH
C++
C++

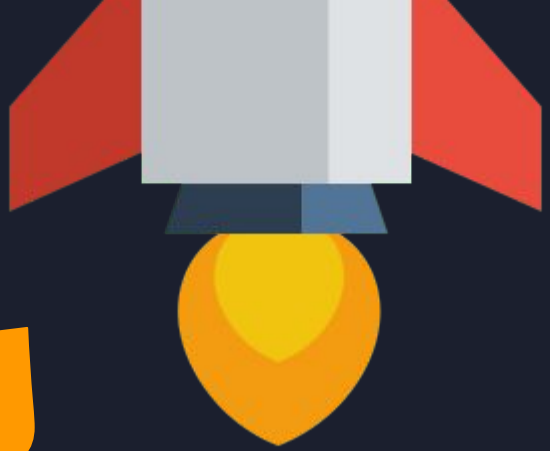
Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Improving Interoperability Between Rust and C++

February 5, 2024

Posted by Lars Bergstrom – Director, Android Platform Tools & Libraries and Chair of the Rust Foundation Board



Lessons FROM Rocket Science

1996

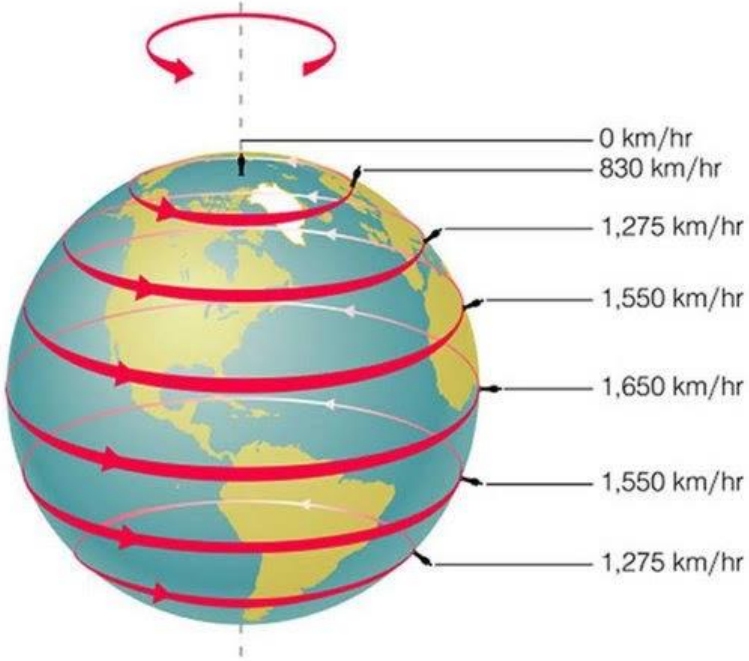
4 June 1996

Kourou, French Guiana

Kourou's facilities:
1 Tracking station of montagne des Pères
2 Paracaibo wharf
3 Jupiter control center
4 Administration
5 Satellite preparation building (EPCU)
6 Tracking and weather station
7 Satellite tracking station (ESA)
8 Soyouz control center
9 Soyouz integration building
10 Cinethéodolite of Royale island

Salut islands

Launch area :
A Sounding rockets
B Diamant (decommissioned)
C Ariane 4 (ELA-2) (decommissioned)
D Ariane 5 (ELA-3)
E Vega (ELV ex ELA-1 Ariane)
F Soyouz (ELS)


0 km/hr
830 km/hr
1,275 km/hr
1,550 km/hr
1,650 km/hr
1,550 km/hr
1,275 km/hr

Launches today!

Cause of explosion:

Honorable self-destruct
(for safety)

Samurai about to perform seppuku

Inertial Reference System

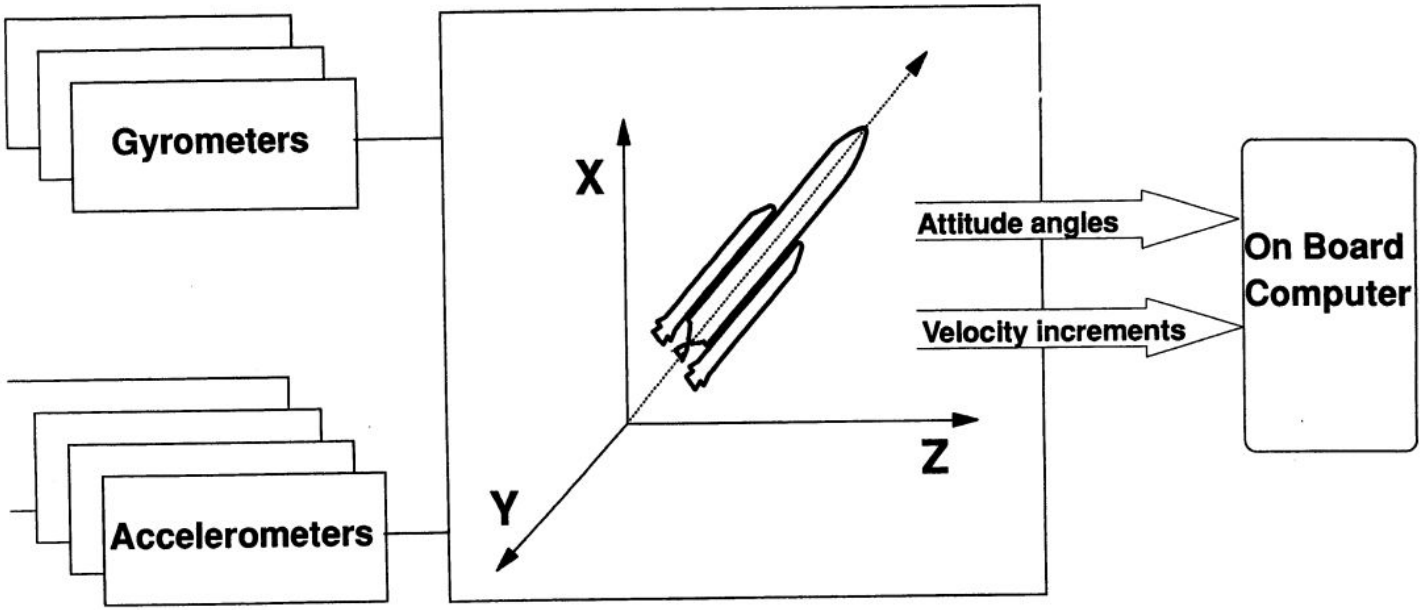
Calibrates trajectory *before* liftoff

Created & used
on the Ariane 4

Moved as-is
onto the Ariane 5

PRINCIPLE

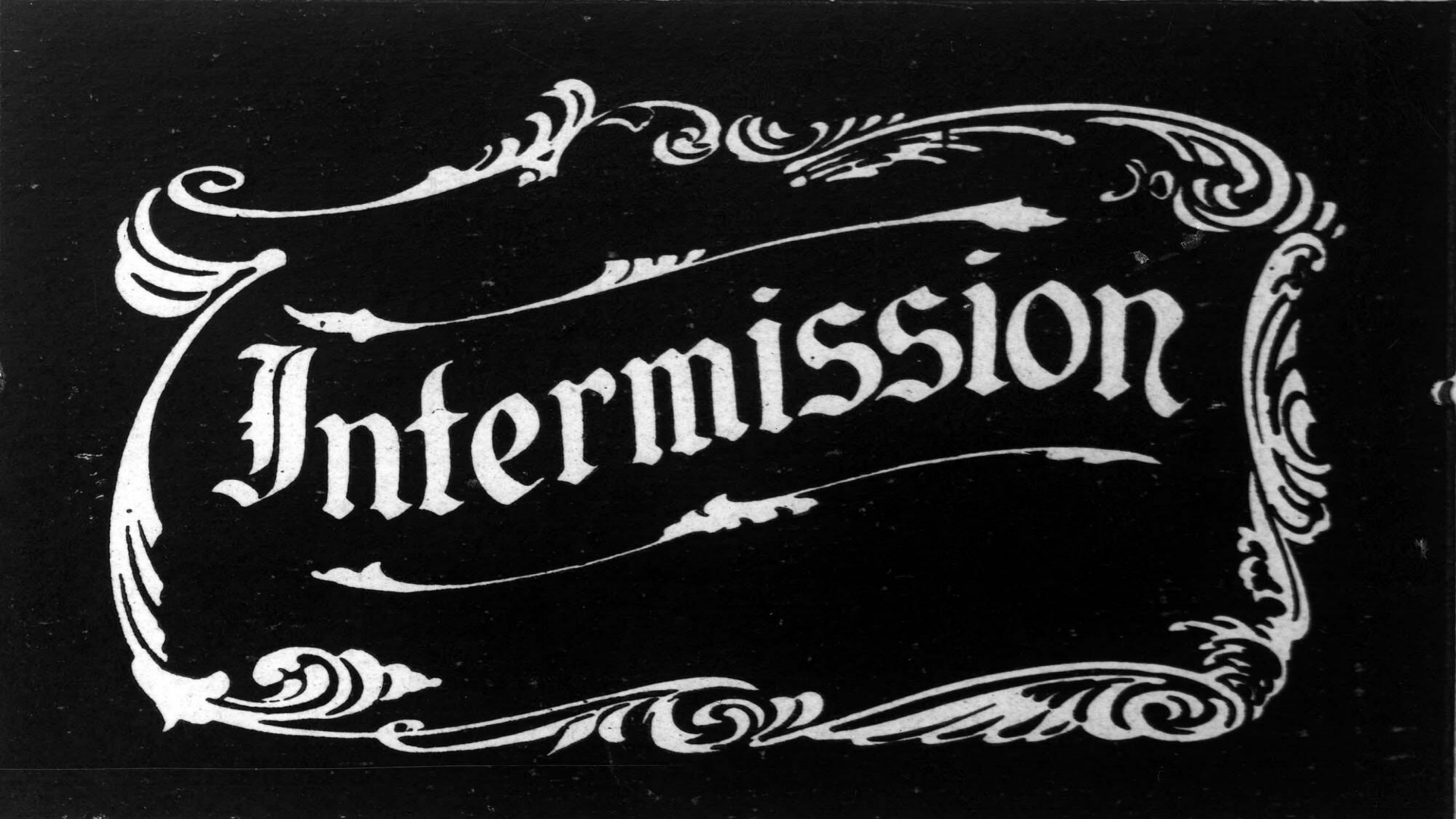
The SRI defines a reference trihedron which is fixed with respect to the stars, called the inertial trihedron within which it provides launcher attitude and velocity data

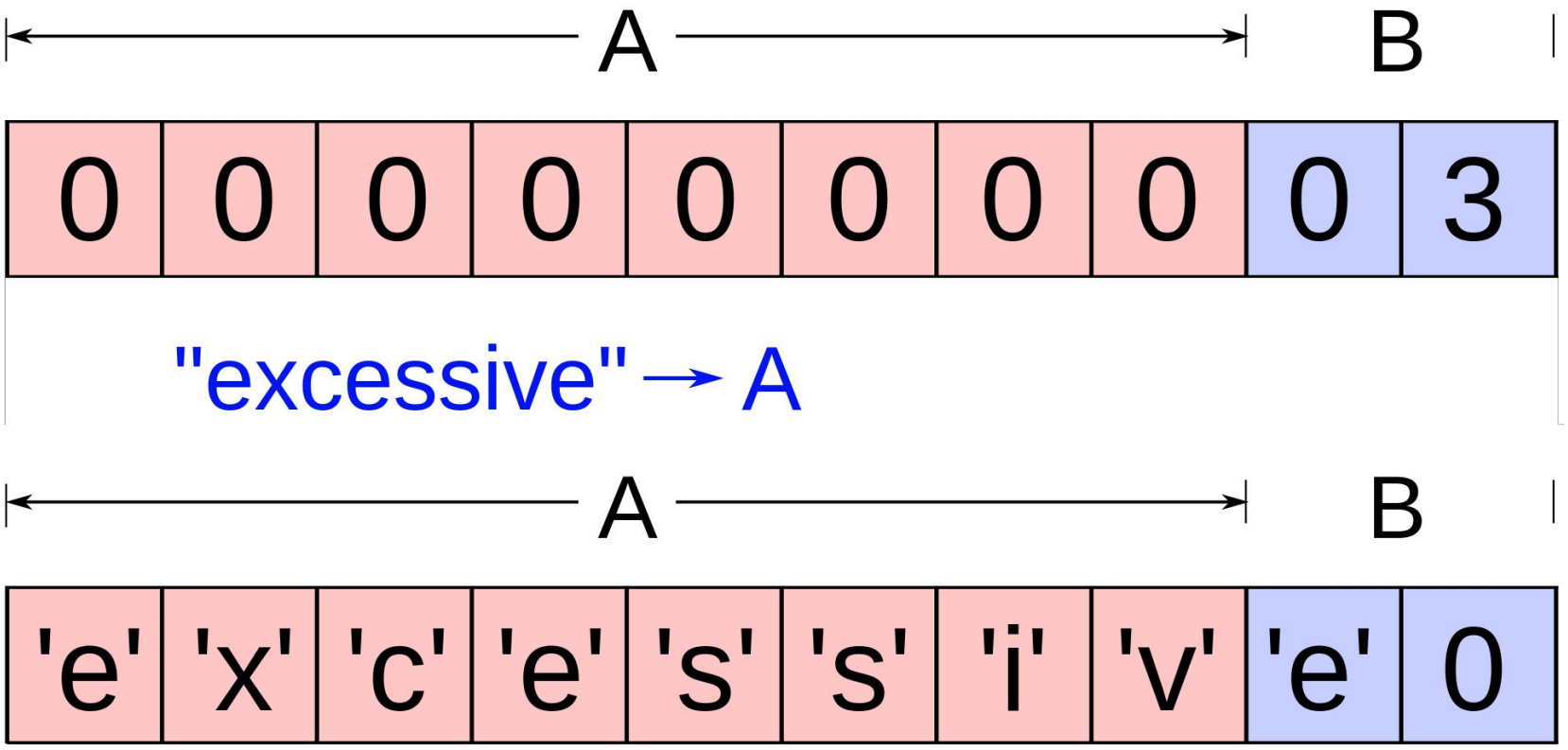


Gyrometers are of the "gyrolaser" type

Integer Overflow?

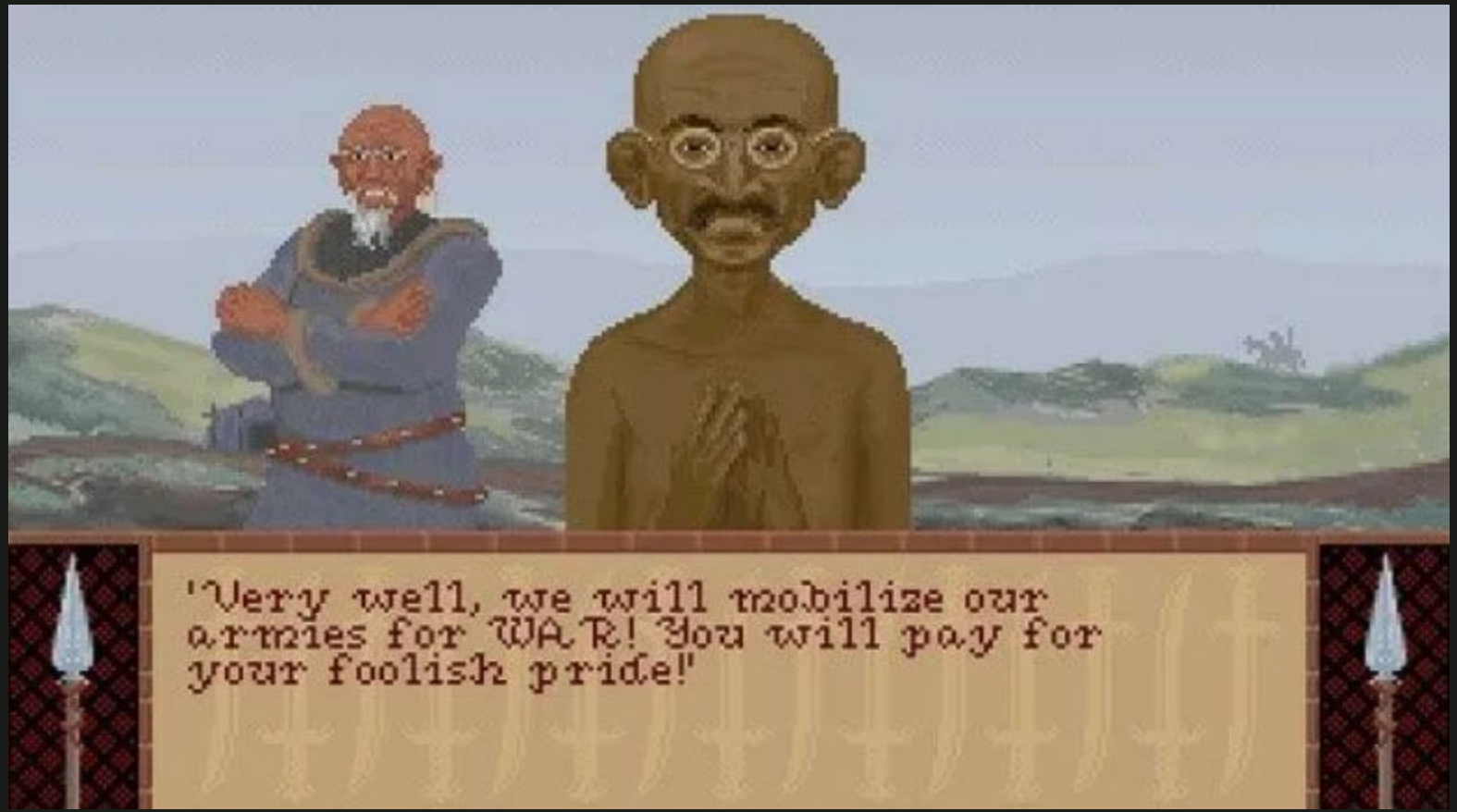
- Horizontal velocity: 64-bit float
- IRS buffer: 16-bit signed int


Intermission


A
B
0 0 0 0 0 0 0 0 0 3
"excessive" → A
A
B
'e' 'x' 'c' 'e' 's' 's' 'i' 'v' 'e' 0


P R N D L2 L1
P R N D L2 L1
P R N D L2 L1

Nuclear Gandhi?


'Very well, we will mobilize our armies for WAR! You will pay for your foolish pride!'


FRAUD
very well, we will mobilize our
armies for WAR! You will pay for
your foolish pride!

Undefined Behavior

- Loss of control flow.
- No idea what happens at runtime.
- Belief that it is the programmer's responsibility to avoid UB.
- `gcc` and `clang` support **`-fsanitize=undefined`** flag (not by default).
- E.g.: Overflows, memory safety violations, use of uninitialized auto var, null ptr dereference, division by zero, etc.

Ada	C / C++
Strongly typed	Pointer sorcery :(
Exception handling	Undefined behavior
Boundary checks	Up to developer
Name references a historical genius	Name references the 3rd letter of the alphabet??

Language	Unsigned integer	Signed integer
Ada	modulo the type's modulus	`raise Constraint_Error`
C, C++	modulo power of two	undefined behavior

* In C11, unsigned integer overflow is defined to wrap around, while signed integer overflow causes undefined behavior.

ARIANE 4 ARIANE 5

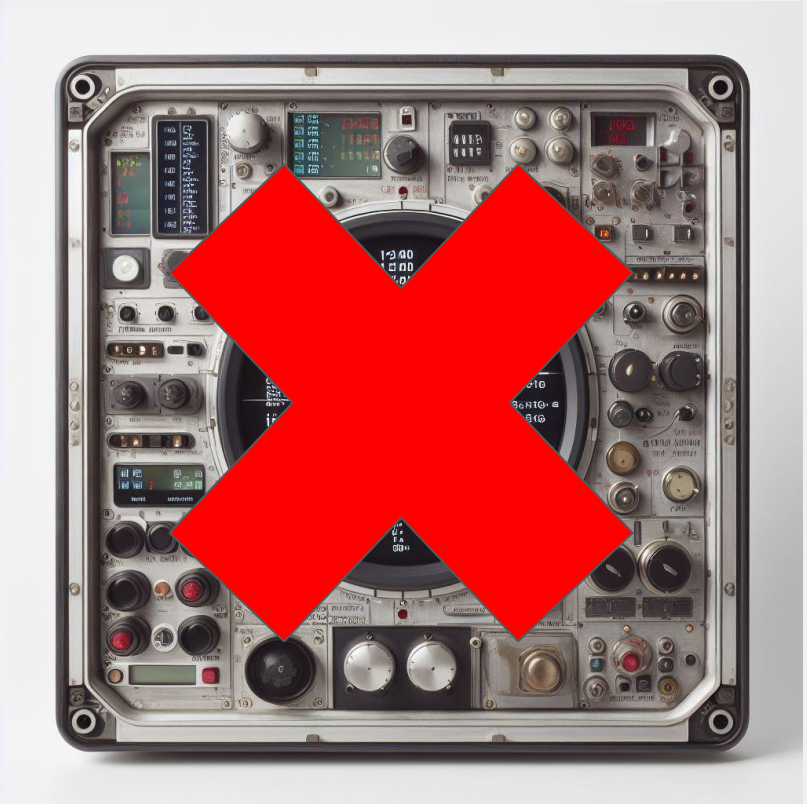
Inertial Reference System

I have detected: Operand_Error

Time for: ✨*Redundancy*✨

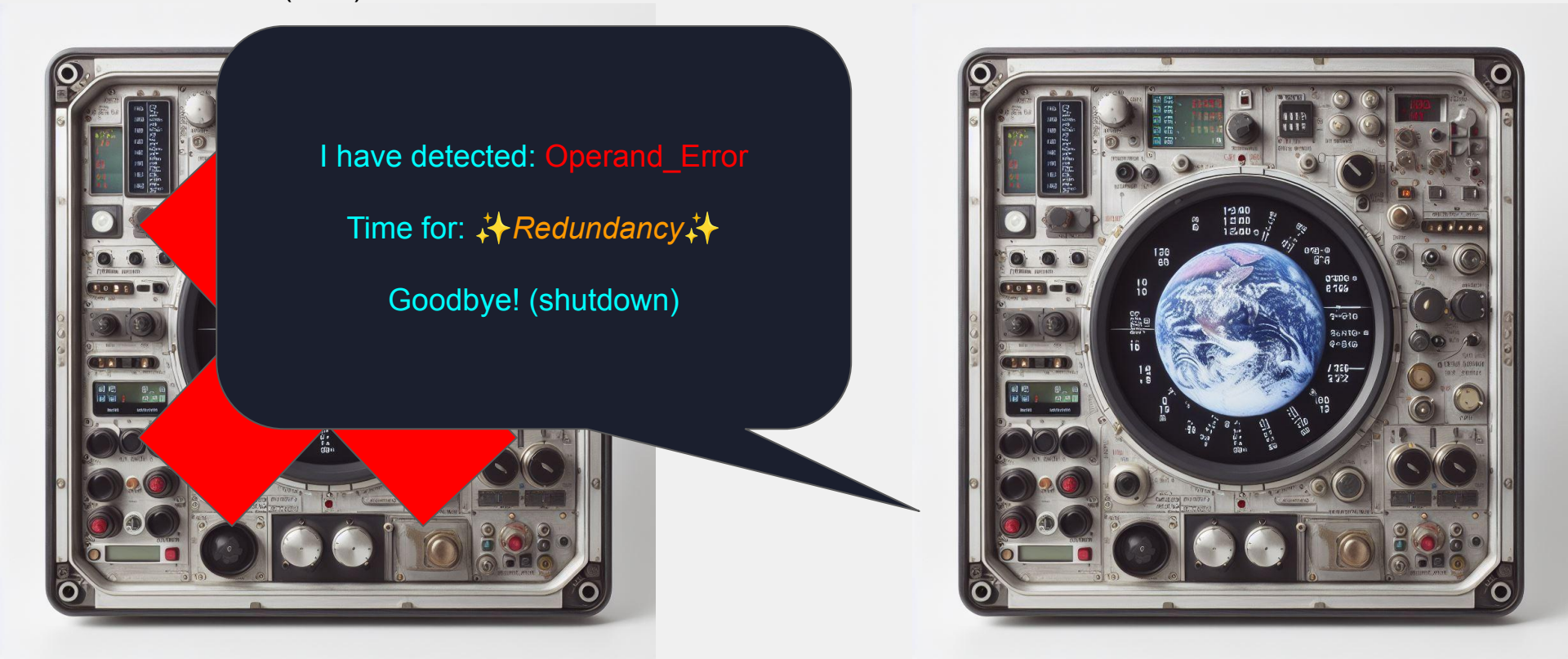
Goodbye! (shutdown)

Inertial Reference System (OFF)



& Inertial Reference System (Backup)

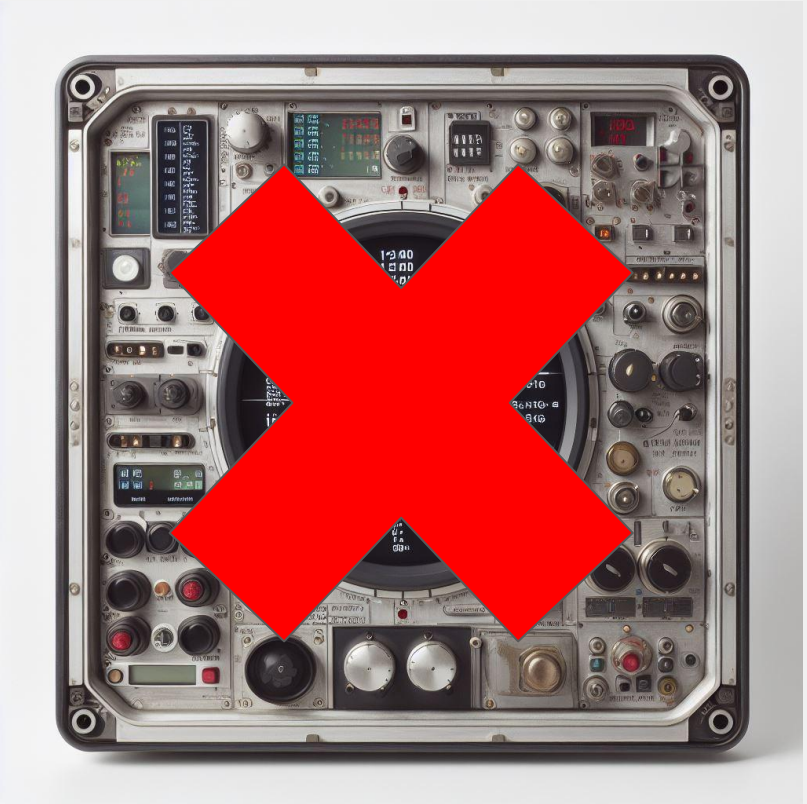
Inertial Reference System (OFF) & Inertial Reference System (Backup)

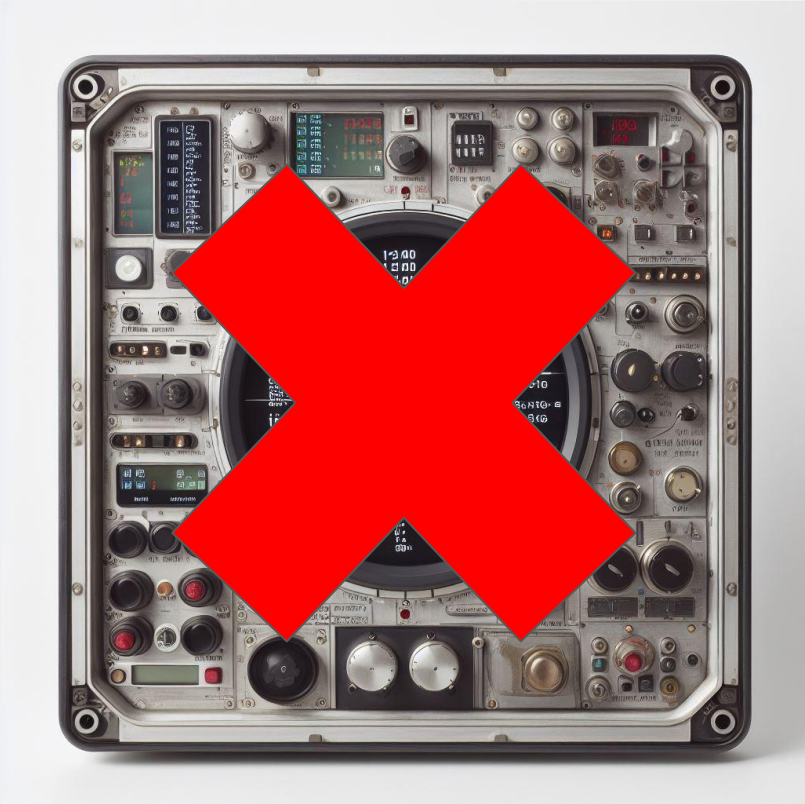


Inertial Reference System & Inertial Reference System

(OFF)

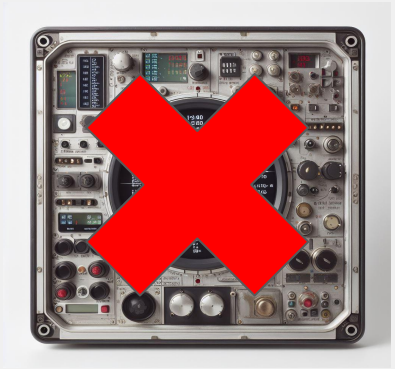
(Also OFF)

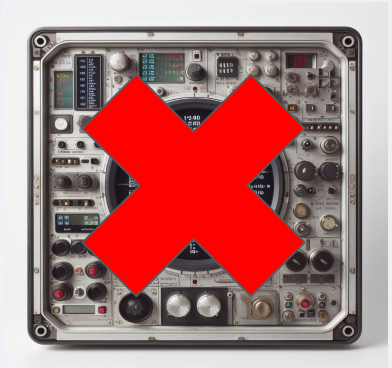




Flight Control System

Raise: Operand_Error





current_speed = Operand_Error

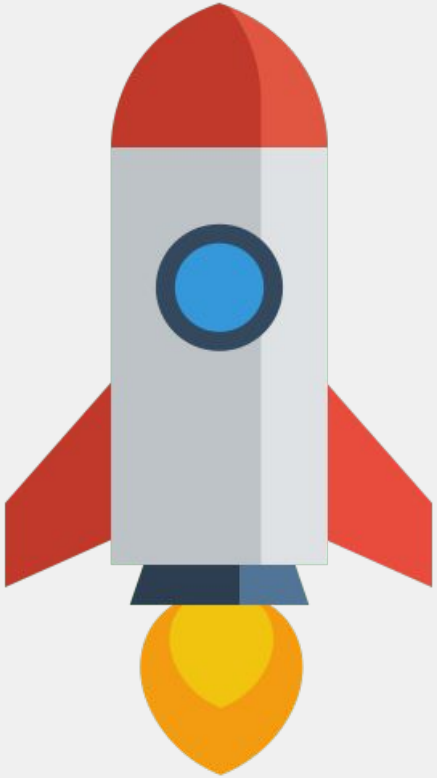
O	p	e	r	a	n	d	_	E	r	r	o	r
79	112	101	114	97	110	100	95	69	114	114	111	114

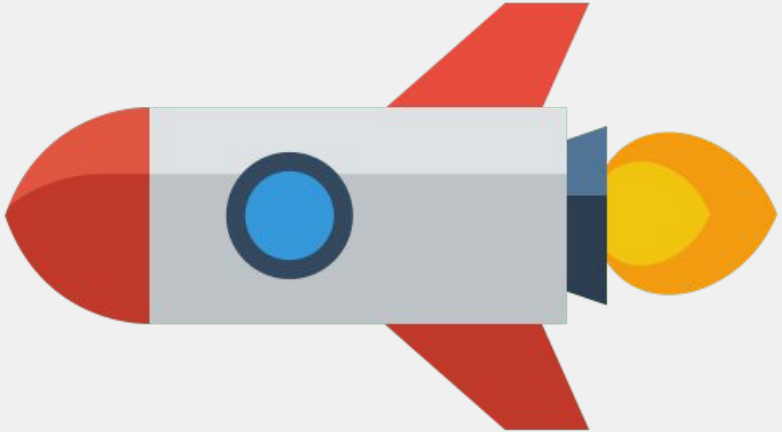
current_speed = 791121011149711010095691141
14111114

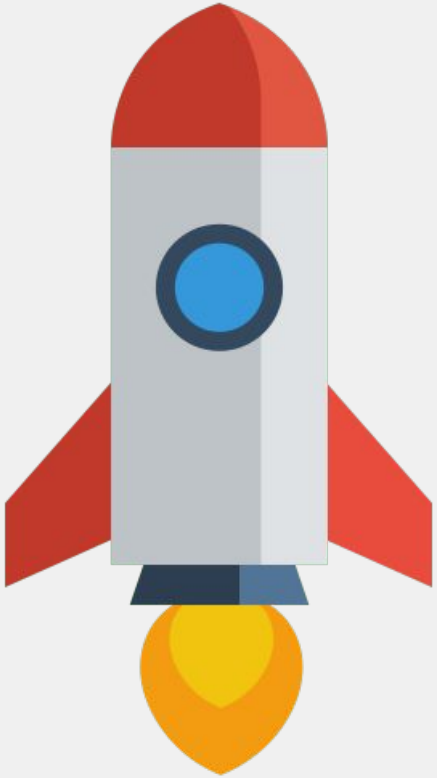
current_speed = 791121011149711010095691141 14111114

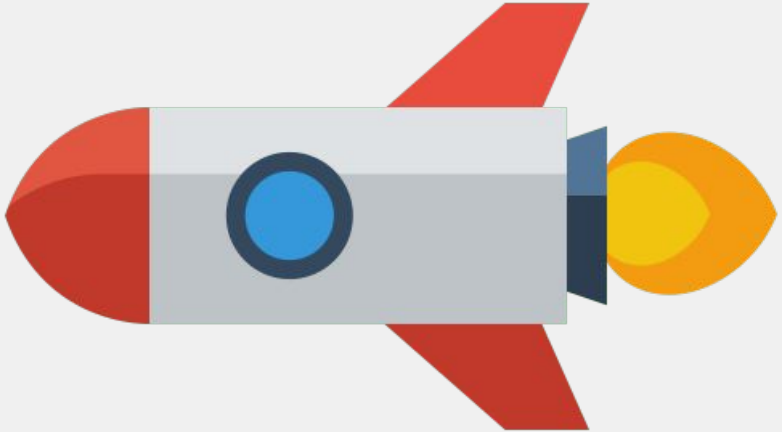
hmmm…
bit fast innit?

Let’s tilt by uhh 90 degrees ish







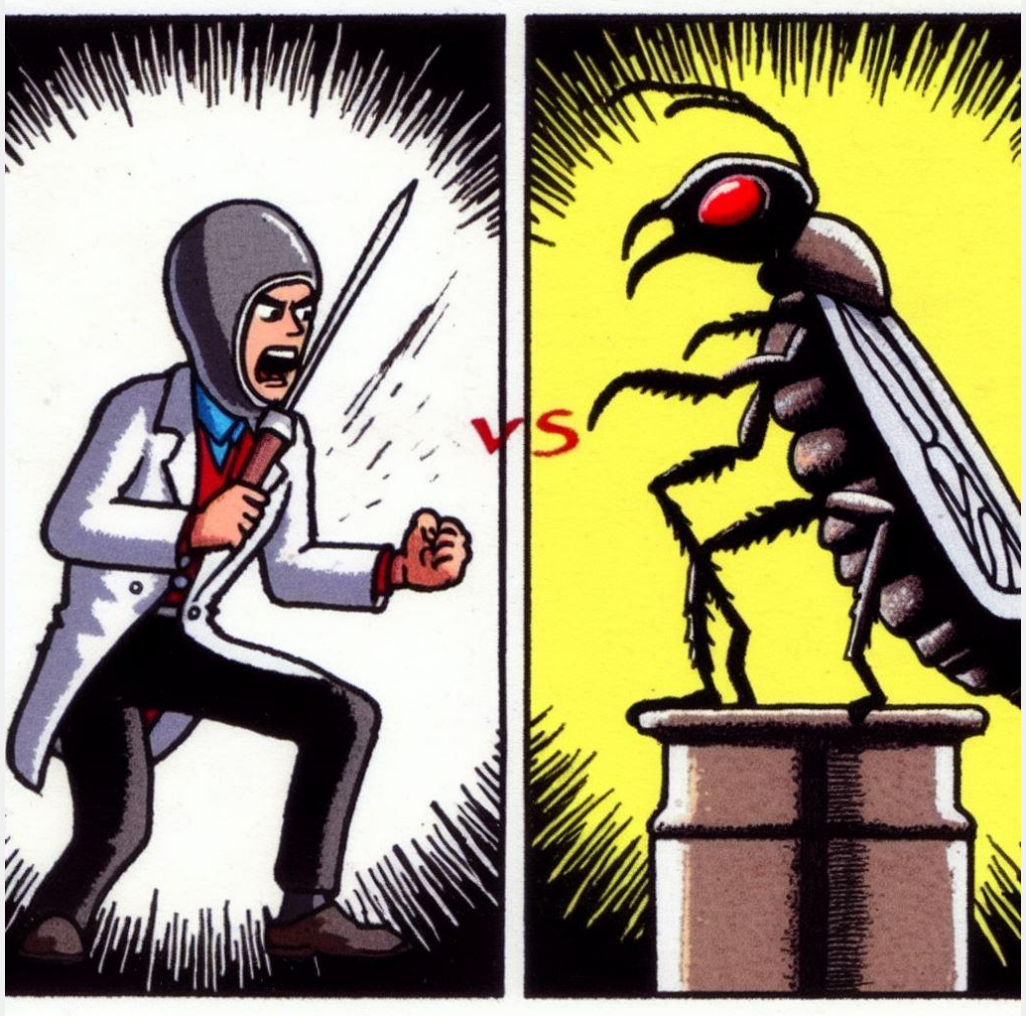


self-destruct program

EU’s smartest
rocket scientists

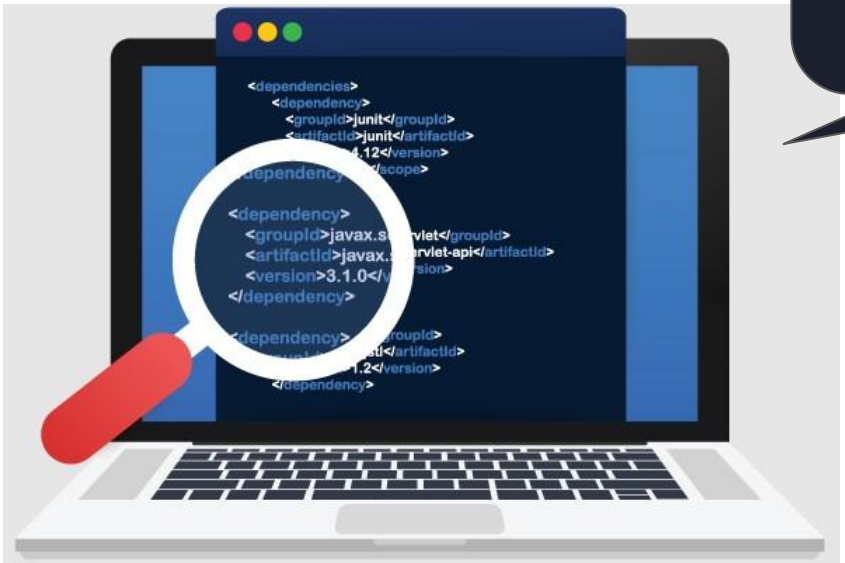
with

$7 Billion in funding



Operand_Error

Static analysis tool



I have found 7 potential buffer or integer overflows in the IRS.


Static
Analysis

⚠️SAST is not a comprehensive security review⚠️

How SAST Works

- All code has vulnerabilities
- SAST analyzes code line by line (instrumentation optional)
- Flags lines with potential issues
- Many different categories of findings
- Empower developers to address findings

Example: Cryptography misconfiguration

```
Cipher.getInstance("AES/ECB/NoPadding");
```

```
Cipher.getInstance("AES/GCM/NoPadding");
```

But it depends on use case!
e.g. should you be hashing instead?

Triaging Security Findings

Send your devs 86,547 untriaged findings

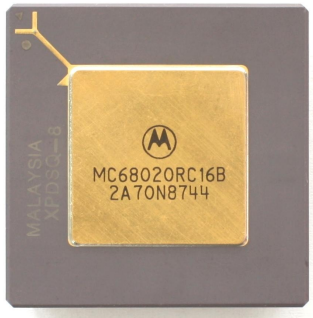
- Devise triage heuristics to deliver meaningful findings
- Tweak tool config
- Combine findings by category & remediation

Tools should be configured to flag what we want to look for!

- Target max CPU util: 80%
- Read docs and comments
- We're writing Ada not C
- Meetings with partners
- Checked assumptions

We have fixed 4 potential buffer/integer overflows in the IRS.

The other 3 will never overflow.



Motorola 68020 (Ariane 5)	**Apple A16 Bionic (iPhone 14)**
Up to **33 MHz**	Up to **3.46 GHz**


STAND BACK EVERYONE!
TESTING IN PRODUCTION

Exception Handling Specification

1. Log the exception
2. Store it in EEPROM
3. SHUT THE HARDWARE DOWN

Design Assumptions

1. Assuming random hardware failures, backup should handle the rest.

2. Unless proven necessary, it was not wise to make changes in software which worked well on Ariane 4.

Inertial Reference System

Calibrates trajectory *before* liftoff

Created & used
on the Ariane 4

Needs it running for 40 seconds.

Moved as-is
onto the Ariane 5

Does not need it to be running at all…

“Just sim it”

- Expensive to test IRS, needs dedicated hardware.
- They built dedicated hardware.
- Injected realistic trajectory details in test.
- **It caught the bug!**

3.2 CAUSE OF THE FAILURE

“The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information **was due to specification and design errors** in the software of the inertial reference system.

The extensive reviews and tests carried out during the Ariane 5 Development Programme **did not include adequate analysis and testing** of the inertial reference system or of the complete flight control system, which could have detected the potential failure.”

https://www-users.cse.umn.edu/~arnold/disasters/ariane5rep.html

“The exception which occurred was not due to random failure but a design error.

The exception was detected, but inappropriately handled because ***the view had been taken that software should be considered correct until it is shown to be at fault.***”

“The Board is in favour of the opposite view, that ***software should be assumed to be faulty until applying the currently accepted best practice methods can demonstrate that it is correct.****”*

“The Board is in favour of the opposite view, that ***software should be assumed to be faulty until applying the currently accepted best practice methods can demonstrate that it is correct.*** *”*

how?

Ariane 5 total launches

- **117 launches** between 1996 and 2023.

- 1 failed launch due to software.

→ **99.14% success rate** for its software.

Ariane 5 containing the James Webb Space Telescope lifting-off from the launch pad

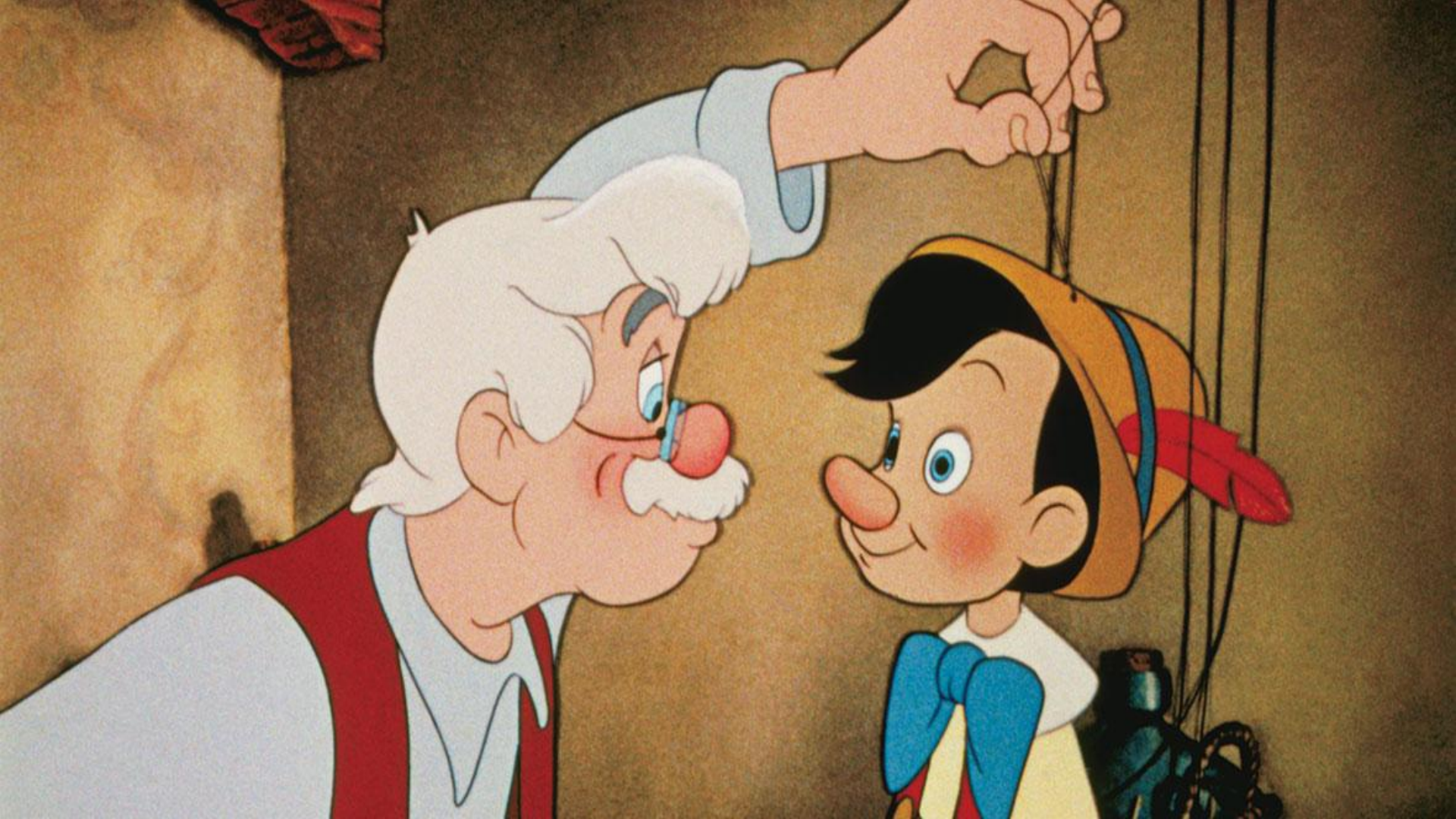

July 9, 2024

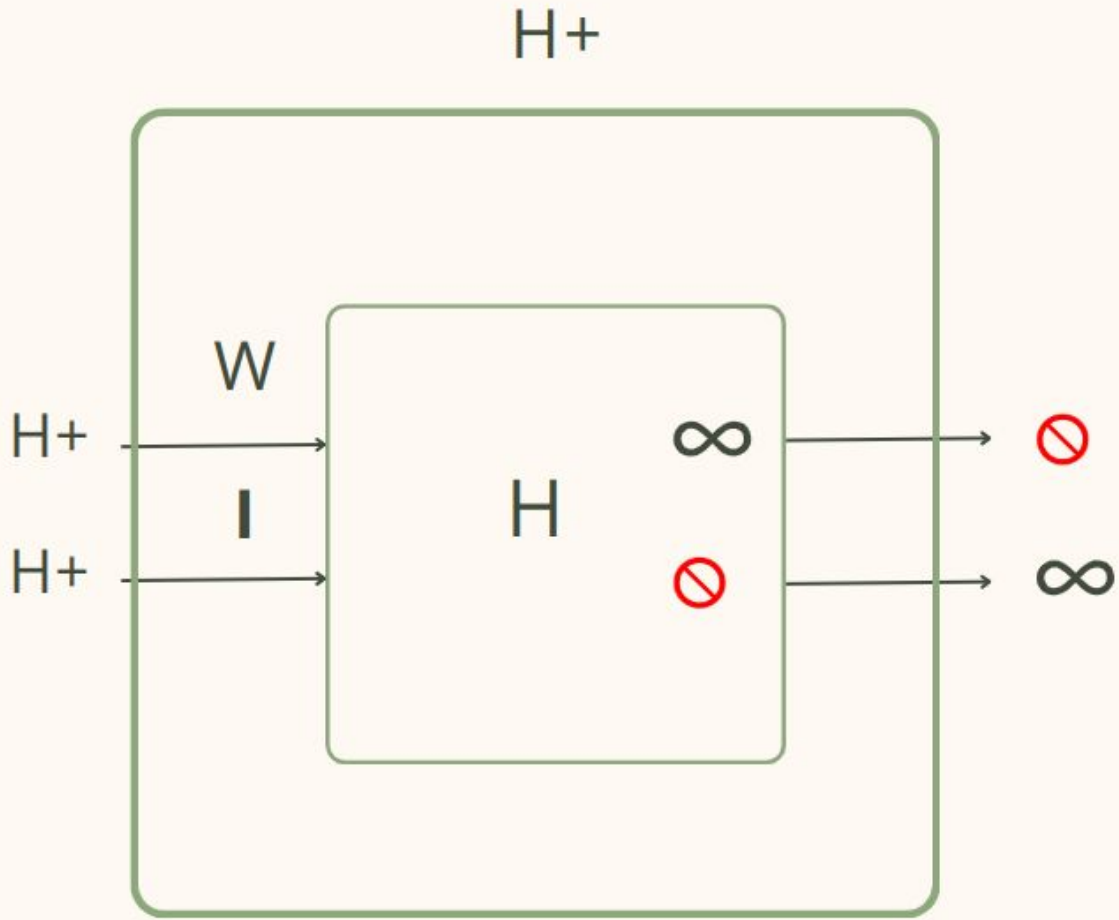
What triggered the failure?

- Wrong design decisions
- Undocumented assumptions
- Software reuse
- Unnecessary legacy code
- Lack of testing
- Mishandled exceptions
- Integer overflow

1931

Gödel's incompleteness theorems




H+
W
H+
I
H+
H
∞
∞

- Does a program halt?

- Is a program secure?

Limits of Formal Verification

Kurt Gödel’s incompleteness theorems:

- *Undecidable* statements: neither provable nor refutable in the system.
- No mathematical system can prove itself free of contradictions.

Turing’s halting problem:

- Determining if any program halts on any input is *undecidable*.

1. ~~Hardware~~
2. ~~Formal Methods~~
3. ~~Memory-Safe Programming Languages~~
4. Check Assumptions

Language	Can I write secure code in it?
C++	yes
Rust	yes
Ada	yes
Python	yes
Ruby	yes
C	yes
Zig	yes
Java	yes
Go	yes
???	yes

Synopsis

- document assumptions
- validate them with tests
- might still explode, its ok