Using Formal Methods Tools to Improve Security in an Autonomous Military Truck

Dariusz Mikulski, Ph.D.
SANS Automotive Cybersecurity Summit
Monday, May 1st, 2017

DISTRIBUTION A. Approved for public release. Distribution is unlimited.
A little about me…

• Who Am I?
  – U.S. Army Civilian (14 years)
  – Senior Researcher

• What do I do?
  – Cooperative Teaming
  – Vehicle Cybersecurity

• Examples
  – Computational Trust Research
  – Autonomous Convoy Operations
  – Collaborations with DARPA, ARL, and Universities
Most of our tools require our explicit direction to do something useful
Tomorrow

Machines will be enhancing the way we create stuff with generative design

Metal Work
Source: arup.com

Lightweight Engine Block
Source: Autodesk

Car Frame
Source: Bandito Bros. & Autodesk
Generative Adversarial Networks

- Generative model (G) maps from a latent space to particular data distribution

- Discriminative model (D) distinguishes between true and generated data

- Have G and D train while competing against each other
  - G takes latent space (noise) as input and generates samples
  - D takes G samples and training data, and back-propagates loss to adjust parameters
  - G back-propagates negated D-loss to adjust parameters, and generate newer samples

Source: Antipov et al., Face Aging with Conditional Generative Adversarial Networks, 2017
What if a machine generated this presentation?

• Key ideas: “Formal Methods”, “DARPA HACMS”, “Cybersecurity”, “Vehicles”

• Duration: 45 minutes
  – Including presenter bio? Yes. -1 minute
  – Including question/answer? Yes. -5 minutes
  – BS Level: 10%

• Style
  – TARDEC template

• Audience: Tech-Savy, Nerdy

• Generate clipart/images/video? Sure.

• Continuously update? Why not.

“Hell-ooo, Computer?”
Source: Star Trek IV
Computer-Generated Software

**Specification**

- Checkmarks indicating the specification sections

**Code**

```c
/* findFactors -- a straightforward C program to find the prime factorization of a whole number greater than one. */
/* Copyright (C) 2018 Alex Roman -- All rights reserved. */
#include <stdio.h>
#include <stdlib.h>

/* A subroutine will do the work for each input number. */
int findFactors(long factorMe); /* Error code */

int main(int argc, char *argv[]) {
  long input_argument; /* Error code */
  if (argc == 1) {
    printf(stderr, "%s: please provide whole numbers greater than 1 as command line arguments.", argv[0]);
    return 1;
  }
  for (i = 1; i < argc; ++i) {
    input_argument = atol(argv[i]);
    if (findFactors(input_argument) != 0) {
      printf(" factors\n");
    } else {
      printf(" not eligible for prime factorization\n", argv[i]);
    }
  }
}

/* Print the factors to stdout. Returns 0 if input wasn't valid, returns 1 if we provided the factors. */
int findFactors(long factorMe) {
  long candidate; /* candidate is used for potential factors. */
  /* Validate input. */
  if (factorMe < 2) return 0;

  /* Proceed to factor the input. */
  if (factorMe < 2) return 0;
  while (candidate = factorMe / 2) {
    if (factorMe % candidate) {
      /* candidate is not a factor of factorMe: keep searching. */
      ++candidate;
    } else {
      /* candidate is a factor. Reduce factorMe. */
      /* Check candidate again, because it might be a factor multiple times. */
      printf(" ", candidate);
      factorMe /= candidate;
    }
  }
  printf(" ", factorMe);
  return 1;
}
```

**Generator**

- A flowchart or diagram illustrating the process of generating code from a specification
Specification $\iff$ Code
Formal Methods

• What?
  – Often described as rigorous system-design techniques
  – Used to mathematically prove facts about a system
  – Used to mathematically prove the absence of certain types of errors

• Why?
  – Tests can only show the situations where a system won't fail
  – Tests cannot say anything about the behavior of the system outside of the testing scenarios
  – In contrast, once a theorem is proven true, it remains true

• Also...
  – Software dependency is increasing
  – Software complexity is increasing
  – Maintaining reliability is getting harder
Formal Methods Concepts

• Formal Specification (Formula)
  – Rigorously define a system using a modeling language
  – Similar to the process of converting a word problem into algebraic notation
  – Precise, eliminates ambiguity
  – **Goal**: To describe behavior without describing/constraining implementation

• Formal Verification (Proof)
  – Complete argument for correctness of some property in system description
  – Constructed as a series of steps
  – May be manual, but usually automated

• Model Checking (Implementation)
  – Operational rather than analytic
  – Determines if some finite state machine (FSM) satisfies Formula
  – Explores all reachable paths in FSM (“brute force”)

DISTRIBUTION A. See first page
**Specification**
A function shall read in two numbers and report the larger of the two numbers.

**Formal Specification**
\[
\left\{(\{Output = x\} \land \{x > y\}\right) \lor \left(\{Output = y\} \land \{y > x\}\right) \lor \left(\{Output = x\} \land \{x \equiv y\}\right)\}
\]

**Formal Verification**
In an ordered field \( K \), there is a distinguished subset \( P \) such that:

1. \( \{P, \{0\}, -P\} \) is a partition of \( K \)
2. \( \forall a, b \in P, a + b \in P \) and \( ab \in P \)

We can then define a *strict order* relation on \( K \) by setting \( a > b \) if \( a - b \in P \).

We can also define an *equivalence* relation on \( K \) by setting \( a \equiv b \) if \( a - b \notin P \).
Benefits of Formal Methods

• Provability
  – Guarantee the absence of certain defects
  – Establish fundamental system properties and invariants
  – Checkable by others

• Discipline
  – More rigor enables a better understanding of the problem
  – Abstraction helps separate specification from design (what vs. how)

• Precision
  – Semantics allow checks for self-consistency
  – Discover defects earlier in the development cycle
Limitations of Formal Methods

- **Expensive**
  - Due to rigor involved
  - Large initial cost followed by less consumption as a project progresses

- **Limits of Computational Models**
  - Cannot assure completeness of a specification
  - Some problems are undecidable (computational theory)

- **Usability**
  - Not a replacement for standard quality assurance
  - Often difficult to learn and apply
  - All-encompassing formal methods are often incredibly complex & nuanced
  - Usually considered intolerable for developers
Reality: Pervasive Vulnerabilities in Platforms

- Long-range coms.
- Short-range coms.
- Cloud / Wi-Fi
- GPS / Telematics
- Off-board Sensors
- V2I
- V2V
- Exposed Data Ports
- Supply Chain
- On-board Sensors

DISTRIBUTION A. See first page
“High-Assurance Cyber Military Systems”

• “High-Assurance” means:
  – Functionally correct
  – Satisfying appropriate safety and security properties
let rec getFactor num proposed acc =
  if proposed = num then
    proposed::acc
  elsif num % proposed = 0 then
    getFactor (num/proposed) proposed (proposed::acc)
  else
    getFactor num (proposed+1) acc
let primeFactors n = getFactor n 2 []
HACMS Program Structure

Penetration Testing
AIS

Ground Vehicle Team
HRL
TARDEC Autonomous Truck & GVR-Bot

Air Vehicle Team
Boeing
Unmanned Little Bird & Quadcopter

Application-Level Software
Galois, CMU, Draper Labs, MIT, Oxford, Princeton, SpiralGen, University of Illinois, University of Pennsylvania

Low-Level Software
Data61 (NICTA), Yale

Architectural-Level
Rockwell Collins, University of Minnesota, SRI

DISTRIBUTION A. See first page
Sorry, no cool video

DARPA still OPSECing it

Look for it on YouTube soon…
Fundamental Requirement: Isolation

- **Processor**
- **Operating System**
- **Trusted Section (Sensitive / Critical)**
- **Untrusted Section**

Strong Isolation

- **Trustworthy Separation Kernel**
- **Communications s.t. Global Security Policy**
HACMS: Key Technologies

- Specification Languages (i.e. Ivory/Tower, AADL)
- Synthesized High Assurance Software
- Verification Tools (i.e. Coq)
- Formally Verified OS (i.e. seL4)
Operating System: seL4 + CAmkES

- **seL4** uses *capability-based access control* to enable formal reasoning about object accessibility

- In 2009, seL4 was formally proved for functional correctness, which guarantees:
  - No deadlocks or livelocks
  - No buffer overflows
  - No arithmetic exceptions
  - No use of uninitialized variables

- Later, proved isolation

- **CAmkES**
  - Component Architecture for microkernel-based Embedded Systems
  - language to describe components and their interfaces
  - Tool to build a complete, bootable, seL4 system image
Specification Languages: Ivory / Tower

• **Ivory**: Domain-Specific Language (DSL)
  – Implemented as a library of the Haskell programming language
  – Currently has a single backend, which produces C source files
  – Not a general purpose programming language

• Gives strong guarantees of type and memory safety
  – Does not allow nullable pointers
  – Does not allow unbounded memory access
  – Does not allow heap allocation

• **Tower**: eDSL for composing Ivory programs into real-time systems
  – Generates Ivory code which implements scheduling/comms for real-time OS
  – Generates AADL description of a system

• **AADL**: Architecture Analysis & Design Language
  – SAE standard for Model-Based Engineering
  – Used to model HW/SW architectures of embedded real-time systems
Trustworthy User-Level Code

- **Ivory/Tower**
  - Control DSL
  - Behavioral Code
  - Generate
  - Compile

- **AADL**
  - Architecture Description
  - Generate
  - Generate

- **CAmkES**
  - Component Description
  - Generate
  - Generate

- **.h, .c**
  - Glue Code

- **seL4 Binary**
  - Compile
  - Compile
Verification Tools: Coq / Bedrock

• Coq: a formal proof management system
  – Expresses mathematical assertions
  – Mechanically checks proofs of assertions
  – Helps to find formal proofs

• Bedrock: turns Coq into a software verification tool
  – Assembly language with an operational semantics (serving as the trusted code base)
  – Semantic module system (orchestrating linking of code and proofs across source languages).
Verification Tools: KeYmaera

- Verification tool for hybrid systems (models with interacting discrete and continuous dynamics)
- Plug-in architecture for solvers and proof rules
- Verification amounts to using differential dynamic logic to specify correctness properties
- Has been used in automotive, aviation, autonomous systems, and surgical robotics
Ex: Collision Avoidance (uncontrolled flight)
Ex: Classical Roundabout Resolution
Counter-Ex: Unsuccessful Classical Approach
Ex: New Maneuver – Tangential Roundabout
Ex: Distributed Aircraft Controllers
So what?
• seL4 + CAmkES
  – https://sel4.systems/
  – https://github/com/seL4/

• eChronos
  – Formally-verified Real-Time Operating System
  – https://github.com/echronos

• CertiKOS
  – Formally-verified Hypervisor from Yale University (not yet open-source)
  – http://flint.cs.yale.edu/certikos/
HACMS Specification Languages

• **AADL**
  - [https://github.com/smaccm/smaccm/releases](https://github.com/smaccm/smaccm/releases)

• **Ivory/Tower**
  - [http://ivorylang.org/](http://ivorylang.org/)
  - [https://github.com/GaloisInc/ivory/](https://github.com/GaloisInc/ivory/)
  - [https://github.com/GaloisInc/tower](https://github.com/GaloisInc/tower)

• **HCOL / Spiral**
  - Hybrid Control Operator Language
  - DSL for control code/proof co-synthesis in Spiral
HACMS Verification Tools

• Coq / Bedrock
  – https://coq.inria.fr/
  – http://plv.csail.mit.edu/bedrock/

• KeYmaera
  – http://symbolaris.com/info/KeYmaera.html
  – https://github.com/LS-Lab/KeYmaeraX-release

• Isabelle
  – Generic proof assistant
  – https://isabelle.in.tum.de/
Other HACMS Tools

• Resolute
  – Language & tool for constructing assurance cases based on AADL models

• Guardol
  – DSL for constructing correct network guards operated over tree-shaped data
  – Generates Ada code

• CompCert C Compiler
  – A high-assurance compiler for almost all of the ISO C90/ ANSI C language for PowerPC, ARM, and x86
  – Rules out the possibility that the compiler introduces bugs
  – http://compcert.inria.fr/

• Other DARPA tools
Happening RIGHT NOW!
Thank You

@DariuszMikulski

(you can also find me on LinkedIn and ResearchGate)