Zeus-IoT: Experimental Evaluation of Integrity Enforcement on OpenWrt

Alireza Roshandel
Manuel Egele
Department of Electrical &
Computer Engineering

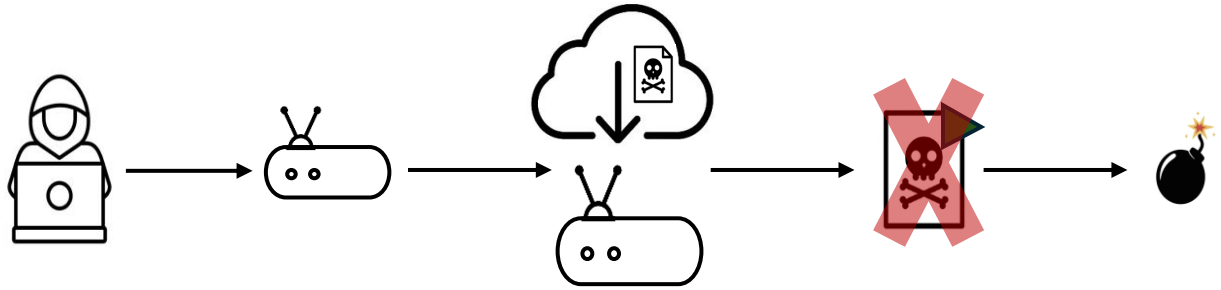


SeclaBU

BOSTON UNIVERSITY

The Problem: IoT Devices Are Being Weaponized

- Billions of IoT devices deployed with minimal built-in security
- Mirai, NoaBot, and other malware show how easily compromised devices become botnets or miners
- Once compromised, attackers run arbitrary binaries or scripts





Goal: Prevent post-compromise code execution on IoT devices

Preventing IoT device weaponization

When was the last time you installed software on your home router?

Insight

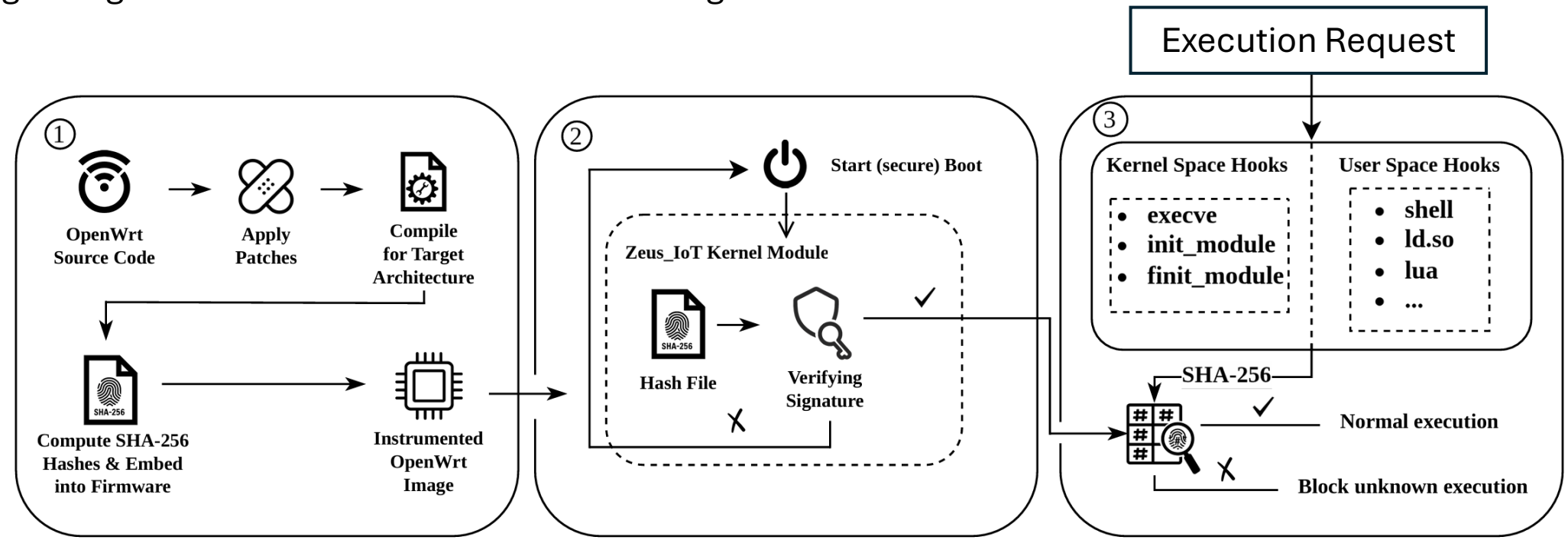
IoT devices run a small, **fixed set of binaries and scripts**, performing specific, well-defined tasks that rarely change after deployment.

Approach

Because the software set on an IoT device is static, we generate an allowlist during the firmware image build process and use it for runtime integrity verification. We also extend this verification into interpreter environments to cover script-based execution.

Zeus-IoT (3 Stage Approach)

A lightweight execution control mechanism designed for IoT devices.



Once per firmware build

Once per device boot

Continuously during the device's lifetime

What This Talk Won’t Cover

- What I won’t cover here
  - Detailed explanation of each enforcement hook
  - Internals of the 3-stage architecture
  - Full design/implementation walk-through
- All of that will be covered in the main Zeus-IoT talk:
  - Thursday, 11 December — 15:30–17:00

What This Talk Will Cover

- The rest of this talk will be:
  - Experimental methodology
  - How we designed and executed the evaluation
  - What our evaluation results reveal
  - Key lessons learned during the development and evaluation process

What We Evaluated

- Evaluation Goals:
  - Malware prevention
  - Hook contribution
  - Runtime overhead
  - Impact on Core Device Functionality
- Environment:
  - OpenWrt 24.0.10 (ARM)
  - QEMU (1 vCPU, 256 MB RAM)
  - GL-MT3000 router





RQ1: Can Zeus-IoT Prevent Malware Execution?

- Dataset
  - Public IoT malware corpus (~166k samples)*
- ARM Subset Provided by Corpus
  - 81,152 ARM IoT malware binaries
- Goal
  - Show Zeus-IoT blocks actual in-the-wild malware

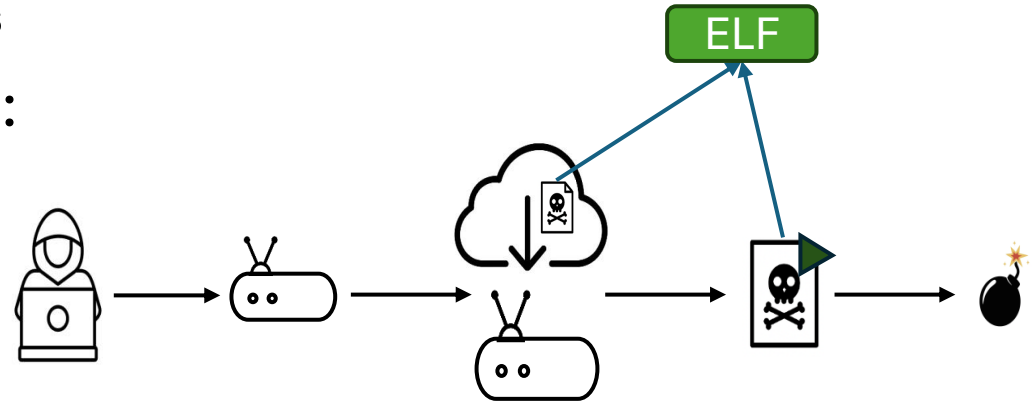
100% prevention
All samples blocked by execve hook

* https://badthings.info/

What This Result Tells Us About IoT Attacks

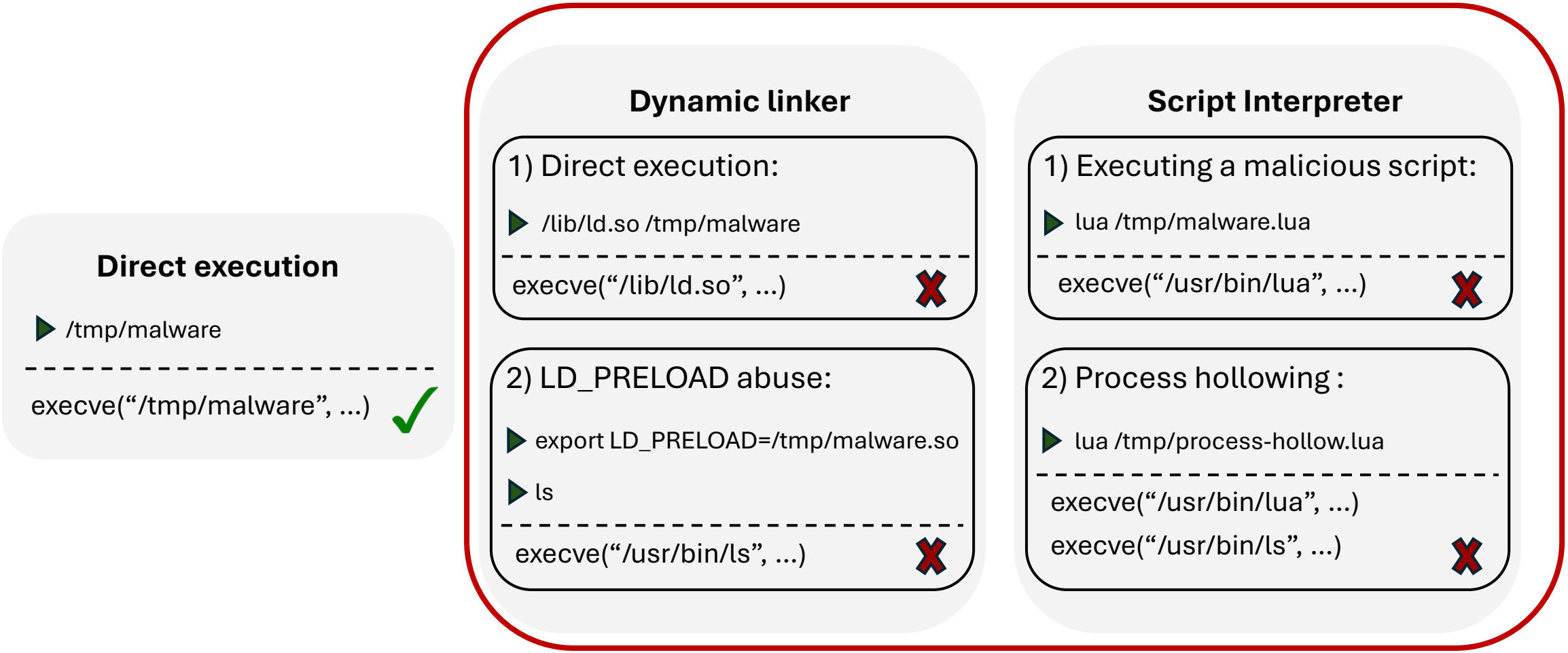
- Almost all caught at **execve** hook
- Dataset = only native binaries
- No real samples (in this dataset) using:
  - ld.so tricks
  - Interpreter scripts
- IoT attacker model:

"We found a set of payloads that rely on script interpreters like Python and Lua for functionality."



Attackers dropped native binaries, and Zeus-IoT stops all of them.
Next: what happens when attackers change techniques?

Different Execution Paths



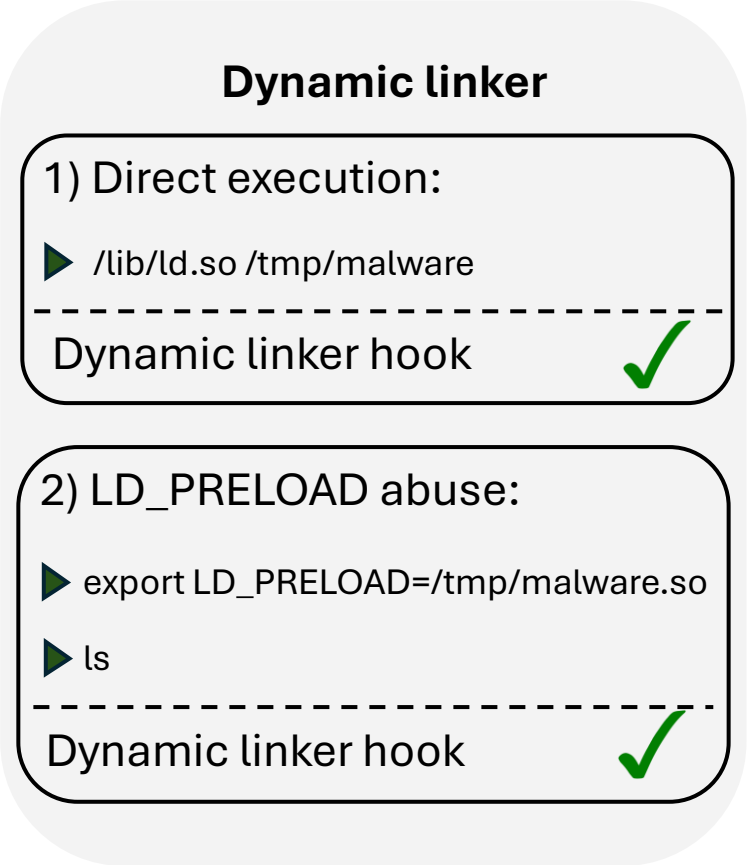
RQ2: Ablation Study of Zeus-IoT's Enforcement Hooks

- Approach
  - Keep all kernel-space hooks active
    (i.e., execve, (f)init_module, write, unlink)
  - Turn OFF all user-space hooks:
    - Dynamic linker hook
    - Lua interpreter hook
    - Shell interpreter hook
  - Attempt attacks on each execution path
  - Re-enable one user-space hook at a time

Map each user-space hook → to the specific execution vector it prevents

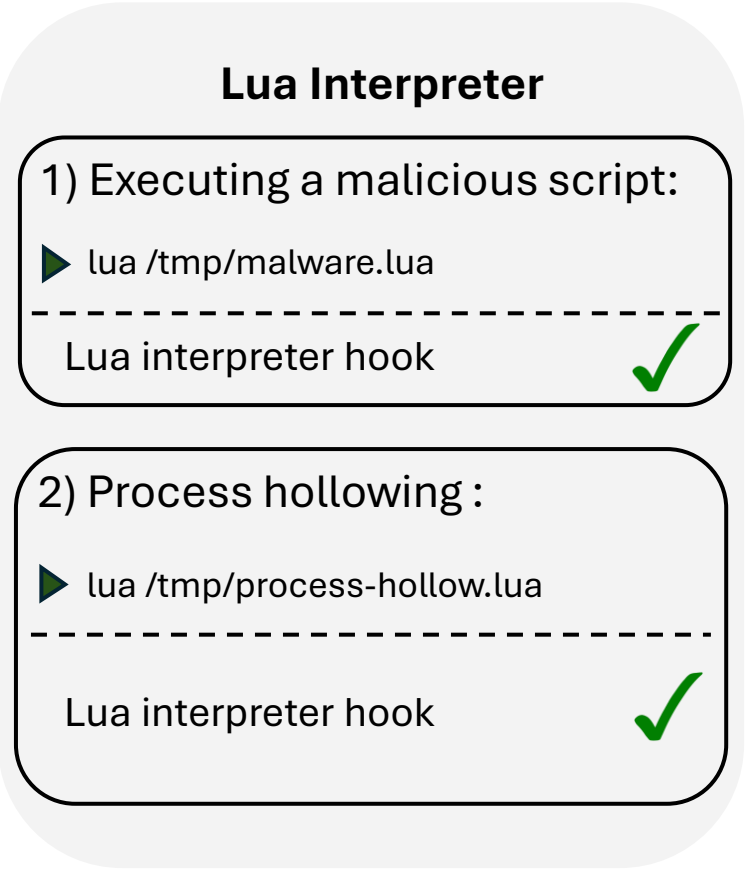
RQ2: Dynamic Linker Hook

- No real IoT malware using this path
  - Found no IoT malware that abuses dynamic linker
  - Built custom PoCs (“Hello World!” payloads)
- Kernel-Only Enforcement Fails!
  - *execve* measures only the linker binary
  - Payload ELF is never verified
- With Zeus-IoT Linker Hook Enabled
  - Linker hook verifies every ELF + shared library before loading
  - Allow only allowlisted payloads



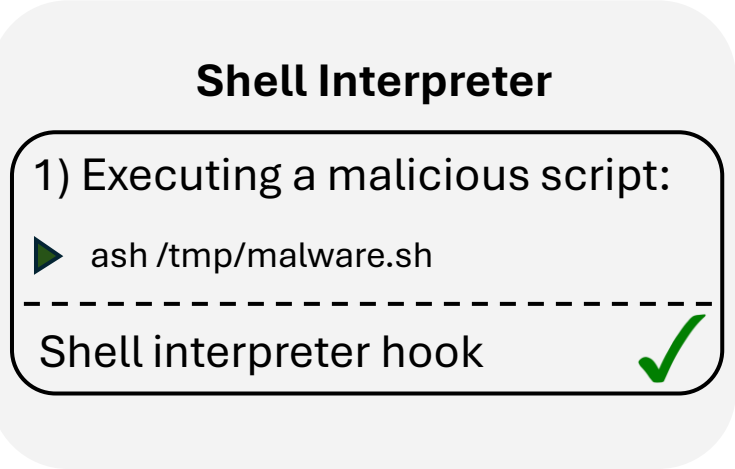
RQ2: Lua Interpreter Hook

- Kernel + Linker Hooks Still Fail!
  - Lua interpreter loads .lua scripts inside its own process
  - execve and linker hooks are not triggered
  - Payload script is never verified
- With Zeus-IoT Lua Hook Enabled
  - Hook hashes each script file at load time
  - Blocks scripts not in the allowlist
  - Blocks inline or stdin execution



RQ2: Shell Interpreter Hook

- Dataset
  - Collected shell-scripted IoT malware from MalwareBazaar
  - Mirai: 157 samples, Gafgyt: 23 samples
- Baseline (Before Shell Hook)
  - Shell runs script content inside its own process
  - Kernel/linker hooks don’t verify script files
  - Payload script is never verified
- With Zeus-IoT Shell Hook Enabled
  - Shell hook verifies every file-based script before execution
  - Blocks scripts not in the allowlist



RQ2: Shell Inline/Stdin Execution (Remaining Gap)

- Bypass via Inline/Stdin Execution
  - cat malware.sh | ash
  - Shell receives script content from stdin → no file to verify
- Behavior of Real Malware Samples
  - 40 representative shell-scripted malware samples
  - All samples act as droppers for multi-arch ELF binaries
  - Even with inline execution, **execve hook blocks all downloaded ELFs**
- Why This Path Remains Open
  - Inline/stdin shell execution required by OpenWrt system utilities
  - Integrating a restricted shell into OpenWrt is **future work**

```
ash -c '(echo %s; sleep 1; echo %s) | passwd %s'
```

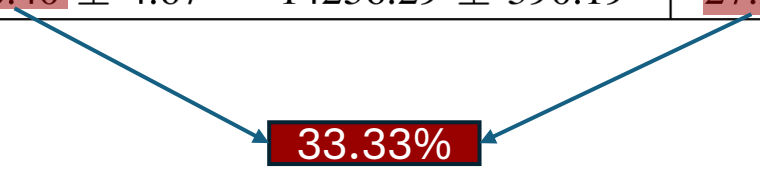
Moving From Effectiveness → Efficiency

- What We've Shown So Far:
  - Zeus-IoT blocks:
    - Native ELF malware (**RQ1**)
    - Linker-based execution paths (**RQ2 – Dynamic Linker**)
    - Interpreter-based execution paths (**RQ2 – Lua / Shell**)
  - Kernel-only enforcement is **not enough**
  - User-space hooks are **necessary and effective**
- What We Evaluate Next?
  - How much **CPU**, **memory**, does Zeus-IoT consume?
  - Is the system usable on **real IoT hardware**?

Zeus-IoT: Runtime Performance Evaluation(RQ3)

- Periodic shell script (1 Hz) executing 4 programs (static ELF, dynamic ELF, shell script, Lua script)
- CPU & memory usage tracked via *sar* over 5-, 10-, 15-, and 20-min intervals, repeated × 10

Time Interval	Stock OpenWrt		Zeus-IoT	
	CPU (%)	Memory (KB)	CPU (%)	Memory (KB)
5 minutes	20.36 ± 4.57	15194.37 ± 1190.33	27.33 ± 6.46	14346.98 ± 331.77
10 minutes	20.37 ± 4.70	13876.20 ± 143.88	27.67 ± 6.47	14083.95 ± 115.92
15 minutes	20.37 ± 4.70	14048.58 ± 110.78	26.89 ± 6.26	14090.59 ± 139.97
20 minutes	20.51 ± 4.73	13906.01 ± 115.76	26.91 ± 6.36	14257.98 ± 142.37
Average	20.40 ± 4.67	14256.29 ± 390.19	27.20 ± 6.39	14199.38 ± 182.51



Current Workload ≠ Real IoT Behavior

- High process-spawn rate (1 Hz) is not representative of real IoT behavior
- Real IoT devices (e.g., routers) spawn far fewer processes in normal operation
- Zeus-IoT overhead appears only when **new processes are spawned**:
  - Boot (19s): ~2.7s of checks
  - First config (300s): ~25s of checks
  - Steady state: 0s of checks

Since startup events are rare, Zeus-IoT's overhead is negligible in practice.

Are CPU & Memory Correct Metrics for Evaluation? Are We Done?

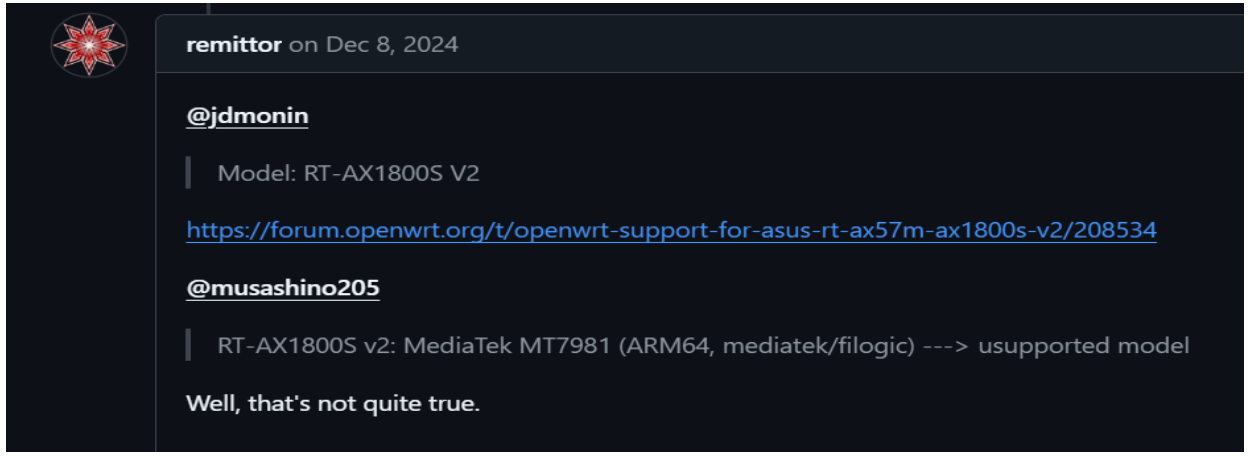
- CPU & memory show resource usage, not real functionality
- IoT devices must preserve their **core function** under enforcement
- For routers, that core function is packet routing
- So CPU/memory alone cannot answer:

Is Zeus-IoT deployable on real IoT devices?

- RQ4: Does Zeus-IoT introduce any overhead on packet routing?

Challenges with OpenWrt on ASUS AX1800s Router

- Router Model Confusion
  - OpenWrt page listed AX1800s with MIPS architecture (v1)
  - **AX1800s v2** with **ARM architecture**
- Flashing Issue
  - Flashed the wrong OpenWrt version, resulting in a **soft bricked router**
  - Recovered using **TFTP recovery mode** and the **stock firmware**





Key Takeaways



- Product (Marketing) names can be misleading
  - Small variations in model names (e.g., AX1800s v1 vs. AX1800s v2)
  - Different architectures (MIPS vs. ARM) lead to flashing issues
- Lack of official support
  - No official OpenWrt support for certain hardware revisions
  - Rely on community contributions for correct firmware and modifications

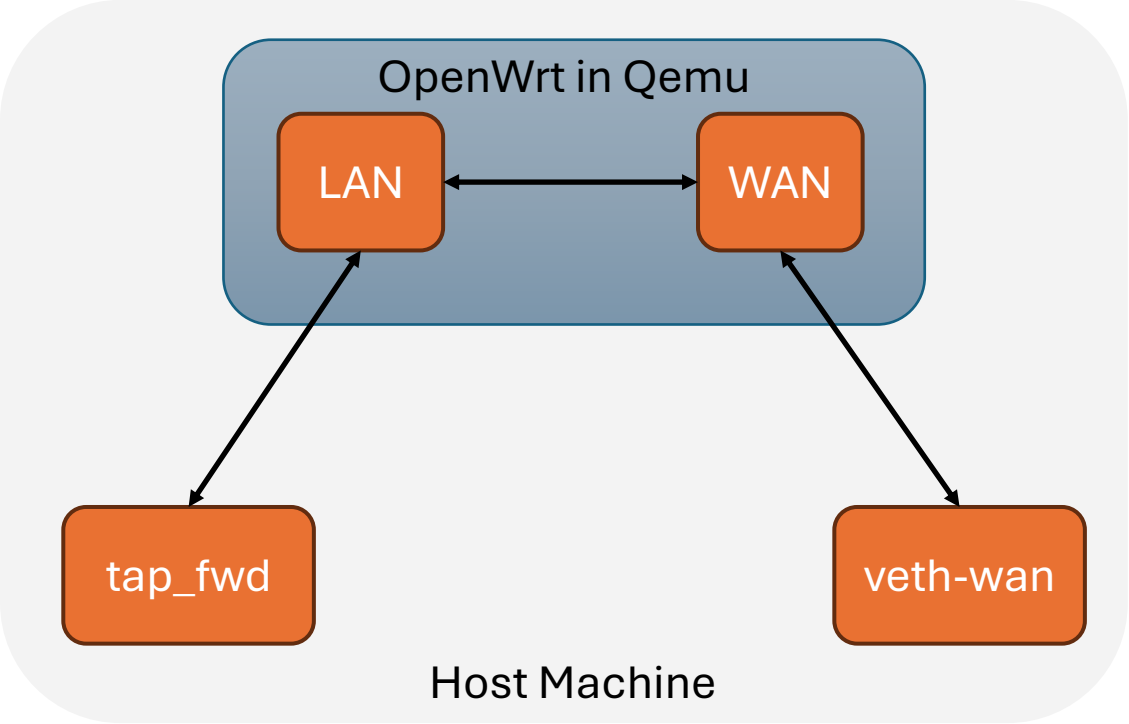
RQ4: Does Zeus-IoT introduce any overhead on packet routing?

- Why We Measure This
  - The **core functionality** of a router is **forwarding packets quickly and reliably**
  - Any security system must **not** degrade routing performance
  - Zeus-IoT performs checks only when:
    - executables load
    - processes spawn
  - Packet forwarding happens entirely in the kernel dataplane
    - No new processes → no integrity checks
    - Zeus-IoT **not** on the forwarding path

Expected result: negligible throughput impact

RQ4: Network Throughput Performance Evaluation Setup

- QEMU Setup:
  - Host machine is connected to OpenWrt running in QEMU, with LAN and WAN interfaces.
  - tap_fwd connects the host to OpenWrt's LAN.
  - veth-wan connects to OpenWrt's WAN interface.
- Router Setup:
  - GL.iNet GL-MT3000, running OpenWrt



RQ4: Measured Throughput Overhead

- Evaluation Methodology
  - Used **iperf3** in client–server mode
  - 10 trials × 60 seconds each

Environment	Stock OpenWrt	Zeus-IoT	Overhead
QEMU	mean=1.631 Mbps, stddev=0.06	mean=1.629 Mbps, stddev=0.07	0.12%
GL.iNetGL-MT3000	mean=871.9 Mbps, stddev=16.27	mean=864.8 Mbps, stddev=9.42	0.8%

Network Throughput: Without vs. With Zeus-IoT

Zeus-IoT: Summary



- IoT devices have specific and fixed functions, unlike general-purpose systems.
- This insight allowed us to use a statically generated allowlist.
- Zeus-IoT extended integrity enforcement to interpreters (Lua, Shell) and the loader, addressing a gap in previous works.
- We evaluated Zeus-IoT on 81 K real IoT malware samples, showing 100% prevention, and tested on real hardware with no impact on core functionality.
- Main Talk: Thursday, 11 December — 15:30–17:00



Code, Results, & Artifacts
https://github.com/BUseclab/ZEUS_IoT

BOSTON UNIVERSITY