Challenges for dynamic analysis of embedded systems and tackling them with avatar2

Marius Muench

Team Tasteless/EURECOM

Team Tasteless

Captain / pwn & embedded

EURECOM

PhD Student / Dynamic Binary Firmware Analysis

Agenda

Motivation

Reasons for analyzing embedded systems

- Ever-growing amount of embedded systems
- Vulnerabilities go beyond weak-keys, misconfigurations etc.
- “Juicy” vulnerabilities require sophisticated analysis

A Look At The Samsung Shannon Baseband.

Amat Insa Cama / Securin Technology Founder

Fuzzing in the mobile world: the challenges, ideas, questions and (some of) answers.

Tomasz Kuchta / Qualcomm

IoT and silicon security: dissecting a real life IoT attack

Asaf Shen VP Business Development, IoT Device Line of Business, Arm

Dynamic Analysis

Analyzing the code while it runs:

- Aid reverse engineering
- Allows for automated techniques
- Sound results

Challenges

The Challenges

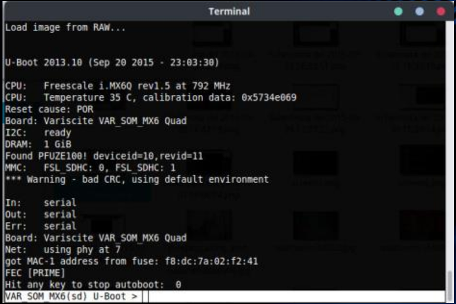
- Obtaining Firmware
- Platform Variety
- Fault Detection
- Scalability
- Instrumentation

(1) Obtaining Firmware

- Obtaining firmware is hard
- Embedded devices are often a black box
- Publicly available source code is the exception
- Various Extraction methods

Firmware Extraction – Software Methods

- Bootloader
- Debug interfaces
- Runtime memory dump
- Firmware Updates



Firmware Extraction – Hardware Methods

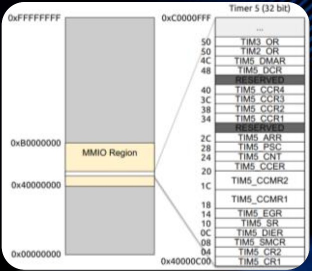
- Flash dump
- Bus tapping
- Glitching

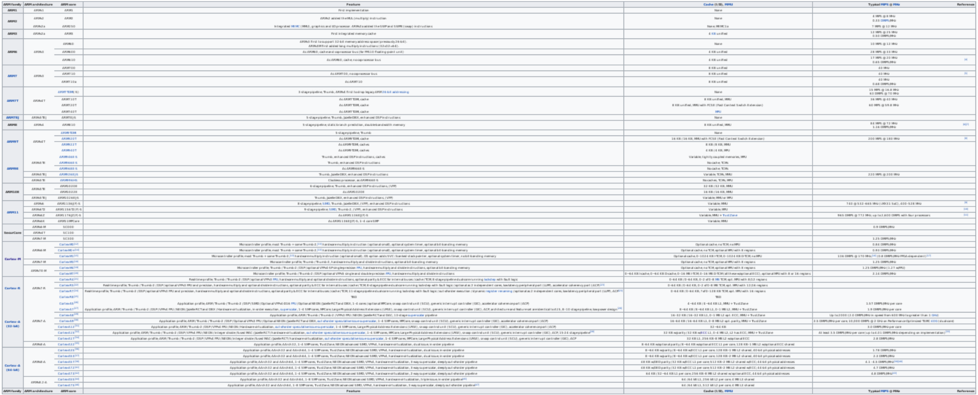
(2) Platform Variety

- Instruction Set Architectures
- Often no OS-level abstractions
- Memory Layout
- Peripherals

Peripherals

- Rarely documented
- Opaque communication
- On-chip vs off-chip





https://en.wikipedia.org/wiki/List of ARM microarchitectures#Designed_by_ARM

Categorization

Type-I:

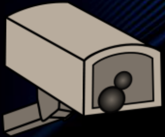
General purpose OS-based

Type-III:

No OS-Abstraction

Type-II:

Embedded OS-based



(3) Fault Detection

Corruption ≠ Crash

(3) Fault Detection

- Lot of techniques rely on *observable crashes*
- Missing methods to turn corruptions into crashes
- Missing I/O
- Active vs. passive probing

(3) Fault Detection

	Platform			
	Desktop	Type-I	Type-II	Type-III
Format String	✓	✓	✗	✗
Stack-based buffer overflow	✓	✓	✓ (opaque)	! (hang)
Heap-based buffer overflow	✓	! (late crash)	✗	✗
Double Free	✓	✓	✗	✗ (malfunc.)
Null Pointer Dereference	✓	✓	✓ (reboot)	✗ (malfunc.)

(4) Scalability

- 1 instance = 1 physical device
- Creating clean state is time-costly

(5) Instrumentation

- Coverage information rarely retrievable
- Available source code is the exception
- Common tools make assumptions about OS or ISA

OS	x86	x86_64	ARM	ARM64	MIPS	MIPS64	PowerPC	PowerPC64
Linux	yes	yes			yes	yes	yes	yes
OS X	yes	yes						
iOS Simulator	yes	yes						
FreeBSD	yes	yes						
Android	yes	yes	yes	yes				

https://github.com/google/sanitizers/wiki/AddressSanitizer

Challenges Recap

Embedded Systems are often:

- Different from one to another (Platform Variety)
- Different from desktop systems (Instrumentation)
- Non-transparent (Obtaining Firmware, Scalability)
- Cost-efficiently produced (Fault Detection)

avatar^2

The big picture

- Dynamic multi-target orchestration framework
- Focus on firmware analysis
- Python-based framework
- Open source: https://github.com/avatartwo

TRITON

Dynamic Binary Analysis

FRIDA

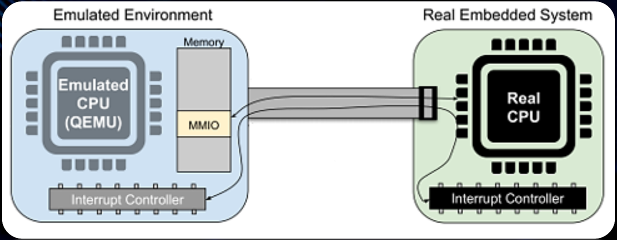
Why another framework?

- Analysis state mostly local to specific tools
- Integrating tools into each other needs effort

avatar2 – the goals

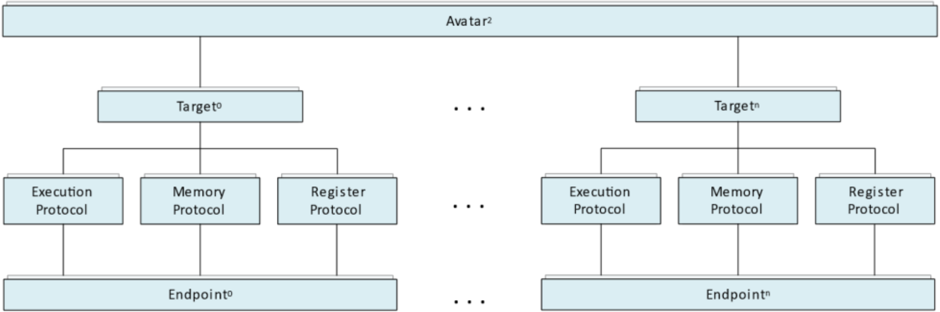
- Interconnecting variety of tools
- Consistent API to the analyst
- Easy scriptability

Partial Emulation



Core concepts

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



Avatar2
Target0
Targetn
Execution Protocol
Memory Protocol
Register Protocol
Execution Protocol
Memory Protocol
Register Protocol
Endpoint0
Endpointn

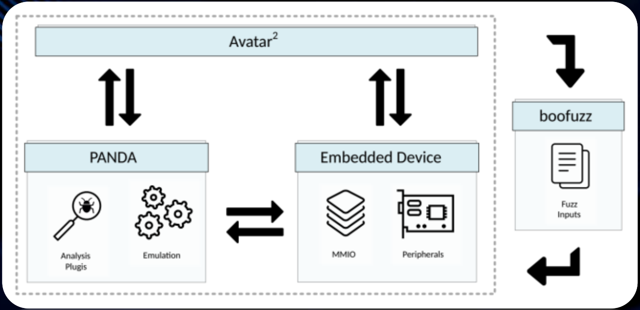
WYCINWYC

What you corrupt is not what you crash [1]

- Focuses on fuzzing embedded devices
- Investigates Fault Detection, Instrumentation & Scalability
- Measurements & Improvements

[1] Muench, M., et.al. "What you corrupt is not what you crash: Challenges in Fuzzing Embedded Devices" NDSS 2018

The setup



Avatar²
PANDA
Analysis Plugis
Emulation
Embedded Device
MMIO
Peripherals
boofuzz
Fuzz Inputs

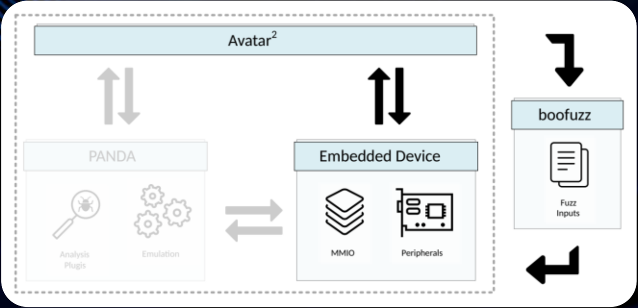
Setup: Fuzzer

- boofuzz, python-based fuzzer based on Sulley
- Used to trigger corruptions with different rations
- Used for 100 fuzzing sessions over 1 hour each

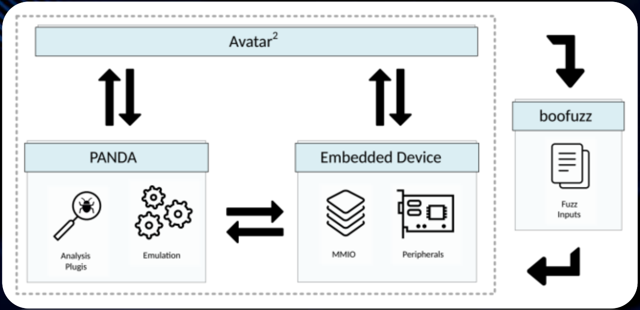
Setup Target

- Vulnerable expat program
- Focus on a Type-III device
- Fuzzed in 4 different configurations

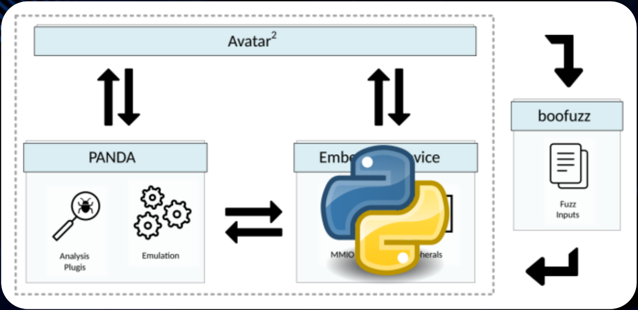
Configurations: Native



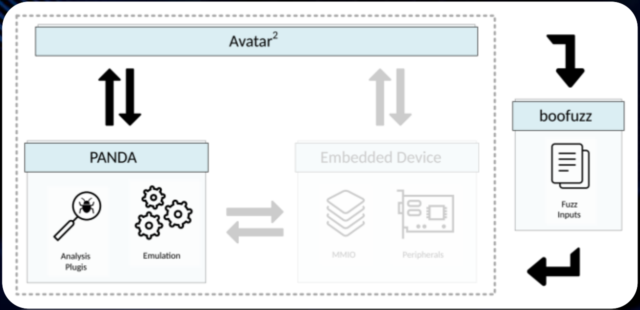
Configurations: Partial Emulation/Memory Forwarding



Configurations: Partial Emulation/Peripheral Modeling



Configurations: Full Emulation



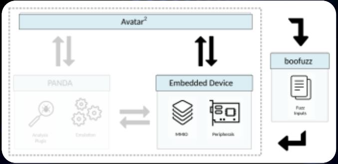
Instrumentation

- Uses plugin infrastructure of PANDA
- Simple heuristics, mimicking already existing techniques

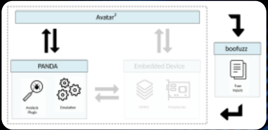
Instrumentation

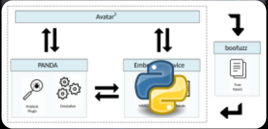
1. Segment Tracking
2. Format Specifier Tracking
3. Heap Object Tracking
4. Call Stack Tracking
5. Call Frame Tracking
6. Stack object Tracking

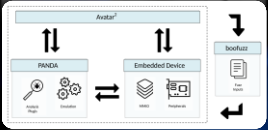
Fault Detection



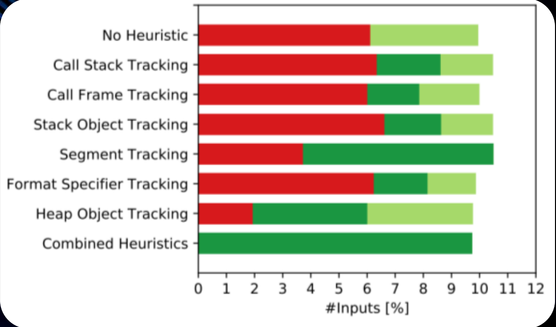
Vs.







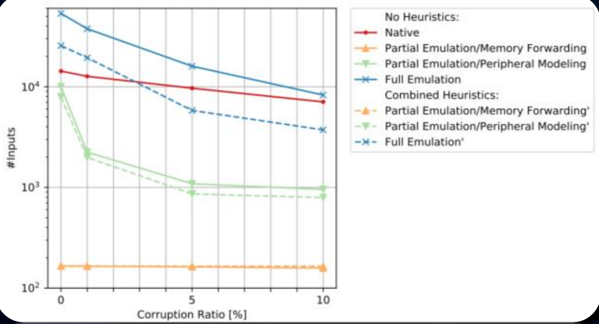
Fault Detection



Corruption detected by:

- *Liveness check*
- *Heuristics*
- *Undetected*

Scalability



Pretender

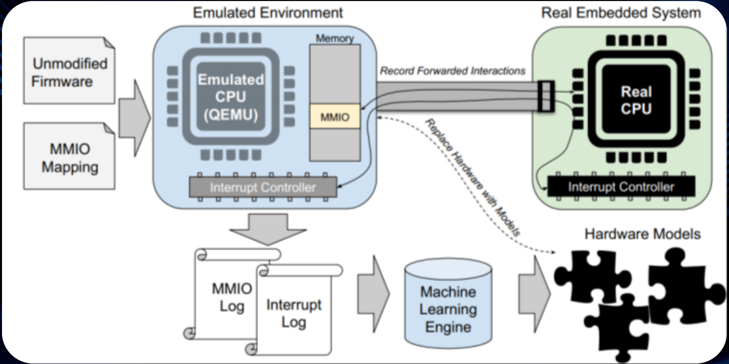
Pretender [2]

- Focuses on automated rehosting
- Tackles platform variety & scalability

[2] Gustafson, E., et.al. " Toward the Analysis of Embedded Firmware through Automated Re-hosting ” RAID 2019

Pretender in a nutshell

- Automated creation of peripheral models
- Based on recorded interaction
- Transition from partial to full emulation


Emulated Environment
Real Embedded System
Unmodified Firmware
MMIO Mapping
Emulated CPU (QEMU)
Memory
MMIO
Interrupt Controller
Record Forwarded Interactions
Real CPU
Interrupt Controller
Replace Hardware with Models
Hardware Models
MMIO Log
Interrupt Log
Machine Learning Engine

Pretender Workflow

- Recording
- Peripheral Clustering
- Interrupt Inference
- Memory Model Training

Evaluation

- Performed on 3 different Type-III devices
- 6 firmware examples:
  - 4 from vendor, 2 custom: thermostat, rf_door_lock

Firmware Name	Peripherals	Blocks Executed			
		Recording	Null Model	Pretender w/ SA	Pretender w/ fuzzing
STM Nucleo L152RE					
blink_led	Timer, GPIO	218	86	218	n/a
read_hyperterminal	Timer, GPIO, USART3	545	85	545	636
i2c_master	Timer, I2C, AM3215	1185	61	1185	n/a
button_interrupt	Timer, GPIO, Button	344	68	314	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1263	62	1261	1276
rf_door_lock *(custom)*	Timer, GPIO, Radio,	665	87	665	758
STM Nucleo F072RB					
blink_led	Timer, GPIO	405	117	405	n/a
read_hyperterminal	Timer, GPIO, USART3	828	102	828	999
i2c_master	Timer, I2C, AM3215	1572	103	1572	n/a
button_interrupt	Timer, GPIO, Button	362	103	362	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1662	103	1662	1918
rf_door_lock *(custom)*	Timer, GPIO, Radio,	960	102	960	972
Maxim MAX32600MBED					
blink_led	Timer, GPIO	280	9	280	n/a
read_hyperterminal	Timer, GPIO, USART3	514	8	514	668
i2c_master	Timer, I2C, AM3215	941	8	942	n/a
button_interrupt	Timer, GPIO, Button	188	8	188	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1009	8	1009	1066
rf_door_lock *(custom)*	Timer, GPIO, Radio,	692	8	692	712

Conclusion

Conclusion (1/2)

- Obtaining Firmware
- Platform Variety
- Fault Detection
- Scalability
- Instrumentation

Conclusion (1/2)

- Obtaining Firmware
- Platform Variety
- Fault Detection
- Scalability
- Instrumentation

Conclusion (2/2)

- Dynamic firmware analysis remains challenging
- Partial emulation can help
- Automated rehosting is promising

THANKS

— TENCENT SECURITY CONFERENCE 2019 —