#RSAC

RSAConference2022

San Francisco & Digital | June 6 – 9

TRANSFORM

SESSION ID: **HTA-M03**

OT Reverse Engineering - What Can Be Automated?

Dr. Ulrich Lang
Founder/CEO

Dr. Reza Fatahi
Senior Research Scientist

Rudolf Schreiner
Founder/CEO (Europe)

ObjectSecurity
@objectsecurity

Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

All information provided in this presentation and these slides is provided « AS IS » without warranty of any kind either expressed or implied including, without limitation, warranties of merchantability, fitness for a particular purpose or non infringement of intellectual property rights.
The ObjectSecurity logo must not be removed from any slides, and any publications must include the ObjectSecurity logo in full

About Us

Dr. Ulrich Lang

- 20+ year cybersecurity
- Founder & CEO of ObjectSecurity
- PhD cybersecurity (Cambridge)
- Master’s cybersecurity
- 1st time RSA presenter in 2007
- Interests: Binary vuln. analysis (OT/ICS), trusted AI, 5G/wireless security, access policy automation, supply chain risk analysis

Contributors

- Dr. Reza Fatahi, Senior Research Scientist at ObjectSecurity
  - leads much of our binary analysis work
- Rudolf Schreiner, CEO, ObjectSecurity OSA (Europe)
  - 20+ years cybersecurity
  - lots of vulnerability analysis, incl. hardware, side-channel, FPGA etc.

OT & security

Operational Technology (OT): "hardware and software that detects or causes a change, through the direct monitoring and/or control of industrial equipment, assets, processes and events."

Why this matters:
OT increasingly an attack target

- Colonial pipeline ransomware – not OT but OT impact (2021)
- Ukraine power grid Sandworm hack – SCADA etc. (2015)
- German steel mill hack - ICS (2014)
- Stuxnet gas centrifuge (nuclear) - SCADA (2010)
- …
- These are critical systems (also cyber/physical safety!)

OT Security != IT Security

- Highly heterogenous vs monoculture
  - Operating systems, buses
- Often offline or on non-IP networks (e.g. sensors, fieldbus…)
- Legacy, system life time
  - Elder systems often very brittle
- Cultural difference (due to safety!): Never change a running system vs quick patches
- Opaque boxes with very little internal documentation
  - Reverse engineering often finds standard CPU and libraries
- Today: Reverse engineers manually assess OT: expensive, no scale
  - Decentralized knowledge base, polyglot exploitation artist, self-sufficient

Some OT Cyber Guidance

- IEC3 62443: Security for industrial communication networks
- NIST SP 800-82 Guide to Industrial Control Systems Security
- Both pretty generic
  - Risk management and assessment
  - Program development and deployment
  - Security architecture and controls
  - Applying controls to ICS
- Advice: Strong segregation and segmentation, esp. IT ⇔ OT

Network Segmentation/Segregation

- Guidelines accept that OT is hard to protect
- So it has to be isolated into domains from IT and “the outside”
- Strictly controlled communication between domains
- In many domains, segmentation does not work
- We found: OT rarely segmented, or can be fully segmented
- Wireless/5G communication make it worse – Industry 4.0 etc.
- IT -> OT for Zero Trust Architecture (ZTA) & Risk Management

OT organizational hurdles

- Where does OT cyber fit into organizations: CISO / Ops?
- Infosec cyber skills don’t fully translate to OT
- Conventional IT security approaches don’t translate well
- Internal IT sec org doesn’t have the staffing to deal with this
- OT cyber needs to be part of an integrated IT security organization but often isn’t today
- Automation is key to manage the growing-scale mess with limited IT sec staff and resources, but is challenging.
  - What can be automated?

Step 1: Know your OT landscape

- Before you can assess risks or protect your systems, you need to know it
  - Which devices? How are they connected? Who is owning/managing them? Etc.
- Seems trivial: Everything is documented :)
  - But in our experience, this is a major challenge: docs != reality
  - Usually no source code
- In many cases, you need to reengineer your systems
  - And: The devil is in the detail
- Automation is key:
  - You cannot analyze your environment manually

Automated network scanning

- Find and identify all devices
  - Active scanning
  - Passive communication analysis
- Scanning of vulnerabilities
  - Consider specific limitations: OT devices not well supported
- Consider safety
  - Legacy devices often crash
- **But this is not enough for OT – less impactful than for IT**

Managed devices vs unmanaged devices

- Many devices are directly managed by the organization
  - PLC visible in supervisory system
  - Software version can be obtained, e.g. directly from supervisory system
- Other devices are embedded in subsystems
  - You know very little about these devices
  - They are part of the attack surface

Lab testing

Lab

- You do not need to test all devices in the field, e.g. all PLC
- If you know the device, incl. software version, you can test one in the lab and then transfer the results to operational devices
- In most cases: PLC is an opaque device
  - You need to do your own testing/reverse engineering

Field

- Devices often not managed
- If a device is unmanaged and unknown, and if it is part of the attack surface, you have to test it in the field
- Can‘t be done manually (scale/skill)
- Network scanning not enough
- RedBox

Vulnerabilities scanning is not testing

- Vulnerabilities scanners look for *known* vulnerabilities
  - No real testing
  - Determination of version, database lookup
  - Signatures
- Does not work for 0-days
- Little OT information in vuln databases
- CWE != CVE

RSAConference2022

Let’s talk OT Reverse Engineering Automation

SBIR Case Study & more

ObjectSecurity
Secure technology

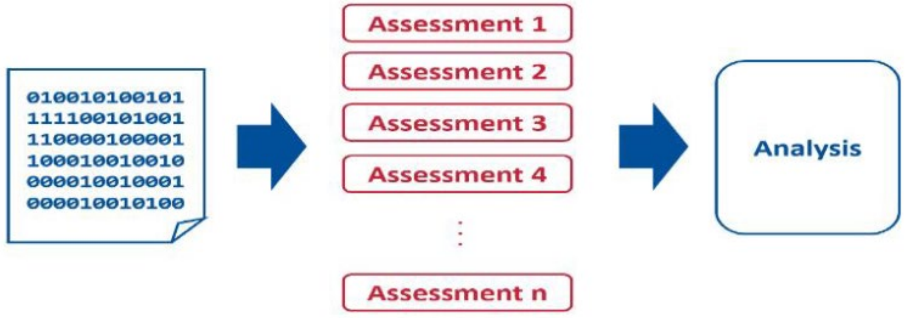
Automated vulnerabilities analysis of devices

- "Red Team in a Box for Embedded and Non-IP Devices" (Navy SBIR 2018.2 - Topic N182-131)
  - need to reverse engineer (automatically) to analyze for vulnerabilities
  - simpler/cheaper and at scale?
  - also for already-fielded embedded systems
  - also for legacy systems
  - operated by non-expert (scale!)
  - offline (cloud use optional) -> SWAP performance, scale, battery, time …
  - Buyer vs. developer -> DevSecOps or source code tools n/a
- Involves some/all of:
  - Connect – extract – analyze – report
  - Orchestrate many tools and approaches coherently…

https://www.navysbir.com/n18_2/N182-131.htm

Firmware Analysis

- Lots of tools for human reverse engineers (RE) out there
  - Some work if human assists, many don't work well in many cases
  - RE tools like IDA Pro, Binary Ninja, Ghidra not geared to non-expert use
  - Find CWEs, but CWE is not always vulnerability or malware
- No single tool can do it all
  - We orchestrate many tools
  - Selected from 100's analyzed



Binary Analysis (1)

- Binaries usually have headers and data
  - Created by compiling source to binary to make an executable
  - Static linking and dynamic linking can inherent vulns from code you never wrote and didn't know your software depends on
- Meta-analysis: headers & strings
- Malware: Self-modifying behavior & evasion techniques
- Faulty software: incl. 0-Days

Binary Analysis (2)

- Disassembly: Linear sweep, recursive traversal, …
- Pattern matching in disassembly: kernel operations, overflow attacks, opcode sequence analysis, …
  - We found ML useful here!
- Vulnerability verification: detect weakness, then confirm vuln
- Intermediate Representation (IR): LLVM, BAP, Binary Ninja, …
- Higher-level representation Decompiling, emulation
- Dynamic analysis with fuzzing

Binary Analysis (3)

- Lifting challenges
  - Faulty representation
  - Time & complexity
- Rewriting -> bleeding edge
  - Software fault isolation
  - Software flow integrity
- Side Channel Analysis
  - Very challenging with full automation, and non-destructively in the field

Binary Analysis Tools

CVE DETECTION

MALWARE DETECTION

TAINT ANALYSIS

CWE DETECTION

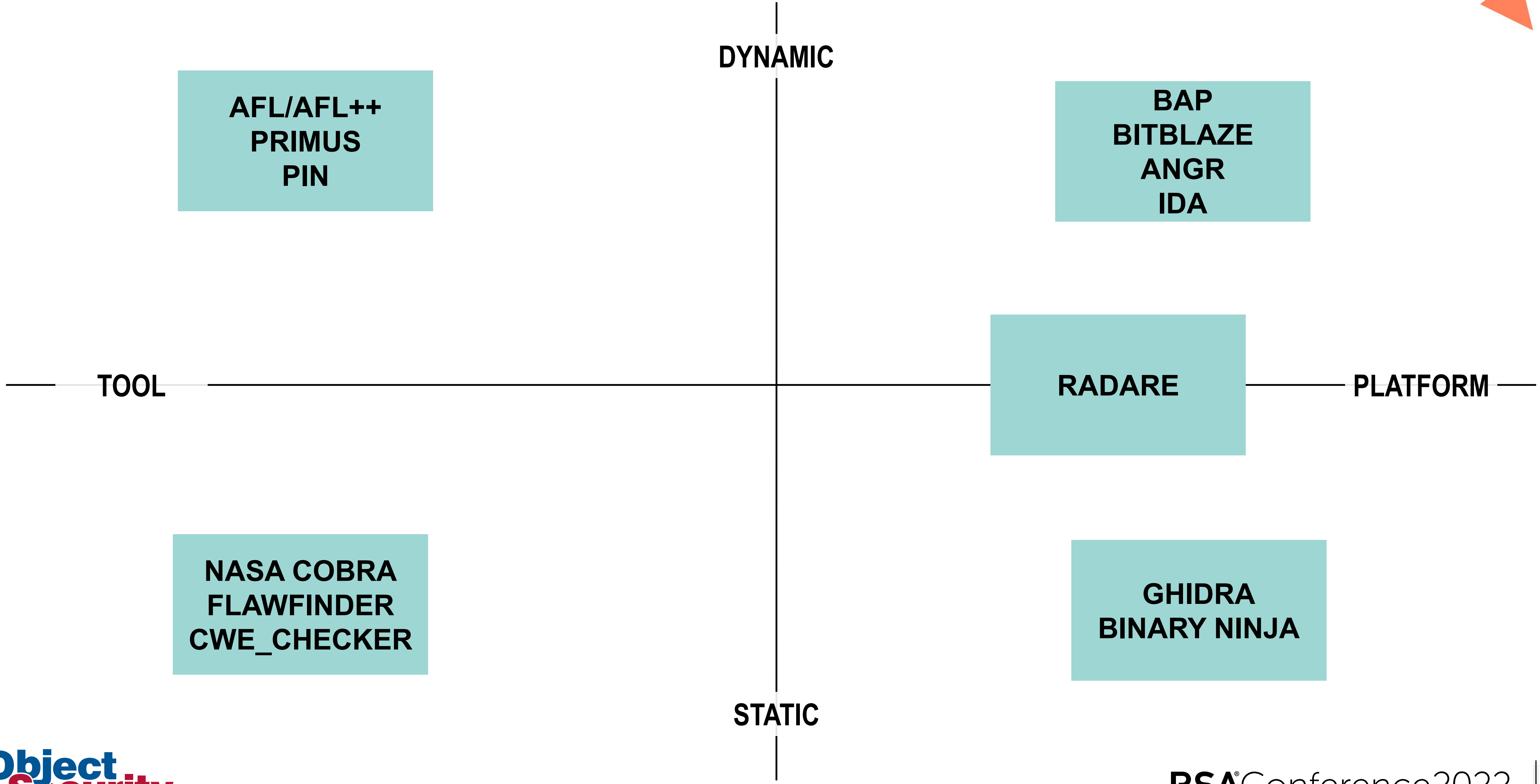
ROP GADGETS

SYMBOLIC EXECTION

DYNAMIC FUZZING

BINARY INSTRUMENTATION

Binary Analysis: Tool Landscape Overview



Binary Analysis: Essential Ingredients

Correct Binaries: Connect / extract automation

- Challenges
  - External: console port, USB, etc. -> often no access
  - Internal: UART, JTAG etc. -> requires some tech skill
  - Often “playing around” that is hard to automate, esp. generically
  - Many CPU archs., OS’s etc.
  - Pulling firmware often slow (USB ok)
  - Ports often not exposed, non-standard
  - Device often not accessible
- Automation success
  - console shells, JTAG
  - not “one size fits all” though

- **Architecture Examples**: i386, x86-64, ARM, MIPS, PowerPC, SPARC, RISC-V, SH, m68k, m680x, AVR, XAP, System Z, XCore, CR16, HPPA, ARC, Blackfin, Z80, H8/300, V810, V850, CRIS, XAP, PIC, LM32, 8051, 6502, i4004, i8080, Propeller, Tricore, CHIP-8, LH5801, T8200, GameBoy, SNES, SPC700, MSP430, Xtensa, NIOS II, Java, Dalvik, WebAssembly, MSIL, EBC, TMS320 (c54x, c55x, c55+, c66), Hexagon, Brainfuck, Malbolge, whitespace, DCPU16, LANAI, MCORE, mcs96, RSP, SuperH-4, VAX, AMD Am29000, …
- **File Type Examples**: ELF, Mach-O, Fatmach-O, PE, PE+, MZ, COFF, OMF, TE, XBE, BIOS/UEFI, Dyldcache, DEX, ART, CGC, Java class, Android boot image, Plan9 executable, ZIMG, MBN/SBL bootloader, ELF coredump, MDMP (Windows minidump), WASM (WebAssembly binary), Commodore VICE emulator, QNX, Game Boy (Advance), Nintendo DS ROMs and Nintendo 3DS FIRMs, various filesystems, …

Disassembly

```
[X] Disassembly (pd)                                            [Cache] Off
              ;-- posix_spawnp:
              0x000e8ea0       4883ec10        sub rsp, 0x10
              0x000e8ea4       6a01            push 1
              0x000e8ea6       e8c5070000      call 0xe9670
              0x000e8eab       4883c418        add rsp, 0x18
              0x000e8eaf       c3              ret
              0x000e8eb0       4157            push r15
              0x000e8eb2       4156            push r14
              0x000e8eb4       4989ce          mov r14, rcx
              0x000e8eb7       4155            push r13
              0x000e8eb9       4d89c5          mov r13, r8
              0x000e8ebc       4154            push r12
              0x000e8ebe       4989fc          mov r12, rdi
              0x000e8ec1       55              push rbp
              0x000e8ec2       53              push rbx
              0x000e8ec3       4881ecd80200.   sub rsp, 0x2d8
```

Targets:

1. Opcodes
2. Registers
3. Bytes
4. Calls
5. Hex

Intermediate Representation

```
r2's esil debugger:

addr: 0x000e8ea0
pos: 0
hex: 4883ec10
asm: sub rsp, 0x10
esil: 16,rsp,-=,16,0x8000000000000000,-,!,63,$o,^,of,:=,63,$s,sf,:=,$z,zf,:=,$p,pf,:=,64,$b,cf,:=,
      --
esil regs:
$$ 0x00000000 (0) ; address
$z 0x00000000 (0) ; zero
$b 0x00000000 (0) ; borrow
$c 0x00000000 (0) ; carry
$o 0x00000000 (0) ; overflow
$p 0x00000000 (0) ; parity
$r 0x00000000 (0) ; regsize
$s 0x00000000 (0) ; sign
$d 0x00000000 (0) ; delay
$j 0x00000000 (0) ; jump
regs:
    rax 0x00000000            rbx 0x00000000            rcx 0x00000000
    rdx 0x00000000            rsi 0x00000000            rdi 0x00000000
     r8 0x00000000             r9 0x00000000            r10 0x00000000
    r11 0x00000000            r12 0x00000000            r13 0x00000000
```

Tools Generating IL/IR:

1. LLVM
2. Binary Ninja(extended API)
3. BAP
4. BitBlaze
5. Radare (limited support)

ML on Sequenced Data

```
[X] Disassembly (pd)                                          [Cache] Off
            ;-- posix_spawnp:
            0x000e8ea0      4883ec10        sub rsp, 0x10
            0x000e8ea4      6a01            push 1
            0x000e8ea6      e8c5070000      call 0xe9670
            0x000e8eab      4883c418        add rsp, 0x18
            0x000e8eaf      c3              ret
            0x000e8eb0      4157            push r15
            0x000e8eb2      4156            push r14
            0x000e8eb4      4989ce          mov r14, rcx
            0x000e8eb7      4155            push r13
            0x000e8eb9      4d89c5          mov r13, r8
            0x000e8ebc      4154            push r12
            0x000e8ebe      4989fc          mov r12, rdi
            0x000e8ec1      55              push rbp
            0x000e8ec2      53              push rbx
            0x000e8ec3      4881ecd80200.   sub rsp, 0x2d8
```

```
,sf,:=,$z,zf,:=,$p,pf,:=,64,$b,cf,:=,
$z 0x00000000 (0) ; zero
$b 0x00000000 (0) ; borrow
$c 0x00000000 (0) ; carry
$o 0x00000000 (0) ; overflow
$p 0x00000000 (0) ; parity
$r 0x00000000 (0) ; regsize
$s 0x00000000 (0) ; sign
$d 0x00000000 (0) ; delay
$j 0x00000000 (0) ; jump
regs:
    rax 0x00000000          rbx 0x00000000          rcx 0x00000000
    rdx 0x00000000          rsi 0x00000000          rdi 0x00000000
     r8 0x00000000           r9 0x00000000          r10 0x00000000
    r11 0x00000000          r12 0x00000000          r13 0x00000000
```

Architectures:

1. NLP Libraries
2. RNN
3. LSTM

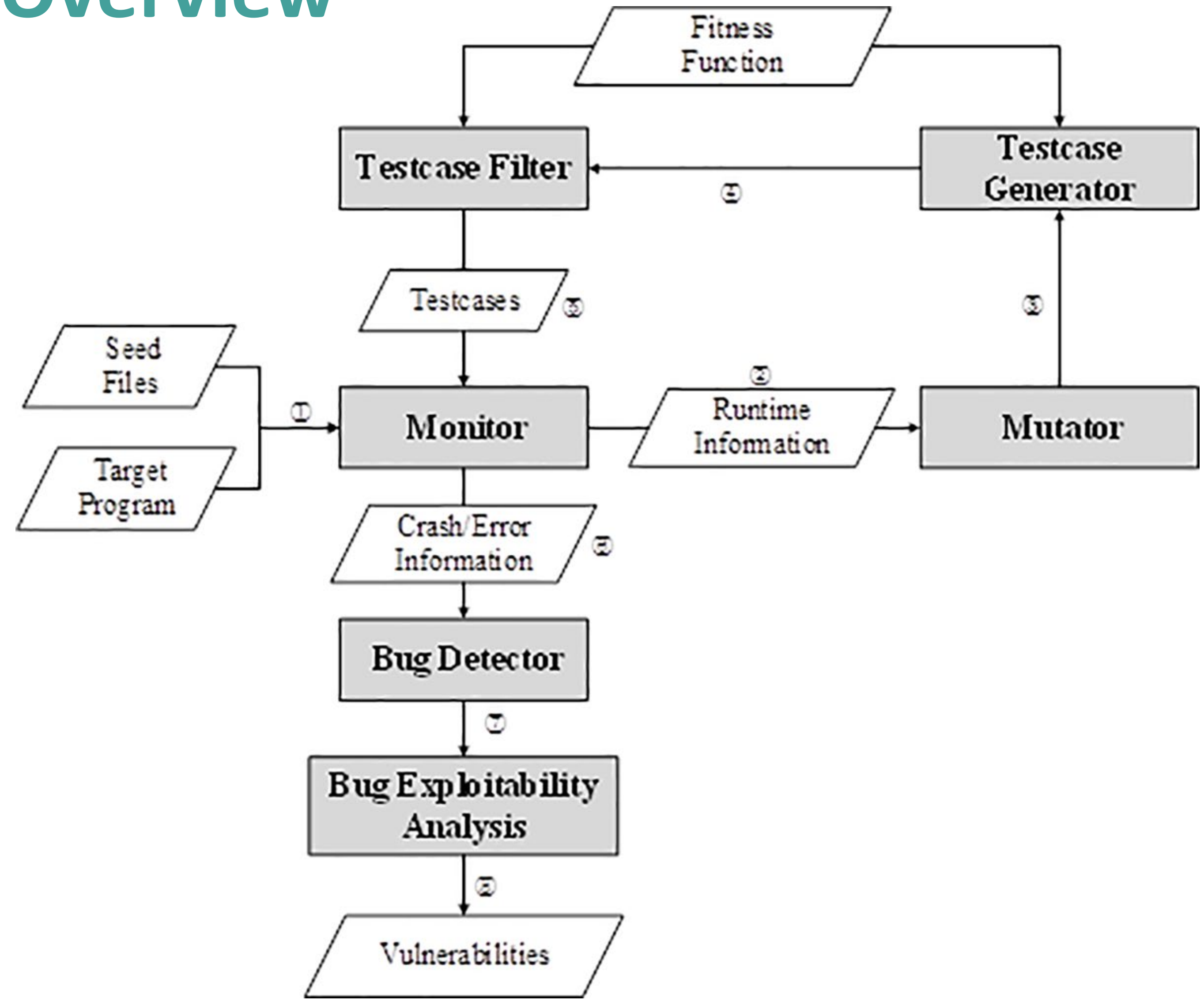
Fuzzing & ML

- Fuzzing Challenges
  - Limitless Analysis Space
    - Potentially unlimited inputs to a function or binary
  - Approaches Correctness
    - Getting further in the space of a function or binary
- Mitigations
  - Smarter Inputs
    - Best guess on initial attempts from data at hand and previously encountered data
  - Reinforcement
    - Identification of correctness and improvements to continuously improve inputs

American fuzzy lop (AFL) is a Google open-source fuzzer that uses genetic algorithms to efficiently increase code coverage of the test cases.

Fuzzers with ML:
AFL, VUzzer, FUZZER, SES, Steelix, Angora, AFL-laf-Intel, InsFuzz, T-fuzz, REDQUEEN, DigFuzz

Fuzzing Overview



Source: https://doi.org/10.1371/journal.pone.0237749.g001

ML Algorithms in Fuzzing

Spaces where ML is applied to Fuzzing

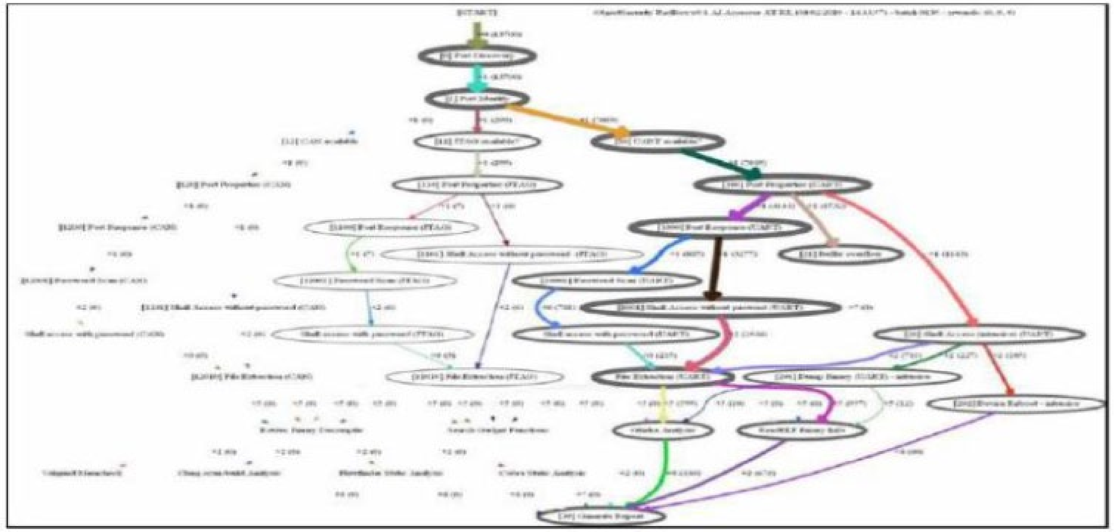
Step
Seed file generation
Testcase generation
Testcase filtering
Mutation operator selection
Exploitability analysis

Algorithm	Category	Studies
LR (*Logistic Regression*)	Traditional Machine learning	Testcase filtering Exploitability analysis Seed file generation Mutation operator selection
NB (*Naive Bayes*)		
BN (*Bayesian Networks*)		
SVM (*Support Vector Machines*)		
KNN (*K-Nearest Neighbor*)		
RF (*Random Forest*)		
DT (*Decision Tree*)		
PCFG (*Probabilistic Context-Free Grammar*)		
PCSG (*Probabilistic Context-Sensitive Grammar*)		
PA (*Passive-Aggressive*)		
Thompson Sampling		
RLS(*Recursive Least Square*)		
RNN (*Recurrent Neural Network*)	Deep learning	Testcase generation Seed file generation
CNN (*Convolutional Neural Network*)		
LSTM (*Long Short-Term Memory*)		
GRU (*Gate Recurrent Unit,*)		
BLSTM (*Bidirectional Long Short-Term memory*)		
Seq2seq (*Sequence-to-sequence*)		
seq2seq-attention (*Sequence-to-sequence-attention*)		
MLP (*Multilayer Perceptron*)		
GCN (*Graph Convolutional Network*)		
Struc2vec		
GAN (*Generative Adversarial Networks*)		
WGAN (*Wasserstein Generative Adversarial Networks*)		
Q-Learning	Reinforcement learning	Exploitability analysis Mutation operator selection
SARSA (*State–action–reward–state–action*)		
Deep Q-Learning		
Deep Double Q-Learning		

https://doi.org/10.1371/journal.pone.0237749.t004

How to orchestrate all this?

- reinforcement learning, but successively minimized once we learned the criteria for orchestration
  - Which tool supports which CPU architecture, file size, binary type, …
  - Remaining battery/time…
  - Predict crashes of tools etc.
  - Easy/automated updates to orchestration as the evolving “hodgepodge” of tools and approaches changes/grows



Other related binary analysis automation R&D

- Automated binary decomposition analysis
  - For any libraries, not just open source dependencies.
  - Noteworthy: scalability/performance; similarity graphing
- Automated “differential stimulus”
  - IDE to (semi-)automated “stimulation” to create “labeled data”
- Automatically analyzing binaries against requirements
  - Challenges:
    - Allow users to meaningfully specify requirements
    - Test these requirements are met, and without a state space explosion

Automation has limitations

- Automated vulnerabilities testing is most useful
  - Not looking for known bugs
- But cannot beat a skilled tester (currently at least)
- Complex attacks
- Beware of the snake oil

Summary

- OT security is different from IT security
  - IT security concepts cannot be directly applied
- OT guidelines are not always helpful
  - Concepts of segmentation often fails
- You need to know your OT/ICS systems in detail
  - All devices, all connections – and vulnerabilities, before an attacker finds them
- Automation is very useful – too many OT/ICS devices
  - Network reengineering, vulnerabilities detection
- But does not solve all problems, and is challenging
  - Esp. firmware and fielded OT/ICS devices
- Beware of the snake oil

Apply What You Have Learned Today

- Do not apply an "IT security" mindset – OT cyber is different
  - Go beyond network scanning, much OT exposure won't be visible that
- Next week, figure out your OT exposure – roughly better than not at all if your organization operates OT
- Do not rely on OT manufacturers – the buck usually stops with your organization (even if they manage your devices)
- Next month, clarify role of CISO in OT
  - budgets, and how stakeholders will come together and be funded
- Within 3 months, implement roadmap towards integrated IT/OT operations in your organization
- Within 6months, look for OT cyber solutions but beware of "snake oil"
  - many newcomers, really understand what vendors bring to the table

RSAConference2022

Thank you!

Questions?