Avatar^2: A Multi-target Orchestration Platform

Marius Muench, Dario Nisi, Aurélien Francillon & Davide Balzarotti

Workshop on Binary Analysis Research 2018

Introduction

FЯIDA

Pin

TRITON
Dynamic Binary Analysis

Motivation

- Having a huge variety of tools is *awesome*
  - **But** analysis state is mostly local to the single tools
  - A lot of effort to integrate specific tools into other

Being able to interconnect debuggers, emulators and analysis frameworks greatly benefits dynamic binary analysis

A link to the past

2014: The avatar framework:

- Connects S^2E and OpenOCD/GDB
- Targets ARM firmware
- Partial emulation together with real hardware

Zaddach, Jonas, et al. "AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares." NDSS 2014.

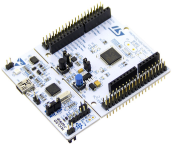
A link to the past

2014: The avatar framework:

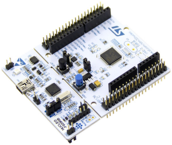
- Connects S^2E and OpenOCD/GDB
- Targets ARM firmware
- Partial emulation together with real hardware
- Tightly coupled to S^2E and OpenOCD/GDB

Zaddach, Jonas, et al. "AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares." NDSS 2014.

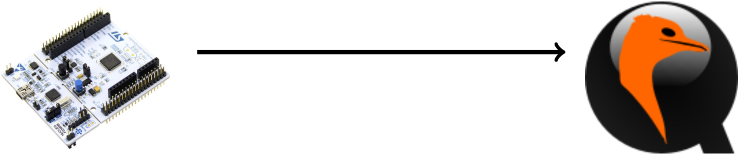
Imagine a tool that ...



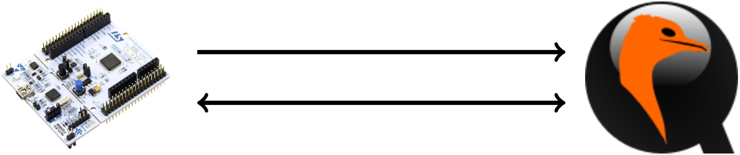
Imagine a tool that ...



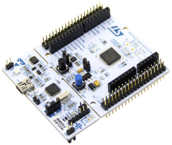
Imagine a tool that ...



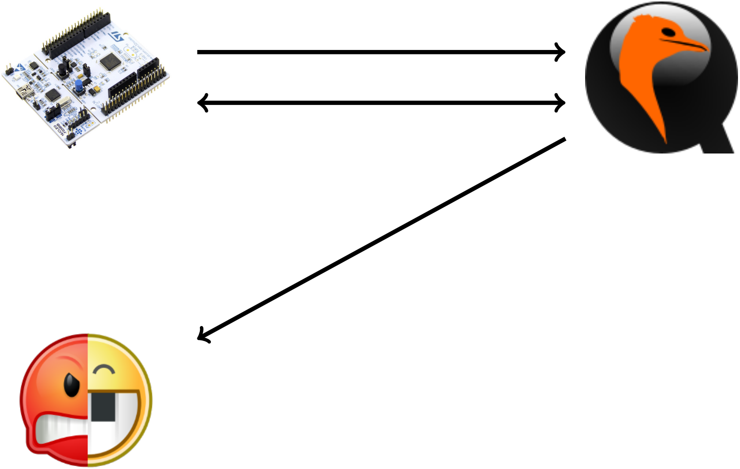
Imagine a tool that ...



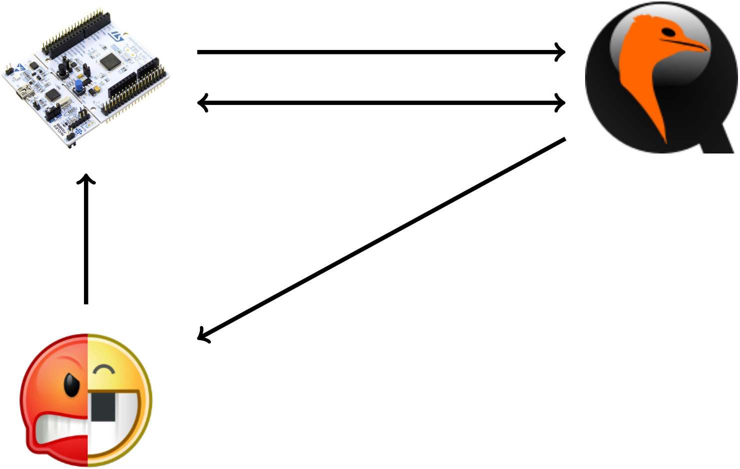
Imagine a tool that ...



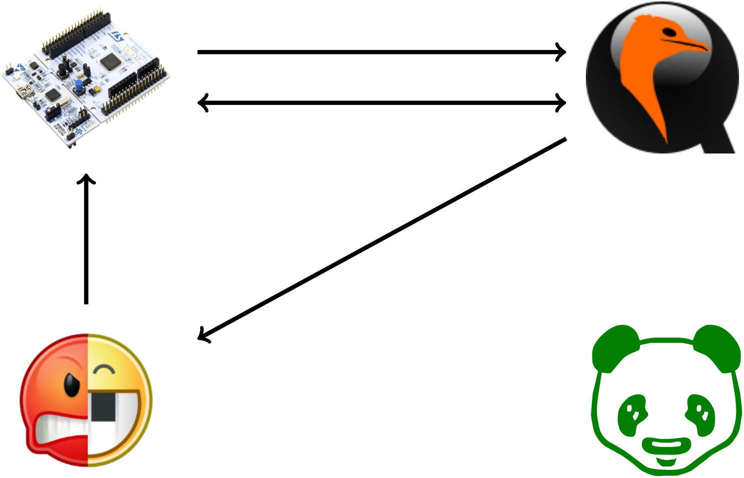
Imagine a tool that ...



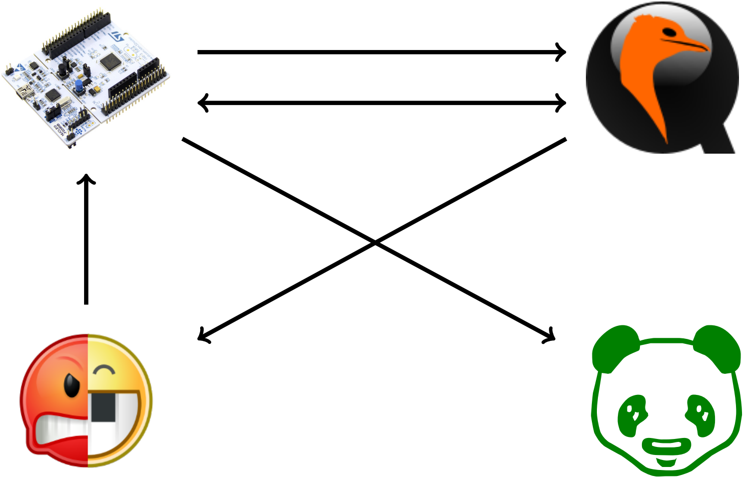
Imagine a tool that ...



Imagine a tool that ...



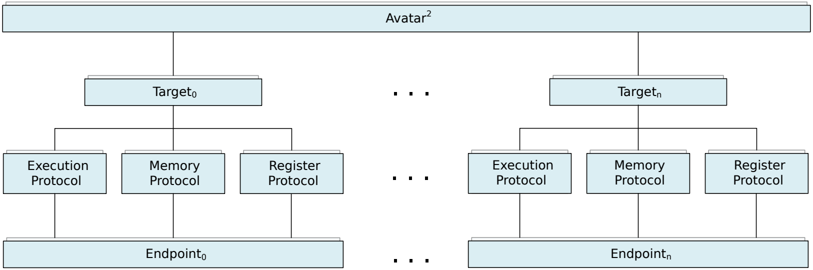
Imagine a tool that ...



One framework to orchestrate them all?

- Capable of interconnecting a variety of tools
- Expose a consistent API to the analyst
- Easy scriptability
- Operate in an highly asynchronous environment
  - ➔ Careful crafted architecture required

Avatar^2- The architecture

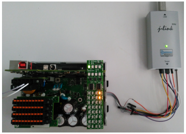


Additional features

- Architecture independent design
- Internal memory layout representation
- Legacy python support
- Peripheral modeling
- Plugin System
  - Assembler/Disassembler
  - Orchestrator
  - Instruction Forwarder

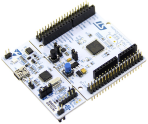
Examples

Example I:
Facilitating replication
& reproduction



Example II:
Symbolic Execution
& Complex Software

Example III:
Record & Replay for Firmware



Example I - Replicating Harvey

- Proof of concept implementation of HARVEY [1]
  - Malware for a COTS PLC
  - The plc utilizes multiple boards
  - Code injection via JTAG

[1] Garcia, Luis, et al. "Hey, My Malware Knows Physics Attacking PLCs with Physical Model Aware Rootkit." NDSS 2016.

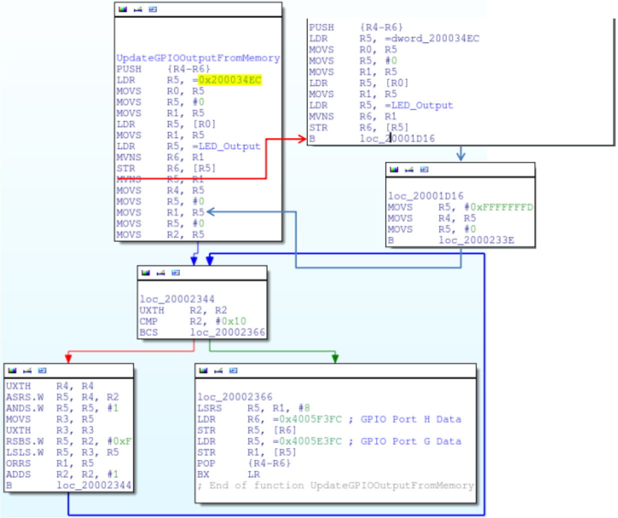


Figure 1: Harvey's modifications to the GPIO-output ISR[1]

[1] Taken from [1]. Original title: "Figure 5. Original GPIO-output update ISR assembly code compared to modified subroutine with branch to malicious code."

```
from avatar2 import Avatar, ARMV7M, OpenOCDTarget

output_hook = '''mov r5,0xfffffffd
                mov r4, r5
                mov r5, 0
                b 0x2000233E'''

avatar = Avatar(arch=ARMV7M)
avatar.load_plugin('assembler')

t = avatar.add_target(OpenOCDTarget, openocd_script='harvey.cfg',
        gdb_executable='arm-none-eabi-gdb')

t.init()
t.set_breakpoint(0xd270)
t.cont()
t.wait()

t.inject_asm('b 0x20002514',addr=0x20002338)
t.inject_asm(output_hook,addr=0x20002514)

t.cont()
```

Example I - Results

- Implementation of PoC in approx. 30 lines of Python
- All of this could –and has – been done without avatar[2]
- *Unified* and *centralized* interface
  - Easy to exchange scripts
  - Modifications can easily be integrated

Example II - Symbolic Execution of Firefox

- Firefox with inserted bug
  - Executed concretely inside gdb until function of interest
- Automated memory layout extraction from gdb
- Transfer of layout into angr
- Memory contents copied-on-read
- Symbolic function arguments
- Analysis of only one thread

Example II - Results

- Implementation in approx. 60 lines of Python
- Execution statistics:
  - Approximatly 10 minutes of runtime[2]
  - 36 executed basic blocks
  - 21 uniquely accessed pages
  - Found the bug
- angr alone was not able to find the bug
  - Could be achieved by tedious population of state without avatar[2]
- Demonstrates *State Transfer* and *Orchestration* capabilities

[2] Hardware: VM with four Intel Xeon E5-2650L cores and 16GB of RAM

Example III - Recording Firmware Execution

- Dynamic binary analysis of firmware often requires the device
- PANDA [2] allows to record and replay execution
- Allows exchange of executions for further analysis without the device

[2] Whelan, Ryan J., et al. "Repeatable Reverse Engineering with the Platform for Architecture-Neutral Dynamic Analysis." MIT Lincoln Laboratory Lexington 2015.

```python
from avatar2 import  ARMV7M, Avatar, OpenOCDTarget, PandaTarget

avatar = Avatar(arch=ARMV7M)
avatar.load_plugin('orchestrator')

nucleo = avatar.add_target(OpenOCDTarget, [...])
panda  = avatar.add_target(PandaTarget, [...])

rom = avatar.add_memory_range(0x08000000, 0x1000000,
    file=firmware)
ram = avatar.add_memory_range(0x20000000, 0x14000)
mmio= avatar.add_memory_range(0x40000000, 0x1000000,
        forwarded=True, forwarded_to=nucleo)

avatar.init_targets()
[...]
panda.begin_record('panda_record')
avatar.resume_orchestration(blocking=False)
[...]
avatar.stop_orchestration()
panda.end_record()
```

Example III - Results

- Implementation in approx. 30 lines of Python
- Successful recording of firmware's execution
  - Replayable *without* presence of hardware
- Without avatar[2], cumbersome implementation of peripherals required
- Demonstration of separation between execution and memory

Discussion

- So far, only five targets implemented
- Achieving genericity is difficult
  - Overhead for implementing new targets varies
- Unsolved challenges for analysis of embedded devices
  - Interrupts
  - Debug access

Conclusion

- Multi-target orchestration is not limited to firmware
- We are just at the beginning ...
- More tomorrow morning!
  - "What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices."
  - Session 1A: IoT, Kon Tiki Ballroom, 12.20pm

Artifacts?

- The full framework is open source:
  `https://github.com/avatartwo/avatar2`
- Presented examples at:
  `https://github.com/avatartwo/bar18_avatar2`
- Pre-built vagrant box:
  `avatar/2bar18_avatar2`

Backup slide #1: Changes to QEMU

Avatar^2 provides a costomized QEMU

- All located in a single subfolder: hw/avatar
- New board: Configurable Machine
  - Already present in the first avatar
  - Allows flexible configuration of emulated hardware
- New peripheral: remote_memory
  - Communicates with avatar^2 via posix message queues
  - Utilizes custom remote-memory protocol

Backup Slide #2: Event handling in avatar^2

- Targets are emitting events
- Events are registered by protocols forwarded to the avatar^2 core
    - Fast queue for execution state updates
- Enables callbacks and inspection mechanisms