# WHOIS

- Jos Wetzels\textsuperscript{1,2}
  - samvartaka.github.io

- Ali Abbasi\textsuperscript{1,3}
  - http://wwwwhome.cs.utwente.nl/~abbasia/

\textsuperscript{1}Distributed and Embedded System Security (DIES) group, University of Twente, Netherlands

\textsuperscript{2}SEC Group, Eindhoven University of Technology, Netherlands

\textsuperscript{3}SYSSEC Group, Ruhr-University Bochum, Germany
EMBEDDED BINARY SECURITY

• **Claim**
  Embedded binary security lags behind General-Purpose World significantly

• **Claim**
  Catching up will be non-trivial due to embedded constraints
BINARY SECURITY & MEMORY CORRUPTION

• Memory Corruption among most common embedded vulns

• Embedded Development Dominated by C*

• Unsafe Language
  • “C is a terse and unforgiving abstraction of silicon”

* 2015 UBM Study, ** 2016 Kaspersky Study
IMPACT: REMEMBER SHADOW BROKERS?

Shadow Brokers launch auction for Equation Group hacking cache

Shadow Brokers' Tools Pose Zero-Day Risks, Cisco and Fortinet Warn

Kaspersky Says that Shadow Brokers Leaked Malware is Genuine

Shadow Brokers' Cisco vulnerability exploited in the wild

FORTINET

CISCO

TOPSEC
How many of the EQUATION 0day would have been fully or mostly mitigated if vendor binaries had basic anti-exploit compiler options enabled?
WHY EXPLOIT MITIGATIONs?

Buffer Overflow  VS  Shell

Buffer Overflow  Infoleak  ROP Payload  Sandbox Escape  Shell
WHY EXPLOIT MITIGATIONS?

• But why not tackle things at the root (eg. safe langs)?
  • Portability → C Toolchain For Nearly Every Platform and OS
  • Functionality → ‘Close To Metal’
  • Performance → Critical for Embedded
  • Existing Codebases → Billions of Lines of Legacy Code

• Need Short-Term Solution
WHAT MITIGATIONS ARE WE TALKING ABOUT?

- Complementary Baseline
  - Executable Space Protection (ESP)
    - DEP, ExecShield, W ^ X

- Address Space Layout Randomization (ASLR)

- Stack Canaries
  - Stack Smashing Protector (SSP), /GS

- Future Work: Advanced Mitigations
  - Control-Flow Integrity (CFI)
EXPLOIT-FLOW VS MITIGATION

1. Overflow stack buffer
   - Overwrite saved return address
     - Stack Canaries
       - Attacker has to know secret canary value
     - With shellcode / ROP gadget address
       - Address Randomization
         - Attacker has to know randomized shellcode / gadget address
       - Execute shellcode
     - Executable Space Protection
       - Data memory is non-executable
   - Execute ROP chain
     - uh-oh...
EMBEDDED OS MITIGATION SUPPORT*

EXECUTABLE SPACE PROTECTION
- Support 44%
- No support 56%

ADDRESS SPACE LAYOUT RANDOMIZATION
- Support 22%
- No support 78%

STACK CANARIES
- Support 33%
- No support 67%

* Based on our research into 36 popular embedded OSes
EMBEDDED SYSTEMS ARE DIVERSE
SUPPORT AMONG MOST CONSTRAINED OSES

**EXECUTABLE SPACE PROTECTION**
- Support 15%
- No support 85%

**ADDRESS SPACE LAYOUT RANDOMIZATION**
- Support 0%
- No support 100%

**STACK CANARIES**
- Support 5%
- No support 95%
ADDRESS SPACE LAYOUT RANDOMIZATION (ASLR)
VIRTUAL MEMORY

* Based on our research into 36 popular embedded OSes
EXECUTABLE SPACE PROTECTION (ESP)

Harvard Architecture

Software ESP

Hardware ESP
HARDWARE FEATURES

- **Von Neumann Cores**
- **Hardware Feature Support**
  - Hardware ESP
  - MMU
  - MPU

*Based on our research into 51 popular embedded core families*
STACK CANARIES

• Compiled without canaries

• Compiled \textit{with} canaries
OS CSPRNGs (eg. /dev/urandom)

*Based on our research into 36 popular embedded OSes*
THAT DOESN’T LOOK GOOD...
ADDRESSING ESP CHALLENGES

- Use Harvard CPUs (e.g. AVR, 8051, PIC)
- Or Neumann with HW ESP (e.g. ARMv6+, MIPS32r3+, x86)

OPEN PROBLEM
- Multi-Arch, Low Overhead SW ESP for Embedded OSes

OPEN PROBLEM
- How to deal with MPU/MMU-less systems
ADDRESSING ASLR CHALLENGES

• ASLR is runtime diversification

• Potential Alternatives w/o VM
  • Compile-time Diversification
  • Install-time Diversification

• Downsides: less effective
  • Only between builds or devices
  • Only code, not data memory

ASLR

Inherent Problem (real-time conflicts, overhead, expensive, etc.)

Virtual Memory

OS Features

Hardware

MMU

OPEN PROBLEM

Need Embedded Alternative for ASLR
ADDRESSING OS CSPRNG CHALLENGES

- Cannot trivially port existing designs*
  - Entropy Problems, Resource Constraints, ...

- **Ideal**: Omnipresent On-Chip high-throughput TRNGs

- PRNG Research Directions
  - Lightweight Crypto (e.g. ACRYPT Project)
  - Omnipresent Entropy Sources: SRAM Start-Up Values, Clock Jitter, ...

* See our 33C3 Talk “Wheel of Fortune: Analyzing Embedded OS Random Number Generators”
FUTURE: ADVANCED EMBEDDED MITIGATIONS

• Control-Flow Integrity (CFI) for High-End Embedded Systems
  • Enforces program execution matches legitimate Control-Flow Graph
  • Prevents Control-Flow Hijacking
  • Recently rolled out in GP world (eg. MS CFG, Clang CFI, PaX RAP, etc.)
EMBEDDED CFI CHALLENGES

• Low *Worst-case* Runtime & Memory Overheads

• Real-Time Friendliness
  • Predictable, deterministic response times

• Availability Preservation
  • Eg. mitigation terminating powerplant PLC runtime

• COTS Binary Support
  • Drivers, PLC runtimes, ...

• Hardware Agnostic
  • eg. no CET, LBR, PMU reliance
OUR WORK: μSHIELD & ECFI

• PREEMPTIVE* EU FP7 Project (papers forthcoming)
  • μShield**: COTS binary support + Heuristics
  • ECFI: Real-Time Friendly

Additional Heuristics:
* Stackframe Integrity Walker
* Kernel ESP enforcement
+ Minimal Sandbox

Availability Preservation:
* Only log alerts, don't terminate OR clean device reset

* http://preemptive.eu, ** https://github.com/preemptive-FP7/uShield
CALL TO ACTION

• (Keep) Raising Awareness
  • Continue to demonstrate urgency & impact of embedded vulns

• Short term solutions
  • Address mitigation challenges (researchers)
  • Adopt mitigations (OS Devs)

• Long term solutions
  • Embedded Safe Language Adoption
  • Secure Embedded Patching & Updating (see FTC $25k prize)
  • Need for IoT Standardization, Policy & Regulation ("Security by Design")