Using Formal Methods to Eliminate Exploitable Bugs

Kathleen Fisher

Tufts University
Original Program Manager, DARPA's HACMS program
Using Formal Methods to Eliminate Exploitable Bugs
Kathleen Fisher
Tufts University
Original Program Manager, DARPA's HACMS program
Using Formal Methods to Eliminate Exploitable Bugs
Kathleen Fisher
Tufts University
Original Program Manager, DARPA's HACMS program
Networked computers are unacceptably insecure

and essentially all computers are networked.
Today many things are networked computers.

http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

And attackers take control with distressing ease.
Why so insecure?

- Faulty Design
- Buggy Specifications
- Implementation Errors
- Side-channel leaks
- Misconfiguration
- Gullible users
- Weak passwords
- Malicious insiders
- Physical security failures
- Reliance on 3rd party software
- Faulty/malicious hardware
- And the list goes on...
- Faulty Design
- Buggy Specifications
- Implementation Errors
- Side-channel leaks
- Misconfiguration
- Gullible users
Many Exploitable Vulnerabilities
In 2012, 3,436 vulnerabilities had public exploits available. 42% of total number of vulnerabilities.

Source: IBM X-Force 2012 Trend and Risk Report
Ubiquitous and Pernicious

Microsoft Security Bulletin MS15-078 – Critical

Remote code execution on all Windows platforms (July 20, 2015). Caused by a buffer underflow in the Adobe Type Manager Library.


Heartbleed: Missing bounds check in OpenSSL allows theft of secret keys, user names and passwords, instant messages, emails, documents without leaving a trace.

http://heartbleed.com
Exploit Kits leverage software bugs

Phoenix Exploit Kit, 2010
CVE-2009-0836: Buffer Overflow
CVE-2009-0927: Missing Bounds Check
CVE-2009-1869: Integer Overflow
CVE-2010-0188: Unspecified
CVE-2010-0840: Unspecified
CVE-2010-0842: Invalid Array Index
CVE-2010-1297: Memory corruption
CVE-2010-1818: Unmarshalling bad ptr
CVE-2010-1885: Mishandling escapes
CVE-2010-2883: Buffer overflow

Enabling the masses to launch pernicious attacks at scale.

Source: The Exploit Intelligence Project, Dan Guido
Networked computers are unacceptably insecure

And essentially all computers are networked.
Hypothesis: Formal Methods can eliminate many exploitable vulnerabilities.

Been there, tried that, wasn't impressed.

Why is now a good time to revisit hypothesis?

We have found that testing the code is inadequate as a method to find subtle errors in design, as the number of reachable states of the code is astronomical. So we looked for a better approach.

*Use of Formal Methods at Amazon Web Services, 2014*
Faster Hardware. More memory.

Source: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software
More automation

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

Picking 80 problem point, the best time has dropped from 1000 (2002) to 40 seconds (2010).

Source: Daniel Le Berre, The International SAT Solver Competitions
More infrastructure

[A] significant part of the effort in existing projects was spent on the further development of verification tools, on formal models for low-level programming languages and paradigms, and on general proof libraries... Future efforts will be able to build on these tools and reach far-ranging verification goals faster, better, and cheaper.

Gerwin Klein, Formal OS Verification—An Overview
Hypothesis:
Formal Methods can eliminate many exploitable vulnerabilities.

Been there, tried that, wasn't impressed.

Why is now a good time to revisit hypothesis?

We have found that testing the code is inadequate as a method to find subtle errors in design, as the number of reachable states of the code is astronomical. So we looked for a better approach.

Use of Formal Methods at Amazon Web Services, 2014
Some Evidence: DARPA's HACMS Program

Using Formal Methods to produce more secure vehicles.

The Setup
Program Goals:
- FI can withstand losses to remain cyber-secure.
- 3rd party:
  - build physical access, distribute agents (e.g. scribble node).
- Onboard tools:
  - Blockchain sensor to be integrated.
Experimental Platforms:
- SMACCM: short range small drone (UAV).

Baseline Security Assessment
Red Team: Attackers could exploit legitimate ground control station & hijack quadcopter in flight.

The Evolving SMACCM Copter

The SMACCM Copter: 18 Month Assessment
- The SMACCM Copter:
  - Drones & onboard AI, Energetic Field, 24/7 operation.
  - GPS network feedback with artificial neural.
- Air: Onboard hardware security properties
- The attacker: battery, GPS feedback, energy
- The attacker: system use and how to sell
  - The sell of a drone: hacking in SMACCM drone linked with the Internet
  - The final: Kali
  - Fiat: security design to identify the attacker with full access to source code.

Pre-Production Testing Phase
- The SMACCM Copter is tested by the initial users & feedback
- Pre-production testing
- Final production testing
**The Setup**

**Program thesis:**
FM can yield vehicles less susceptible to remote cyber attack.

**Threat model:**
No physical access, full knowledge of system and source code.

**Out-of-scope:**
Hardware assumed to be correct.

**Experimental Platforms:**
Arducopter and Boeing's Unmanned Little Bird (ULB)
Baseline Security Assessment

Red Team: Attacker could crash legitimate ground control station & hijack quadcopter in flight.

Source: Darren Cofer, Rockwell Collins
The Evolving SMACCMCopter

Phase 1

Rockwell Collins / UMN

- System requirements
  - AADL model of HW & SW

NICTA

- NICTA RTOS
  - FM Workbench
  - Verification of system requirements
    - AADL translation, generate glue code

Galois

- Embedded DSL (Ivory)
- Factored autopilot tasks

Response to DoS

Research Vehicle

- Monolithic Ardupilot Software
  - HW Abstraction Layer
  - FreeRTOS
  - PX4: ARM Cortex M4

Stability

Other

Legacy Ardupilot

- Stability
- Other
- Monitor

- Glue code
- HAL
- FreeRTOS / eChronos
- PX4: ARM Cortex M4

Source: DARPA HACMS program slides
The SMACCMAcopter: 18-Month Assessment

- The SMACCMAcopter flies:
  - Stability control, altitude hold, directional hold, DOS detection.
  - GPS waypoint navigation 80% implemented.

- Air Team proved system-wide security properties:
  - The system is memory safe.
  - The system ignores malformed messages.
  - The system ignores non-authenticated messages.
  - All “good” messages received by SMACCMAcopter radio will reach the motor controller.

- Red Team:
  - Found no security flaws in six weeks with full access to source code.

- Penetration Testing Expert:
The SMACCMAcopter is probably “the most secure UAV on the planet”.

Open source: autopilot and tools available from http://smaccmpilot.org
Some Evidence: DARPA's HACMS Program

Using Formal Methods to produce more secure vehicles.

The Setup
Program Status:
- DARPA is funding a project to develop a secure V2X communication system.
- The goal is to create a system that is resistant to cyber threats.

The Evolving SMACCMcaptor
- The SMACCMcaptor is a new communication system designed to enhance security.
- The system uses advanced cryptographic techniques to ensure data integrity and confidentiality.
- The system is being tested in various environments to evaluate its effectiveness.

Red Queen Security Assessment
- The assessment involves simulating cyber attacks on the system to test its resilience.
- The results of the assessment will be used to improve the system's security features.

The SMACCMcaptor: 18 Month Assessment
- The system has been under development for 18 months and has met all initial goals.
- The next phase of development will focus on enhancing its performance and scalability.
- The system is expected to be deployed in 2023.
**Formal Methods: An Overview**

**What are “Formal Methods”?**

Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals to problems in software and hardware specification and verification. Understanding Formal Methods, Jean-Francois Mounin, 2003

“I know it when I see it” — Justice Potter Stewart

- Based on math
- Machine-checkable
- Capable of proving properties of code and models
- Built, read the fine print
- Assumptions may be unreasonable
- Guarantees may be too weak

**Survey Results on Using Formal Methods**

![Survey Results](image_url)

**Formal Method Based Tools**

- [Tool 1]
- [Tool 2]
- [Tool 3]

**Characteristics**

- Robustness
- Flexibility
- Simplicity

**Strength of Formal Methods**

- Readability
- Maintainability
- Reliability
What are "Formal Methods"?

Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals ... to problems in software and hardware specification and verification. *Understanding Formal Methods, Jean-Francois Monin, 2003*

"I know it when I see it" — Justice Potter Stewart

**Characteristics:**

- Based on math
- Machine-checkable
- Capable of proving properties of code and models
  - But, read the fine print!
  - Assumptions may be unreasonable.
  - Guarantees may be too weak.
Survey Results on Using Formal Methods

Fig. 6. Did the use of formal techniques have an effect on time, cost, and quality?

Formal Methods: An Overview

What are "Formal Methods"?

Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals to problems in software and hardware specification and verification. Understanding Formal Methods, Jean-Francois Monin, 2003

“Know it when I see it” — Justice Potter Stewart

Characteristics:
- Based on math
- Machine-checkable
- Capable of proving properties of code and models
  - But, read the fine print
  - Assumptions may be unreasonable
  - Guarantees may be too weak

Formal Method Based Tools

Survey Results on Using Formal Methods

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>2%</td>
</tr>
<tr>
<td>Cost</td>
<td>18%</td>
</tr>
<tr>
<td>Quality</td>
<td>83%</td>
</tr>
</tbody>
</table>

Q: Did the use of formal techniques have an effect on time, cost, and quality?

What software is worth verifying?

- **Separation Kernel**: seL4 [SOSP 2009, ToCS 2014]
- **Hypervisor**: mCertiKOS [POPL 2015]
- **RTOSes**: eChronos [echronos.org]
  ORIENTIAS [ICECCS 2012]
- **C Compiler**: CompCert [POPL 2006]
- **Web Browser**: QUARK [USENIX 2012]
- **Browser Sandbox**: RockSalt [PLDI 2012]
- **Crypto Algorithms**: SHA-256, HMAC [USENIX 2015]
  AES-128, SHA-384, ECDSA(NIST P-384) [HILT 2013]
- **Garbage Collector**: Nucleus [PLDI 2010]

All verified to be functionally correct!
seL4 Microkernel

- General purpose, operating system microkernel.
- Implemented and proven correct in Isabelle/HOL.
- Size: 10K LoC, 480K LoP
- Time to build: 13 (8) person years
- Speed: 206 vs 227 cycles in 1-way IPC fastpath
- Machine-checked theorems include
  - Access-control enforcement
  - Non-interference
  - Compilation to binary
  - IPC fast-path correctness

Available open source.

CompCert Verifying C Compiler

Xavier Leroy, Formal certification of a compiler back-end, POPL 2006

- Subset of C used by aviation industry
- Implemented and proven correct in Coq.
- Size: 42K LoC+P
- Time to build: 3 person years
- Speed: 2x speed of gcc -O0,
  7% slower than gcc -O1,
  12% slower than gcc -O2
- Poised to become the compiler for Airbus software.
What software is worth verifying?

- **Separation Kernel**: seL4 [SOSP 2009, ToCS 2014]
- **Hypervisor**: mCertiKOS [POPL 2015]
- **RTOSes**: eChronos [echronos.org] ORIENTIAS [ICECCS 2012]
- **C Compiler**: CompCert [POPL 2006]
- **Web Browser**: QUARK [USENIX 2012]
- **Browser Sandbox**: RockSalt [PLDI 2012]
- **Crypto Algorithms**: SHA-256, HMAC [USENIX 2015] AES-128, SHA-384, ECDSA(NIST P-384) [HILT 2013]
- **Garbage Collector**: Nucleus [PLDI 2010]

All verified to be functionally correct!
Impediments to Using Formal Methods

The Problem of Relevance

The Required Level of Effort

Performance Consequences

Keeping Up

The Flow Print

Due to the nature of the problem, the following issues may arise:

1. Inadequate training for users
2. Lack of suitable tools
3. Complex formal methods

Due to this, there is a significant challenge to maintaining the benefits of formal methods.
The Problem of Expertise


<table>
<thead>
<tr>
<th>Region</th>
<th>Researchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>1,000</td>
</tr>
<tr>
<td>UK</td>
<td>500</td>
</tr>
<tr>
<td>Europe</td>
<td>1,000</td>
</tr>
<tr>
<td>China</td>
<td>250</td>
</tr>
<tr>
<td>Nordic</td>
<td>500</td>
</tr>
<tr>
<td>Japan</td>
<td>250</td>
</tr>
<tr>
<td>Others</td>
<td>1,000</td>
</tr>
</tbody>
</table>

The Required Level of Effort

Significant overhead in terms of lines of code/proof, but level of effort is becoming a reasonable investment for certain kinds of software.

- seL4 Separation Kernel [SOSP 2009; ToCS 2014]
  10K LoC, 480K LoP; 13(8) person years
- CompCert Verifying C compiler [POPL 2006]
  42K LoC+P; 3 person years
- FSCQ File System [SOSP 2015]
  24K LoC+P; <5 person years
- certiKOS Hypervisor [POPL 2015]
  3K LoC, 18.5K LoP; 1 person year
- SHA-256/HMAC [USENIX 2015]
  407 LoC, 14.4K LoP; not reported
- Rocksalt Sandbox [PLDI 2012]
  100 LoC, 10K LoP; <2 person years
- Nucleus Allocator/Garbage Collector [PLDI 2010]
  6K LoP+C; 0.75 person years
- QUARK Web Browser [USENIX 2012]
  5.5K LoP+C; 0.5 person years
Performance Consequences

Verified code is not intrinsically slower, but verifying faster code can be more time consuming.

- seL4 Separation Kernel [SOSP 2009; ToCS 2014]
  206 vs 227 cycles in 1-way IPC fastpath
- CompCert Verifying C compiler [POPL 2006]
  2x speed of gcc-O0, 7% slower than gcc -O1, 12% slower than gcc -O2
- FSCQ File System [SOSP 2015]
  Performance roughly 80% of xv6 file system.
- certiKOS Hypervisor [POPL 2015]
  <2x slowdown on most lmbench benchmarks
- SHA-256/HMAC [USENIX 2015]
  Performance equal to OpenSSL 0.9.1c (March 1999)
- Rocksalt Sandbox [PLDI 2012]
  1M instructions/second; faster than Google's checker
- Nucleus Allocator/Garbage Collector [PLDI 2010]
  “competitive performance on macro-benchmarks when compared to native garbage collection.”
- QUARK Web Browser [USENIX 2012]
  24% overhead wrt WebKit baseline on the top 10 Alexa Web sites
Keeping Up

Time to produce a proof can be serious impediment to adoption.

(Many of the artifacts we've considered lack features.)

Facebook's INFER sound static analyzer processes millions of lines of code and thousands of diffs per day.
- 4 hours for full Android/iOS code base
- <10 minutes on average for single diffs

But, proves only the absence of null pointer exceptions and resource leaks.

The Fine Print

All proofs come with assumptions. Violate those assumptions and all bets are off.

John Regehr et al found 325 bugs in various C compilers, including CompCert [PLDI 2011].

CompCert bugs were in:
• the unverified front end (subsequently fixed & verified)
• a hardware model

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. ... CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task."

Source: Finding and Understanding Bugs in C Compilers, PLDI 2011
Impediments to Using Formal Methods

The Problem of Relevance

The Required Level of Effort

Performance Consequences

Keeping Up

The Fine Print

1. Articles in this issue with asterisks are available online at our website.

2. Some authors have requested that their names not be published.

3. This research was supported by the National Science Foundation under Grant No. XXX.
Taking Stock

Formal Methods:
- Not just for implementation stage!
- Formal methods solve security challenges as well!
- Rust
- Scala
- Futhark
- The Activity Model for Miro
- TLA+
- Property-based testing
- PhD student Chi-Xuan steering a research domain named "Formal Security of Distributed Systems"
- On the rise of the "Formal methods for security" community

Lessons Learned:
- Proving very complex formal proofs is not practical in large organizations.
- Performance of formal proofs can be significantly improved.
- The static semantics of the target language is not always the same as the dynamic semantics.
- Formal methods are not a silver bullet.
- The OSE messages need to be very precise.
- Formal proofs are very time-consuming.
- Formal verification is hard.
- A lot of work still needs to be done to make formal methods more accessible.

Ongoing Research & Challenges:
- Producing and validating models of real systems across SDLC, System Design, APIs, and code base
- Introducing formalization in the organization
- Tooling
- Proof automation
- Aligning with formal development processes
- Getting buy-in from project managers
- Developing our tools and infrastructure
- Making formal methods more accessible
- Building more community
Formal Methods:
Not just for implementation bugs!

Formal methods can help with other security challenges as well:
  • Faulty design
    Ex: Amazon Web Services use of TLA+
  • Buggy specifications
  • Ex: Rockwell Collins detecting unencrypted comm channel
  • Side-channel information leaks
    Ex: NICTA analysis of seL4
  • Dependence on 3rd party software
Lessons Learned

- Don't verify existing code artifacts
  seL4, CompCert, FSCQ, certiKOS, Rocksalt, Nucleus, QUARK
- Use static analyzers to eliminate obvious bugs before starting formal verification.
- Don't verify all code:
  Secure essential infrastructure & contain the rest.
  QUARK, Verve/Nucleus, Rocksalt
- Use DSLs to generate code and correctness proofs
  Rocksalt, Ivory/Tower, SpiralGen, BilbyS
- Composition enables scaling
  Facebook INFER; UC-KLEE, Rockwell Collins Workbench; mCertiKOS
- Automation is essential:
  Tactic libraries and SMT/SAT solver integration
  FSCQ, Nucleus, mCertiKOS
On-Going Research & Challenges

- Producing and validating models of real systems x86, LLVM, Linux, Browser APIs, POSIX interfaces
- Increasing automation
- Scaling
- Proof engineering
- Integrating with normal development processes
- Getting buy-in/adoptions & training
  "Exhaustively testable pseudo-code"
  [Using Formal Methods at Amazon Web Services]
- Handling concurrency
Within Reach?

- Critical parts of critical systems are built out of verified components & their composition is verified.
- Buggy software is no longer an easy attack vector.

- Black Hat is reduced to non-scalable attack vectors:

   ![Comic Strip](http://xkcd.com/1559/)
verified components & their composition is verified.

• Buggy software is no longer an easy attack vector.

• Black Hat is reduced to non-scalable attack vectors:

http://xkcd.com/1559/
QUESTIONS?

Thanks to:

Andrew Appel  John Launchbury
Princeton  DARPA
Adam Chlipala  Gerwin Klein
MIT  NICTA
Darren Cofer  Gary McGraw
Rockwell Collins  Cigital
Drew Dean  Greg Morrisett
SRI  Cornell
Dan Guido  Aaron Tomb
Trail of Bits  Galois
Joe Hendrix  Mike Walker
Galois  DARPA