Avatar² - Enhancing Binary Firmware Security Analysis with Dynamic Multi-Target Orchestration

Marius Muench <marius.muench@eurecom.fr>
PART I
Avatar² - Enhancing Binary Firmware Security Analysis with Dynamic Multi-Target Orchestration
Enhancing Binary Firmware Security Analysis
Dynamic Binary Firmware Security Analysis?

- Majority of nowadays vulnerabilities are “low-hanging fruits”
- Often 3\textsuperscript{rd} party analysis
- Lack of sophisticated tooling
(Some) Challenges in Dynamic Binary Firmware Analysis

- Intransparency
- Performance & Scalability
- Instrumentation capabilities
Avatar² - Enhancing Binary Firmware Security Analysis with Dynamic Multi-Target Orchestration
Avatar² - Dynamic Multi-Target Orchestration
**Avatar²**

- **Developed by:**
  - Marius Muench
  - Dario Nisi
  - Aurélien Francillon
  - Davide Balzarotti

- **Open source:**
  - [https://github.com/avatartwo/avatar2](https://github.com/avatartwo/avatar2)

- Re-designed and re-implemented from scratch

The general picture

Avatar²

Target₀

Execution Protocol Memory Protocol Register Protocol

Endpoint₀

... ...

Targetₙ

Execution Protocol Memory Protocol Register Protocol

Endpointₙ
Core Concepts

- Target Orchestration
- Separation of Execution and Memory
- State Transfer and Synchronization
Supported Targets

- * Not yet available
from os import getcwd
from avatar2 import *

def main(trace_name):
    sample = "%%/..\binaries/expat_panda.bin" % getcwd()
    openocd_conf = "%%/..\config/nucleo-l152re.cfg" % getcwd()
    panda_path = "%%/..\avatar2\targets\build\panda\arm-softmmu\qemu-system-arm"

    avatar = Avatar(output_directory="/tmp/myavatar", arch=ARM_CORTEX_M3)
    panda = avatar.add_target(PandaTarget,
        gdb_executable="arm-none-eabi-gdb",
        executable=panda_path)

    nucleo = avatar.add_target(OpenOCDTarget,
        gdb_executable="arm-none-eabi-gdb",
        openocd_script=openocd_conf)

    panda.gdb_port = 1234
    nucleo.gdb_port = 1235

    avatar.add_memory_range(0x40000000, 0x10000000, 'mmio',
        forwarded=True, forwarded_to=nucleo)

    avatar.add_memory_range(0x00000000, 0x10000000, 'rom', file=sample)
    avatar.add_memory_range(0x20000000, 0x140000, 'ram')

    avatar.init()

    nucleo.set breakpoint(0x0800b5ac)
    nucleo.cont()
    nucleo.wait()

    print("Syncing State")
    avatar.transfer_state(nucleo, panda, synced_ranges=[ram])
    panda.write_memory(0x200000c26, 4, 1)

    print("Start recording the execution")
    panda.begin_record[trace_name]

    import IPython; IPython.embed()

    avatar.shutdown()}
Phase #0: Preambel

```python
from os import getcwd
from avatar2 import *

def main(trace_name):
    sample = "%s/../binaries/expat_panda.bin" % getcwd()
    openocd_conf = "%s/../configs/nucleo-l152re.cfg" % getcwd()
    panda_path = "%s/../avatar2/targets/build/panda/arm-softmmu/qemu-system-arm"
    avatar = Avatar(output_directory='tmp/myavatar', arch=ARM_CORTEX_M3)
```
Phase #1: Target Definition

```python
panda = avatar.add_target(PandaTarget,
                          gdb_executable="arm-none-eabi-gdb",
                          executable=panda_path)

nucleo = avatar.add_target(OpenOCDTarget,
                          gdb_executable="arm-none-eabi-gdb",
                          openocd_script=openocd_conf)

panda.gdb_port = 1234
nucleo.gdb_port = 1235
```
Phase #2: Memory Layout Definition

```python
avatar.add_memory_range(0x40000000, 0x10000000, 'mmio', forwarded=True, forwarded_to=nucleo)
avatar.add_memory_range(0x08000000, 0x10000000, 'rom', file=sample)
avatar.add_memory_range(0x20000000, 0x14000, 'ram')
```
Phase #3: Orchestration!

```python
avatar.init()
nucleo.set_breakpoint(0x0800b5ac)
nucleo.cont()
nucleo.wait()

print("Syncing State")
avatar.transfer_state(nucleo, panda, synced_ranges=[ram])
panda.write_memory(0x20000c20, 4, 1)

print("Start recording the execution")
panda.begin_record(trace_name)

import IPython; IPython.embed()

avatar.shutdown()
```
A note on peripherals

- Main source of complication for emulation

- Avatar\(^2\) offers different strategies:
  - Full emulation
  - Partial emulation using peripheral forwarding
  - Partial emulation using python abstractions
PART II
(WYCINWYC)
WYCINWYC - Overview

- Joint Work with Siemens
- Utilizes Avatar\(^2\) to improve fuzz testing on embedded systems

WYCINWYC - Setup

Avatar²

PANDA
- Analysis Plugis
- Emulation

Embedded Device
- MMIO
- Peripherals

boofuzz
- Fuzz Inputs

Diagram showing the flow between Avatar², PANDA, and Embedded Device.
**WYCINWYC - Analysis Plugins**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Call Stack Tracking</td>
<td>✗</td>
<td>✓</td>
<td>×</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>b) Call Frame Tracking</td>
<td>✗</td>
<td>✓</td>
<td>×</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>c) Stack Object Tracking</td>
<td>✗</td>
<td>✓</td>
<td>×</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>d) Segment Tracking</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>e) Format Specifier Tracking</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>f) Heap Object Tracking</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Combined</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

WYCINWYC - Results

S1: Native
S2: Partial Emulation (Peripheral Forwarding)
S3: Partial Emulation (Avatar Peripheral)
S4: Full Emulation
Related Tools

- **AVATAR ;)**

- **Firmadyne**
  

- **Luaqemu**

  https://github.com/Comsecuris/luaqemu

- **PROSPECT**

  Kammerstetter, M, Platzer, C., & Kastner, W.: “Prospect: peripheral proxying supported embedded code testing.” *ASIA CCS 2014*
Conclusion

- Appropriate tooling is important
- … so are good emulators
- Until then, avatar\(^2\) might be helpful
- We are just at the beginning…