The Avatar project:
Improving embedded security with S²E, KLEE and Qemu

http://www.s3.eurecom.fr/tools/avatar/

Luca Bruno <lucab@debian.org>,
J. Zaddach, A. Francillon, D. Balzarotti
About us

• Eurecom, a consortium of European universities in French riviera
• Security research group
  – 9 people
• Applied system security
  – Embedded systems
  – Networking devices
  – Critical infrastructures
Outline

- Embedded security
- Avatar overview
- Framework components
- Field testing
- Conclusions
Software everywhere

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

• Embedded devices are ubiquitous
  – Even if not visible, your lives depend on them

• Can operate for many years
  – Legacy systems, no (security) updates

• Have large attack surfaces
  – Networking, forgotten debug interfaces, etc.

• Sometime too easy to take-over/backdoor
Challenges in embedded security

• **No source code** available
  – Often monolithic binary-only firmwares

• **No toolchain** available

• **No documentation** available

• **Unique tools** (to flash and debug) for each manufacturer
Wishlist for security evaluation

• Typical PC-security toolbox
  – Advanced debugging techniques
    • Tracing
    • Fuzzing
    • Symbolic Execution
    • Tainting
  – Integrated tools
    • IDA Pro
    • GDB
    • Netzob
Outline

• Embedded security
• Avatar overview
• Framework components
• Field testing
• Conclusions
Why Avatar

- Provide a framework for
  - In-vivo analysis of any kind of device
  - Advanced debugging
  - Easy prototyping

- Integrated workbench
  - To use all techniques together on a live system

- Not only focused on security
  - Debugging/profiling/tracing is hard in embedded environments
Avatar: basics

- Emulate embedded devices’ firmwares
- **Forward peripheral accesses** to the device under analysis
- **Do NOT** attempt to emulate peripherals
  - No documentation
  - Reverse engineering is difficult
Avatar overview

Firmware

Emulator

Emulator Backend

Target Backend

Proxy

Plugins

read/write memory

value

interrupt

read/write memory

value

interrupt

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

...
Avoid NIH syndrome

- **$S^2E$ (Qemu+Klee)**
  - for emulation and symbolic execution
- **GDB and OpenOCD**
  - to attach components and devices
- **Your own tools** for analysis
  - IDA Pro, Capstone, Netzob...
Outline

• Embedded security
• Avatar overview
• **Framework components**
• Field testing
• Conclusions
LLVM under the hood

- **S²E** combines existing tools to achieve symbolic execution of x86/ARM binary code
  - **Qemu** translates binary code to an intermediate representation (TCG)
  - **QEMU-LLVM** translates TCG to **LLVM** bytecode
  - **KLEE** executes LLVM bytecode symbolically
S²E in a nutshell
Python3 framework

Analysis script

Emulator
- Config writer
- GDB interface
- QMP/Lua interface
- Memory forwarder

Avatar

Emulator backend

Analysis Plugins

Target backend
- GDB/MI adapter
- BinProto adapter
- Telnet adapter
- GDB adapter

Target

Emulator

02/02/2014
Analysis platform

• Avatar provides analysis glue
  – Orchestrate execution
  – Bridge between emulator ↔ device
  – Intercept/manipulate memory accesses
  – External integration, exposing GDB or JSON interfaces
Embedded target

Avatar

Open OCD

In-memory stub

Target device

Target state
- Registers
- CPU state
- Memory

UART

JTAG
Target communication

- Either a debugging interface
  - JTAG
  - Debug Serial Interface
- Or code injection and a communication channel
  - GDB Stub + Serial Port
Outline

• Embedded security
• Avatar overview
• Framework components
• Field testing
• Conclusions
Usecases

- Check for hidden backdoors in HDD firmware
- Fuzzing/symbolic execution of SMS decoding on feature phone
- Vulnerabilities check on programmable wireless sensors
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 instr./sec.

• Interrupts are tricky, can overwhelm emulation
Improving performance

- Point of Interest is often far down in the firmware
  - Trap execution on device and transfer state to the emulator

- A large part of forwarded accesses are to non-IO memory
  - Detect and drop forwarding for non-IO memory regions (stack, heap and code in the emulator)

- High-periodicity interrupts can be synthesized to avoid saturation
Outline

- Embedded security
- Avatar overview
- Framework components
- Field testing
- Conclusions
Limitations

- **State** consistency
  - DMA memory changes not tracked

- **Timing** consistency
  - Emulated execution time much slower than real execution time

- **Symbolic** execution
  - Coherency between HW and SW

- **Bug-finding** strategies to be improved
Recap

• Avatar is a tool to
  – Enable dynamic analysis
  – And perform \textit{symbolic execution}
  – On \textit{embedded} devices
  – Where \textit{only} binary code is available
Questions?

Thank you for listening!

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

- AVATAR web page: http://www.s3.eurecom.fr/tools/avatar/
- 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/
- S2E webpage: https://s2e.epfl.ch/
- S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems, Vitaly Chipounov, Volodymyr Kuznetsov, George Candea
- The S2E Platform: Design, Implementation, and Applications, Vitaly Chipounov, Volodymyr Kuznetsov, George Candea
- QEMU webpage: http://qemu.org
- Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations, Istvan Haller, Asia Slowinska, Matthias Neugschwandtner, Herbert Bos
Extra: GDB stub

• GDB can connect to targets using a serial interface and a simple protocol

• There is a stub implementation in the source code tree, but not for ARM and it’s bloated (for our purposes)

• 6 primitives are enough to give debugging support with software breakpoints:
  Read bytes, write bytes, read registers, write registers, continue and get signal