AVATAR: A Framework for Dynamic Security Analysis of Embedded Systems’ Firmwares

Jonas Zaddach
(zaddach@eurecom.fr)
Luca Bruno, Aurélien Francillon, Davide Balzarotti
Outline

• Introduction
• AVATAR overview
• Framework components
• Use cases
• Conclusion
Software is everywhere

- Embedded devices are **diverse** – but all of them run **software**
Reasons for embedded security

• Embedded devices are ubiquitous
  – Even if invisible, they are essential to your life
• Can operate for many years
  – Legacy systems, no (security) updates
• Have a large attack surface
  – Networking, forgotten debug interfaces, etc
Third party security evaluation

- No source code available
- No toolchain available
- No documentation available
- Distinct tools (to flash and debug) for each manufacturer
Wishlist for security evaluation

- Typical PC security toolbox
  - Advanced debugging techniques
    - Tracing
    - Fuzzing
    - Tainting
    - Symbolic Execution
  - Integrated tools
    - IDA Pro
    - GDB
Challenges

• Advanced dynamic analysis needs emulation
• Full emulation
  – Unknown peripherals
  – Firmware fails if peripherals are missing
• Integration
  – Support multiple vendors and platforms
Outline

• Introduction
• AVATAR overview
• Framework components
• Use cases
• Conclusion
AVATAR

• Orchestrate execution between emulator and device
• Forward peripheral accesses to the device under analysis
• Do not attempt to emulate peripherals
  – No documentation
  – Reverse engineering is difficult
Avatar overview

...  
mov r2, r0  
mov r3, r1  
add r3, r3, #1  
ldr r2, [r2, #0]  
cmp r2, r3  
...
Avatar overview

... mov r2, r0
mov r3, r1
add r3, r3, #1
ldr r2, [r2, #0]
cmp r2, r3
...

Analysis script

Analysis plugins

In-memory stub

Memory Registers CPU state
Avatar overview

... mov r2, r0
mov r3, r1
add r3, r3, #1
ldr r2, [r2, #0]
cmp r2, r3...

Emulator

Avatar

Analysis script

Analysis plugins

Device

In-memory stub

Memory Registers
CPU state

2/24/14
Avatar overview

Emulator

```
... mov r2, r0
mov r3, r1
add r3, r3, #1
ldr r2, [r2, #0]
cmp r2, r3
...```

Avatar

Analysis plugins

Analysis script

Device

In-memory stub

Memory Registers

CPU state

IRQ
Outline

- Introduction
- AVATAR overview
- Framework components
- Use cases
- Conclusion
Emulator

S²E
- LLVM
- Qemu
- Klee
- Analysis plugins

Memory
- Registers
- CPU state

Configuration
- Qemu GDB
- Remote Memory

Avatar
Avatar core

- Emulator
  - Configuration interface
  - GDB interface
  - Remote memory

- Avatar
  - GDB interface
  - Analysis plugins

- Device
  - Analysis script
Target communication

• Either a debugging interface
  – JTAG
  – Debug Serial Interface

• Or code injection and a communication channel
  – Custom GDB Stub + Serial Port
Bottlenecks

- Emulated execution is much slower than execution on the real device
  - Memory access forwarding through low-bandwidth channel is the bottleneck
  - In one case down to ~10 memory accesses/sec.
- Interrupts can saturate debug connection
Improving performance

• Transfer execution/state
  – From the device to the emulator
  – From the emulator to the device

• Migrate memory and code snippets
  – Keep memory regions in the emulator
  – Execute IO-intensive pieces of code on the device
Full separation mode
Memory access optimization

Emulator

State
Register
Memory

Avatar

Device

State
IO Memory

2/24/14 22
Execute code snippets on the device

Emulator

Device

Avatar

Code

State

State
Execute code snippets on the device
Outline

• Introduction
• AVATAR overview
• Framework components
• Use cases
• Conclusion
Use case: Hard Disk

- Recover bootloader protocol with symbolic execution
  - Inject GDB stub
  - Instrument flash loading
  - Inject symbolic values for data read from serial port
  - Keep track of which input leads into which code flow

Use case: GSM Phone

• Search vulnerabilities in SMS decoding routine
  – Connect through JTAG
  – Execute on device until SMS decoding
  – Replace SMS payload with symbolic values
  – Check for symbolic values in
    • program counter
    • load/store address
Use case: Econotag

- Find proof-of-concept bug in user application
  - Connect through JTAG
  - Execute on device until Zigbee packet arrives
  - Replace payload with symbolic values
  - Check for symbolic values in
    - program counter
    - load/store address
We are adding more devices
Outline

• Introduction
• AVATAR overview
• Framework components
• Use cases
• Conclusion
Future work

• Enhance **state** consistency
  – DMA memory changes not tracked
• Automatically **emulate** peripherals
• Improve **symbolic** execution
  – Coherency between HW and SW
  – Improve bug-finding strategies
Conclusion

• AVATAR is a modular open-source tool to
  – Enable dynamic analysis
  – And perform symbolic execution
  – On embedded devices
  – Where only binary code is available

→ A first step towards better analysis tools for embedded systems!
Questions?

• Thank you for listening!
• Open source on github: https://github.com/eurecom-s3/avatar-python
• Project page: http://s3.eurecom.fr/tools/avatar/

Thanks to Pascal Sachs and Luka Malisa who built an earlier prototype of the system, and Lucian Cojocar for applying and extending AVATAR
References

- AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares, Jonas Zaddach, Luca Bruno, Aurelien Francillon, Davide Balzarotti
- Howard: a dynamic excavator for reverse engineering data structures, Asia Slowinska, Traian Stancescu, Herbert Bos
- KLEE webpage: [http://ccadar.github.io/klee/](http://ccadar.github.io/klee/)
- S2E webpage: [https://s2e.epfl.ch/](https://s2e.epfl.ch/)
- S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems, Italy Chipounov, Volodymyr Kuznetsov, George Candea
- The S2E Platform: Design, Implementation, and Applications, Vitaly Chipounov, Volodymyr Kuznetsov, George Candea
- QEMU webpage: [http://qemu.org](http://qemu.org)
- Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations, Istvan Haller, Asia Slowinska, Matthias Neugschwandtner, Herbert Bos
Injecting a debugger

• Requires writing and executing memory
  – Debug menus allow this sometimes
  – A code execution vulnerability can be used
• Requires a communication channel
  – Serial port, GPIO, Power consumption, …
  – GPIO
• Requires an unused memory location in the firmware
  – Stub is about 3k of code
Full separation mode

Emulator

Avatar

Device

State

Register

State

Memory
Memory access optimization
Transfer execution from emulator to device

Emulator

State

Register

Memory

Device

State
Transfer execution from emulator to device
Transfer execution from device to emulator
Transfer execution from device to emulator

Emulator

Device

Avatar

State

Register

Memory

State
Software interrupts

• Software Interrupts
  – Are issued by an interrupt instruction in the code

• Can be entirely emulated
  – Qemu manages calling of software interrupt handlers

http://home.netcom.com/~swansont/interrupt.jpg
Task completion interrupts

• Triggered by application requests
  – Responses aligned with firmware execution speed
  – E.g., signal that a requested DMA transfer has finished

• Can be forwarded from the device to the emulator
  – A stub on the device traps interrupts and forwards them
External event interrupts

• Signals an external event
  – Events aligned to wall-clock instead of execution time
  – E.g., that a time span has elapsed
• Solution depends
  – Controllable interrupts can be forwarded
  – Uncontrollable interrupts need to be synthesized
    • Original interrupts are suppressed
    • Emulated interrupts are inserted according to emulated execution speed