Proof Engineering for Secure Systems

Greg Morrisett

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

All too familiar headlines…

September 26, 2014

19 million Windows PCs still vulnerable to Stuxnet zero-day

SECURITY security, encryption, heartbleed

Devastating 'Heartbl
was unknown befor
disclosure, study fir

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

SHELLSHOCK-LIKE WEAKNESS MAY AFFECT WINDOWS

In Cyberattack on Saudi Firm, U.S.
Back

Security researcher says many of his iOS 'backdoor'
vulnerabilit

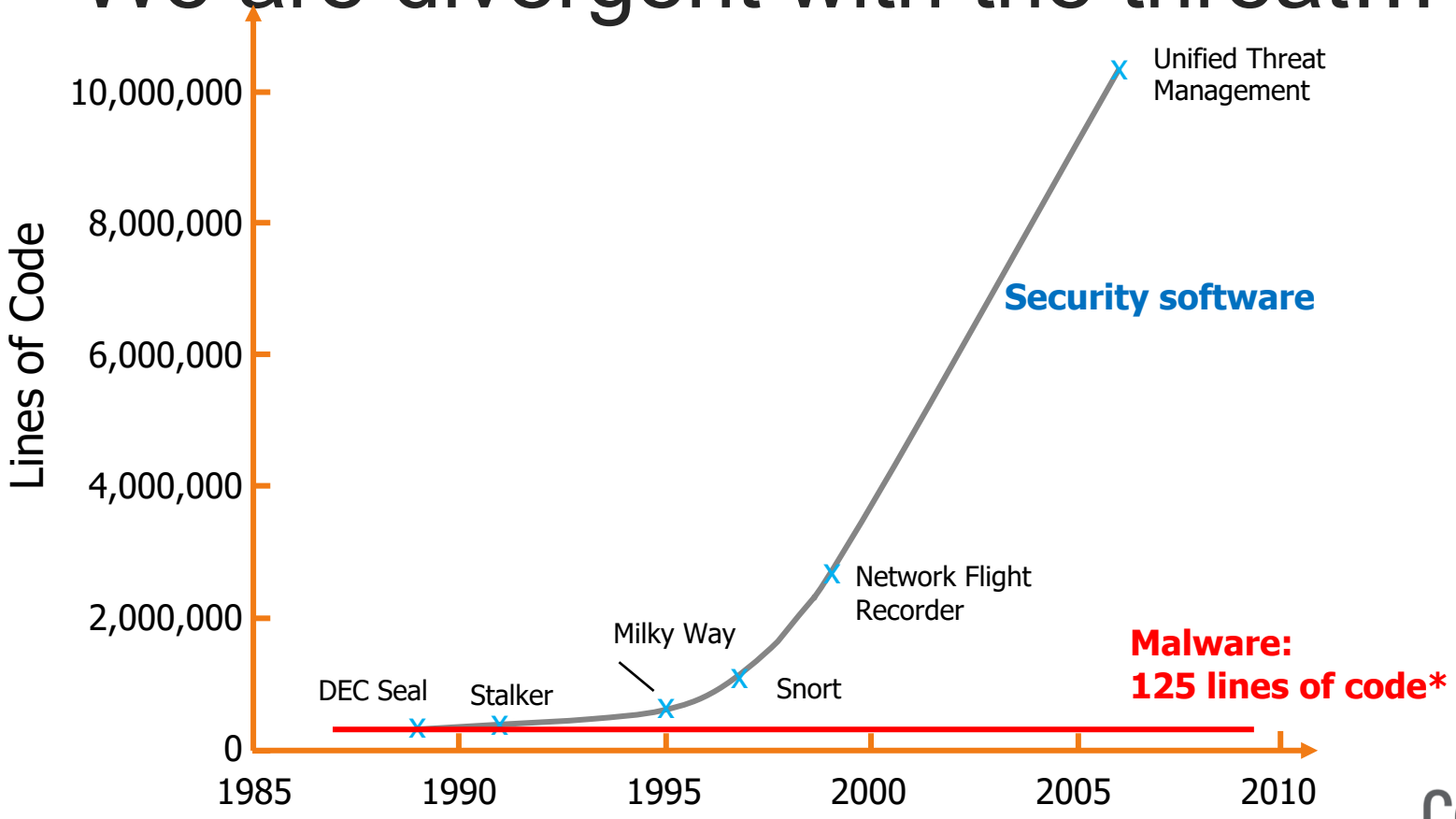
A Hospital Paralyzed by

The Stuxnet Attack Or
Was 'Far More Danger
Thought

MICHAEL B KELLEY

Bad news: A Spectre-like flaw will probably happen again

We are divergent with the threat…



* Malware lines of code averaged over 9,000 samples

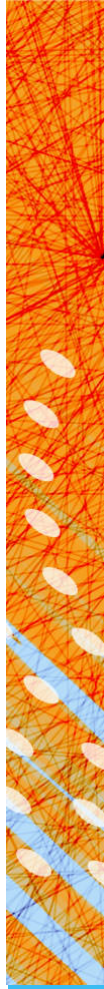
Cornell CIS

Many Things Need to be Fixed

- User interfaces (and users)
- Underlying Architecture
- Underlying Protocols
- Configuration & Operation tools

But one huge issue dominates right now:

The code we depend upon is full of bugs.



A particularly bad bug:

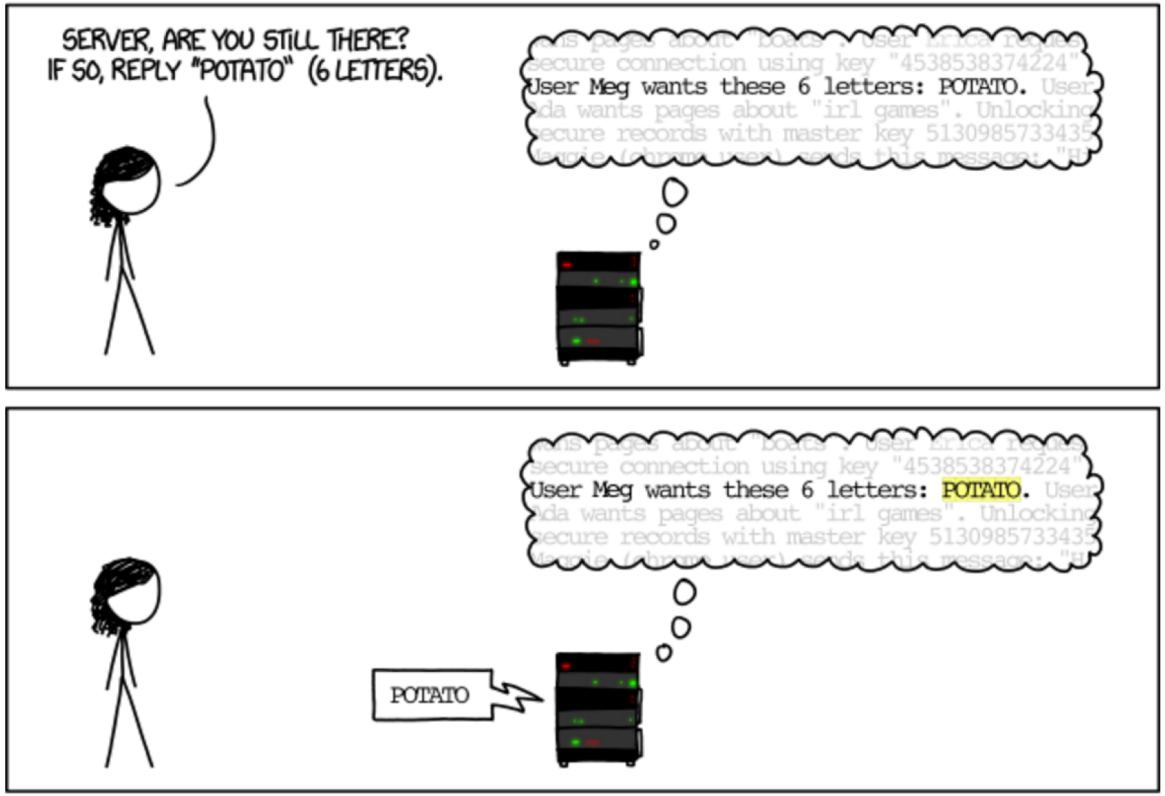
The Heartbleed Bug

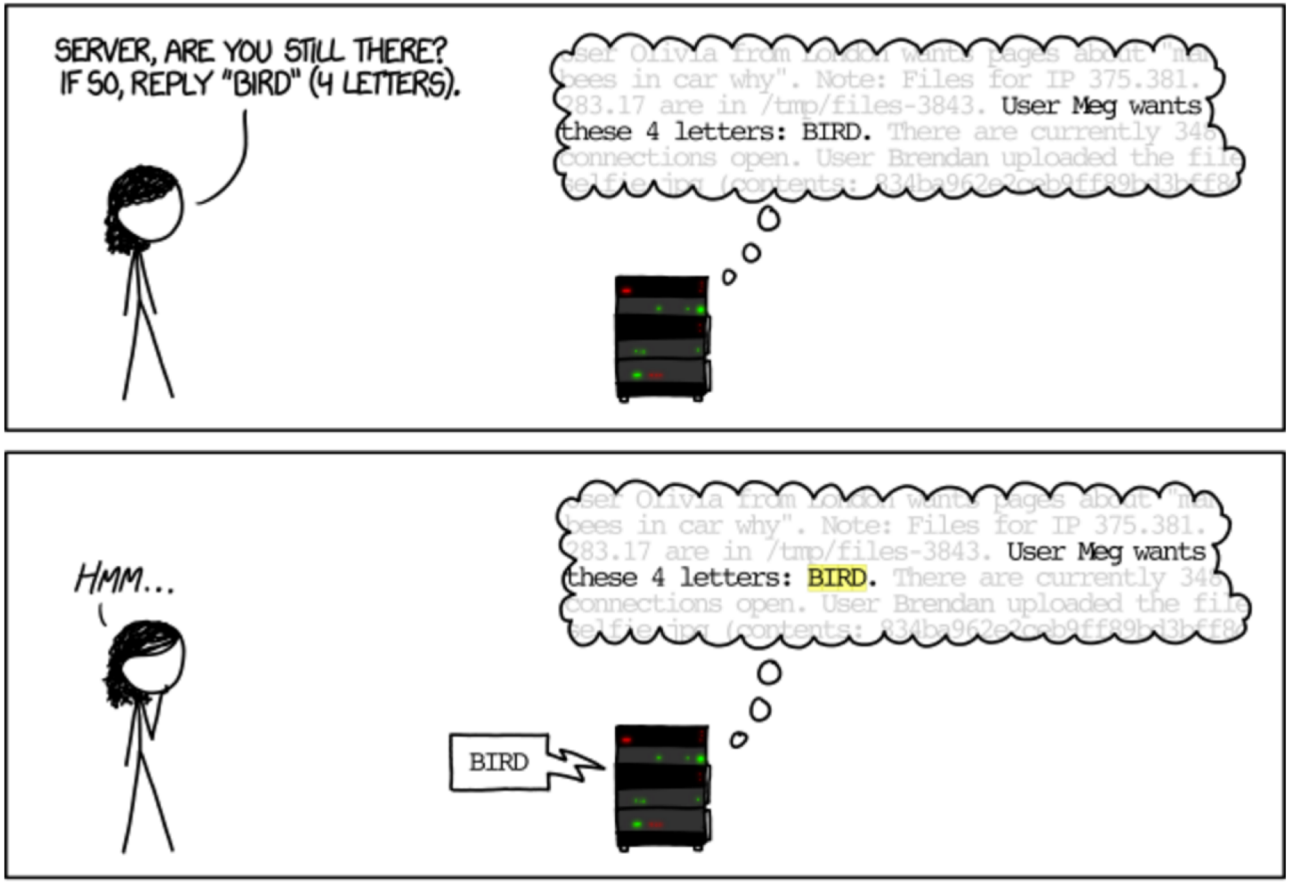
The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet.

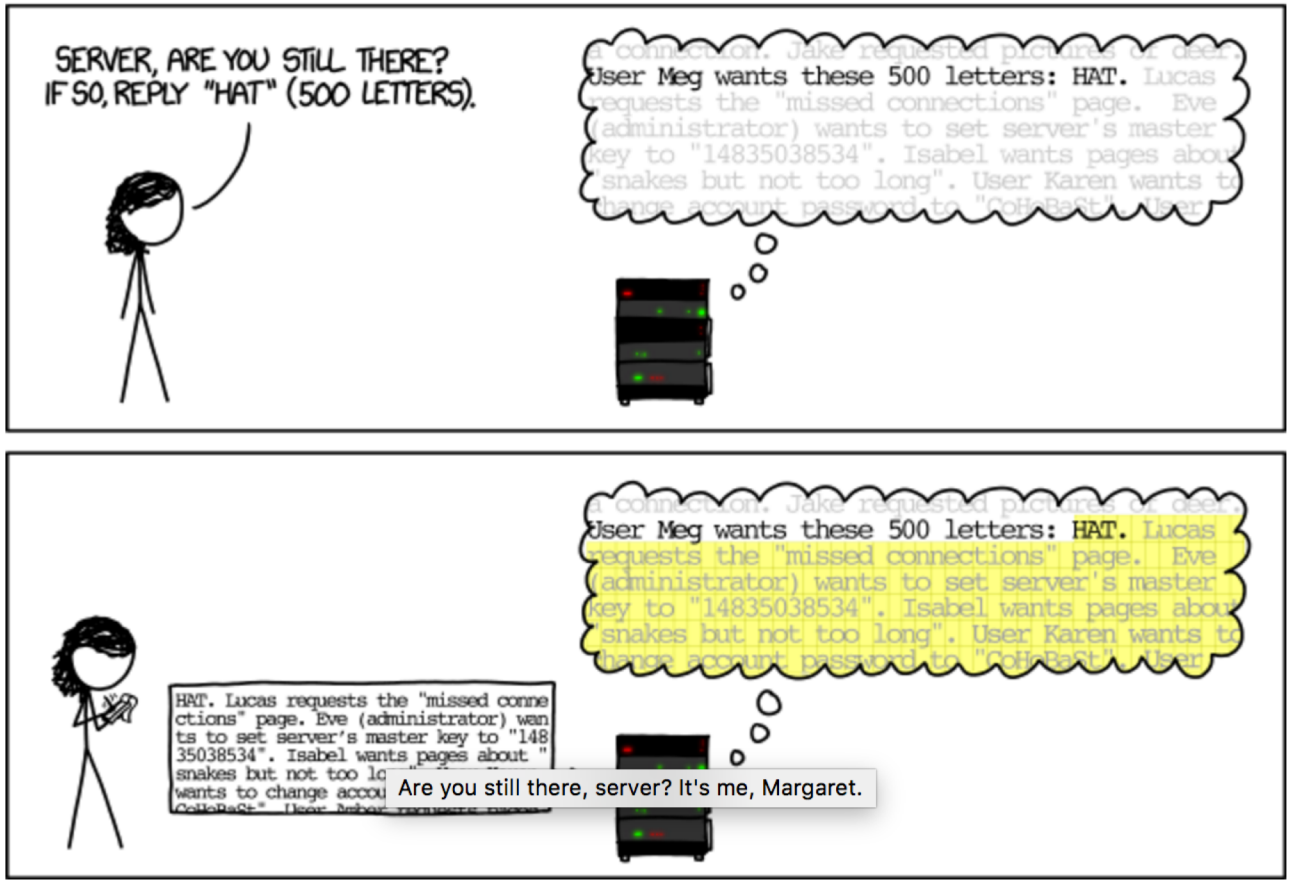
"Catastrophic" is the right word. On the scale of 1 to 10, this is an 11.

- Bruce Schneier

HOW THE HEARTBLEED BUG WORKS:




SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
983.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
HMM...
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
983.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
BIRD


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT.
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
HAT. Lucas requests the "missed conne ctions" page. Eve (administrator) wan ts to set server's master key to "148 35038534". Isabel wants pages about " snakes but not too lo wants to change accou
Are you still there, server? It's me, Margaret.

What's going wrong?

Development processes are ineffective.

- Human code review doesn't work.

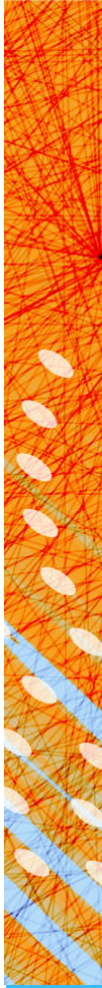
Certification processes are ineffective.

- Based on who authored, not the code itself.

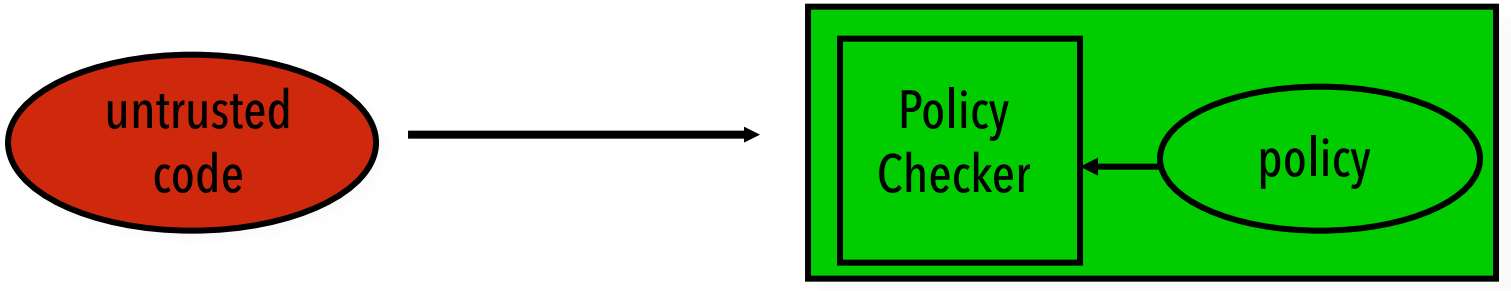
Current automated defenses are worse than ineffective.

- Based on *syntax* or *provenance,* not semantics.
- Introduce new classes of vulnerabilities.

So how do we dig ourselves out of this mess?



Ideal Architecture



Policies capture intended *behavior.*

The checker automatically rules out any code that will violate the policy.

The checker is small, simple, trustworthy, and automatic.

Unfortunately…

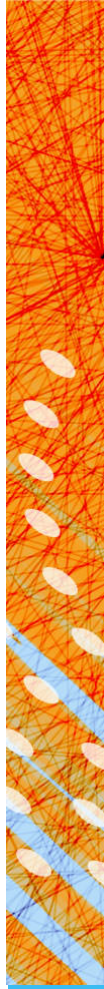
Even simple policies are undecidable.

- e.g., Does the code have a buffer overflow?
- So any checker is either incomplete or unsound.

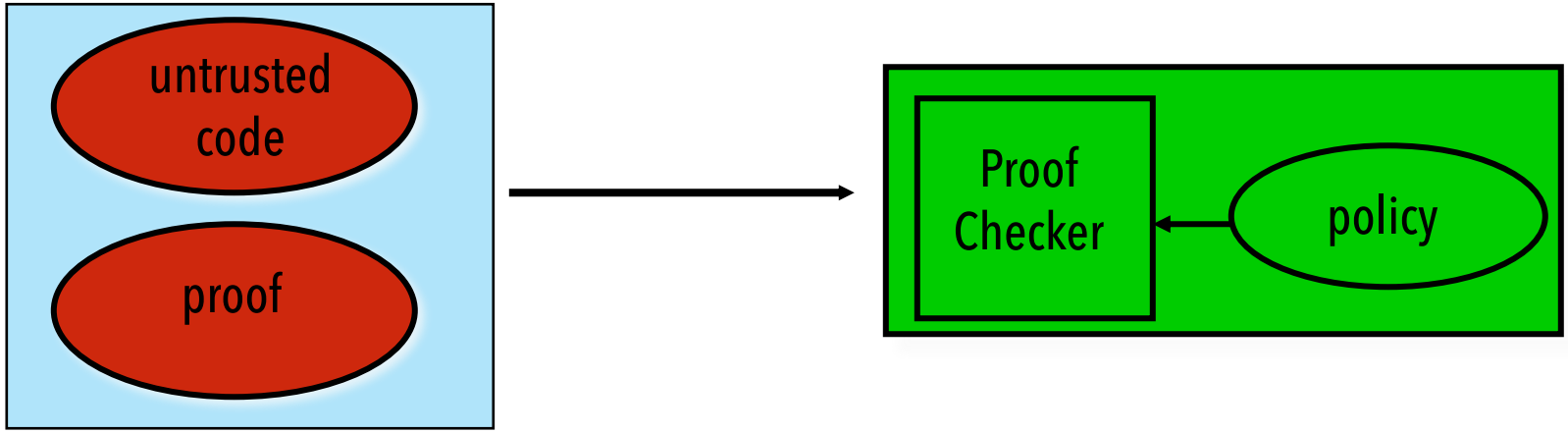
Analyzing machine code is *hard*.

- It's hard enough to analyze real source code for simple policies.
- Any machine-level analysis requires a big, complicated checker.

One idea: *shift the burden.*



Proof-Carrying Code [Necula & Lee]



Code comes with a *proof* that it satisfies the policy.

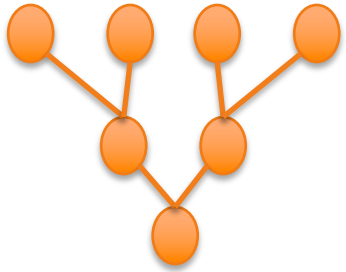
The proof checker ensures that:

a) the proof is valid

b) the conclusion says "this code respects the policy"

What is a (CS) proof?

It's a data structure (tree).



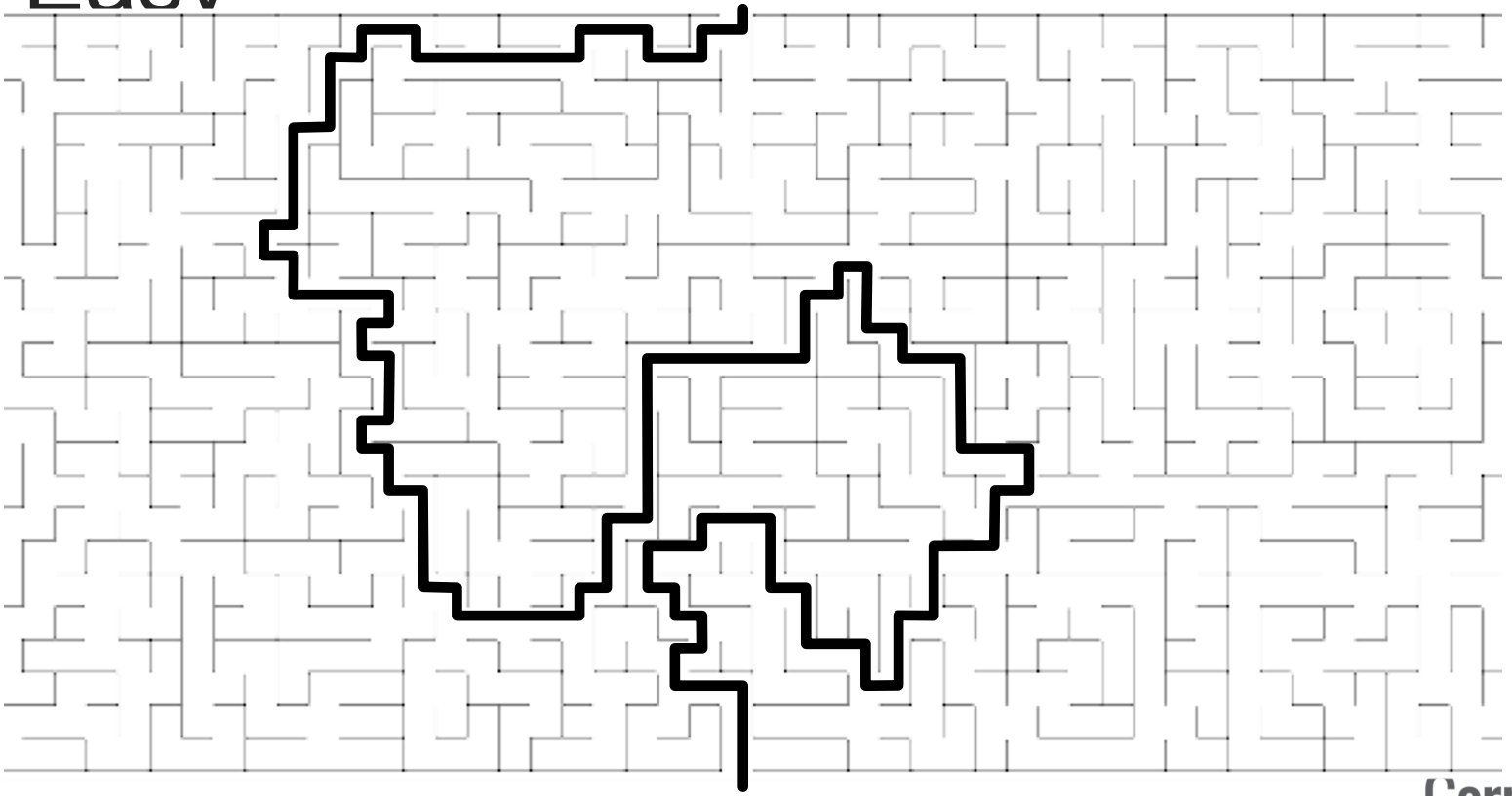
Each node is an instance of a rule:

$$\frac{x = y \qquad y = z}{x = z}\ \text{(transitivity)}$$

Where a rule has some assumptions, and a conclusion.

We stack these proofs together so that the assumptions of the parent correspond to the conclusions of the child.

Finding Proofs is Hard; Checking is Easy

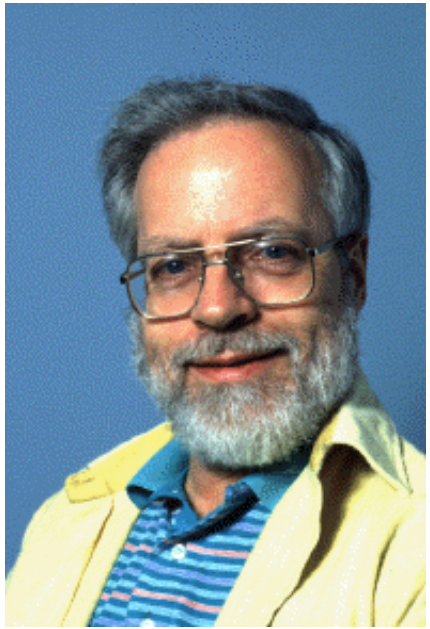


Proof Engineering

- We want proofs grounded in minima assumptions (e.g., ZFC).
- But then resulting proofs are HUGE.
- The burden shifts to the code producer to construct and maintain these enormous objects.
- Proof *languages* that support modularity, sharing, and re-use are of vital importance.

Not a new idea…

Bob Constable



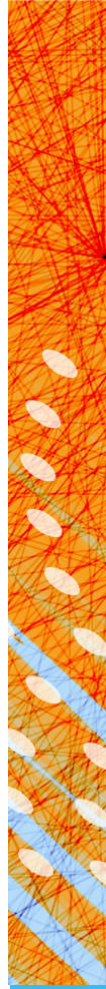
David Gries

And not the only domain…

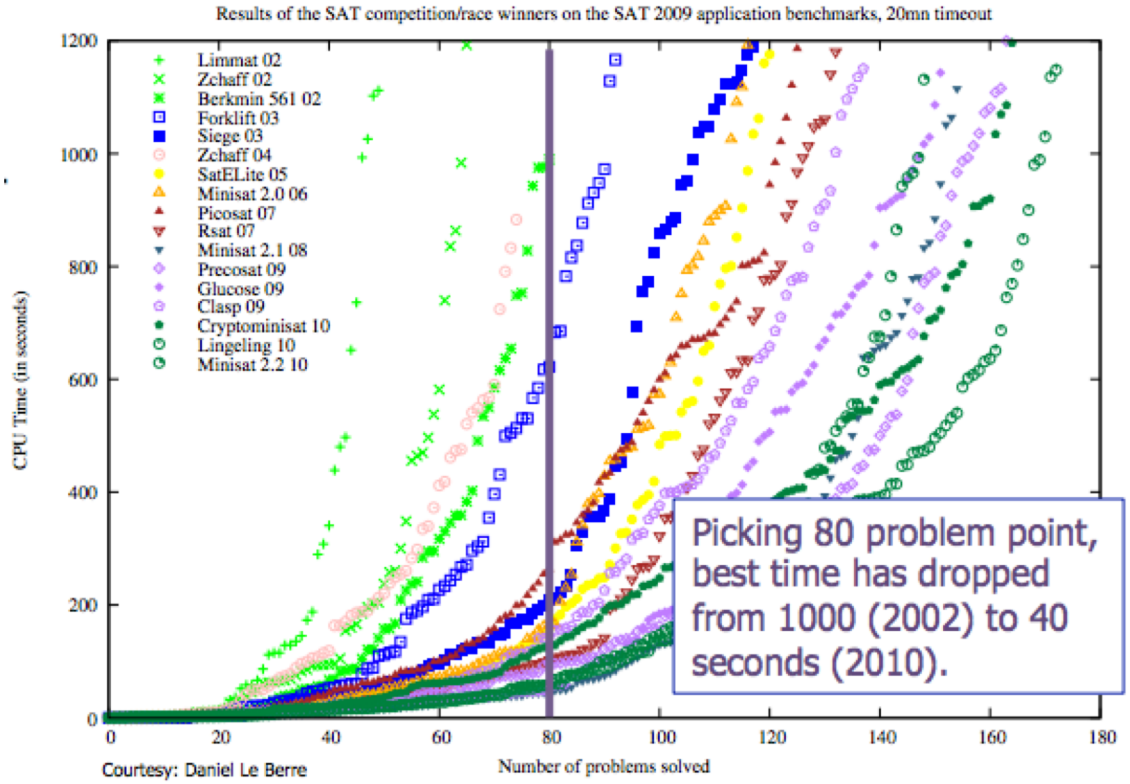
Vladimir Voevodksy

What has changed?

- Foremost: The need!
- Languages, environments, and logics for constructing proofs have matured.
  - Coq, Isabelle, ACL2, Agda, Lean, PVS, NuPRL, …
  - Books such as *Software Foundations*.
- Tools that automate constructing proofs.
  - SAT & SMT solvers (e.g., Z3)



Example: SAT solvers



Many new academic projects

- Compilers: CompCert, CertiCoq, …
- Operating Systems: SeL4, Verve, CerticOS, …
- Web browsers: Quark
- Databases & File Systems: YnotDB, FSCQ, ...
- Hardware: Kami
- Cryptography: FCF, CertiCrypt, Everest, …

An Example Success Story

DARPA's high assurance drones (HACMS)

HACMS: 18-month program

- Clean slate software stack
  - Stability control, altitude hold, direction hold, DOS detection & response
  - GPS waypoint navigation (80%)
- Proved system-wide properties
  - System is memory safe
  - System ignores mal-formed messages
  - System ignores non-authenticated messages
  - All “good” messages will reach the controller

Evaluation

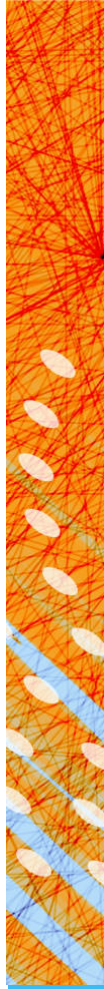
A red team was given full access to the source code for six weeks and told to break it.

They weren't able to.

Penetration expert:
"The most secure UAV on the planet."

Still many hard challenges

- The kernel (SeL4) is simple and yet took 20 person years to prove correct.
- The policies have to be right.
- Models of the environment have to be faithful.
- Need architectures to handle legacy code.



To Summarize

Mechanizing proofs of correctness, security, etc. is a viable way to support open source development without needing the same (misplaced) trust that we have today.

The tools are rapidly coming together to reason about real code executing on real systems.