



**Don't trust the Internet;
Verify it!**

Pierre Louis Aublin
2026 年 8 月 4 日

What is Formal Verification?

- **Mathematical proof** that a system **satisfies** its **specification**
- Different flavours
 - Dynamic (testing at runtime)
 - Static
- Different approaches
 - Model checking
 - Deductive verification
 - ...

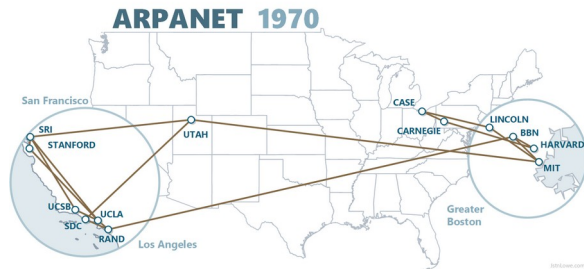
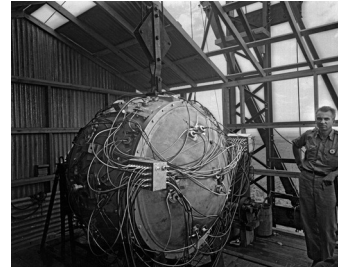
Testing: hope to find bugs

Formal Verification: prove there are no bugs

How we think of Formal Verification



From Specialized Fields to the General Public



Why Verify the Internet

- **Correctness**

- Check for specification compliance and protocol errors

- **Security**

-
-

**Formal Verification Increases
Trust in the Infrastructure**

- **Reliability**

- Avoid bugs or misconfigurations

- **Interoperability**

- Ease standardization and development of various implementations



Why Verify now?

- Protocols become increasingly complex and difficult to maintain
- Attacks increase both in quantity and sophistication

DEV Powered by Algolia

**Prasanna Gautam**
Posted on Dec 26, 2025 • Originally published at [nextdoorhacker.com](#) on Dec 25, 2025

Don't Unwrap in Production: A Formal Verification Guide

[#rust](#) [#testing](#) [#computerscience](#) [tutorial](#)

TL;DR : On November 18, 2025, a single `.unwrap()` call caused a 3-hour global Cloudflare outage. Formal verification tools like Verus could have caught this at compile time—before it reached production. This post shows how Verus's built-in

ICS/OT

Nearly 100 TCP/IP Stack Vulnerabilities Found During 18-Month Research Project

An 18-month research project has resulted in the discovery of nearly 100 vulnerabilities across more than a dozen TCP/IP stacks.



By [Eduard Kovacs](#) | November 11, 2021 (12:59 PM ET)

The **A** Register



CrowdStrike blames a test software bug for that giant global mess it made

Something called 'Content Validator' did not validate the content, and the rest is history

[Simon Sharwood](#)

Wed 24 Jul 2024 · 05:17 UTC

Search blog

[Support](#) [Webinars](#) [Trust](#)

[← Blog Home](#)

The Mythos Inflection Point: Dealing With the Upcoming Vulnerability Disclosure Avalanche and Compressed Exploitation Window



[Shallesh Athalye](#), Senior Vice President, Product Management, Qualys
May 1, 2026 · 10 min read

[Share](#)

Internet Outages vs Formal Verification

AWS, October 2025

- Impact on critical services: DynamoDB, EC2, Lambda...
- Day-long disruption for Amazon, Disney+, Reddit, Signal, Zoom...
- Root cause: DNS race condition
- **Formal Verification** would have prevented this issue when **designing** the system

Cloudflare, November 2025

- ~3h global outage
- CloudFlare handles ~20% of web traffic
- Root cause: panic in Rust code
- **Formal Verification** would have avoided the situation while **writing** the code

Formal Verification for QUIC

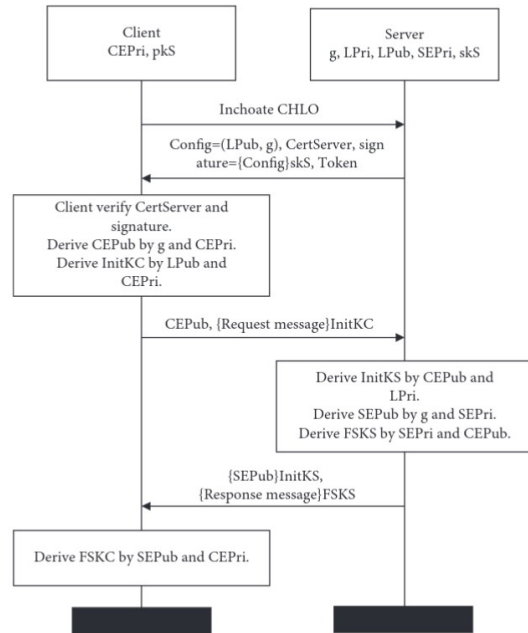


FIGURE 3: The detailed steps of the QUIC handshake process.

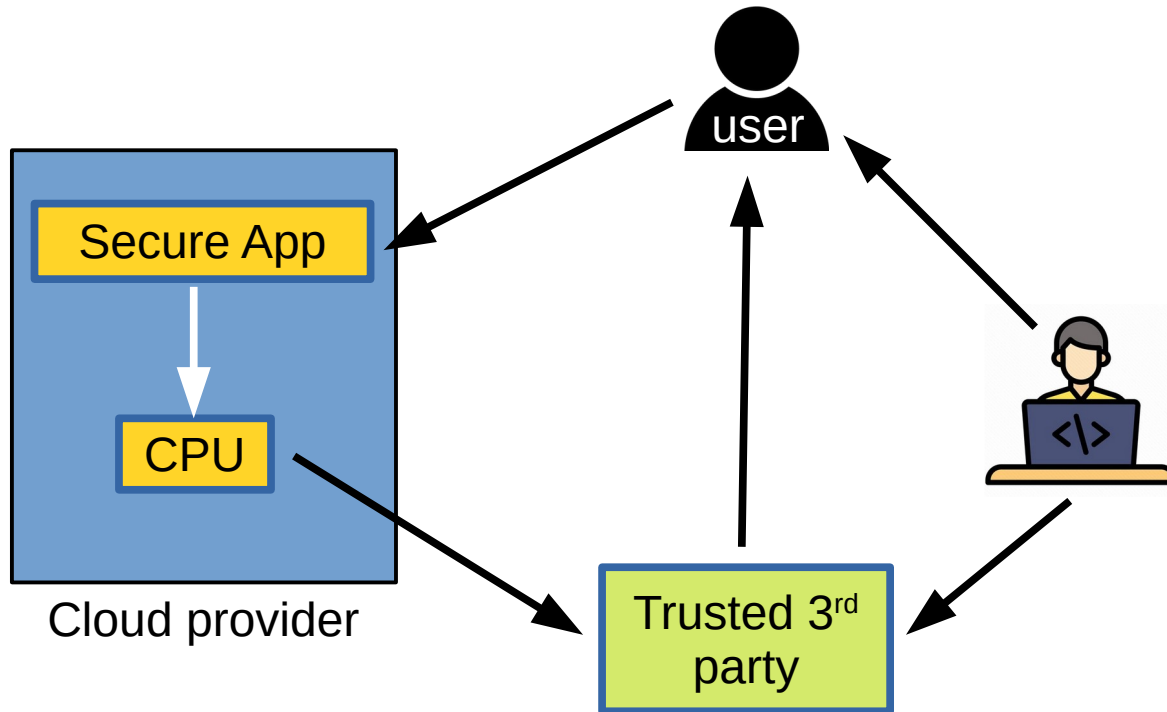
Handshake protocol: Fast but Complex

- What could go wrong?
 - Specification errors
 - Attacks
 - ...

Formal verification proves the protocol is correct:

- “it is not possible that a packet encrypted with an old key will be accepted after a key update”
- “an attacker cannot replay a valid packet to deceive the server

Formal Verification for Trusted Execution



Need to **trust** many participants

- Developer
- Application code & dependencies
- Application image repository
- CPU vendor
- Cloud provider

Formal Verification Tools (not exhaustive)

Target
Language
Proof Engine
Learning Curve
Abstraction Level
Link with code
Use-cases

Formal Verification Tools (not exhaustive)

	TLA+
Target	Design and architecture
Language	Temporal Logic
Proof Engine	Model Checking
Learning Curve	Medium
Abstraction Level	High
Link with code	None
Use-cases	Distributed Protocols

Formal Verification Tools (not exhaustive)

	TLA+	Verus
Target	Design and architecture	Implementation
Language	Temporal Logic	Rust w/ Annotations
Proof Engine	Model Checking	SMT solver
Learning Curve	Medium	Medium
Abstraction Level	High	Medium
Link with code	None	Direct
Use-cases	Distributed Protocols	Systems code

Formal Verification Tools (not exhaustive)

	TLA+	Verus	Lean
Target	Design and architecture	Implementation	Mathematical Proof
Language	Temporal Logic	Rust w/ Annotations	Dedicated Proof Language
Proof Engine	Model Checking	SMT solver	Proof Assistant
Learning Curve	Medium	Medium	High
Abstraction Level	High	Medium	Low
Link with code	None	Direct	Separated or Translation (e.g., Aeneas)
Use-cases	Distributed Protocols	Systems code	Cryptographic proofs

Formal Verification Tools (not exhaustive)



	Design Structure	Infrastructure	Mathematical Proof
Target	Design Structure	Infrastructure	Mathematical Proof
Language	Rust	Rust	Dedicated Proof Language
Proof Engine	Model Checking	Creusot	Proof Assistant
Learning Curve	Medium	Medium	High
Abstraction Level	High	Medium	Low
Link with code	None	Direct	Separated or Translation (e.g., Aeneas)
Use-cases	Distributed Protocols	Systems code	Cryptographic proofs

Formal Verification Examples

min() function in Verus

```
use builtin::*;

verus! {

fn min(x: i32, y: i32) -> (m: i32)
  ensures
    m == x || m == y,
    m <= y,
    m <= x,
  {
    if x <= y {
      y
    } else {
      x
    }
  }

} // verus!

example.rs [rust] utf-8 84% 21:0
```

```
$ ./verus example.rs
note: verifying root module

error: postcondition not satisfied
--> example.rs:12:1
11 |         m <= x,
    |         ----- failed this postcondition
12 | / {
13 | |   if x <= y {
14 | |       y
15 | |   } else {
16 | |       x
17 | |   }
18 | | }
    | |_^ at the end of the function body

error: aborting due to previous error

verification results:: verified: 0 errors: 1
$
```

TLA+ Modeling of AWS outage DNS race condition

- November 05, 2025

```
4  CONSTANTS
5  ENACTORS,           \* set of concurrent Enactors
6  PLAN_AGE_THRESHOLD, \* number of plans after which a plan can be deleted
7  MAX_PLAN,           \* maximum number of plans (bounded model checking)
8
9  VARIABLES
10 current_plan,       \* currently active Route53 plan
11 latest_plan,        \* last plan generated by the Planner
12 highest_plan_applied, \* highest version ever applied
13 plan_deleted,       \* set of deleted plans
14 plan_channel,        \* queue of plans awaiting enactment
15 enactor_processing,  \* which plan each Enactor is processing
16 enactor_pc,         \* Enactor program counter
17
18 (*-----*)
19 (* Initial state *)
20 (*-----*)
21 Init ==
22   /\ current_plan = 0
23   /\ latest_plan = 0
24   /\ plan_deleted = {}
25   /\ plan_channel = []
26   /\ enactor_processing = {}
27   /\ enactor_pc = {}
28   /\ enactor_processing <= [e \in ENACTORS |> 0]
29   /\ enactor_pc <= [e \in ENACTORS |> 0]
30
31 (*-----*)
32 (* Planner behavior *)
33 (*-----*)
34 PlannerStep ==
35   /\ latest_plan < MAX_PLAN
36   /\ latest_plan = latest_plan + 1
37   /\ plan_channel = Append(plan_channel, latest_plan + 1)
38   /\ UNCHANGED <<current_plan, highest_plan_applied, plan_deleted,
39   /\ UNCHANGED enactor_processing, enactor_pc>>
40
41 (*-----*)
42 (* Enactor behavior *)
43 (*-----*)
44 EnactorReceiveStep(self) ==
45   /\ plan_channel < <<
46   /\ enactor_pc[self] = 0
47   /\ enactor_processing =
48     [enactor_processing EXCEPT ![self] = Head(plan_channel)]
49   /\ enactor_pc[self] =
50     [enactor_pc EXCEPT ![self] = 1]
51   /\ plan_channel = Tail(plan_channel)
52   /\ UNCHANGED <<current_plan, latest_plan,
53   /\ UNCHANGED highest_plan_applied, plan_deleted>>
54
55 (*-----*)
56 (* Enactor processing *)
57 (*-----*)
58 EnactorProcessingStep(self) ==
59   /\ enactor_processing[self] = Head(plan_channel)
60   /\ enactor_processing[self] = Tail(plan_channel)
61   /\ enactor_processing[self] = Tail(plan_channel)
62   /\ enactor_processing[self] = Tail(plan_channel)
63   /\ enactor_processing[self] = Tail(plan_channel)
64   /\ enactor_processing[self] = Tail(plan_channel)
65   /\ enactor_processing[self] = Tail(plan_channel)
66   /\ enactor_processing[self] = Tail(plan_channel)
67   /\ enactor_processing[self] = Tail(plan_channel)
68   /\ enactor_processing[self] = Tail(plan_channel)
69   /\ enactor_processing[self] = Tail(plan_channel)
70   /\ enactor_processing[self] = Tail(plan_channel)
71   /\ enactor_processing[self] = Tail(plan_channel)
72   /\ enactor_processing[self] = Tail(plan_channel)
73   /\ enactor_processing[self] = Tail(plan_channel)
74   /\ enactor_processing[self] = Tail(plan_channel)
75   /\ enactor_processing[self] = Tail(plan_channel)
76   /\ enactor_processing[self] = Tail(plan_channel)
77   /\ enactor_processing[self] = Tail(plan_channel)
78   /\ enactor_processing[self] = Tail(plan_channel)
79   /\ enactor_processing[self] = Tail(plan_channel)
80   /\ enactor_processing[self] = Tail(plan_channel)
81   /\ enactor_processing[self] = Tail(plan_channel)
82   /\ enactor_processing[self] = Tail(plan_channel)
83   /\ enactor_processing[self] = Tail(plan_channel)
84   /\ enactor_processing[self] = Tail(plan_channel)
85   /\ enactor_processing[self] = Tail(plan_channel)
86   /\ enactor_processing[self] = Tail(plan_channel)
87   /\ enactor_processing[self] = Tail(plan_channel)
88   /\ enactor_processing[self] = Tail(plan_channel)
89   /\ enactor_processing[self] = Tail(plan_channel)
90   /\ enactor_processing[self] = Tail(plan_channel)
91   /\ enactor_processing[self] = Tail(plan_channel)
92   /\ enactor_processing[self] = Tail(plan_channel)
93   /\ enactor_processing[self] = Tail(plan_channel)
94   /\ enactor_processing[self] = Tail(plan_channel)
95   /\ enactor_processing[self] = Tail(plan_channel)
96   /\ enactor_processing[self] = Tail(plan_channel)
97   /\ enactor_processing[self] = Tail(plan_channel)
98   /\ enactor_processing[self] = Tail(plan_channel)
99   /\ enactor_processing[self] = Tail(plan_channel)
100  /\ enactor_processing[self] = Tail(plan_channel)
```

Formal Verification: Not Difficult

TLA+ is not hard to learn. [Amazon engineers report](#) that:

"Engineers from entry level to principal have been able to learn TLA+ from scratch and get useful results in two to three weeks, in some cases in their personal time on weekends and evenings, without further help or training."

THOUGHT LEADERSHIP

Learn Formal the Easy Way

August 31, 2021 • 3 MIN READ

June 5, 2025

AI is a gamechanger for TLA+ users

There has never been a better time to learn formal specification.

TLA+ By Example

Learn TLA+ specifications through interactive examples. Write specs and run the TLC model checker directly in your browser.

TLA+ Made Simple with ChatGPT

Sep 24, 2023

Conclusion: Please verify your systems

- Internet is too important and complex for testing only
- Formal Verification is now more accessible than ever
- Make the Internet safe and robust
 - **For you: less headaches** during **development**
 - **For everyone: more security**, more **trust**

The verification has already started!

TECHNOLOGY

Signal's protocol gets glowing reviews in first security audit

Signal is widely considered the gold standard of secure encrypted messaging apps but it hasn't been subject to a fine-toothed audit. What did researchers have to say?

BY PATRICK HOWELL O'NEILL • NOVEMBER 8, 2016

Revolutionizing Network Protocol Testing: A New Era of Validation

"Towards verification of QUIC and its extensions"



EINIak

Follow

10 min read · Jan 26, 2024



re:Invent

Discover AWS

Products

Solutions

AWS Cloud Security

Security Services

Use Cases

Continuous Formal Verification of Amazon s2n

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...

Internet Security Research Group -- Memory Safety project



Initiatives ▾ Blog ▾ About ▾

Support this Work

Memory Safety
for the Internet's most critical infrastructure

What is Memory Safety?

How We Work

INITIATIVES

TLS (Rustls)

Let's get the Rustls TLS library ready to replace OpenSSL in as many projects as possible.

View Initiative

DNS (Hickory)

Let's create a memory safe, high performance, fully recursive DNS resolver.

View Initiative

sudo/su (sudo-rs)

Let's make the utilities that mediate privileges safer.

View Initiative

NTP (ntpd-rs)

Let's create a memory safe NTP implementation.

View Initiative

AV1 (rav1d)

Let's create a memory safe AV1 decoder that delivers great performance.

View Initiative



sudo
su

NTP



FUNDERS

Google

aws

Flyio

FUTUREWEI

CISCO

Open Networking Foundation

Sovereign TechFund

Alpha-Omega

Open Networking Foundation

Changguard

CLOUDFLARE

shopify

FROM OUR BLOG

July 16, 2025

sudo-rs Headed to Ubuntu

A security tool incubated by Prossimo takes on a big role.

June 5, 2025

Compatibility with C is Key for Memory Safe Software

We're in the beginning phases of a journey towards memory safety for the Internet's critical software infrastructure, and as we get going it makes the most sense to break down big problems into smaller ones by focusing on replacing components within existing C and C++ software.

May 14, 2025

AV1: AAV and AV1 Reader Reference