

Overview and Usage of Binary Analysis Frameworks

Florian Magin fmagin@ernw.de

whoami

- Security Research at ERNW Research GmbH from Heidelberg, Germany
- Organizer of the Wizards of Dos CTF team from Darmstadt, Germany
- Reach me via:
 - Twitter: @0x464D
 - Email: fmagin@ernw.de



Who we are

- Germany-based ERNW GmbH
 - Independent
 - Deep technical knowledge
 - Structured (assessment) approach
 - Business reasonable recommendations
 - We understand corporate
- Blog: *www.insinuator.net*
- Conference: *www.troopers.de*



Agenda

- I. General Intro to Program Analysis
- II. Concepts and Analysis Techniques
- III. Tools and Frameworks
- IV. Applications and Examples
- V. Takeaway/Recap

What is Automated Binary Analysis?

What is Automated Binary Analysis?

➡ Binary Analysis performed by algorithms

Wait, isn't that impossible?

- It's impossible to generally tell if a program halts for a given input (Halting Problem)
- Also Rice's Theorem
- Also, what exactly are we even looking for?
 - Crashes?
 - Memory Corruptions?
 - Logic Errors?

Bit of History

- The ideas themselves are 40 years old
 - Robert S. Boyer and Bernard Elspas and Karl N. Levitt, SELECT--a formal system for testing and debugging programs by symbolic execution, 1975
- Analysis is resource intensive
 - Cray-1 supercomputer from 1975 had 80MFLOPS (8MB of RAM)
 - iPhone 5s from 2013 produces about 76.8 GFLOPS (1GB of RAM)
- Advances in SMT solving around 2005 made it feasible

DARPA CGC

- Task: Develop a “Cyber Reasoning System”
- Big push in moving the ideas from academia to practicability
- Qualification Prize: \$750,000
- Final Prizes:
 1. \$2,000,000
 2. \$1,000,000
 3. \$750,000



DARPA CGC

- CRS needs to:
 - Find vulnerabilities
 - Patch them
 - Partially exploit them
- Ran on 64 Nodes, each:
 - 2x Intel Xeon Processor E5-2670 v2 (25M cache, 2.50GHz) (20 physical cores per machine)
 - 256GB Memory
- 7 finalists, winner competed at DEFCON CTF
- It was better than some of the human teams some of the time



Agenda

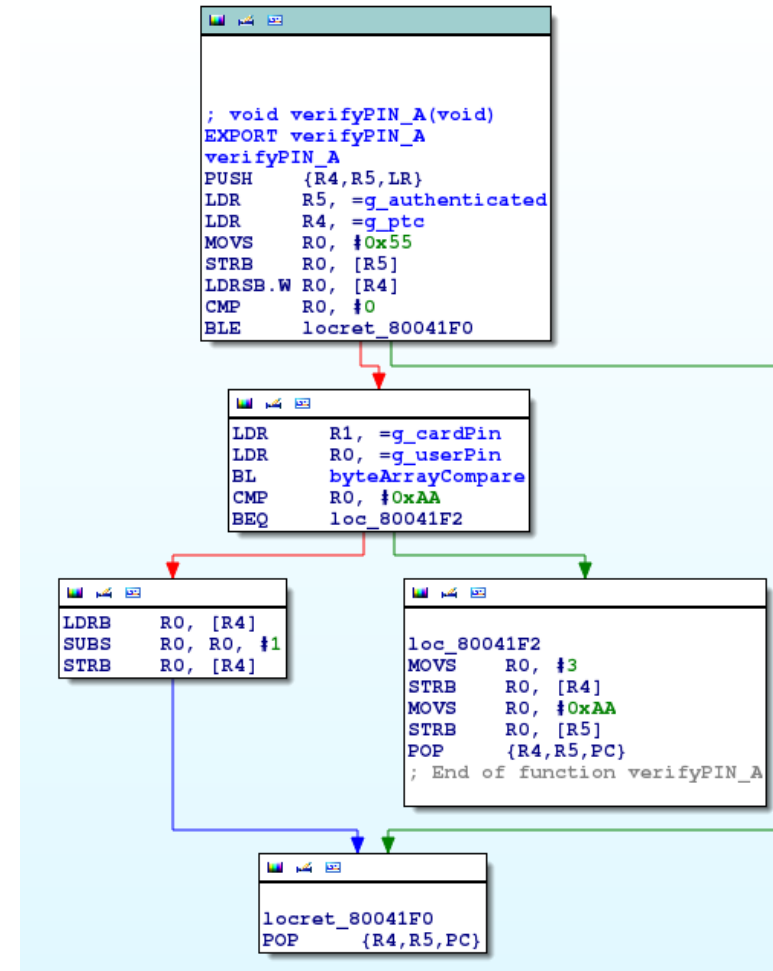
- I. General Intro to Program Analysis
- II. Concepts and Analysis Techniques
- III. Tools and Frameworks
- IV. Applications and Examples
- V. Takeaway/Recap

Intermediate Representations

- Every architecture is different
- What do we actually analyze?
 - Abstract/Common Representation
- Typical case of too many standards
 - VEX IR (used by Valgrind and angr)
 - Binary Ninja IR
 - LLVM IR
 - So many more
- Different IRs for different purposes
 - Easy to read (as a human)
 - Easy to analyze
 - Easy to optimize

CFG Recovery

- Recursively build a graph with jumps as edges and basic blocks as nodes
- Easy with calls and direct jumps
- But what about “jmp eax”?
 - Jump table
 - Callbacks/Higher Order Functions
 - Functions of Objects in OOP
- “Graph-based vulnerability discovery”



Data-Flow Analysis/Taint Analysis

- Track where data ends up
- Data dependencies
- Discover functions that handle user input
- Discover possible data exfiltration

Slicing

- Reducing the program statements to those dealing/changing with a specific variable from a specific point
 - Backward: All statements before the point
 - Forward: All statements after
- Static or dynamic
 - Static looks at all statements
 - Dynamic looks at statements in the execution trace

Slicing Example

```
int i;  
int sum = 0;  
int product = 1;  
for(i = 1; i < N; ++i) {  
    sum = sum + i;  
    product = product * i;  
}  
write(sum); //Slicing criterion  
write(product);
```



```
int i;  
int sum = 0;  
for(i = 1; i < N; ++i) {  
    sum = sum + i;  
}  
write(sum); //May or may not  
be included
```


Symbolic Execution

- Symbolic instead of concrete variables
- Following example is from a presentation about angr

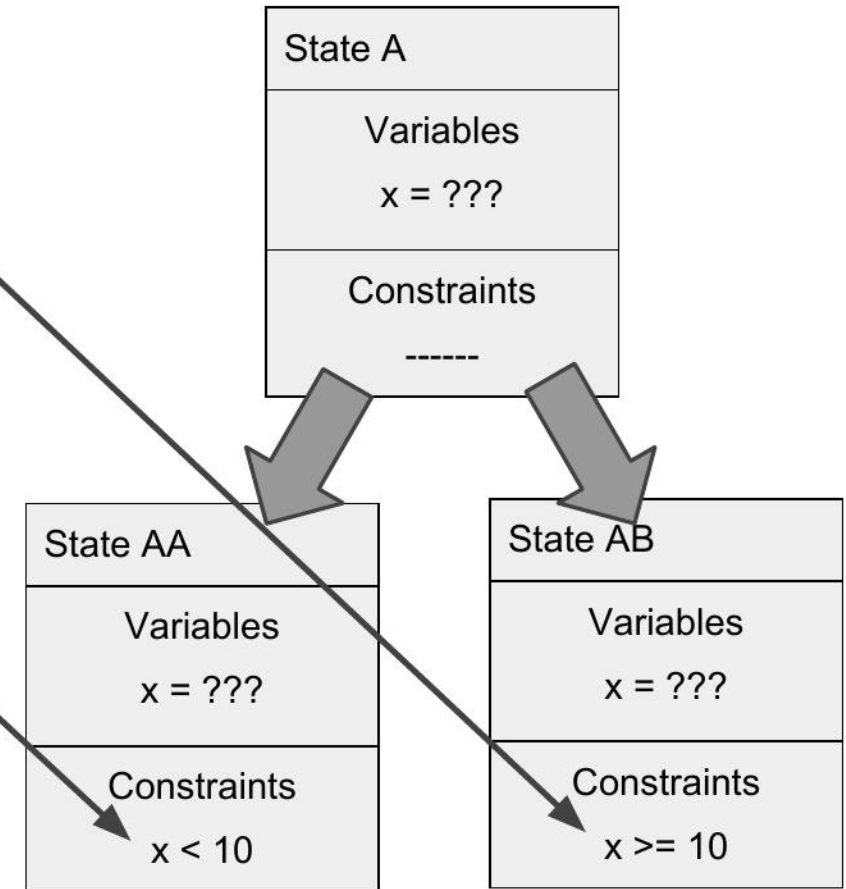
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



State A
Variables x = ???
Constraints -----



```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



State AA
Variables $x = ???$
Constraints $x < 10$

State AB
Variables $x = ???$
Constraints $x \geq 10$



```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

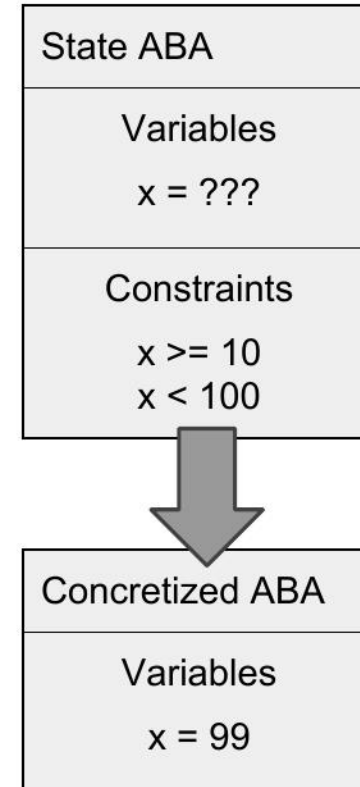
State AA
Variables x = ???
Constraints x < 10

State AB
Variables x = ???
Constraints x >= 10

State ABB
Variables x = ???
Constraints x >= 10 x >= 100

State ABA
Variables x = ???
Constraints x >= 10 x < 100

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



SMT/Constraint Solving

- Abstract your problem, model it, solve it
 - Can be as simple as in previous slide
 - Can be significantly more complicated
- Dedicated tools available
 - Complicated theory involved

Agenda

- I. General Intro to Program Analysis
- II. Concepts and Analysis Techniques
- III. Tools and Frameworks
- IV. Applications and Examples
- V. Takeaway/Recap

Z3 Theorem Prover

- Developed by Microsoft Research
- Effort by Microsoft to formally verify some parts of their products with it
 - Windows Kernel
 - Hyper-V
- MIT License
- Provides a SMT Solver and theories for
 - Bitvectors
 - Strings
 - Arrays
 - etc
- Seems to be the standard for program analysis

Microsoft®
Research

angr

- Most beginner friendly of all tools
- Written in Python
- Good Documentation
- Plenty of available research
- Used by Mechaphish (3rd at Darpa's CGC)
- Developed at University of California, Santa Barbara



Triton

- x86 and x86_64 only
- Designed as a library (LibTriton.so)
 - Should be easier to integrate into C Projects
- Has python bindings
- Not focused on automating but assisting
- Sponsored by Quarkslab



Others

- Bitblaze
- bap: Binary Analysis Platform
 - Carnegie Mellon University/ForAllSecure
 - Written in OCaml
 - Used by Mayhem (1st Place at Darpa's CGC)
- Miasm
- S2E(2)
- Microsoft Sage
- Manticore by Trail of Bits

Agenda

- I. General Intro to Program Analysis
- II. Concepts and Analysis Techniques
- III. Tools and Frameworks
- IV. Applications and Examples
- V. Takeaway/Recap

What to do with all this?

- These techniques don't scale
 - State Explosion
 - Constraint solving is generally NP-Complete
- Combine with something smart but slow: a human
- Combine with something dumb but fast: a fuzzer



Source:

<https://twitter.com/matalaz/status/580600098092105728>

Augmented Fuzzing

- Usual fuzzing to detect likely code paths
- Taint Analysis to discover what branch depends on what input to fuzz that
- Symbolic Execution with constraint solver to build input to take a specific new branch

Installing Angr

- Libraries modified by angr don't mix well with their originals
 - stripped down z3
 - forked libvex
 - etc.
- Run it isolated and officially supported
 - Python2 virtualenv (pip install angr)
 - Official Docker Container (docker pull angr/angr)

SMT Solving Example

```
void xmalloc(unsigned_int sz)
if(sz > PAGE_SIZE) {
    s = ((sz+PAGE_SIZE+1) &
        ~(PAGE_SIZE-1));
}
else {
    s = sz;
}
return malloc(s)
```

- sz is the parameter supplied to xmalloc()
- s is the parameter for malloc()
- PAGE_SIZE is 0x1000 or 4096
- Assumption: Allocating a buffer of certain size via xmalloc() and getting a smaller buffer from malloc() is bad

How to solve this?

```
void xmalloc(unsigned_int sz)
if(sz > PAGE_SIZE) {
    s = ((sz+PAGE_SIZE+1) &
        ~(PAGE_SIZE-1));
}
else {
    s = sz;
}
return malloc(s)
```

1. Get the function into Python
2. Formulate the problem
3. Let the solver do its magic

Get the function into Python

- Handwrite it in python
- If it's just some simple logic
it's often copy-paste able
from source or decompiler
- Just remember that your
variables are bit vectors
- Foreign Function Interface
- See next Slide

Foreign Function Interface

- Automagically import binary functions
 - angr detects calling convention
 - Maps python types to binary representation
- Call them from python
 - With concrete values
 - With symbolic value

```
>>> import angr
>>> b=angr.Project('/path/binary')
>>> f = b.factory.callable(address)
>>> f?
```

```
Type: Callable
[...]
```

Callable is a representation of a function in the binary that can be interacted with like a native python function.

```
[...]
```




DEMO: Finding Bugs with Math

DEMO: Finding Bugs with Math

```
def f(x):  
    return (x+PAGE_SIZE+1) & ~(PAGE_SIZE-1)  
solver = claripy.Solver()  
sz = claripy.BVS('sz', WORD_SIZE)  
constraints = [ sz > PAGE_SIZE,  
                f(sz) < sz,  
                f(sz) != 0 ]  
solver.add(constraints)  
return solver.eval(sz, 3)
```

Inversing Functions

- Get an input so that a function returns a certain value
- Function can be from Python or from binary(see FFI)
- $f(x, y) \rightarrow (x * 3 \gg 1) * y$
- $f(?, 0x42) \rightarrow 0x76E24$

Inversing Functions

- Get an input so that a function returns a certain value
- Function can be from Python or from binary(see FFI)
- $f(x, y) \rightarrow (x * 3 \gg 1) * y$
- $f(?, 0x42) \rightarrow 0x76E24$

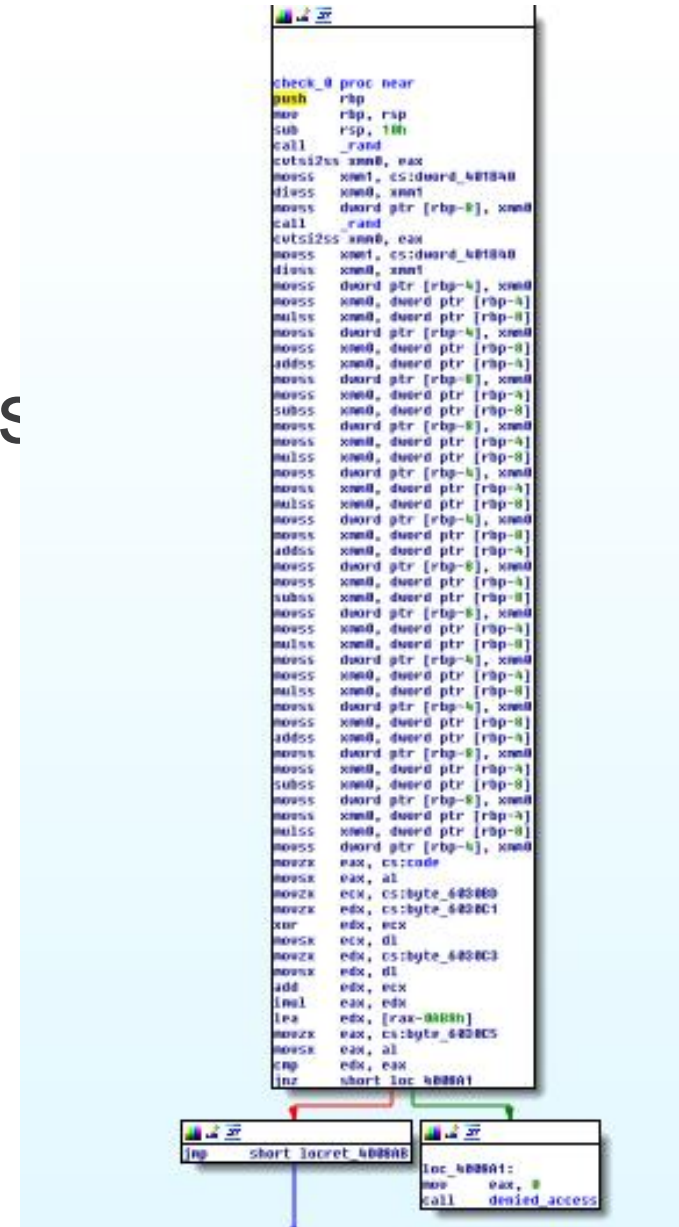
DEMO

Inversing Functions

- Get an input so that a function returns a certain value
- Function can be from Python or from binary(see FFI)
- $f(x, y) \rightarrow (x * 3 \gg 1) * y$
- $f(?, 0x42) \rightarrow 0x76E24$
- We got two possible solutions
 - 0x1337 (intended)
 - 0x5555555555555688c which returns 0x76e24
-> Integer Overflow

Automagical Solving of Crackmes

- Binary that takes some user input
 - stdin
 - argv
 - Some file
- Checks it against constraints
- Determines if it's valid



Automagical Solving of Crackmes

- Binary that takes some user input
 - stdin
 - argv
 - Some file
- Checks it against constraints
- Determines if it's valid

DEMO

Automagical Solving of Crackmes

- Binary that takes some user input
 - stdin
 - argv
 - Some file
- Checks it against constraints
- Determines if it's valid
- We just declare that input as symbolic
- Choose a starting point and explore the possible paths from there
- Solve for an input that brings us down the wanted path ➡ that's the solution

Debugging capabilities

- Breakpoints with callbacks before or after:
 - Instructions or address
 - Memory read/write
 - Register read/write
 - Many others
- Hooks
 - Optimized libc functions
 - Own Python Code

```
>>> import angr, simuvex
>>> b=angr.Project('/path/binary')
>>> s = b.factory.entry_state()
>>> def debug_func(state):
...     print 'Read',
...     state.inspect.mem_read_expr, 'from',
...     state.inspect.mem_read_address
>>> s.inspect.b('mem_read',
...             when=simuvex.BP_AFTER, action=debug_func)
```

Anti-Anti-Debugging

- angr is not a debugger
 - Some anti debug tricks wont work
 - Others accidentally break angr in other ways
- Simuvex or Unicorn can be used as an emulator
 - Breakpoints without the program noticing
 - Invisible Hooks
- Overall it needs a different approach

Agenda

- I. General Intro to Program Analysis
- II. Concepts and Analysis Techniques
- III. Tools and Frameworks
- IV. Applications and Examples
- V. Takeaway/Recap

Takeaway

- Reverse Engineering is already regarded as some arcane art
- Adding tons of complicated math seems to make it true black magic
- BUT it actually makes many things easier if you know which parts to treat as magic and which to understand
 - You don't need to understand how Z3 works

Interested in the theory and want to learn more?

- Extensive List of Materials
<http://www.msreverseengineering.com/program-analysis-reading-list/> for self learning
- If possible check your uni for lectures on the topics in the above list
 - Some aspects are part of the “Computer Science” curriculum
 - Other aspects are typical math topics

Want to contribute without getting too deep into the theory?

- Plenty of work that would be cool to have but no one is doing yet
 - Proper UIs to visualize the concepts (angr-management needs more work)
 - <https://github.com/angr/angr-doc/blob/master/HELPWANTED.md>
- Documentation could be better
 - Triton lacks integrated python doc like angr
- Building tools can be annoying
 - Package it for your favorite distro

Thank you for your attention

Any Questions?



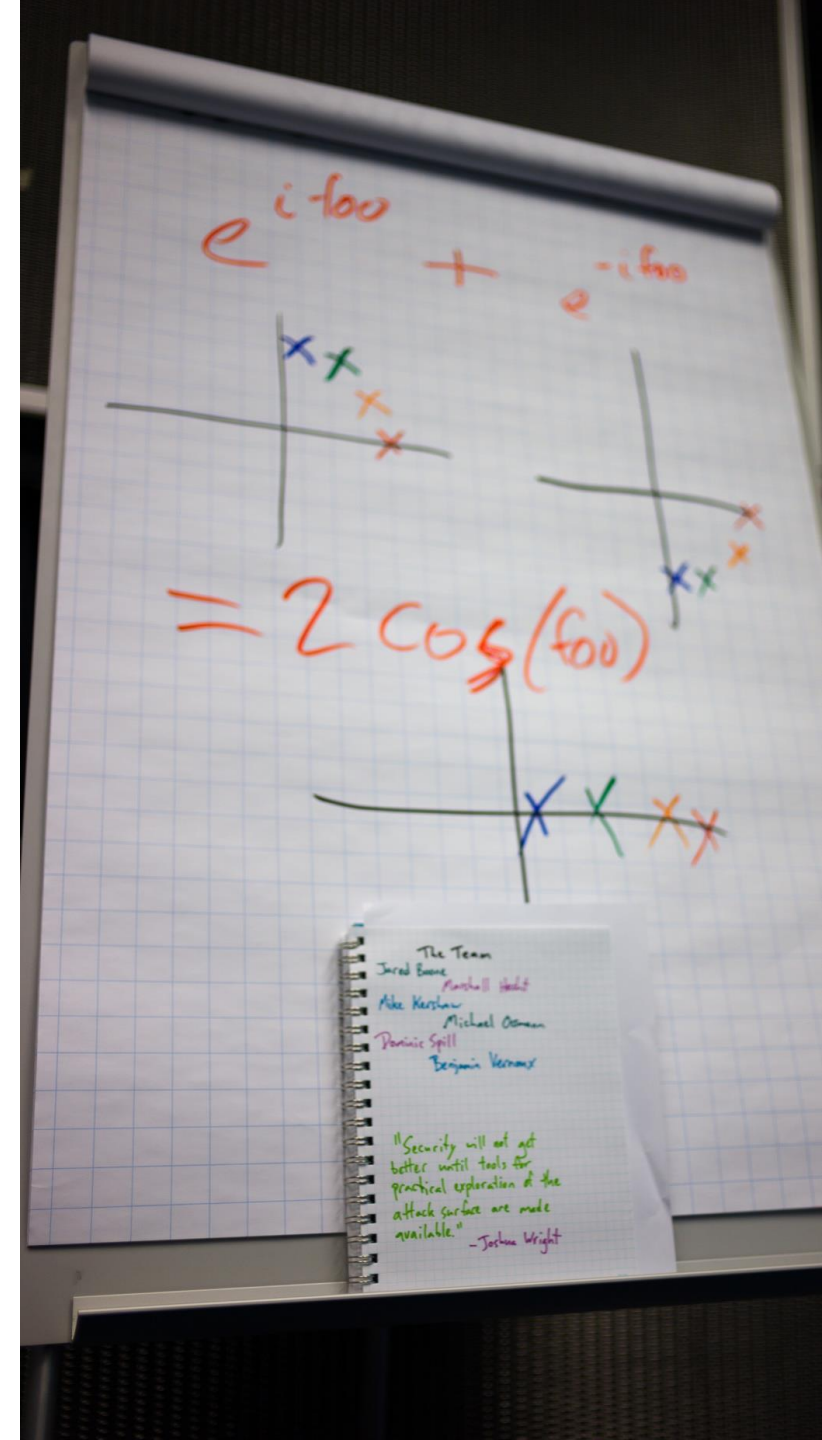
fmagin@ernw.de



Florian Magin @0x464D

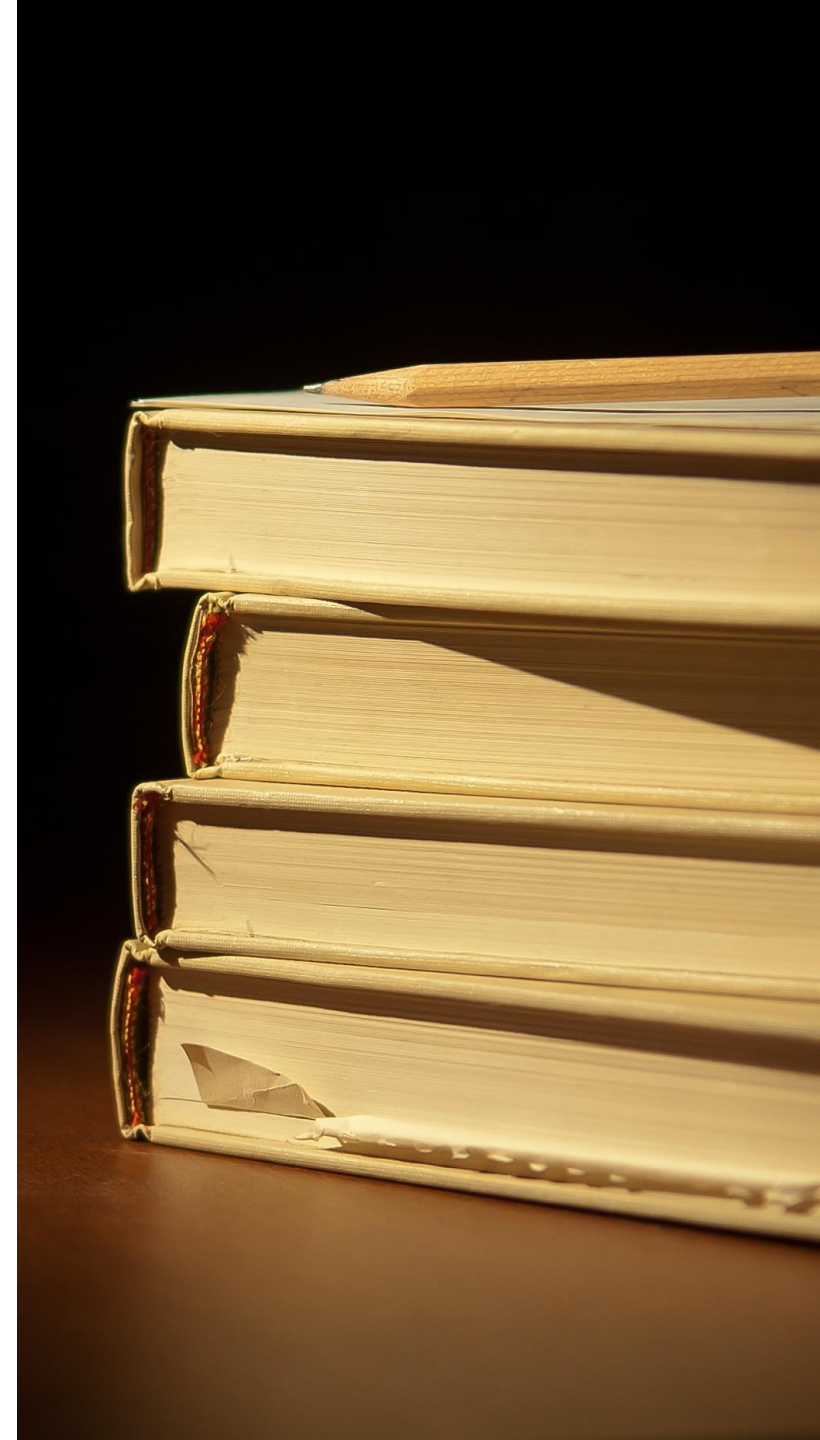
www.ernw.de

www.insinuator.net



References & Literature

- “SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis”
- <https://docs.angr.io/>



Side Note: LLVM Compiler Infrastructure

- Own IR (LLVM IR)
- Own symbolic execution engine (KLEE)
- Own constraint solver (Kleaver)



Value-Set Analysis

- Approximate program states
- Values in memory or registers
- Reconstruct buffers
- Can be enough to detect buffer overflows
- Helps with complex CFG recovery