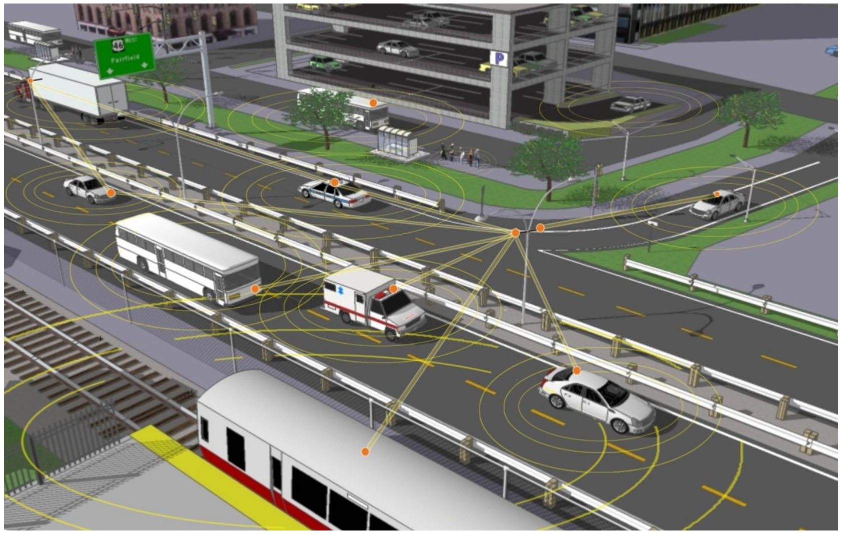
galois

Development of a Verified Message Encoder/Decoder for Automotive Vehicle to Vehicle (V2V) Communications

Mark Tullsen, tullsen@galois.com (presenting)
Lee Pike, leepike@galois.com
Nathan Collins, conathan@galois.com
Eric Woldridge, ericw@galois.com
Aaron Tomb, atomb@galois.com

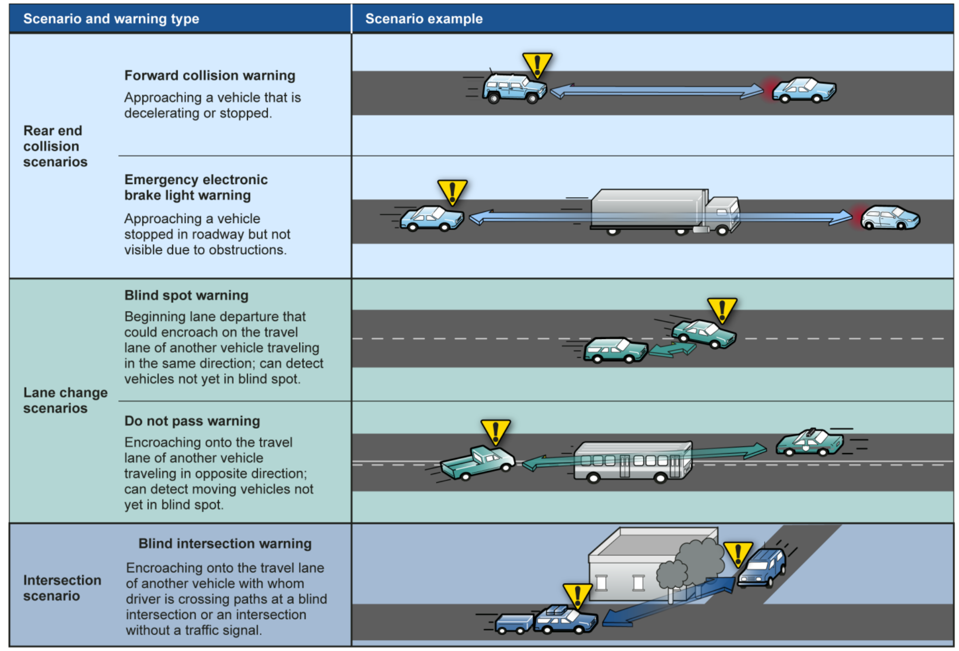
ITS (Intelligent Transportation Systems)

V2V (Vehicle to Vehicle)



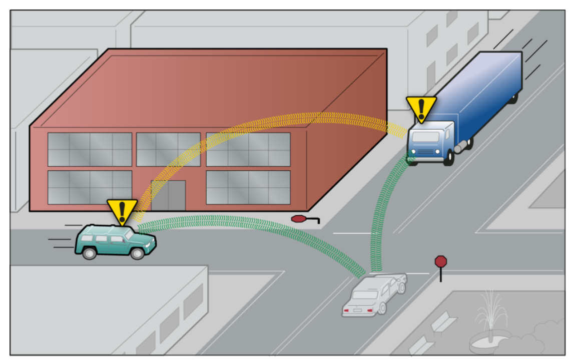
- Emergency brake light warning
- Forward collision warning
- Intersection movement assist
- Blind spot and lane change warning
- …

V2I (Vehicle to Infrastructure)



Scenario and warning type		Scenario example
Rear end collision scenarios	**Forward collision warning** Approaching a vehicle that is decelerating or stopped.	
	Emergency electronic brake light warning Approaching a vehicle stopped in roadway but not visible due to obstructions.	
Lane change scenarios	**Blind spot warning** Beginning lane departure that could encroach on the travel lane of another vehicle traveling in the same direction; can detect vehicles not yet in blind spot.	
	Do not pass warning Encroaching onto the travel lane of another vehicle traveling in opposite direction; can detect moving vehicles not yet in blind spot.	
Intersection scenario	**Blind intersection warning** Encroaching onto the travel lane of another vehicle with whom driver is crossing paths at a blind intersection or an intersection without a traffic signal.	

Source: GAO analysis of Crash Avoidance Metrics Partnership information.

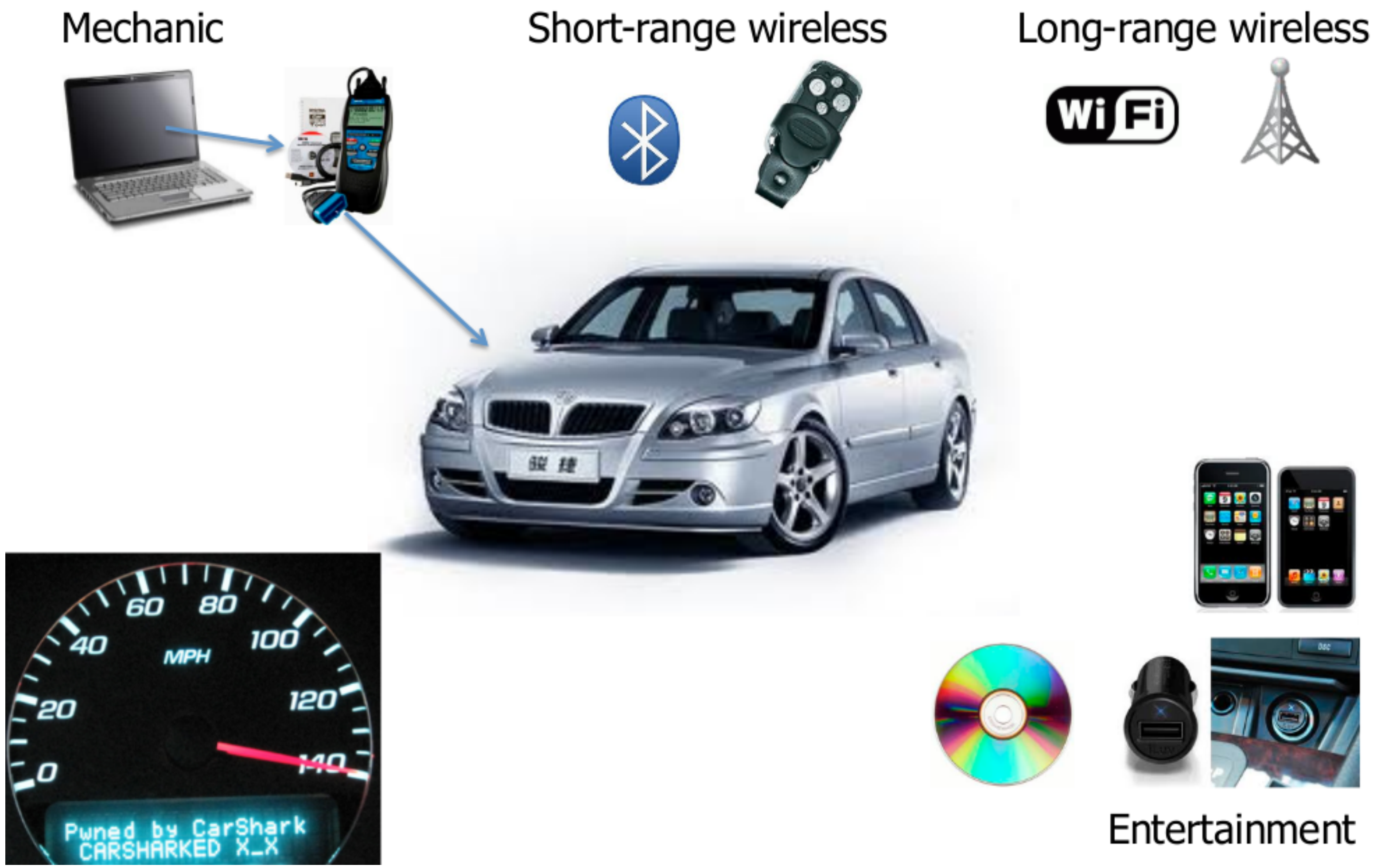


Source: GAO.

This Project

- Small, research-oriented pilot study
  - Can we develop a formally verified encoder/decoder for the messages between vehicles?
- Funded by DOT/NHTSA (Art Carter, POC)
- Partners
  - Battelle (Prime, Management)
  - Galois (Sub, Technical work)
    - Expertise in ASN.1, security, embedded-systems, formal methods
- Galois Team: Lee Pike, Mark Tullsen, Nathan Collins, Eric Woldridge, Aaron Tomb

From Embedded Systems to Cyber Physical Systems



src: Kathleen Fisher, http://www.cyber.umd.edu/sites/default/files/documents/symposium/fisher-HACMS-MD.pdf

Hacking Cars

Researchers Show How a Car's Electronics Can Be Taken Over Remotely

By JOHN MARKOFF
Published: March 9, 2011

New York Times

ANDY GREENBERG SECURITY 08.01.16 3:30 PM

THE JEEP HACKERS ARE BACK TO PROVE CAR HACKING CAN GET MUCH WORSE

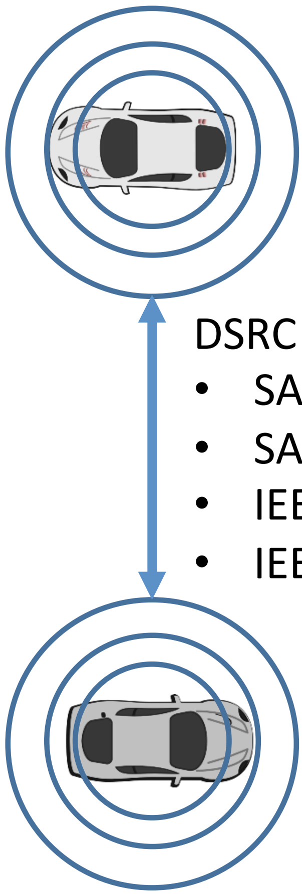
WIRED

Example Attacks

Vulnerability Class	Channel	Implemented Capability	Visible to User	Scale	Full Control	Cost
Direct physical	OBD-II port	Plug attack hardware directly into car OBD-II port	Yes	Small	Yes	Low
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High
	PassThru	WiFi or wired control connection to advertised PassThru devices	No	Small	Yes	Low
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)	No	Large	Yes	Medium-High
	Cellular	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High
	DSRC	???		???	???	???

Comprehensive Experimental Analyses of Automotive Attack Surfaces, Stephen Checkoway et al. (2011)

Secure V2V



DSRC

- SAE J2945/*
- SAE J2735
- IEEE 1609.x
- IEEE 802.11p

Pilot-study security focus:
J2735 (ASN.1)

SAE J2735 Basic Safety Message	
Basic Vehicle State (Vehicle Size, Position, Speed, Heading, …) *Mandatory in all Messages*	Part 1
Vehicle Safety Extensions (Exterior Lights, Trailer Data, …) *Optional in all Messages*	Part 2

What is ASN.1?

- It is not a single specification, not a library (that we implement once)
- It is the language by which we define hundreds of protocols and data-formats

Where Is ASN.1 Used? (Everywhere)

Telecomm

- Cellular protocols including UMTS, 4G, LTE
- Call control SS7, CSTA
- H.323

Networking in General

- SNMP, X.500, LDAP
- PKI X.509

Automotive (Intelligent Transportation Systems "ITS")

- Telematics
- *DSRC (dedicated short range communications)*
- GPS
- Toll booths
- Anti-theft applications

ASN.1: Security Problems?

In Theory: A great idea; In Practice: Easy to get wrong

- Very large, complex language
  - language features interfere with each other
- Evolving standards
- Multiple encoding schemes (BER, DER, PER, XER, …)
- Numerous opportunities for low-level software errors in the bit-fiddling code

Commercial ASN.1 libraries, compilers have had flaws/vulnerabilities!

… Yet, this is the first line of interface for many mission-critical systems, so it must be correct. (Typically on the attack surface.)

Patch and Pray Doesn't Work

Common Vulnerabilities and Exposures

The Standard for Information Security Vulnerability Names

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ASN.1

Search Results

There are **95** CVE entries that match your search.

Name	Description
CVE-2016-5080	Integer overflow in the rtxMemHeapAlloc function in asn1rt_a.lib in Objective Systems ASN1C for C/C++ before 7.0.2 allows context-dependent attackers to execute arbitrary code or cause a denial of service (heap-based
	protection mechanism and discover an authentication key via a crafted application, aka internal bug 26234568. NOTE: The vendor disputes the existence of this potential issue in Android, stating "This CVE was raised in error: it referred to the authentication tag size in GCM, whose default according to ASN.1 encoding (12 bytes) can lead to vulnerabilities. After careful consideration, it was decided that the insecure default value of 12 bytes was a default only for the encoding and not default anywhere else in Android, and hence no vulnerability existed."
CVE-2016-2176	The X509_NAME_oneline function in crypto/x509/x509_obj.c in OpenSSL before 1.0.1t and 1.0.2 before 1.0.2h allows remote attackers to obtain sensitive information from process stack memory or cause a denial of service (buffer over-read) via crafted EBCDIC ASN.1 data.
CVE-2016-2109	The asn1_d2i_read_bio function in crypto/asn1/a_d2i_fp.c in the ASN.1 BIO implementation in OpenSSL before 1.0.1t and 1.0.2 before 1.0.2h allows remote attackers to cause a denial of service (memory consumption) via a

http://www.theregister.co.uk/2016/07/19/asn_objective_systems_asn_compiler_memory_bug/

Guilt by ASN: Compiler's bad memory bug could sting mobes, cell towers

Telco, embedded systems may inherit remote vulns

19 Jul 2016 at 03:40, Richard Chirgwin

0

Our Approach: Security In Depth

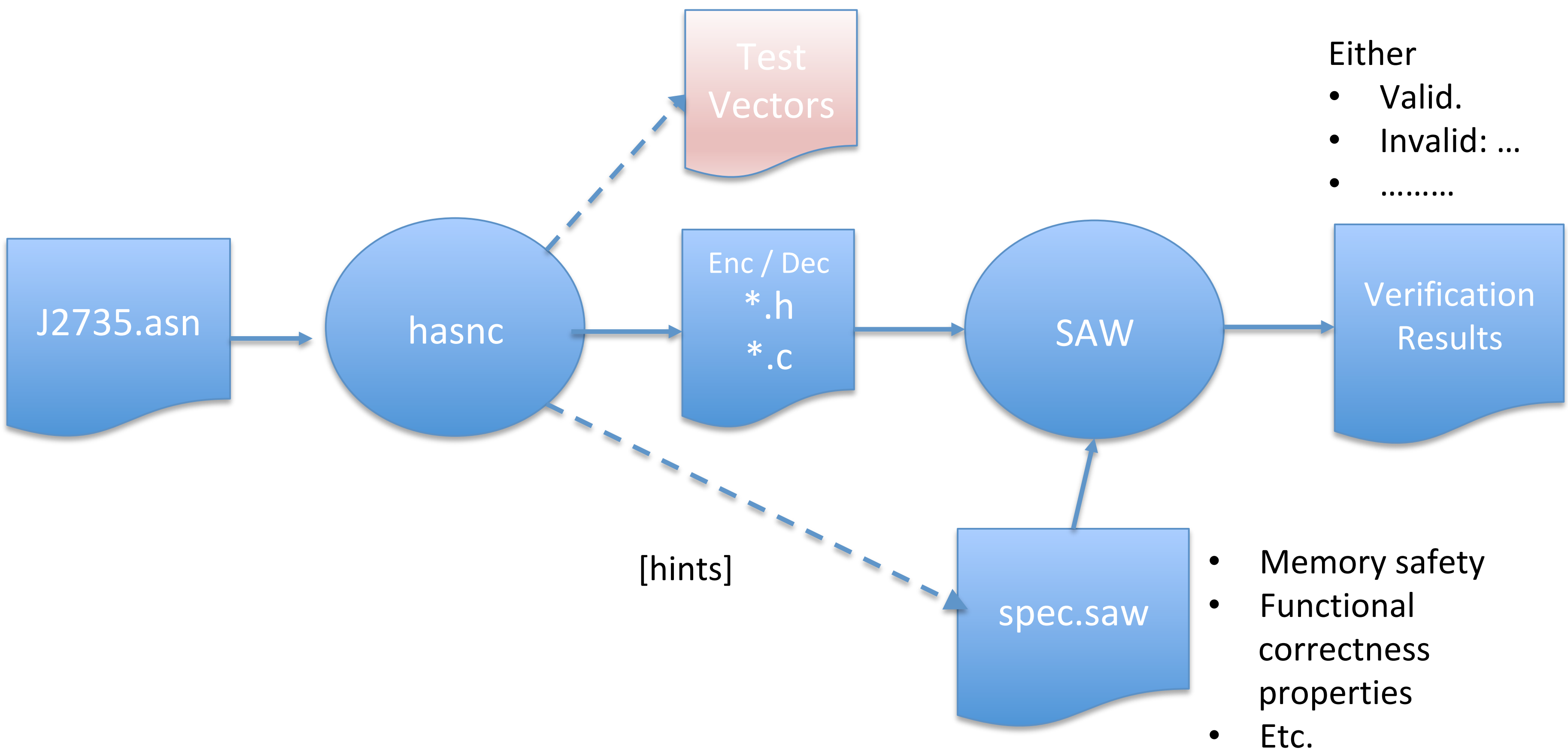
- Generate correct code
  - Galois ASN.1 compiler "correct by construction"

- Test the code
  - test vectors
  - compare to other ASN.1 com

- Verify the code
  1. Motivations
  2. Properties
  3. Approach
     - tools
     - methods

- Code Generation via correctness-preserving transformations of ASN.1 interpreter
- Optimized for verification of compiler
- Optimized for correctness of generated code

Overview


Test Vectors
Either
Valid.
Invalid: …
………
J2735.asn
hasnc
Enc / Dec
*.h
*.c
SAW
Verification Results
[hints]
spec.saw
Memory safety
Functional correctness properties
Etc.

Verify Code: 1. Motivation

Why Is Testing Hard?

```
dec( uint64_t x
   , uint64_t y
   , uint64_t z) {

  ...

}
```

Execution time: 1ms

And J2735 contains dozens of functions. Complexity grows exponentially!

- Number of unique inputs: $(2^{64})^3$ ≈ $6.3*10^{57}$
- Volume of the Pleiades star cluster (cm^3) ≈ $226*10^{54}$

- Time to execute $6.3*10^{57}$ tests (years) ≈ $2*10^{47}$
- Age of universe (years) ≈ $1.4*10^{10}$

How Do You Know When You've Tested Enough?

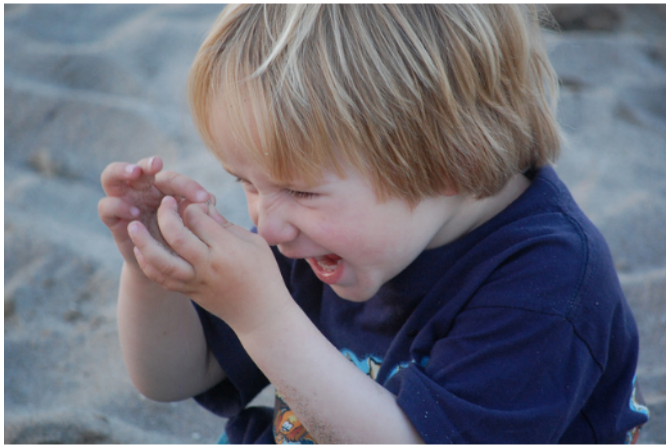
```
dec( uint64_t x
   , uint64_t y) {

  if(   x == 1029384756
     && y == 6574932010
    ) { launch_attack(); }

  ...

}
```

Only detected if you tested
`dec( 1029384756`
`   , 6574932010)`



Creative Commons https://www.flickr.com/photos/kevinkrejci/4735243774

Verify Code: 2. Properties?

Verify Code: 2. Properties

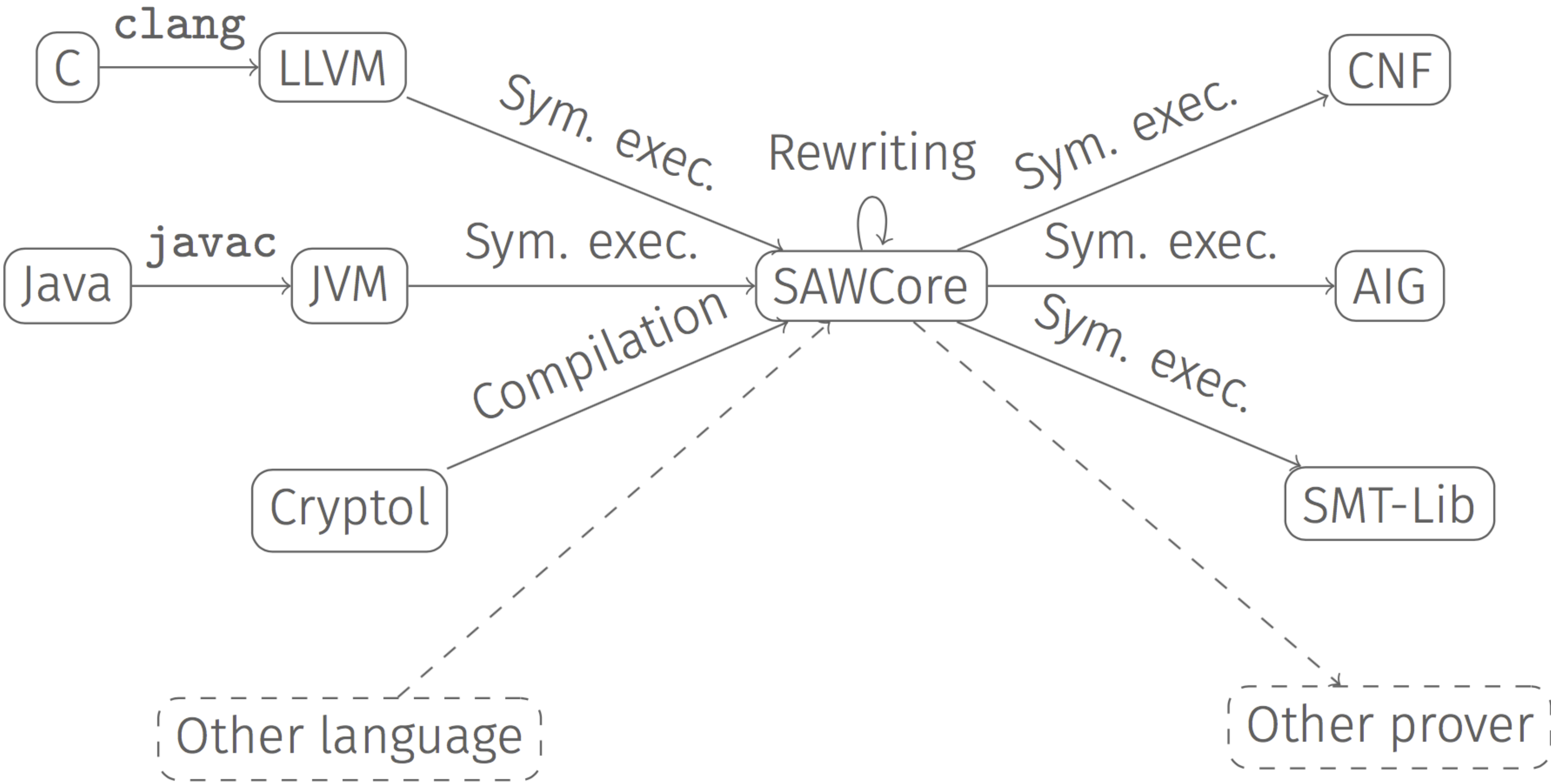
- For any piece of software we want to know:
  - Does it behave correctly and is it secure?

- For J2735 (ASN.1) encoders/decoders:
  - Behaves correctly
    - Round trip: encoding then decoding gives back the original message
      - Forall msg. dec(enc(msg)) = VALID msg
    - Rejection: bad messages are detected (not decoded). Forall bits, either
      - dec(bits) = INVALIDMSGDETECTED , Or
      - dec(bits) = VALID msg, and enc(msg) = bits
  - Is secure
    - Is it "good/valid/safe/…" C (e.g., no buffer overruns, no seg-faults, etc.)

- N.B.: Not full functional correctness.

Verify the Code: Approach

SAW Architecture



"Automated" Formal Methods Applied

1. In SAW, write property P on the code
2. Iterate

```
until SAW proves P repeat:
    if No: counter-example then
        <fix code and/or P>
    else if tool-issues OR no-results then
        <tricks-of-the-trade/incantations/etc>
```

nope

- Switch SMT solver
- Write SAW Overrides
- Specialize SAW overrides

SAW Overrides

- If we have this in C:

```
int f_implem (...) { ... }
```

- We can write this:

```
f_spec =  ... – in Cryptol
let thm1 = {{ \x -> f_implem x == f_spec x }};
```

- Now SAW can use **`f_spec`** in the symbolic simulation of programs that use **`f_implem`**.

Using SAW Overrides for V2V verification

```
/* becb - big endian copy bits */
int becb (dst,dst_i,src,src_i,length) {
  /* ugly bit-manipulation … */
  }
```

- Wrote spec in Cryptol to override.
- Still not getting proof!
- Problem: loop with dynamic bounds in **`becb`**
- AHA:
  - Iterations of loop determined by **`src_i`** and **`length`**
  - Small number of statically known **`src_i`**, **`length`** combinations
- Solution:
  - Enumerate the cases and write overrides for each **`becb`** call

Summary

Accomplished:

- Verified Encoder/Decoder for Basic Safety Message Part I

Lessons:

- Automated Formal Methods Work!
  - … with help from an expert SAW user
  - … with detailed knowledge of code structure

Next Steps

- Extend to full Basic Safety Message (Parts II & III)
    - More challenging ASN.1 constructs
- Apply method to other parts of V2V software stack
    - Below: IEEE 802.11p, IEEE 1609
    - Above: J2945/*
- Apply work to 3rd party code for J2735
    - Do verification and test generation for
        - Hand-written code / code from other compilers

Thank You

BACKUP

Symbolic Simulation
in a Nutshell

```
dec( uint64_t x
   , uint64_t y) {

  uint64_t z = 0;

  if(x < 100) {
    if(y < x) {
      z = x-y;
    }
    else {
      z = x;
    }
  else {
    z = 42;
  }
ASSERT(z<100);
}
```

Proof for all values, no false-positives (unlike static analysis)

Not runtime checks or code instrumentation

Prove:

```
x<100 and y<x implies z=x-y<100   ✓
x<100 and y≥x implies z=x<100     ✓
x≥100         implies z=42<100    ✓
```

Galois Technologies

High-Assurance ASN.1 Workbench (HAAW)

- ASN.1 compiler, interpreter, automated test coverage
- Funded by U.S. Government for security-critical application

Test | Interpret | Compile

Software Analysis Workbench (SAW)

- Symbolic analysis for Java, C, C++…
- Open-source: http://saw.galois.com/
- In use by government, Amazon, others

amazon web services | Security Blog

Typically, formal verification can be tedious and is performed as research by skilled specialists using mathematical toolsets. As a part of our commitment to automated reasoning, we have contracted with Galois to simplify this process and make it more developer friendly. Combining a domain-specific language called Cryptol and a software analysis tool called SAW, Galois has produced a tool chain that allows us to formally verify important aspects of s2n.

https://aws.amazon.com/blogs/security/automated-reasoning-and-amazon-s2n/

Project Results

Release to NHTSA in January 2017

- SAE J2735 BSM (ver. MAR2016) encoder/decoder using our ASN.1 compiler, HAAW
- Verification with SAW of the Basic Safety Message, Part I (BSMCoreData)
- Scientific report, experience, recommendations

High Assurance ASN.1 Workbench (HAAW)

- hasni – high assurance ASN.1 interpreter
  - Load, type check, and browse ASN.1 specifications
  - Encode ASN.1 values to octet strings
  - Decode octet strings to ASN.1 values
  - Generation of random data that conforms to ASN.1 types
  - Round-trip (encode-decode) tests of user-defined/generated values
- hasnc – high assurance ASN.1 compiler
  - Generates C code encoders and decoders

SAW Example: Find First Set Bit

Find first set bit

```c
uint32_t ffs1(uint32_t w) {
  int c, i = 0;
  if(!w) return 0;
  for(c = 0; c < 32; c++)
    if((1 << i++) & w)
      return i;
  return 0;
}
```

```c
uint32_t ffs2(uint32_t w) {
  uint32_t r, n = 1;
  if(!(w & 0xffff))
    { n += 16; w >>= 16; }
  if(!(w & 0x00ff))
    { n += 8;  w >>= 8; }
  if(!(w & 0x000f))
    { n += 4;  w >>= 4; }
  if(!(w & 0x0003))
    { n += 2;  w >>= 2; }
  r = (n+((w+1) & 0x01));
  return (w) ? r : 0;
}
```

SAW Example: Find First Set Bit

ffs_llvm.saw

```
m <- llvm_load_module "ffs.bc";
ref <- llvm_extract m "ffs1" llvm_pure;
imp <- llvm_extract m "ffs2" llvm_pure;
time (prove_print abc {{ \x -> ref x == imp x }});
```

Output

```
# saw ffs_llvm.saw
Loading module Cryptol
Loading file "ffs_llvm.saw"
Time: 0.030429s
Valid
```

High-Assurance Cyber-Military Systems (HACMS)

Can we prove the software is secure?

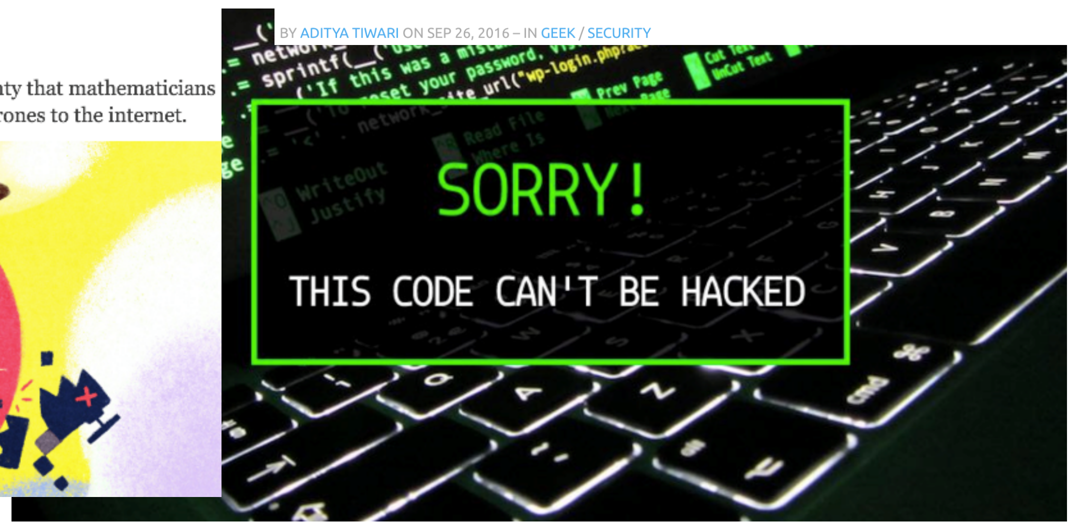
To Creating A

COMPUTER SECURITY

Hacker-Proof Code Confirmed

Computer scientists can prove certain programs to be error-free with the same certainty that mathematicians prove theorems. The advances are being used to secure everything from unmanned drones to the internet.

BY ADITYA TIWARI ON SEP 26, 2016 – IN GEEK / SECURITY



HACMS

- Galois developed a full-featured, provably secure, unpiloted air vehicle autopilot
- Vehicle + source given to U.S. Government-sponsored penetration team for 2-month evaluation
- Result: no software security flaws found that allowed attacker access

Can we achieve the same for V2V?