Overview and Usage of Binary Analysis Frameworks

Florian Magin fmagin@ernw.de

ERNW
providing security.

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

ERNW
providing security.

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*

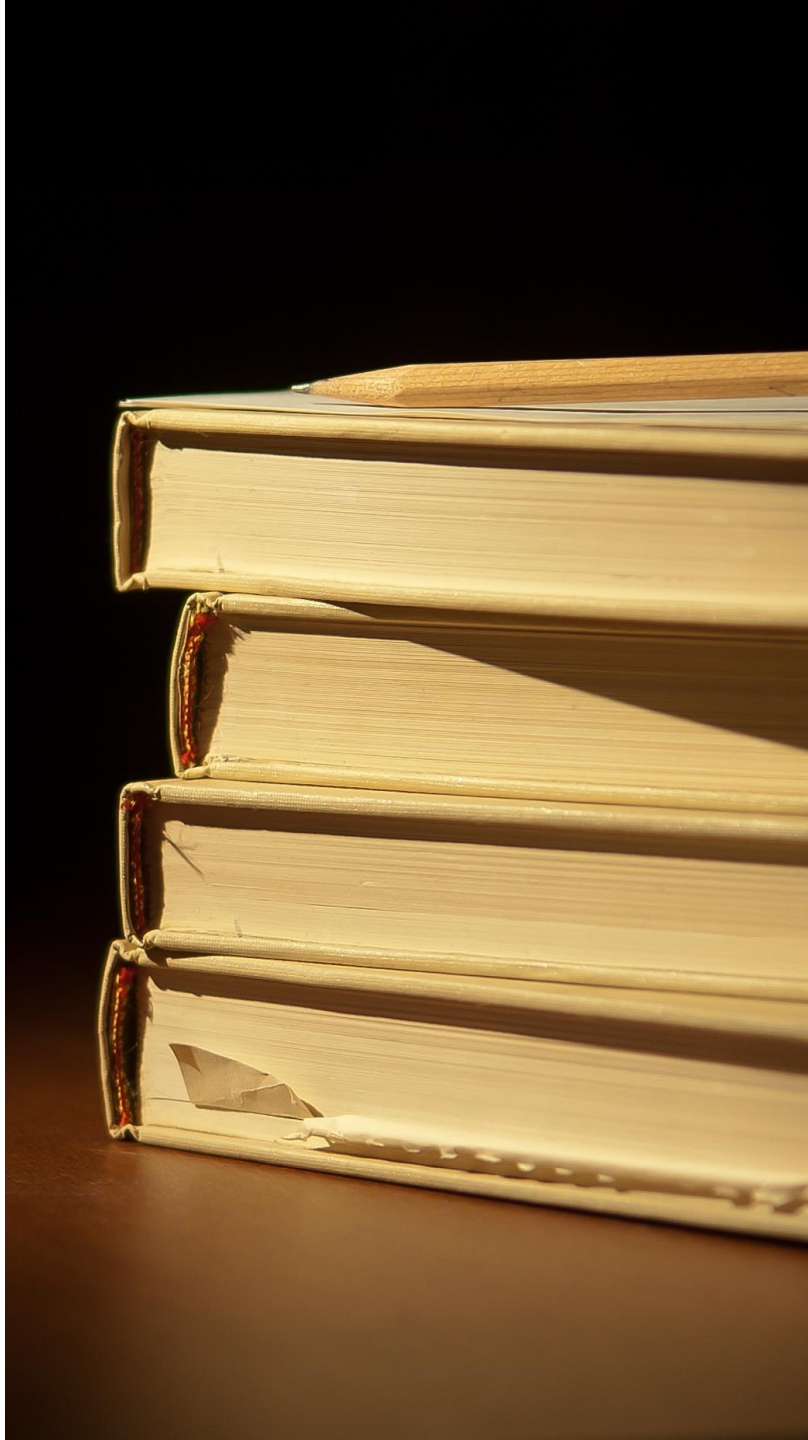
ERNW
providing security.

Agenda

What actually is automated analysis?

How does it work?

What else are some of the frameworks capable of? (In this case angr)



ERNW
providing security.

What is Automated Binary Analysis?

ERNW
providing security.

What is Automated Binary Analysis?

-> Binary Analysis performed by algorithms

ERNW
providing security.

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?

ERNW
providing security.

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)

ERNW
providing security.

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

ERNW
providing security.

DARPA CGC

- CRS needs to:
  - Find vulnerabilities
  - Patch 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

ERNW
providing security.

How do they work?

Overview of the basic concepts

ERNW
providing security.

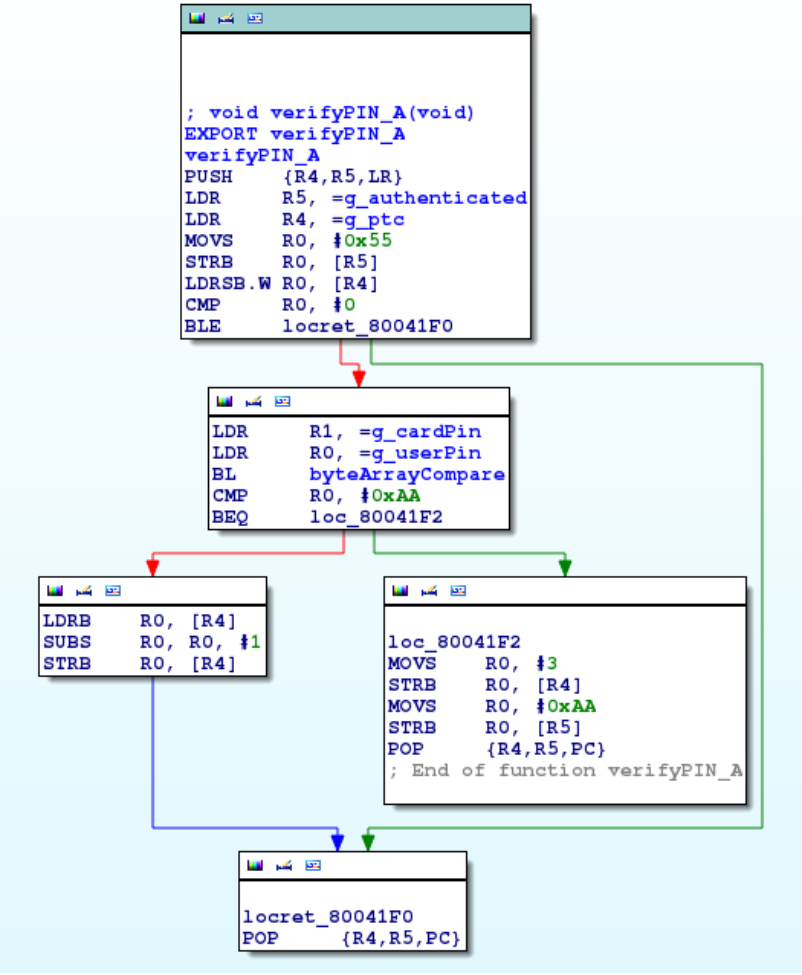
Intermediate Representations

- What do we actually analyze?
- Every architecture is different
  - ➡ Common Representation
- Typical case of too may standards
  - VEX IR (used by Valgrind and angr)
  - Binary Ninja IR
  - LLVM IR
  - So many more

ERNW
providing security.

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"



ERNW
providing security.

Value-Set Analysis

- Approximate program states
- Values in memory or registers
- Reconstruct buffers
- Can be enough to detect buffer overflows
- Research paper is in the last slide (18 Pages)

ERNW
providing security.

Data-Flow Analysis/Taint Analysis

- Track where data ends up
- Data dependencies
- Discover functions that handle user input
- Different granularity:
  - Bits
  - Bytes

ERNW
providing security.

Constraint Solving

- A LOT of Math involved
- Highly simplified:
  - a = 5
  - b = 15
  - x < b && x > a
  - x = ?

ERNW
providing security.

Z3 Theorem Prover

- Developed by Microsoft Research
- Microsoft uses it to formally verify some parts of their products
  - Windows Kernel
  - Hyper-V
- MIT License
- Provides a SMT Solver

Microsoft®
Research

ERNW
providing security.

Z3/Claripy Example

- Claripy is the abstraction layer for constraint solvers like Z3 used by angr
- Only exposes needed functionality for binary analysis

DEMO

Symbolic Execution

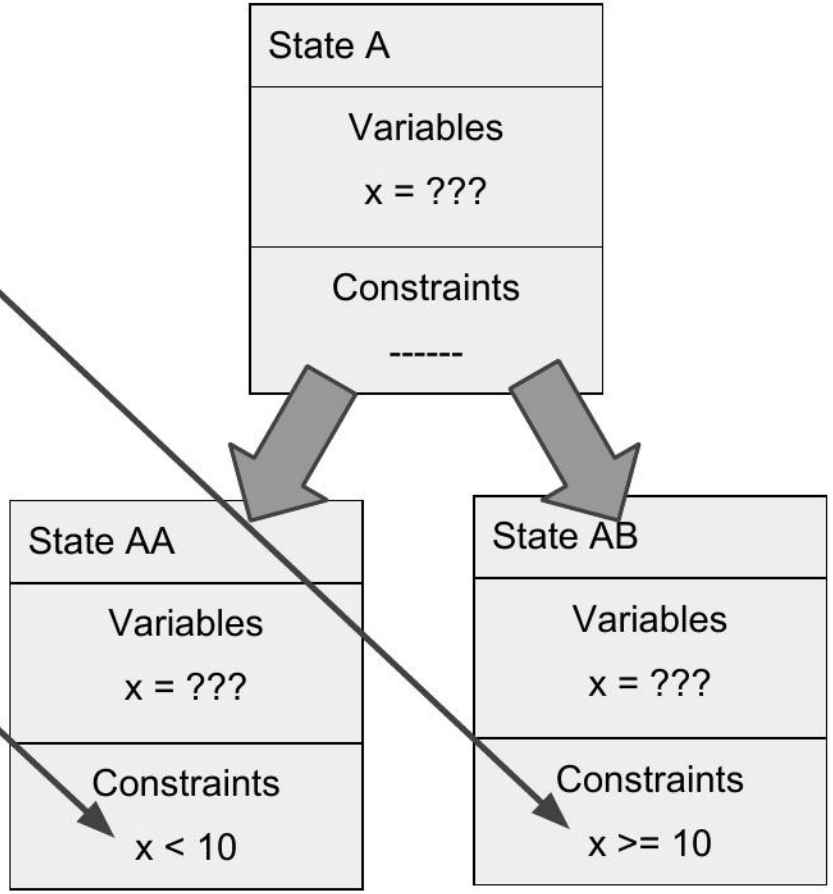
- Symbolic instead of concrete variables
- Following example shamelessly stolen from the angr presentations


```
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 ------

ERNW
providing security.

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



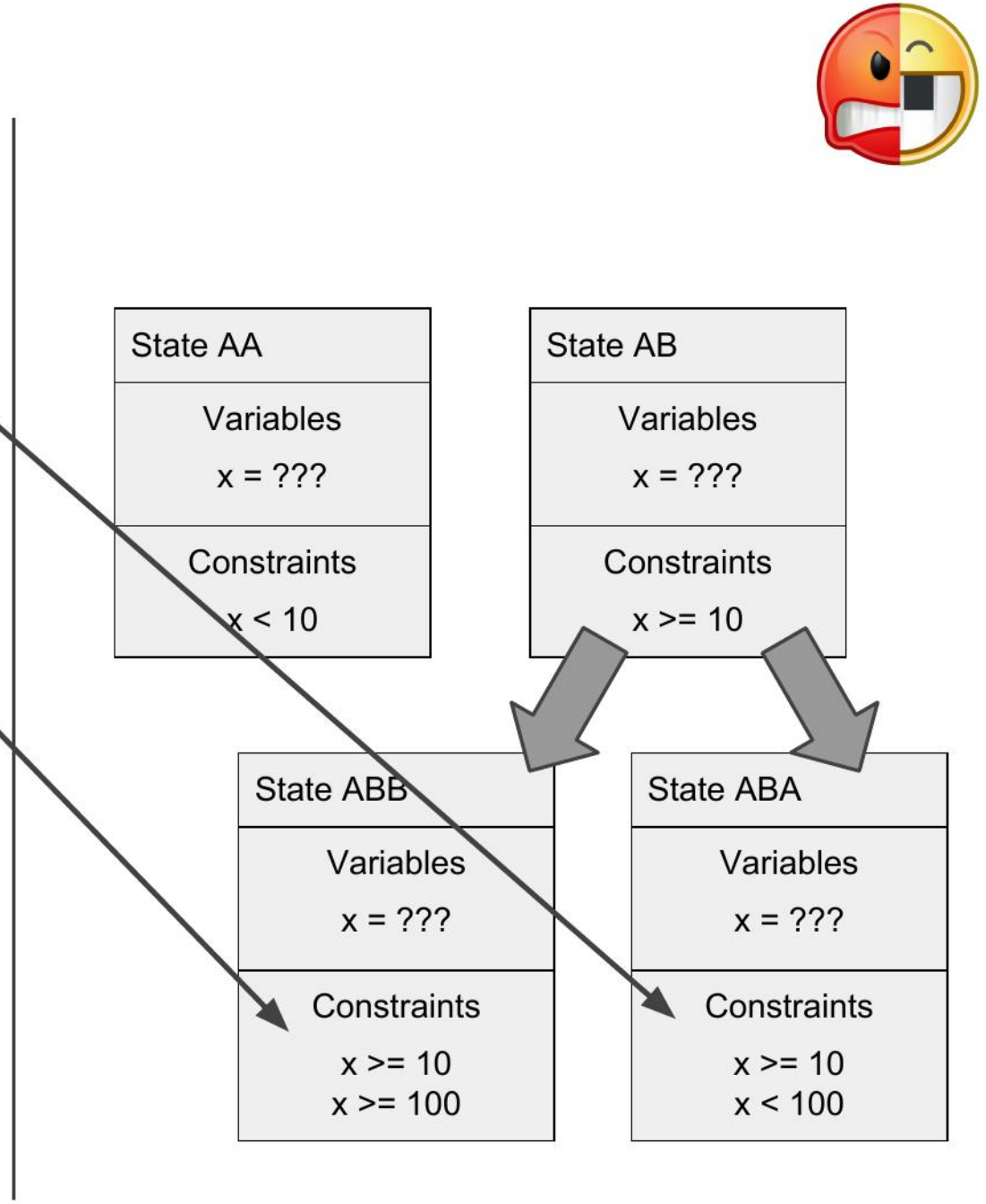
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

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

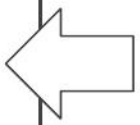
State AB
Variables x = ???
Constraints x >= 10

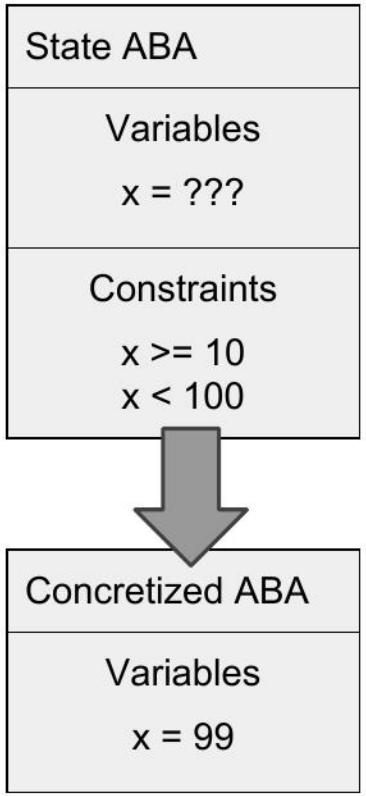
ERNW
providing security.

```
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!"
```





ERNW
providing security.

AST

- Abstract Syntax Tree
- Basically a representation for the constraints from the previous slide

ERNW
providing security.

Side Note: LLVM Compiler Infrastructure

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

ERNW
providing security.

What to do with all this?

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

ERNW
providing security.

Augmented Fuzzing

- Taint Analysis to discover what branch depends on what input
- Symbolic Execution with constraint solver to build input to take that branch

ERNW
providing security.

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
- Even the CIA uses angr!

ERNW
providing security.

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

ERNW
providing security.

Others

- Bitblaze
  - University of California, Berkeley
- bap: Binary Analysis Platform
  - Carnegie Mellon University/ForAllSecure
  - Written in OCaml
  - Used by Mayhem (1st Place at Darpa's CGC)
- Miasm

ERNW
providing security.

Comparison with other projects

Items	angr	KLEE	BAP	BitBlaze	S2E	Triton	Microsoft SAGE	Mayhem
Work on binaries w/o src								
Online symbolic execution								
Offline symbolic execution		?		?			?	
Cross-platform analysis								
Static analysis								
Multi-platform/arch support								
Open source								
Actively maintained					?			
Free license								

Overview of Frameworks and Projects

Source: Angr Tutorial (so obviously biased)

ERNW
providing security.

Installing Angr

- My setup: Vagrant Box ~~with Archstrike Repositories~~ (turns out that's outdated)
  - pacman –S angr
- Alternatives:
  - Official Docker Image
  - pip2 install angr

ERNW
providing security.

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.
[...]
```

ERNW
providing security.

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) -> (x*3 >> 1) * y
- f(?,0x42) - > 0x76E24

ERNW
providing security.

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) -> (x*3 >> 1) * y
- f(?,0x42) - > 0x76E24

DEMO

ERNW
providing security.

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) -> (x*3 >> 1) * y
- f(?,0x42) - > 0x76E24

- We got two possible solutions
  - 0x1337 (intended)
  - 0x55555555555688c which returns 0x21000000000076e24
    -> Integer Overflow

ERNW
providing security.

Automagic Solving of Crackmes

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

DEMO

ERNW
providing security.

Automagic 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

ERNW
providing security.

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_write',
when=simuvex.BP_AFTER, action=debug_func)
```

ERNW
providing security.

Anti-Anti-Debugging

- angr is not a debugger
  - Most tricks wont work
  - Might 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

ERNW
providing security.

Angr Cheat Sheet

- Currently developing an angr cheat sheet
  - Common Commands to look up
  - Features that are hidden somewhere in the docs
- Release $soon
  - Probably as part of the angr/angr-doc repo

Other Tools to know

- Unicorn Emulator
  - Emulate all the architectures!
- Capstone
  - Disassemble all the architectures!

- Interesting Mechaphish Components
  - angrop (generates ROP Chains)
  - Driller (Augmented Fuzzing)
  - Everything in https://github.com/mechaphish

ERNW
providing security.

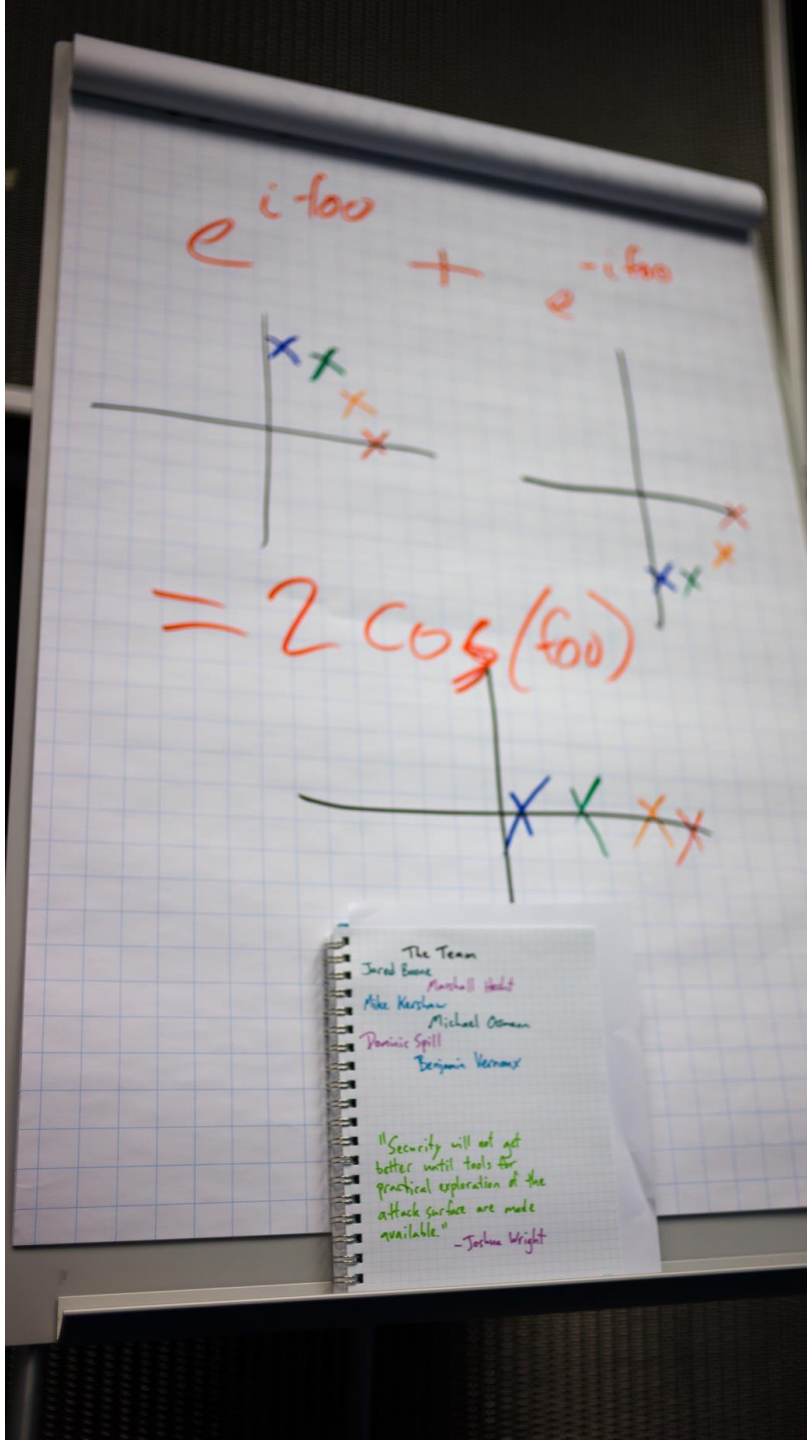
Thank you for your attention

fmagin@ernw.de

0x464D

www.ernw.de

www.insinuator.net



ERNW
providing security.

References & Literature

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

  Analyzing Memory Accesses in x86 Executables

  Gogul Balakrishnan and Thomas Reps

  Comp. Sci. Dept., University of Wisconsin; {bgogul,reps }@cs.wisc.edu