Formal Proofs, the Fine Print and Side Effects

Toby Murray and Paul C. van Oorschot
Formal Proofs: Successes

- Formally verified OS microkernel
  - Project Everest

- Formally verified crypto libraries
Formal Proofs: Successes

- sel4
- Project Everest
- Formally verified crypto libraries
Formal Proofs: Successes

- sel4
- Project Everest
- formally verified OS microkernel
- formally verified crypto libraries
- mozilla
Formal Proofs: Successes

- sel4
- Project Everest
- formally verified OS microkernel
- formally verified crypto libraries
Formal Proofs: Successes

CoCon

Project Everest

Formally verified OS microkernel

Formally verified crypto libraries

CertIKOS

mozilla
Formal Proofs: Successes

CoCon

CoSMed

Project Everest

sel4

CertIKOS

mozilla
Formal Proofs: Successes

CoCon

CoSMed

SAFE

Project Everest

mozilla
Formal Proofs: Successes

- CoCon
- CoSMed
- sel4 (Security, Performance, Proof)
- CERTIKOS
- SAFE
- Quark
- Project Everest
- mozilla
What is a high-fidelity proof?

(current state of the art)
What is a high-fidelity proof?

(current state of the art)
What is a high-fidelity proof?

(current state of the art)

Real World

Binary
What is a high-fidelity proof?

(current state of the art)

Formal World

Real World

Binary
What is a high-fidelity proof?

(current state of the art)

ISA Semantics with Binary in Memory

Formal World

Real World

Binary
What is a high-fidelity proof?

(current state of the art)

Formal World

ISA Semantics with Binary in Memory

Real World

Binary
What is a high-fidelity proof?

(current state of the art)

Formal World

ISA Semantics with Binary in Memory

Proof

Real World

Binary

IEEE Cybersecurity Development Conference (SecDev) 2018
What is a high-fidelity proof?

(current state of the art)

Formal World

ISA Semantics with Binary in Memory

Real World

Binary

Specification

Proof
What is a high-fidelity proof?

(current state of the art)

Proof

Model captures all attacker behaviours we care about

 Formal World

 ISA Semantics with Binary in Memory

 Real World

 Specification

 Binary
What is a high-fidelity proof?

(current state of the art)

Formal World

ISA Semantics with Binary in Memory

Proof

Model captures all attacker behaviours we care about

Real World

Binary

IEEE Cybersecurity Development Conference (SecDev) 2018
What is a high-fidelity proof?

Specification correctly defines what “secure” means

Model captures all attacker behaviours we care about

Formal World

ISA Semantics with Binary in Memory

Proof

Real World

Binary
Example

A claim we might want to prove:

The code below when run on modern x86 CPUs, can cause modifications only within those data cache sets that can be occupied by the physical memory corresponding to the program variables i, r and the array a, whose length is ARRAY_LEN.

```c
if (i < ARRAY_LEN) {
    r = a[i];
}
```
Example

A claim we might want to prove:

The code below when run on modern x86 CPUs, can cause modifications only within those data cache sets that can be occupied by the physical memory corresponding to the program variables \( i \), \( r \) and the array \( a \), whose length is `ARRAY_LEN`.

```c
if (i < ARRAY_LEN){
    r = a[i];
}
```

Is it true?
Example

A claim we might want to prove:

The code below when run on modern x86 CPUs, can cause modifications only within those data cache sets that can be occupied by the physical memory corresponding to the program variables $i$, $r$ and the array $a$, whose length is ARRAY_LEN.

```java
if (i < ARRAY_LEN) {
    r = a[i];
}
```

Is it true? No (e.g. Spectre)
Example

A claim we might want to prove:

The code below when run on modern x86 CPUs, can cause modifications only within those data cache sets that can be occupied by the physical memory corresponding to the program variables \( i, r \) and the array \( a \), whose length is \text{ARRAY\_LEN}.

\[
\text{if} \ (i < \text{ARRAY\_LEN}) \{
\quad r = a[i];
\}
\]

Is it true? No (e.g. Spectre)

But it can be proved if the ISA model doesn’t include speculative execution
What are proofs good for?

**P1:** Proofs as (Qualified) Guarantees

**P2:** Proofs as Structured Exploration

**P3:** Proofs as Commercially Valuable Commodities
As of 19 Feb. 2018, Google records over 8,300 pages that mention “seL4” alongside terms such as “un-hackable”, “invulnerable”, “hack-proof”, “bug free” or “zero bugs”
Proofs and Guarantees

As of 19 Feb. 2018, Google records over 8,300 pages that mention “seL4” alongside terms such as “un-hackable”, “invulnerable”, “hack-proof”, “bug free” or “zero bugs”
Proofs and Guarantees

As of 19 Feb. 2018, Google records over 8,300 pages that mention “seL4” alongside terms such as “un-hackable”, “invulnerable”, “hack-proof”, “bug free” or “zero bugs”
P1: Proofs as Qualified Guarantees

A proof provides guarantees subject to the accuracy of the proof’s assumptions and model ...
P1: Proofs as Qualified Guarantees

Security-related guarantees are: claims of invulnerability to specific attack(s), qualified by lists of assumptions.

A proof provides guarantees subject to the accuracy of the proof’s assumptions and model ...
**P1: Proofs as Qualified Guarantees**

Security-related **guarantees** are: claims of invulnerability to specific attack(s), qualified by lists of assumptions

non-experts (almost everyone) misinterpret these as absolute guarantees against **all** possible attacks

*A proof provides guarantees subject to the accuracy of the proof’s assumptions and model …*
P1: Proofs as Qualified Guarantees

Security-related guarantees are: claims of invulnerability to specific attack(s), qualified by lists of assumptions.

Non-experts (almost everyone) misinterpret these as absolute guarantees against all possible attacks.

(What else should we expect if we use language like “proven secure”)

A proof provides guarantees subject to the accuracy of the proof’s assumptions and model …
Understanding Guarantees
Understanding Guarantees

Proofs cannot make guarantees above whatever formal statement you proved
Understanding Guarantees

Proofs cannot make guarantees above whatever formal statement you proved

How difficult are they to understand, even for experts?
Understanding Guarantees

Proofs cannot make guarantees above whatever formal statement you proved

How difficult are they to understand, even for experts?

Benjamin Pierce (UPenn)
Understanding Guarantees

Proofs cannot make guarantees above whatever formal statement you proved

How difficult are they to understand, even for experts?

2015: studied seL4 proofs to teach them as part of a graduate class.

Benjamin Pierce
(UPenn)
Understanding Guarantees

Proofs cannot make guarantees above whatever formal statement you proved

How difficult are they to understand, even for experts?

2015: studied seL4 proofs to teach them as part of a graduate class.

Took one person week to understand the top level statements well enough to produce 2 lectures

Benjamin Pierce (UPenn)
Fine Print

Are proof statements and assumptions akin to fine print on insurance contracts?
Fine Print

Are proof statements and assumptions akin to fine print on insurance contracts?

Never read by those who most need to read them
Fine Print

Are proof statements and assumptions akin to fine print on insurance contracts?

Never read by those who most need to read them

“User”’s responsibility to ensure assumptions match reality
Fine Print

Are proof statements and assumptions akin to fine print on insurance contracts?

Never read by those who most need to read them

“User”’s responsibility to ensure assumptions match reality

Non-experts in no position to validate assumptions and model
Fine Print

Are proof statements and assumptions akin to fine print on insurance contracts?

Never read by those who most need to read them

“User”’s responsibility to ensure assumptions match reality

Non-experts in no position to validate assumptions and model

 Often not written down but **buried deep in formal models**

(e.g. subtleties of modelling **UNPREDICTABLE** behaviour in ARM ISA)
Language

- Laypeople think “proof” means 100% guarantee of security
- But software proofs have so many assumptions
  - This makes them quite different to theorems people learned in high school (e.g. Pythagoras)
Language

“Engineers in established fields use applied mathematics to predict the behavior and properties of their designs with great accuracy. Software engineers, despite the fact that their creations exhibit far more complexity than physical systems, do not generally do this . . .

”[T]he applied mathematics of software is formal logic, and calculating the behavior of software is an exercise in theorem proving. Just as engineers in other disciplines need the speed and accuracy of computers to help them perform their engineering calculations, so software engineers can use the speed and accuracy of computers to help them prove the . . . theorems required to predict the behavior of software.”

John Rushby, (FM’89)
Language

• Laypeople think “proof” means 100% guarantee of security
• But software proofs have so many assumptions
  – This makes them quite different to theorems people learned in high school (e.g. Pythagoras)

De Millo, Lipton and Perlis, CACM 1979
Language

• Laypeople think “proof” means 100% guarantee of security
• But software proofs have so many assumptions
  – This makes them quite different to theorems people learned in high school (e.g. Pythagoras)

• Rhetorical: Is this why you’ve never heard a civil engineer say that they “proved” or “formally verified” that a bridge won’t fall down?
  – Civil engineers don’t generally have adaptive attackers trying to break their models, yet still avoid this language
  – Should FM-security folks (me!) use different language?
  – “in particular, it would help if they did not call their verifications ‘proofs.’”
    • De Millo, Lipton and Perlis, CACM 1979
Proofs as Non-Guarantees

On the (Alleged) Value of Proof for Assurance

Ehud is about to be annoyed with me. :-)

I am lately suffering from doubts about whether the value of proof is justified by its cost. Some questions, in no particular order:

- Proof seems to be an all-in or all-out exercise. There does not seem to be a viable strategy for initial, limited adoption followed by progressive improvement. Type systems seem to be more "continuous" in their adoption options. Given the (admittedly incomprehensible) expressive power of something like ATS, what is the real semantic gap between provers and type systems these days? Do type-based property embeddings offer the possibility of more incremental and more cost/benefit-aware adoption of checking techniques?
- The goal of proof isn't to be correct. It is to be confident. This has two parts: (1) confidence that you got something right, and (2) ability to convince a second party that your confidence is justified. Assume that you succeed in your proof. To what degree has use of a prover inherently failed objective [2]? If the final objective is credible correctness, then there is presumably a trade-off between accuracy and comprehensibility. Does proof (in its current form, at least) err too much on the side of accuracy?
- Given the likelihood of (possibly retrospective) failures of total abstraction, to what degree is confidence in proof misplaced? If so, is the cost of discharge justified?

There is no question in my mind that proof processes generate more robust code. Yet the general consensus seems to be that this robustness is much more an emergent consequence of rigorously understanding the problem then a result of the proof discharge. In this view, the primary benefit of proof (in the context of assurance) is largely to keep the specification "honest". If this is in fact the goal, is proof the best way to accomplish that goal? How much of that goal can be undertaken by type systems?

In a private conversation, Gernot Heiser (NICTA, OK Labs) was recently asked how use of proof impacted their QA costs. As I recall it, his answer was that they run between 1.5x and 2x the end-to-end cost of conventional development and testing, but they achieve much higher confidence. My questions:

- Might there be a better trade-point between cost and value (in the form of confidence)?
- To what degree is their confidence well-founded?

In a second private conversation, Eric Rudder observed that one of the costs of proof-based methodology was a loss of ability to rapidly adapt software to new requirements and new demands. It follows that proof is not always appropriate, and that a more continuous cost/time/benefit option space would be desirable.

My own observation is that in the end, systems need to be robust, and they include components that lie well outside our ability to prove them. In many cases, type can be usefully exploited when proof cannot, and there is a training cost to educating people in both domains.

So finally my question:

Once we step away from formal semantics and PL insights (which are certainly good things), what is the proper role of proof in real-world production? And what is the proper role of advanced type systems?
Proofs as Non-Guarantees

“There is no question in my mind that proof processes generate more robust code. Yet the general [consensus] seems to be that this robustness is much more an emergent consequence of rigorously understanding the problem [than] a result of the proof discharge.”
P2: Proofs as Structured Exploration

Proofs force careful, rigorous examination of a system (model)
• may find vulnerabilities during proof exercise
• better understanding leads to better designs
• partial proofs thus have residual value if proof result evaporates
  – not true for P1
  – (but little known on how to measure it)
• Do exploration methodologies differ in residual value?
  – structured, manual (BAN)
  – interactive (Isabelle/HOL)
  – fully automated (after idealization)
P3: Proofs as Commercially-Valuable Commodities

Proofs have economic value:

• sales of commercial products
  – tangible product differentiation
  – purchaser requirement
• liability cover / compliance with standards / best practice
• claims of superiority (marketing)
  – intangible
  – may exploit leading language, misunderstanding implications
Proof Side Effects

Things you do to the system being verified, during the proof.
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**

e.g. because the proof found a vuln
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**

e.g. because the proof found a vuln

or to simplify the code to make proof easier
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**

e.g. because the proof found a vuln

or to simplify the code to make proof easier

**Deployment Restriction**
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**

e.g. because the proof found a vuln

or to simplify the code to make proof easier

**Deployment Restriction**

e.g. attacker cannot have physical access
Proof Side Effects

Things you do to the system being verified, during the proof.

**Code change**

- e.g. because the proof found a vuln
- or to simplify the code to make proof easier

**Deployment Restriction**

- e.g. attacker cannot have physical access
to ensure proof assumptions match reality
Proof Side-Effects
Proof Side-Effects

things that stop attacks but not needed for proofs
Proof Side-Effects

things needed for proofs but don’t stop attacks

things that stop attacks but not needed for proofs
Proof Side-Effects

- Things needed for proofs but don’t stop attacks
- Proof induced security fixes
- Things that stop attacks but not needed for proofs
Proof Side-Effects

- Things needed for proofs but don’t stop attacks
- Proof induced security fixes
- Things that stop attacks but not needed for proofs

Changes that introduce new vulnerabilities, unseen by proof.
Example: Constant Time Comparison

```c
/* Unsigned comparisons */
/* Return 1 if condition is true, 0 otherwise */
int ct_isnonzero_u32(uint32_t x)
{
    return (x|-x)>>31;
}
```

https://cryptocoding.net/index.php/Coding_rules
Example: Constant Time Comparison

Vulnerability-Inducing Simplification

```c
/* Unsigned comparisons */
/* Return 1 if condition is true, 0 otherwise */
int isnonzero_u32(uint32_t x)
{
    if (x)
        return 1;
    else
        return 0;
}
```
Proof Side-Effects

proof induced security fixes
Proof Side-Effects

proof induced security fixes
Deployment Restriction: Example
Deployment Restriction: Example

seL4 confidentiality proofs require device interrupts be disabled
Deployment Restriction: Example

seL4 confidentiality proofs require device interrupts be disabled
Deployment Restriction: Example

seL4 confidentiality proofs require device interrupts be disabled

Suppose we care only about isolating **memory** (not interrupts)

Do the seL4 confidentiality proofs help?
WireGuard is an extremely simple yet fast and modern VPN that utilizes state-of-the-art cryptography. It aims to be faster, simpler, leaner, and more useful than IPSec, while avoiding the massive headache. It intends to be considerably more performant than OpenVPN. WireGuard is designed as a general purpose VPN for running on embedded interfaces and super computers alike, fit for many different circumstances. Initially released for the Linux kernel, it plans to be cross-platform and widely deployable. It is currently under heavy development, but already it might be regarded as the most secure, easiest to use, and simplest VPN solution in the industry.
Change the Formal Model

A Cryptographic Analysis of the WireGuard Protocol

Benjamin Dowling and Kenneth G. Paterson

Information Security Group
Royal Holloway, University of London
benjamin.dowling@rhul.ac.uk, kenny.paterson@rhul.ac.uk

and simplest VPN solution in the industry.
To overcome this proof barrier, and as an alternative to performing a monolithic analysis of the entire WireGuard protocol, we **add an extra message to the protocol**. ... This change enable us to prove strong authentication and key indistinguishability properties for the key exchange component of WireGuard under standard cryptographic assumptions.
Change the Formal Model

WireGuard either cannot be proven secure as a key exchange protocol using standard key-indistinguishability notions, or it is vulnerable to key-recovery attacks in the [Key Compromise Impersonation] setting.
Change the Formal Model

WireGuard either cannot be proven secure as a key exchange protocol using standard key-indistinguishability notions, or it is vulnerable to key-recovery attacks in the [Key Compromise Impersonation] setting. Which is it?

and simplest VPN solution in the industry.
Change the Formal Model

WireGuard either cannot be proven secure as a key exchange protocol using standard key-indistinguishability notions, or it is vulnerable to key-recovery attacks in the [Key Compromise Impersonation] setting. Which is it?

How to judge the value of a proof over a model that intentionally differs from reality?
Some Research Questions

Q1: Can we find means to know and measure the relationship between proof side effects and changes that stop attacks, how these sets intersect, and the intersection sizes?

Q2: Can we find means to measure the residual value of proofs, when not all assumptions hold in practice; can we presently even begin to attempt such a measurement?

Q3: How can we better tag formally verified software to explain the fine print that accompanies the proofs?

Q4: What effort can be undertaken to explore formal or other methods to track and validate that (both implicit and explicit) security assumptions in large-scale formal models hold in practice?
Thank You

- toby.murray@unimelb.edu.au
- @tobycmurray