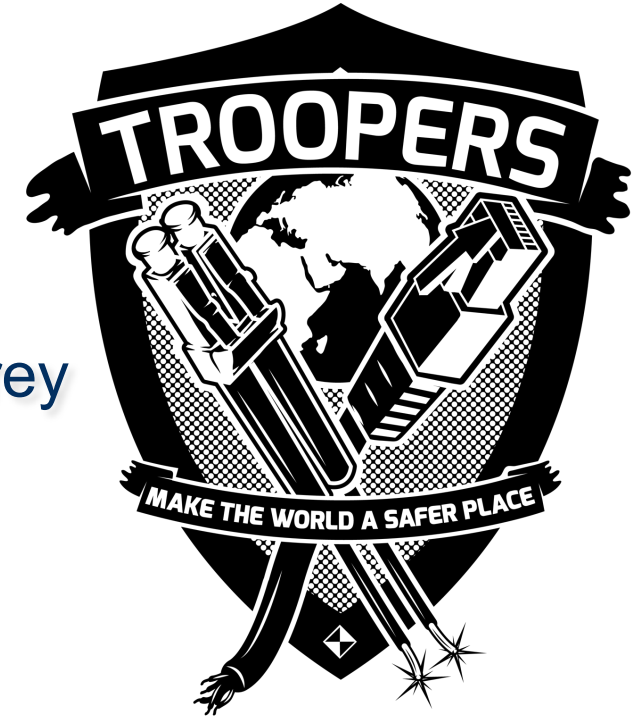
devastating capability, revolutionary advantage

Towards a LangSec-Aware SDLC

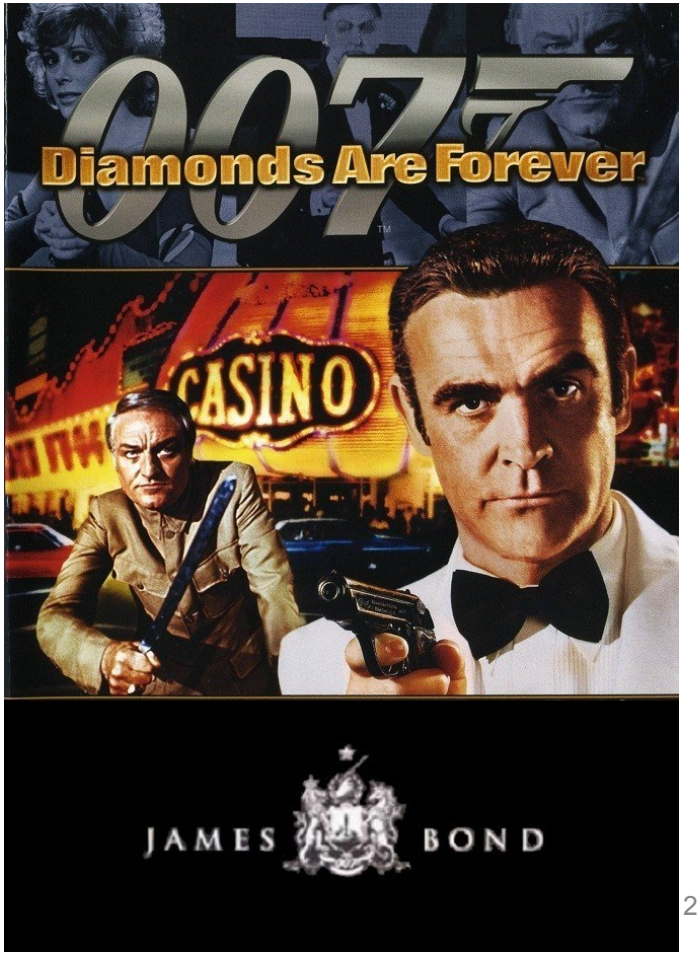
Mr. Jacob I. Torrey
TROOPERS'16
@JacobTorrey



Thanks for coming!

devastating capability, revolutionary advantage

- Hard to “compete” with the very talented Felix presenting exploits
- But…
- Exploits will be patched
- LangSec is FOREVER!



5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Who am I?

- Advising security researcher at Assured Information Security
  - Leads Denver, CO office
  - Leads the low-level computer architectures group
  - Plays in:
    - SMM
    - VMM
    - BIOS
- **LangSec Co-conspirator**
- Avid outdoorsman/traveler
  - SEE YOU AT THE RUN!

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Outline

- Problem, Introduction & Goals
- Background
  - Halting Problem & “Undecidability Cliff”
  - Verification
  - Parsing & Parser Differentials
- Programming Conventions
  - JPL Top 10
  - Strict Parsing
  - Maximal clarity, minimal inference (Verification-Oriented Paradigm)
  - Reduce complexity
- Tools for Enforcing Compliance
- Conclusions

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Problems

- Lack of objective and *comprehensive* metrics in security and software development has led to *ad hoc* development practices
  - Development based on “tradition” (“I’ve always done it that way”)
  - Biases towards your “cult’s” model
  - Current issues highlight the failing of the status quo

- More software being written now by more diverse group, secure composition is ***hard***
  - Pwned by a cloud-enabled light bulb!

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Introduction

- Last year's keynote by Sergey Bratus showed the theoretical underpinnings of *cyber insecurity*
  - "My Favorite Things"

- Field of Language-Theoretical Security (LangSec) aims to use a computational complexity argument to reduce vulnerabilities
  - Identify and kill off the "weird machines"
  - Exploits are *proofs* of insecurity

- Need a recipe book to augment software development life-cycle (SDLC) to "field" LangSec

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Talk Goals

- My goals for the audience after this talk:
  - Understand how LangSec has far-reaching impacts into software security
  - Have a framework to transition theory into practice
- For developers:
  - Recognize dangerous constructs
  - Avoid defect-prone semantics
- For project managers:
  - Audit compliance automatically (continuous integration for LangSec)
  - Sell the theoretical underpinnings of the changes to SDLC to increase corporate buy-in

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

LangSec SDLC Goals

- Bugs will happen, how your SDLC is designed dictates where in the process they'll be found

- By finding bugs sooner in the development process, defect rate in production goes down
  - Improving security
  - Reducing QA costs
  - Less "putting out fires" when production bugs are reported

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Design & Architecture

- See how happy they are?

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Development and Compile-time

- See how happy he is?

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Dynamic testing and QA

- Feeling lucky?

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Production bug reporting

- Yikes!

3rd-party notification and oversight

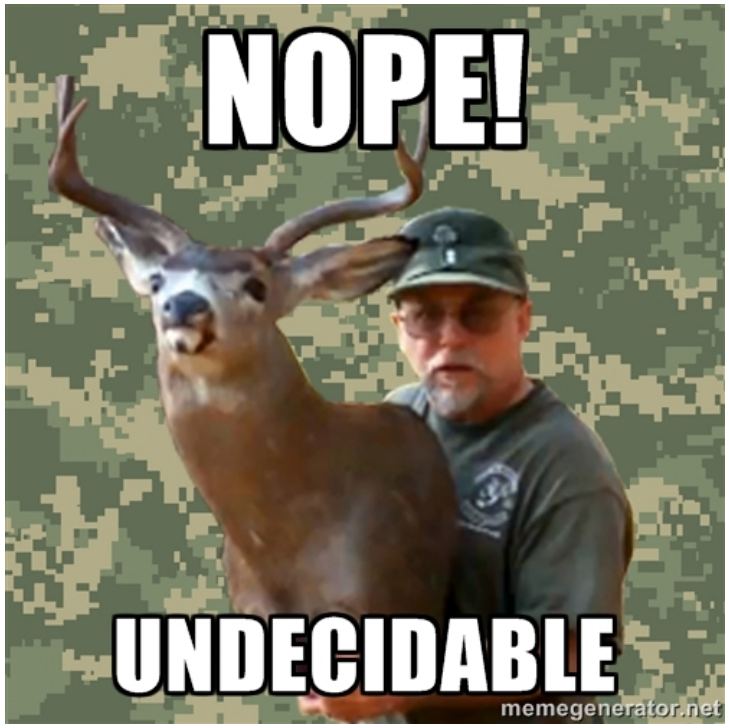
devastating capability, revolutionary advantage

- Hope you have a good lawyer!

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Background: Halting Problem

- Halting Problem
  - Determine if a program will halt on a given input
  - Pretty simple-sounding property to verify, right?



- In general, on Turing-complete programs, this is provably undecidable
- Undecidable: may run *forever* without returning

Background: Undecidability Cliff

- Complexity does not grow uniformly
  - As complexity increases, so too does verification difficulty
  - Once Turing-completeness is hit, you've fallen off the verification cliff

Background: Verification

devastating capability, revolutionary advantage

- Developers are not infallible
  - "Trust but verify"
- Static analysis look for bugs in source or binary without execution
  - Certain run-time semantics lost
- Dynamic analysis looks for bugs through instrumented execution
  - Challenge of coverage

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Background: Verification II

- Static analysis cannot infer all state or the "intent" of a programming construct
  - Mark Dowd's sendmail crackaddr() bug
  - A while() loop expanding email address
  - Semantics too abstracted for easy verification

- Dynamic analysis typically is underpinned by an NP-complete problem
  - As state-space grows, runtime quickly becomes untenable

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Background: Crackaddr

- Halvar Flake/Thomas Dullien proposed this as an example of a “hard problem” for verification
- Bug in while() loop expanding and matching “(“s and “<“s in email addresses
- Can be statically detected if looking for it, hard in practice

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Background: Parsing

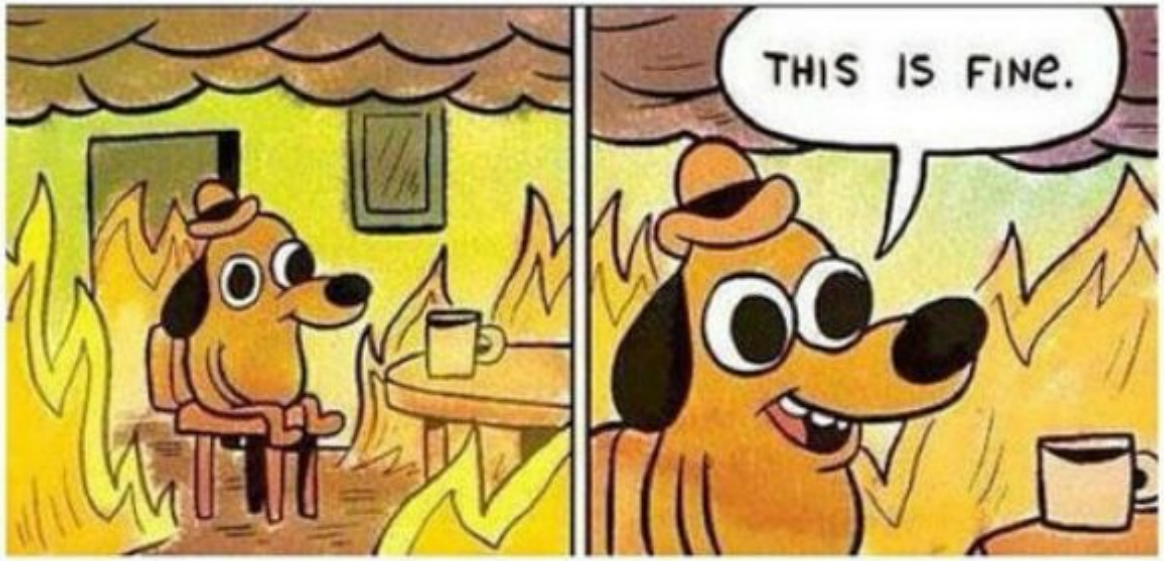
- Term that *should* encapsulate all boundaries in a program or interface where input is converted from one format to another
  - Reading in user input
  - RPC calls
  - Removing encapsulation
  - Reading data from files/network into structured, typed data

- A data specification should be generated first, and non-compliant input rejected!

See how happy she is the invalid data was rejected? →

Background: Parser Differentials

- Once complexity of input language falls off the undecidability cliff, intractable to determine if two parsers for same specification will ingest the same input identically
  - Bitcoin's OpenSSL ASN.1 BER parsing on 32 vs. 64 bit systems
  - SSL certificate parsing in Mozilla Firefox



Programming Conventions: MISRA-C

- Motor industry safety & reliability association's C programming guidelines for safety-critical code
  - Used for automotive control code

- Many tools (FOSS & commercial) to validate code bases!

Programming Conventions: JPL Top 10

- NASA's JPL has a "top ten" for safety-critical code
  - Goals: reduce defects, **ease verification** for code running in ***space***
  - Other than a few for readability/clarity, they map nicely to LangSec principles
  - Next few slides will detail the conventions and their theoretical underpinnings

JPL Top 10

- Restrict to simple to verify control-flow where possible (Rule #1)
  - Eliminate/minimize *goto*s, *longjmp*s and recursion
  - Makes the control-flow graph easier to analyze (acyclic)
  - Forces more programmer “intent” into syntax
- All loops must have an upper bound on iterations (Rule #2)
  - Must be possible for analyzer to determine termination (Walter recursion)
  - Some loops should be provably non-terminating (e.g., scheduling loop)

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

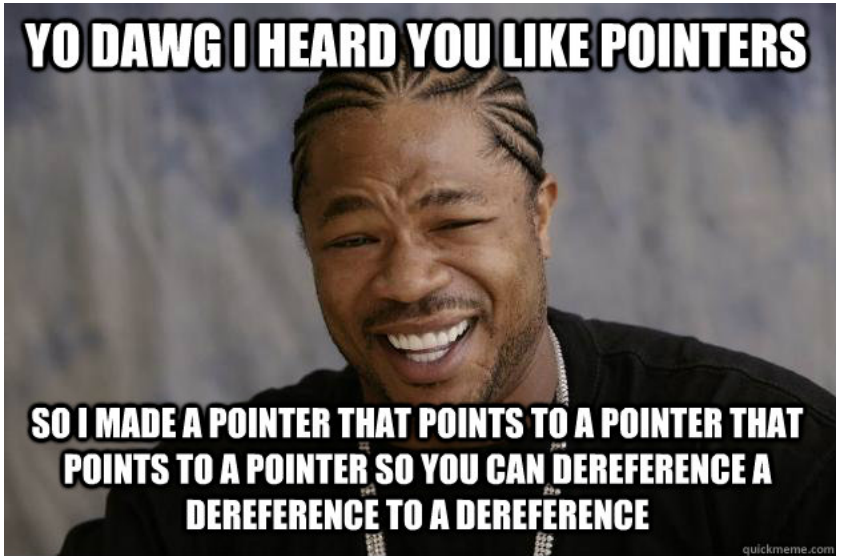
JPL Top 10 II

- Memory allocations should all be performed before business logic execution (Rule #3)
  - Makes the memory map easier to analyze
  - Verification is easier when memory allocations are guaranteed
  - Optionally, valgrind and LD_PRELOAD a malloc() wrapper that randomly fails
- Check all parameters in each function (Rule #7)
  - Ensure a parser bug did not propagate malicious input
  - Add fuzz testing to your unit testing QA step

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

JPL Top 10 III

- Limit use of pointers (especially recursive pointers) and disallow function pointers (Rule #9)
  - Function pointers shift problems from compile-time to run-time – this makes static analysis much less powerful
  - Recursive pointers leads to unbounded computations (e.g., PDF specification)



Strict Parsing

- Input data must be subjected to as much scrutiny as code is by a compiler!
  - When parsing is done in *ad hoc* fashion, the developer's assumptions may lead to vulnerabilities!
  - Invalid input ***must*** be rejected!
  - **NEVER** rewrite invalid input to "fix" it
    - You are now allowing input to operate the weird machine you've created

- Using a specified interface will ease interactions between teams and components
  - Jeff Bezos mandated that all Amazon software will act as a "service", lead to its dominance over the cloud market

Verification-Oriented Paradigm

- A meta development paradigm: aim to provide the maximum semantic information about intent to compiler and verification tools
  - If looping, aim for *induction* variables to be clearly identifiable (for/foreach instead of while)
    - Can improve performance due to better loop unrolling
  - Types should not be overloaded
    - E.g., MISRA-C requires *char* only be used for a single character, not for small integer values (which should be *int8_t*)
  - Minimize data scope
    - If an object cannot be referenced, cannot be corrupted

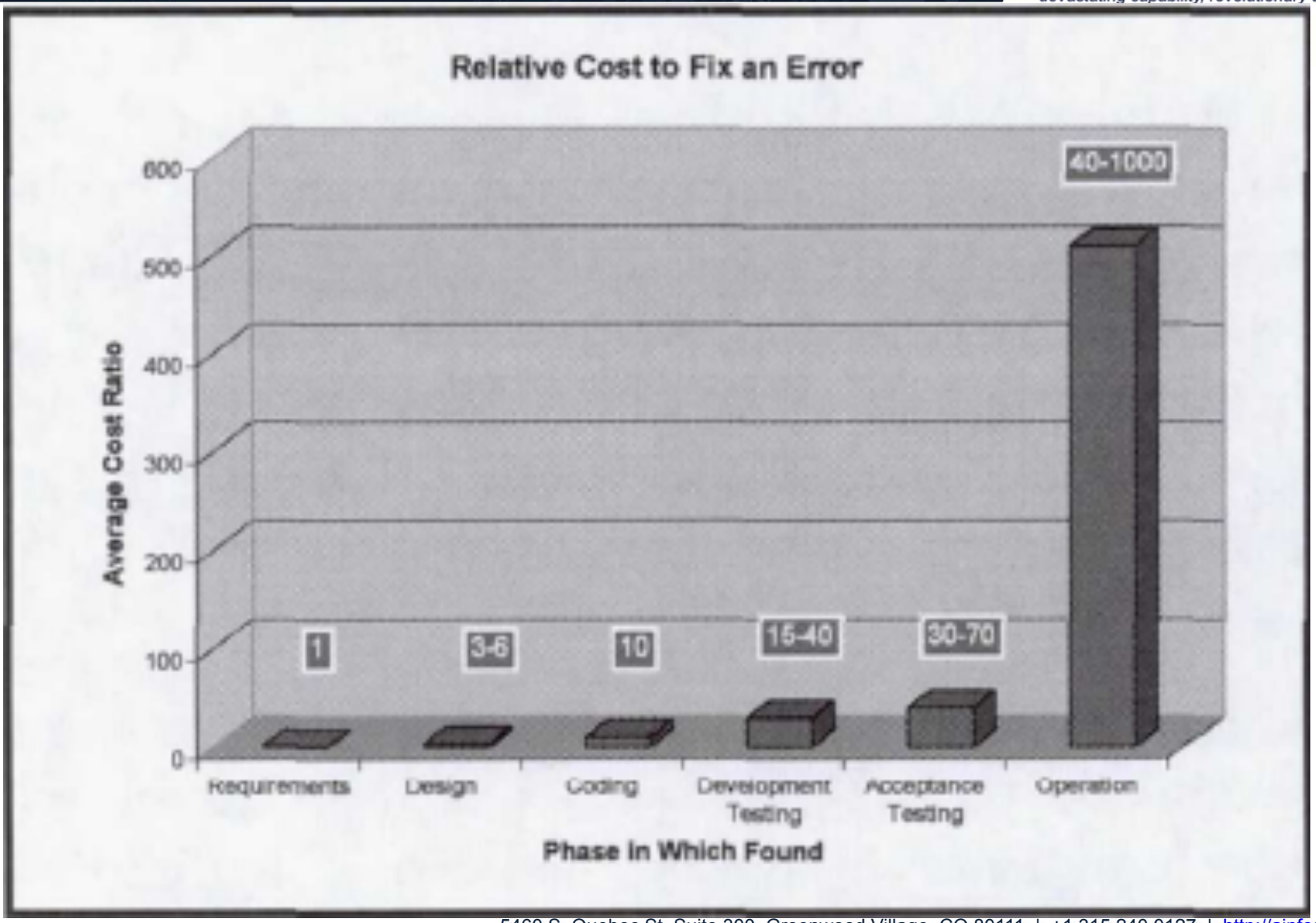
5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Verification-Oriented Paradigm II

- Benefits of VOP:
  - Code is more self-documenting, easier to read & review
  - Verification is easier
  - More bugs can be discovered at compile/unit testing time rather than patching run-time code

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Verification-Oriented Paradigm III



Verification-Oriented Paradigm IV

- Example of pushing bug detection to compile-time:
  - if (variable == CONSTANT) { …
  - versus
  - if (CONSTANT == variable) { …
- Semantically equivalent when implemented correctly
- If the second = is omitted:
  - First will compile, yielding unexpected results
  - Second will fail to compile

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

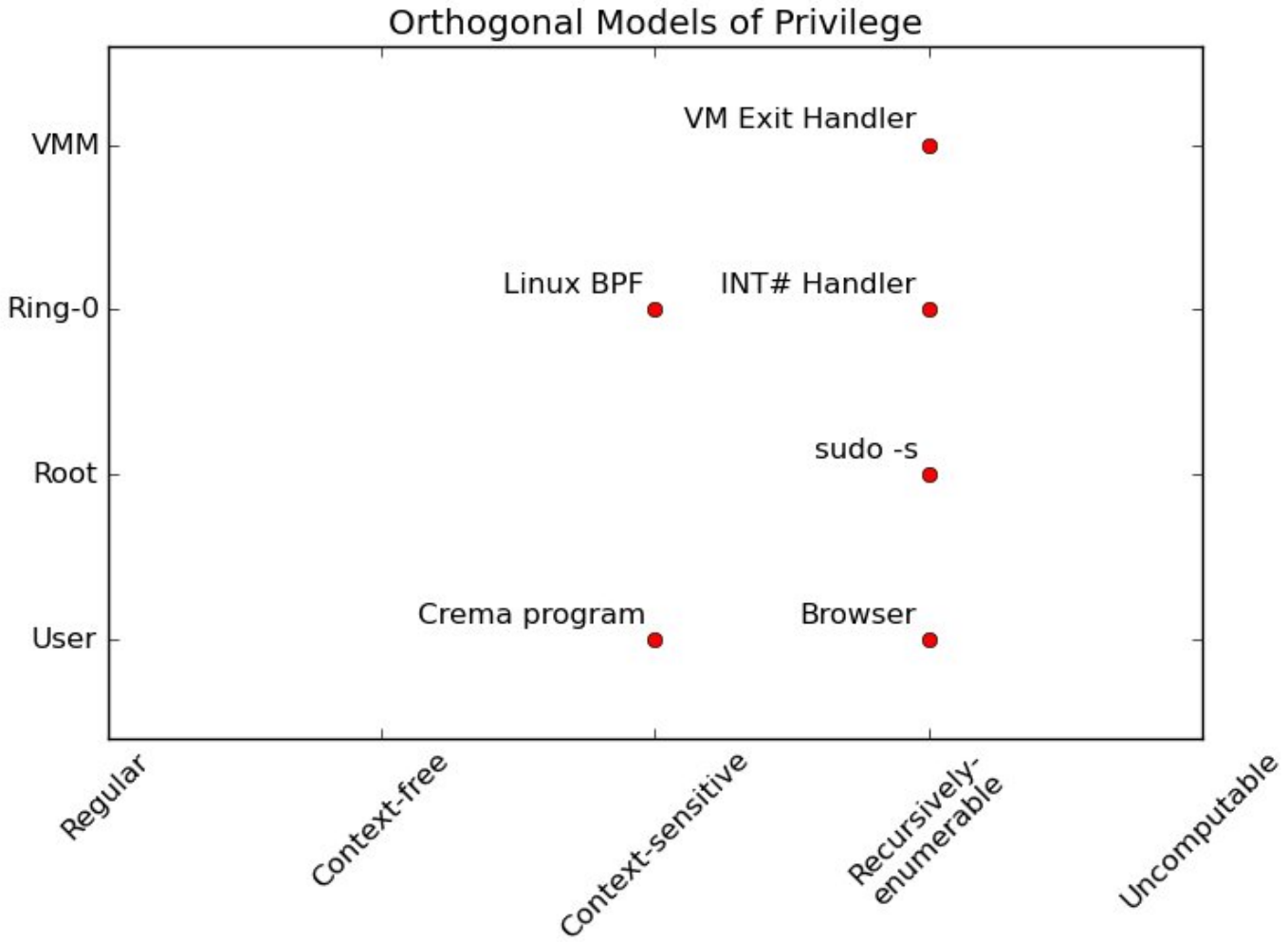
Reducing Complexity

devastating capability, revolutionary advantage

- As security practitioners, we aim to implement the "least privilege principle"
- "Don't run your IRC client or browser as root"
- Computational power is a form of privilege, and we're running everything with "root"
  - AV relies on having more privilege than malware
  - Doesn't work without defender's advantage

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Reducing Complexity II



5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Reducing Complexity III

- More complex programs lead to more bugs
  - More chances of programmer error
  - Less chances of detection in testing, QA and analysis
- JPL Rule #2 to limit looping will restrict state-space growth, improving verification
- In IEEE LangSec workshop, *Crema* showed the verification benefits from bounded execution
  - Very few computations need unbounded looping
  - Ex: seL4 manually segregated bounded and unbounded to formally prove OS correctness

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Tools for Enforcement

- Development guidelines and coding conventions are excellent so long as they are followed
  - Need to have audit capabilities
  - “Trust but verify”
- A good SDLC formalizes development process to allow checks of compliance
  - Code reviews
  - Unit & functional testing
  - QA
  - Commit hooks & continuous integration tools

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Tools for Enforcement

- Scale for effort and results:
  - Protocol design with computational complexity in mind
  - Programming conventions for maximum bounding
  - Static analysis
  - Runtime testing:
    - Dynamic analysis
    - Fuzzing
    - Unit tests
  - Production bug reporting / bug bounties
  - Getting on front page of newspaper for breach

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Developing with Semantics

- Aim to put as much semantic mindset in your code as possible
  - For-each macro to create common looping structure
- Not only valuable for verification, also for readability
- Tools like cpp-check can help warn developers of common traps

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Crema

- Research project to create an open-source compiler for a provably-halting programming language and runtime
- Based on LLVM, can be embedded in C for parsers
- Familiar syntax
- Demonstrated security benefits

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Strict Parsing

- Many mainstream parser generator frameworks are designed with code in mind:
  - Lex/Yacc
  - ANTLR

- Can be used, but Hammer and Nom are designed with *data parsing* in mind
  - Simple parser constructor libraries in C and Rust, respectively

Strict Parsing II

- Design your data format specifications early, get buy-in from parties
  - Similar to interface planning

- When planning specifications, consider the complexity required to parse
  - IPv6 fragmentation and extended attributes is example
  - Specification adds new features, but hard to inspect while maintaining QoS

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Nail

- Nail is an effort by Julian Bangert et al to automatically generate parsers from grammar description
  - Can reverse and output structured data to input format
  - Automatically can handle length and offset fields
- Reduces the risk of implementation or security concerns when parsing a structure into memory from untrusted input (and all inputs should be considered untrusted)
- Rejects invalid inputs

Nail II

- Create data format description, parsing function and structure will be automatically generated
  - Ambiguous parsing (e.g., whitespace) will prevent reversing parsing steps
  - Example: personnel database
    - Employee ID: uint32
    - Name: cstring
    - Manager bool: uint1
    - Remote employee: uint1

Nail III


```
person = {
        id uint32
        name <uint8='"'; many int8 | 'a'..'z'; uint8='"'>
        manager uint1
        remote uint1
}
db = {
        records sepBy uint8=',' (many person)
}
```


5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Nail IV


```
person *parse_person(NailArena *arena, NailStream *data);
db *parse_db(NailArena *arena, NailStream *data);

int gen_person(NailArena *tmp_arena,NailOutStream *out, person * val);
int gen_db(NailArena *tmp_arena, NailOutStream *out, db * val);
```

```
struct person {
    uint32_t id;
    struct {
        int8_t*elem;
        size_t count;
    } name;
    uint8_t manager;
    uint8_t remote;
}
;
```

```
struct db {
    struct {
        struct {
            person*elem;
            size_t count;
        }*elem;
        size_t count;
    } records;
};
```

LLVM/Clang Static Analyzer

- Part of Apple's Xcode IDE, but can be used on other platforms in standalone mode
- scan-build replaces the CC environment variables and performs static checks for common programming errors
- scan-view provides web UI to explore found bugs

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

LLVM/Clang Static Analyzer

devastating capability, revolutionary advantage

- In checking for NULL from malloc, I “forgot” to break out and handle the error

- Ran scan-build while building and then scan-view to see the bug report:

LLVM/Clang Static Analyzer

devastating capability, revolutionary advantage

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main(int argc, char **argv)
5  {
6    char *foo = malloc(32);
       1  'foo' initialized here →
7    if (!foo) {
         2  ← Assuming 'foo' is null →
         3  ← Taking true branch →
8      fprintf(stderr, "Malloc returned 0!\r\n");
9    }
10   foo[31] = '\0';
         4  ← Array access (from variable 'foo') results in a null pointer dereference
```


5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

LLVM/KLEE Dynamic Analyzer

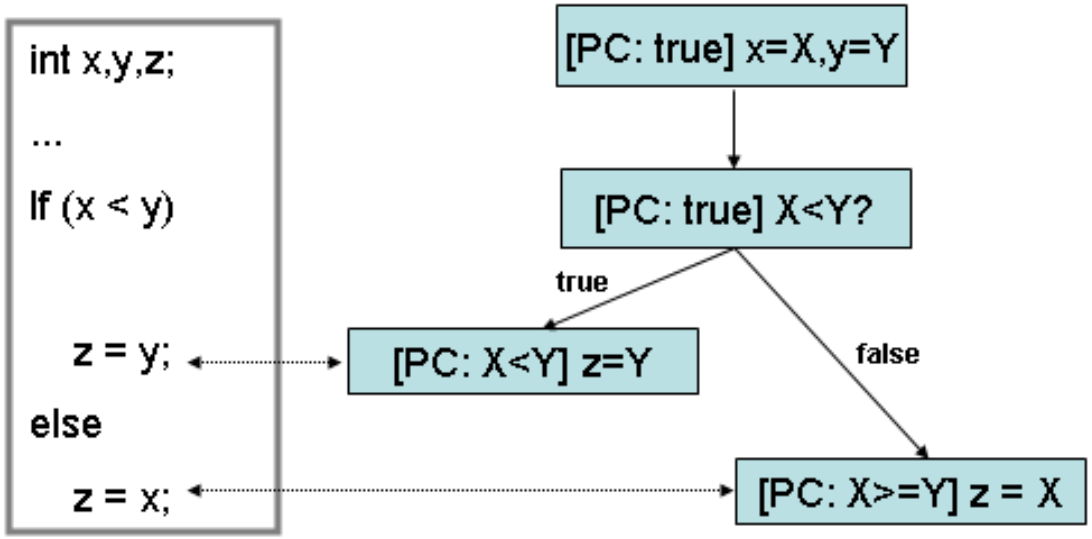
- Another tool based on LLVM intermediate representation
- Performs dynamic analysis through *symbolic computation* to gain high-coverage of code
- Can find crash cases, or be used to verify semantic equivalence between different code bases
  - Can be used to check for parser differentials

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

LLVM/KLEE Dynamic Analyzer

- Whenever a branch is reached, *both* paths are executed, maintain the constraints on the input to reach that state

- SMT/SAT solver used to create concrete value



AFL-Fuzz

- Fuzzer that compiles in instrumentation to improve coverage
  - Provides afl-gcc
  - Provides tools to minimize crashing input case
  - Can run distributed
- Provided input corpus will mutate
- Random, thus can get “trapped” in loops, hard to “steer”

But wait! There's more!

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Enforcement Tool: Sledge Hammer

Enforcement Tool: Sledge Hammer

devastating capability, revolutionary advantage

- Open source suite of tools that a sadistic program manager can run on code base to audit compliance and safety
  - Could be added to CI

- Combination of:
  - LD_PRELOAD to simulate memory management failures
  - Header file with to poison “bad” *ad hoc* parsers and semantically vague looping constructs
  - Automatic symbolic testing for parameter verification **on every function**

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

LD_PRELOAD

- Environment variable to temporary alter library load order
- Allows easy override of library functions on existing binaries
  - $ LD_PRELOAD=./libsledge.so program
- malloc(), calloc(), realloc(), etc. can fail and return NULL pointer, must be checked before dereference
  - libsledge.so replaces these functions with ones that fail with designated probability

LD_PRELOAD II

- Demo

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Poison Pill

- Header file to be included (or with -include) that "poisons" banned keywords
  - Unsafe string operations
  - While loops
  - Non-strictly parsed input (e.g., cin/scanf)
- Will forbid compilation if keywords are found
- Rapid way to audit and quickly ensure compliance (or valid reason for usage)

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

KLEE-Unit

KLEE-Unit

- In order to provide assurances that input is sanitized and function arguments vetted
- Framework to couple KLEE (symbolic execution) with unit test methodology
- Will identify all functions, their arguments and create test harness for each to be SE'd
- Crashes can be analyzed to determine cause and fix

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Function Fuzzing II

- Demo

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Sledgehammer Details

- https://github.com/ranok/sledgehammer

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Concluding Remarks

- At scale, and as perimeter grows weaker, network security must shift to more hardened applications
  - Alex Stamos: AppSec is “eating” security
  - Jacob Torrey: LangSec is “eating” AppSec

- Google’s Beyond Corp shows that perimeter leads to false sense of security, and that well-built applications can stand on their own

- “To err is human; to be caught at compile-time; divine”

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Concluding Remarks II

devastating capability, revolutionary advantage

- Whatever languages and tooling your organization uses, aiming to maximize the semantic quality and verifiability will yield positive results
- Not just for security, but:
  - Less expensive through reductions in run-time bugs (less QA)
  - Faster through more semantics for compilers to use during optimization
- Currently state of software quality highlights need to adjust strategy

5460 S. Quebec St, Suite 300, Greenwood Village, CO 80111 | +1 315.240.0127 | http://ainfosec.com

Questions


YOU COME TO ME AT RUNTIME
TO TELL ME THE CODE YOU ARE EXECUTING DOES NOT COMPILE
memegenerator.net

References

- Alex Stamos talk:
  - https://www.youtube.com/watch?v=-1kZMn1RueI
- AFL-fuzz:
  - http://lcamtuf.coredump.cx/afl/
- Nail:
  - https://people.csail.mit.edu/nickolai/papers/bangert-nail-langsec.pdf
- Nom:
  - https://github.com/Geal/nom
- Hammer:
  - https://github.com/UpstandingHackers/hammer

References II

- LangSec:
  - http://langsec.org/
- JPL Top 10:
  - http://spinroot.com/gerard/pdf/P10.pdf
- MISRA-C:
  - http://caxapa.ru/thumbs/468328/misra-c-2004.pdf
- Crema:
  - http://spw15.langsec.org/papers.html#ver
- Crackaddr():
  - https://bytebucket.org/mihaila/bindead/wiki/resources/crackaddr-talk.pdf