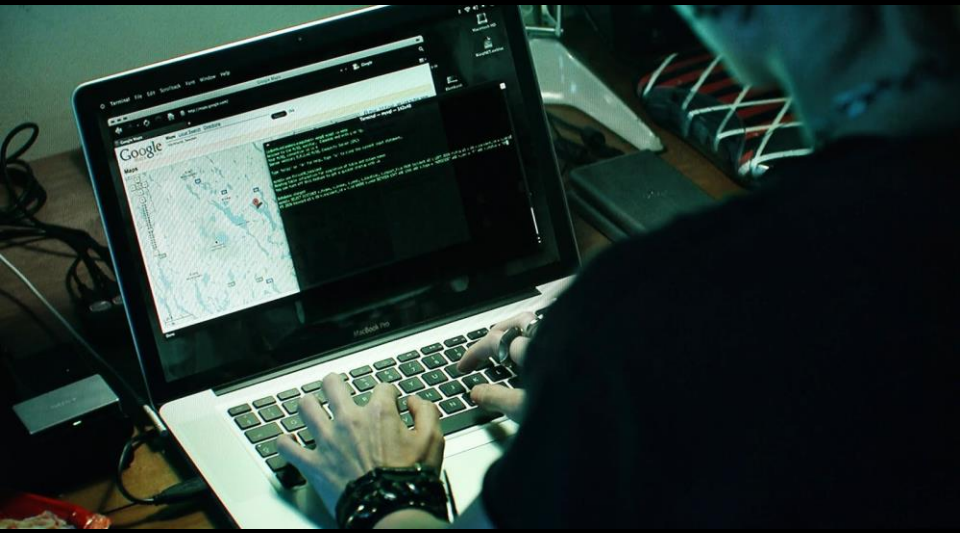
The RTOS Exploit Mitigation Blues

Jos Wetzels

HARDWEAR.IO



Jos Wetzels

Midnight Blue

Independent Security Researcher

ICS
IoT
Automotive
Medical
Access Controls
Networking & Firewalls

UNIVERSITY OF TWENTE.

(Former) Security Researcher

Critical Infrastructure
Embedded BinSec
HIDS / NIDS

@s4mvartaka

www.midnightbluelabs.com
samvartaka.github.io

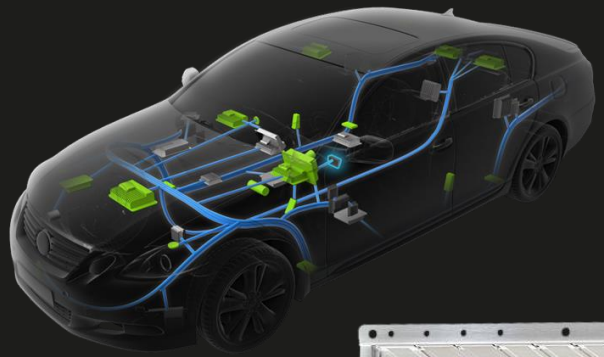
ROADMAP

- Background
- Quantitative Analysis
- RTOS Mitigation Examples
- Embedded Mitigation Criteria
- µArmor

What this talk is ***not*** about

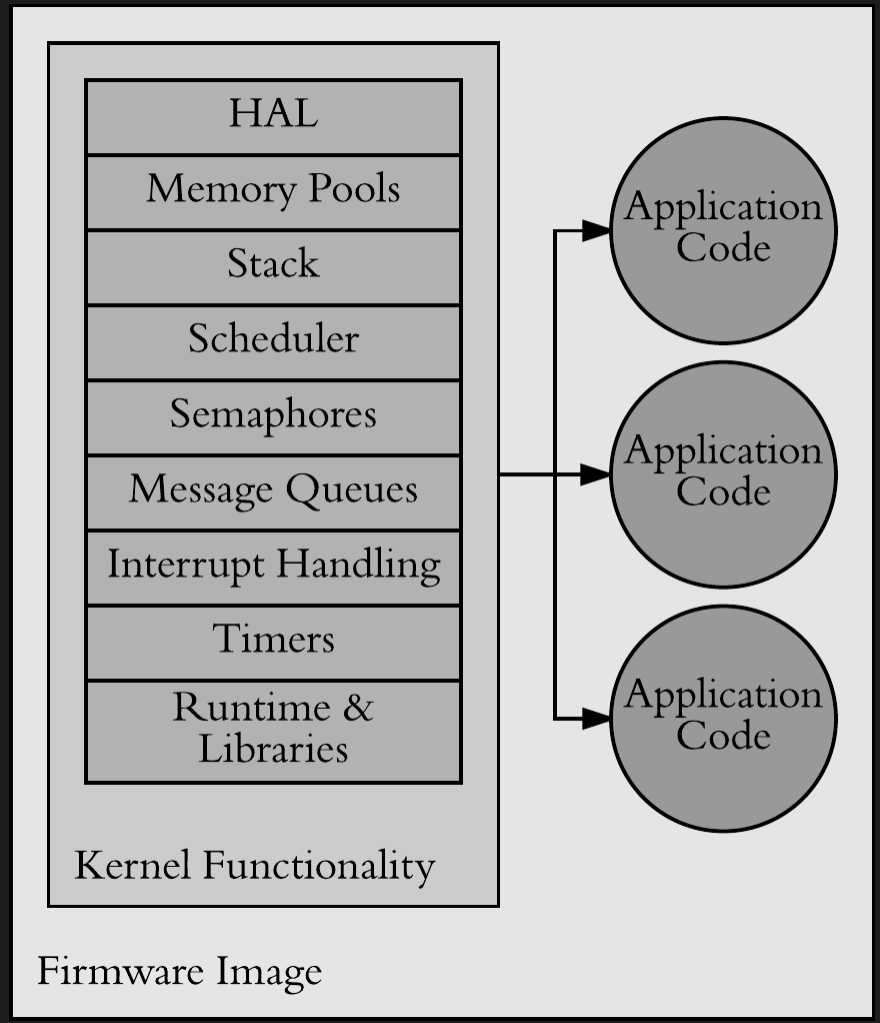
What this talk is also *not* about

What this talk *is* about



Real-Time Operating System (RTOS)

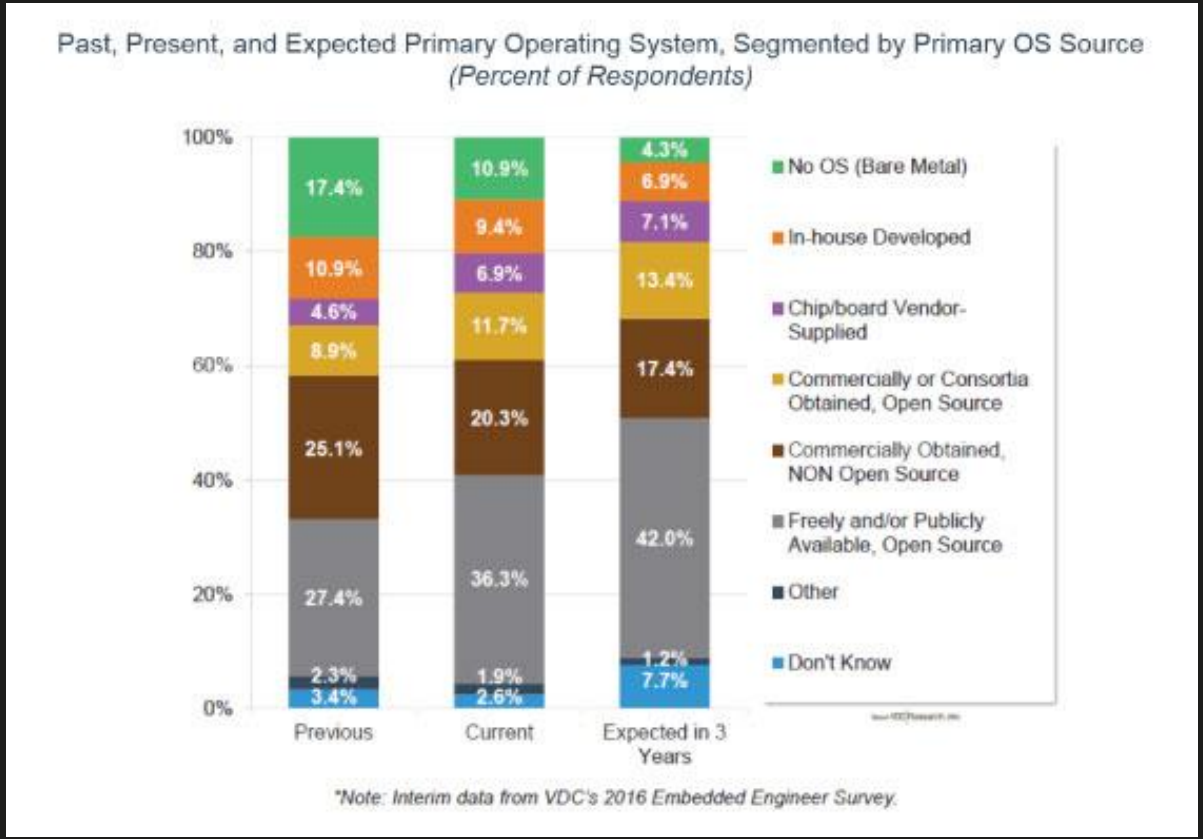
- **Guarantee Response within Time Constraints**
  Important in safety-critical sys.
- **Predictability & Determinism > Speed**
- **Soft RT -> Usefulness of result degrades after deadline**
- **Hard RT -> Missing deadline = system failure**

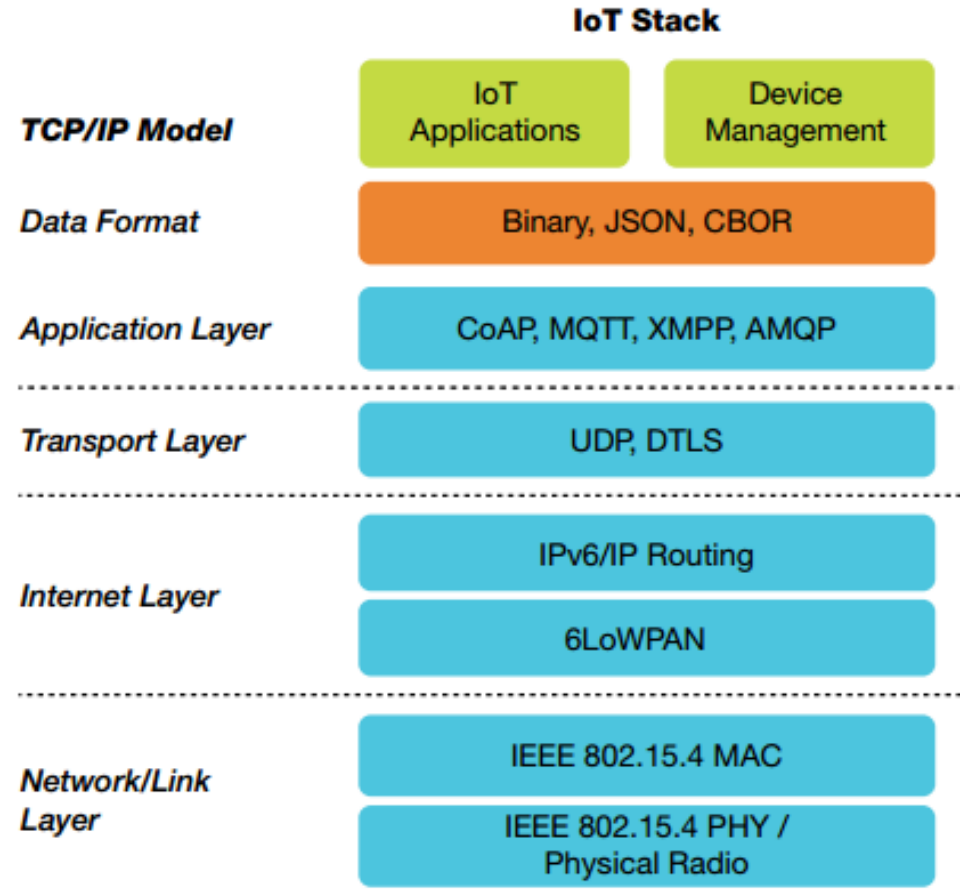


Library / Unikernel OS

- **OS as collection of libraries**
  Linked with application code into single address space image
- **No loader, Syscalls implemented as function calls**
  Saves context-switching

Rising Embedded RTOS Usage





Increasing Connectivity

- **Protocol Stacks Everywhere**

  Talk to gateway, other nodes, smartphones, integration with cloud, etc.

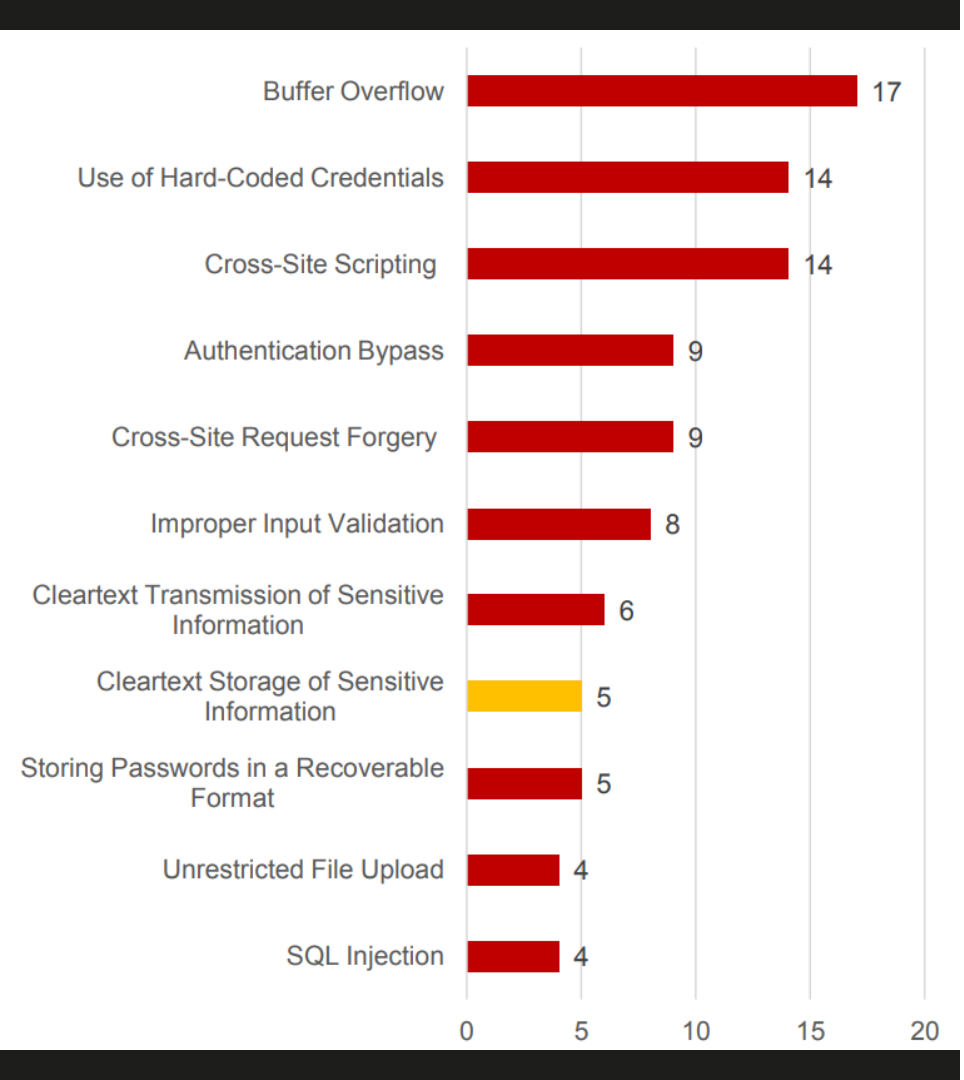
# Processors	Bare Metal	RTOS	Linux
1	**33%**	18%	13%
2-3	22%	**24%**	17%
4+	11%	25%	**32%**

** 2017 Barr Group Embedded Systems Safety & Security Survey*

Increasing Complexity

- More processors, memory, peripherals
- Simple bare metal control loops & state machines not enough
- Hence rise of RTOS


Embedded Binary Security



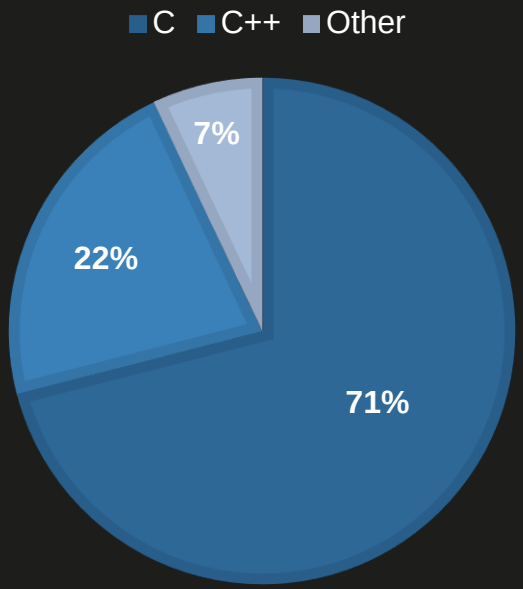
Memory Corruption is a Big Deal™ in Embedded

** 2016 ICS Vulnerabilities Statistics, Kaspersky*

Heightened Impact

- **Patching Issues**
  Fragmented Responsibility
  Lack of Secure Update
  Availability / Safety
- **Long Exposure Windows**
  ‘Forever Days’
- **No Privilege Separation**
  Flat Memory Model
- **Exploits Scale Well**
  Single protocol driver bug can affect billions of devices

PRIMARY EMBEDDED PROGRAMMING LANGUAGE



* 2017 Barr Group Embedded Systems Safety & Security Survey

Unsafe Languages Are Here To Stay

- **Ideally use safe languages**
- **But unsafe continues to dominate**

Despite old (Ada) and upcoming (Rust) embedded-suited langs

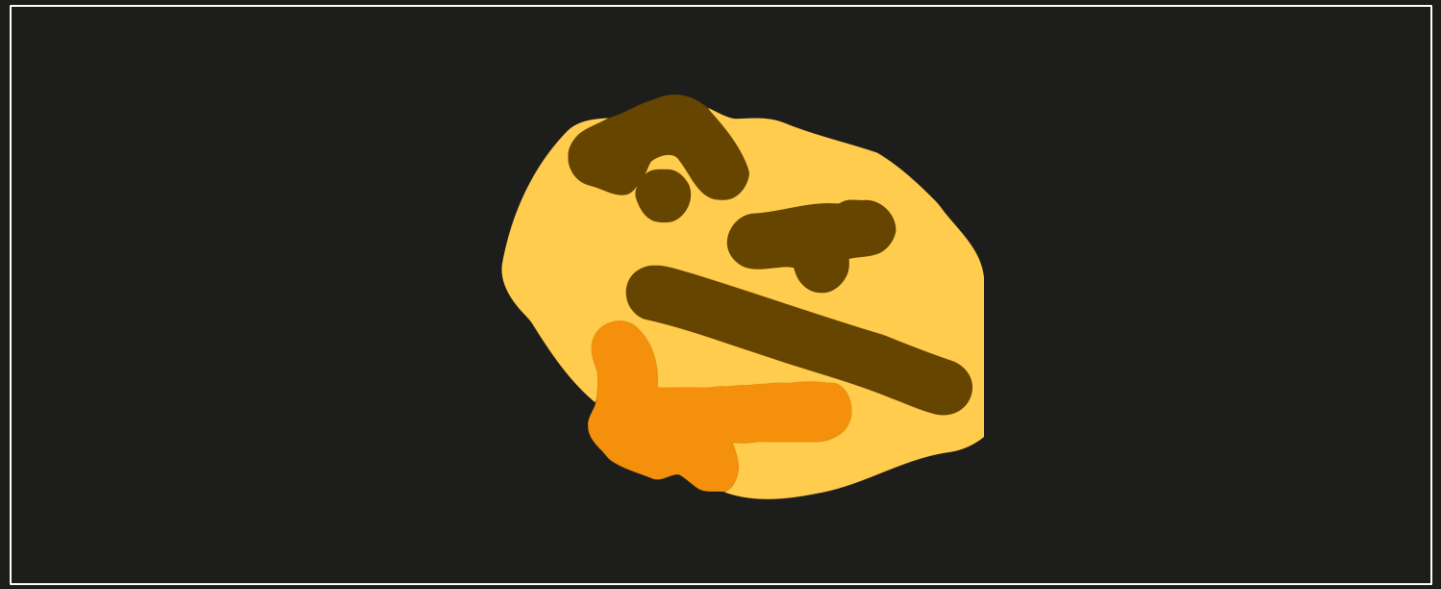
Exploit Mitigations

General Purpose Exploitation Has Been Getting Harder

2010

2016

What About Embedded?




Right...

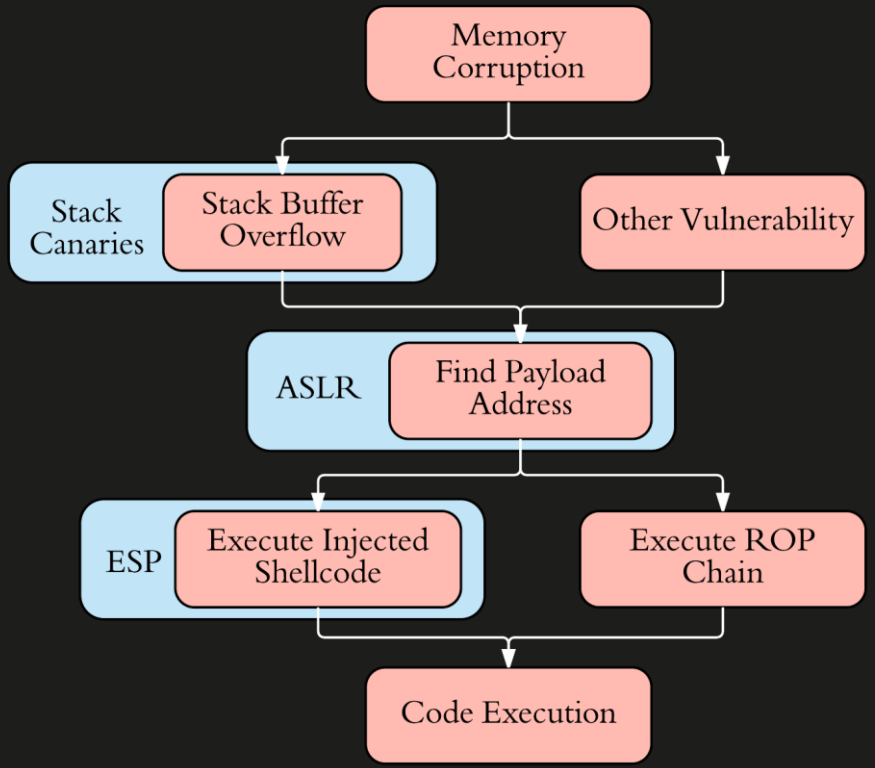
RTOS Exploit Examples

Name	Target	Type	RTOS	Mitigations
BroadPWN / CVE-2017- 9417	Broadcom WiFi	Heap Buffer Overflow	VxWorks	NO
CVE-2017-0561	Broadcom WiFi	Heap Buffer Overflow	VxWorks	NO
VxWorks DHCP / DNS	VxWorks DHCP / DNS Client & Server	Heap / Stack Buffer Overflow	VxWorks	NO
CVE-2015-7599	VxWorks RPC	Integer Overflow	VxWorks	NO
CVE-2010-3832	Intel/Comneon Baseband	Heap Buffer Overflow	ThreadX / Nucleus+	NO[1]
Qualcomm UMTS AUTN	Qualcomm Baseband	Stack Buffer Overflow	REX	NO[2]

[1] XMM6180 on iPhone4 has DEP, see '*All Your Baseband Are Belong To Us*'

[2] Newer Qualcomm chips run BLAST RTOS w. stack cookies / DEP / kernel-user separation, see '*Baseband Exploitation in 2013*'

Quantitative Analysis



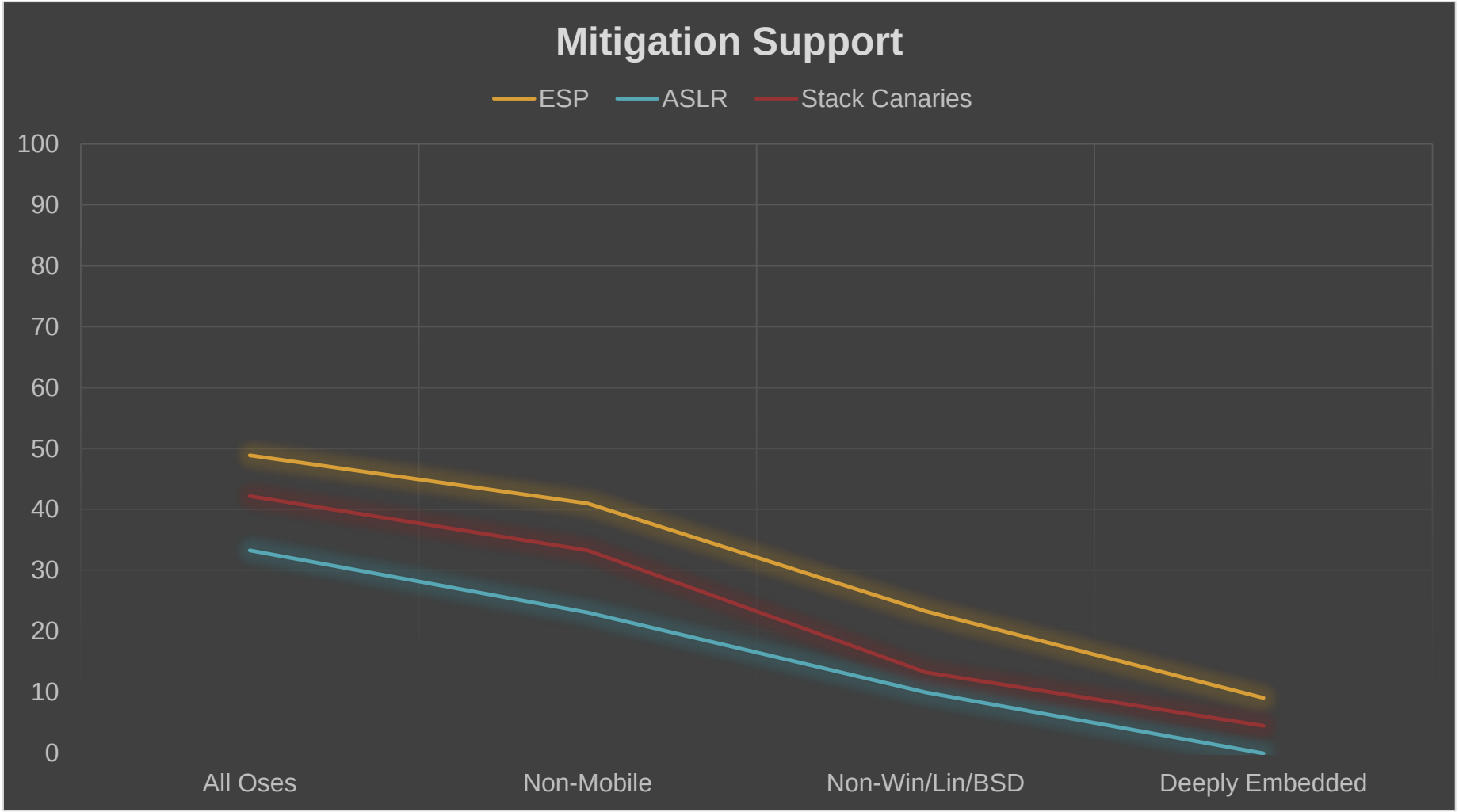
Minimum Mitigation Baseline

- **ESP / DEP / NX / W^X**
  Non-exec. data memory
- **ASLR**
  Address Space Layout Randomization
- **Stack Canaries / SSP**
  Stack buffer overflow protection

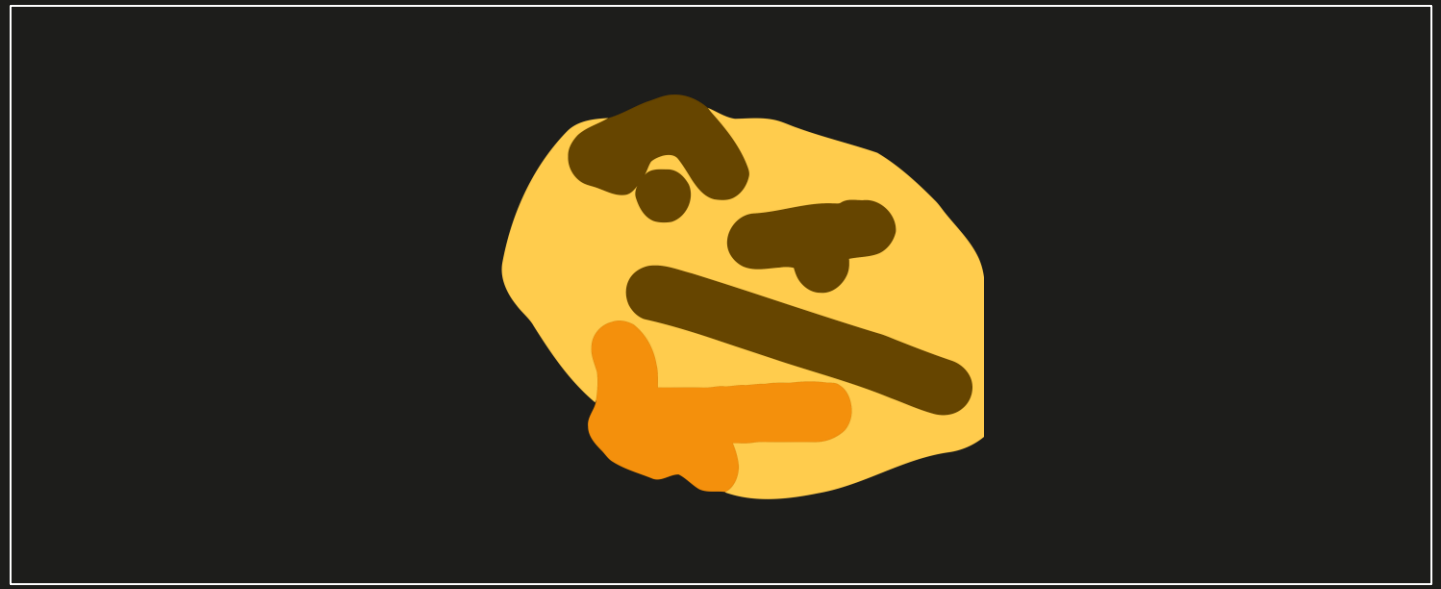
Operating System Selection

- Selected 45 Popular Embedded Oses
- High-end, Low-end, Linux/Windows/BSD-based, proprietary, etc.
- Evaluated Support For Mitigation Baseline

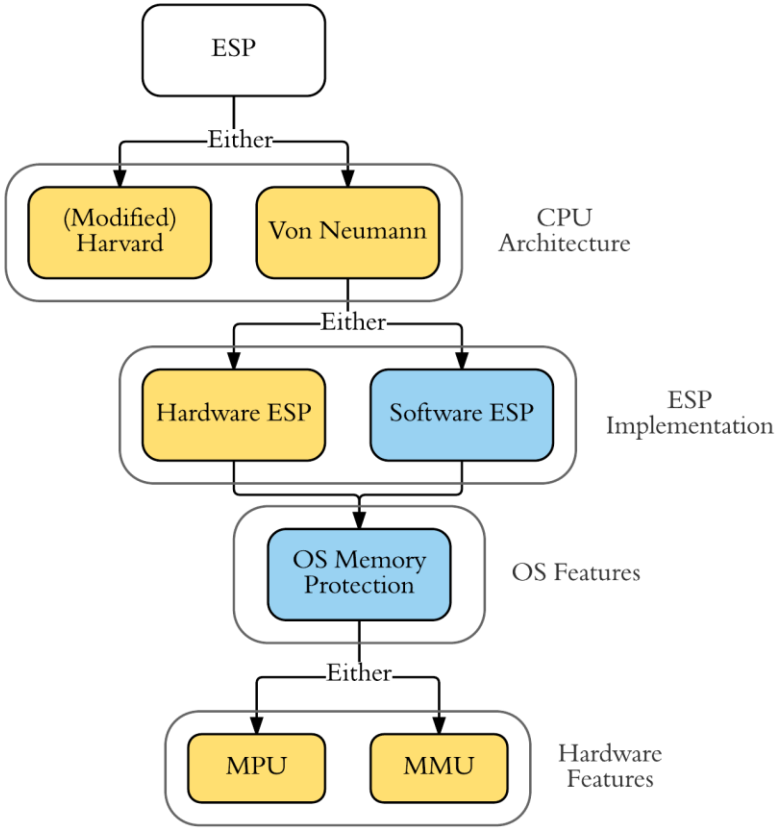
Optimistic Assesment

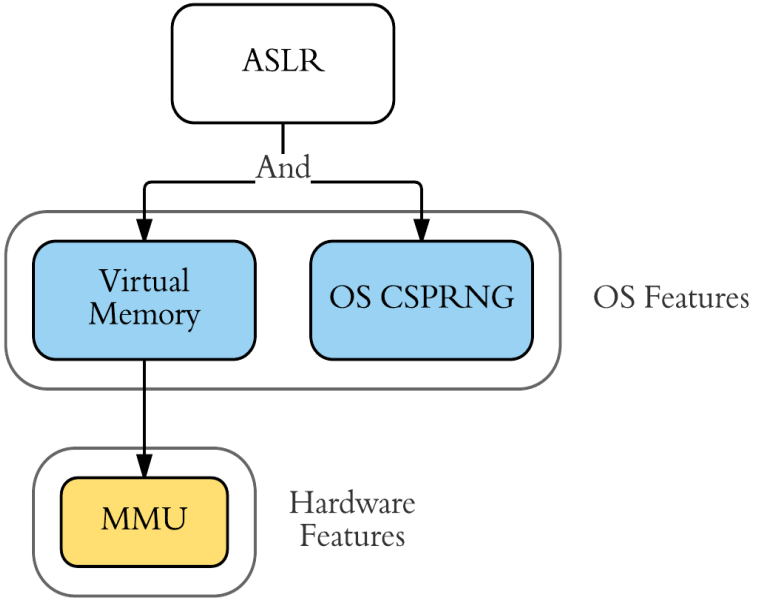

Mitigation Support
ESP
ASLR
Stack Canaries
100
90
80
70
60
50
40
30
20
10
0
All Oses
Non-Mobile
Non-Win/Lin/BSD
Deeply Embedded

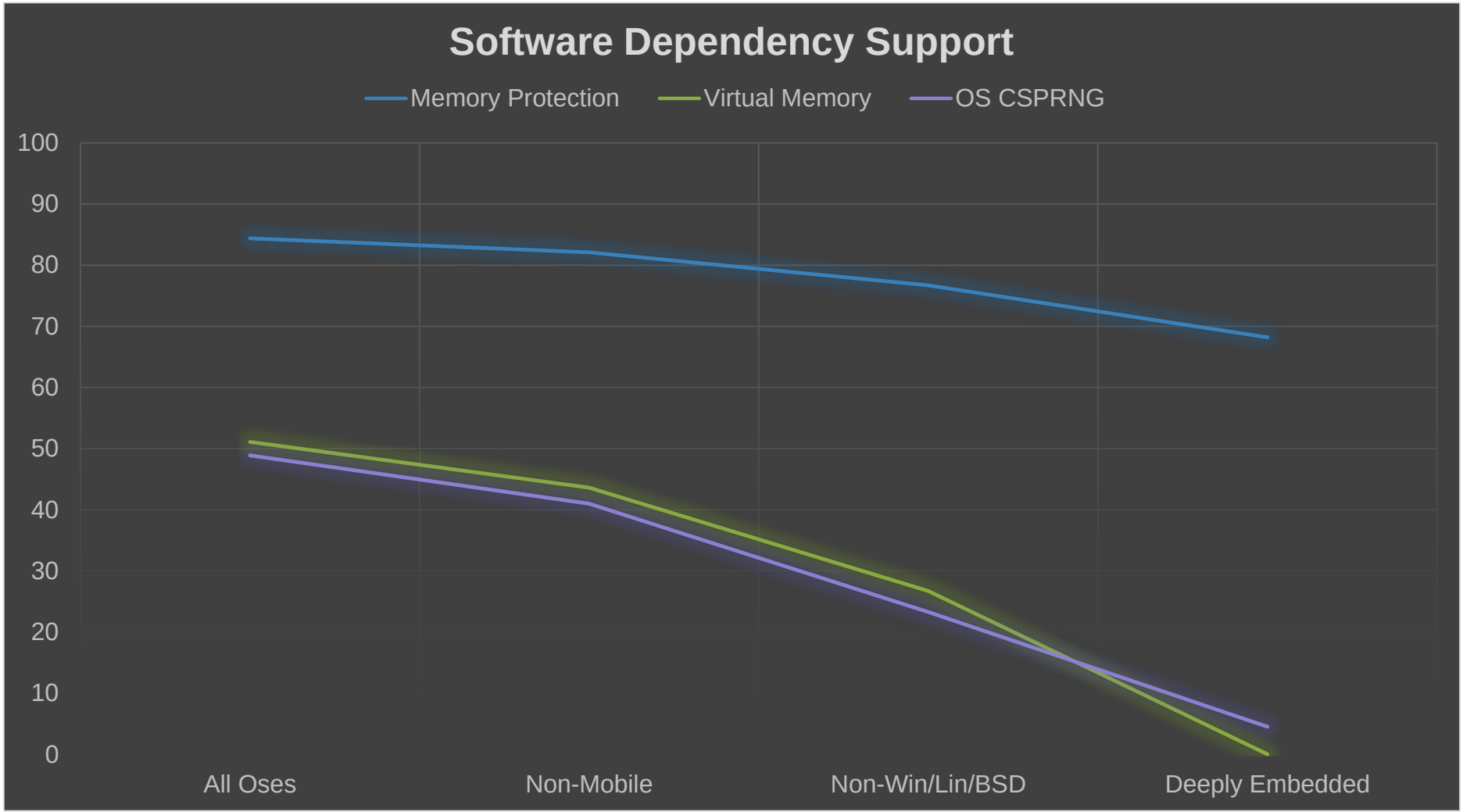
What's Going On Here?



Hardware & Software Dependencies

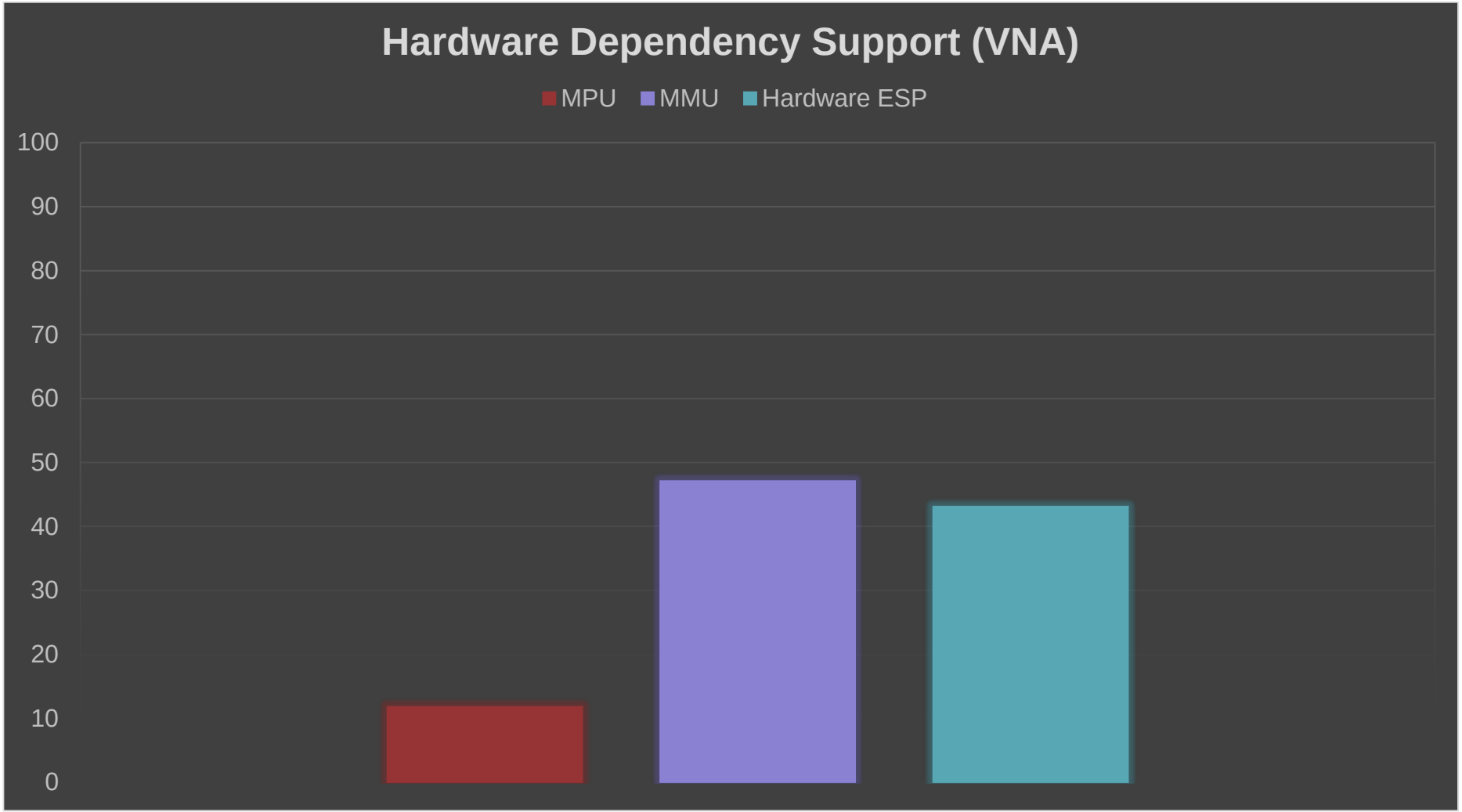





Software Dependency Support
Memory Protection
Virtual Memory
OS CSPRNG
100
90
80
70
60
50
40
30
20
10
0
All Oses
Non-Mobile
Non-Win/Lin/BSD
Deeply Embedded

Hardware Selection

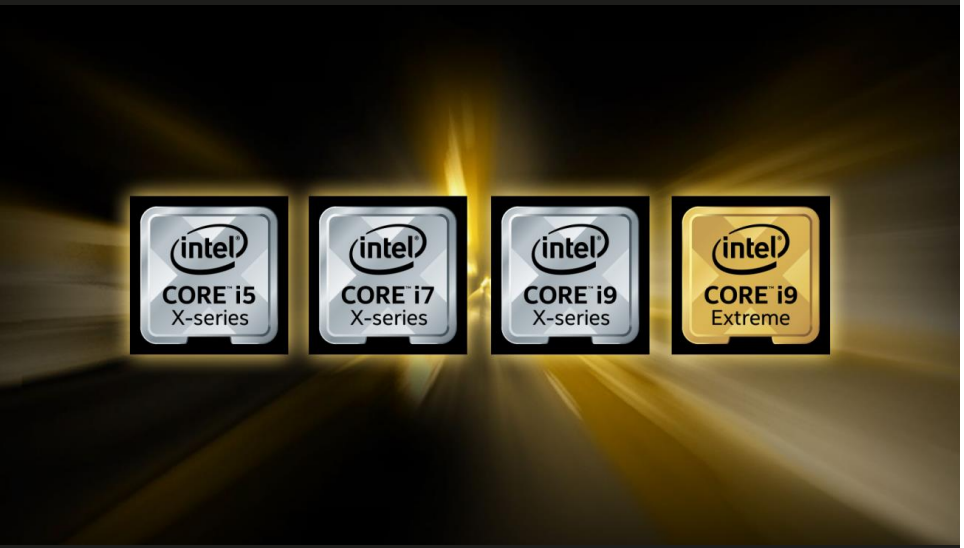
- Selected 78 Popular Embedded '*Core Families*'
- Evaluated for Hardware Dependency Support


Hardware Dependency Support (VNA)
MPU
MMU
Hardware ESP
100
90
80
70
60
50
40
30
20
10
0


Embedded Mitigation Challenges

Resource & System Constraints

- **Limited Resources**
  Overhead influences adoption likelihood
- **Availability Requirements**
  Guarantee high system uptime
- **Real-Time Conflicts**
  Issues with eg. virtual memory caching, page faults, etc.



Advanced Processor Features

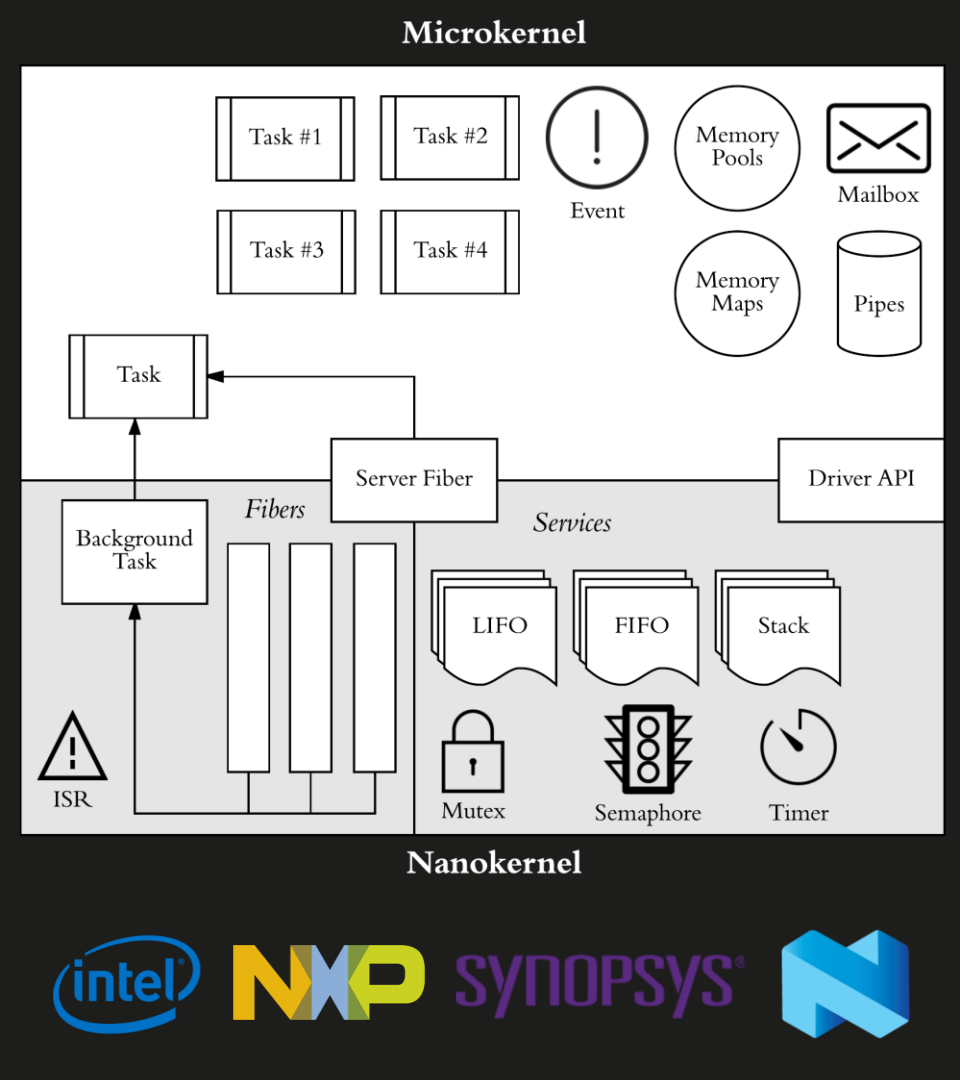
- **Modern mitigations increasingly use advanced processor features**
  eg. SGX, MPX, CET, LBR, PMU
- **Only for modern high-end CPUs**
- **Embedded HW diversity -> OS cannot assume support**

Embedded Random Number Generators

* *Wheel of Fortune – Jos Wetzels, 33C3, 2016*

- **Secure Randomness is OS Service!**
- **Hardware Diversity -> Little Entropy Source Omnipresence**
- **Low Entropy Environment**
  - Little / predictable activity
  - "*Boot-time Entropy Hole*"
  - Can't wait for entropy too long..
- **<u>Result</u>: not-so-random NGs, bad workarounds***

RTOS Mitigation Examples



Zephyr™

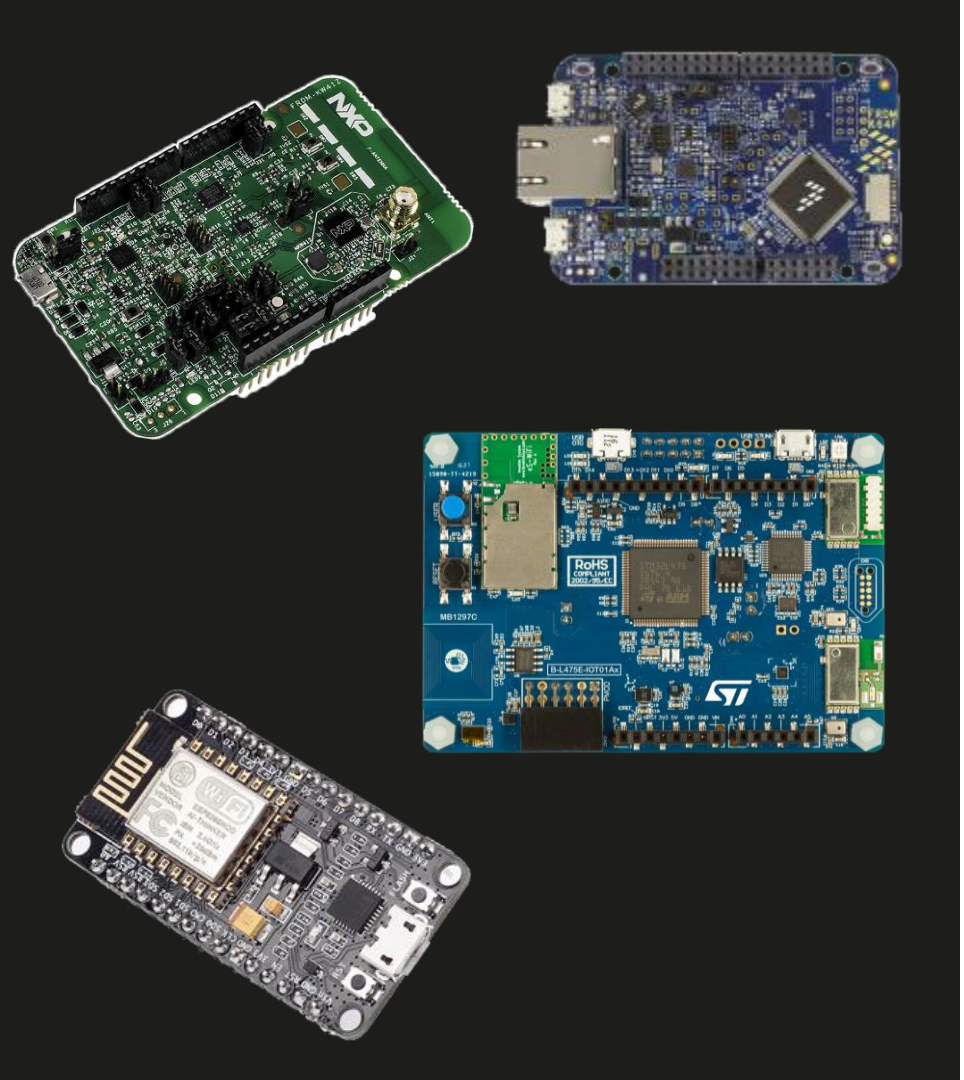
- **Library based RTOS**
  Based on Wind River Rocket
- **OS Linux Foundation Project**
  Aimed at resource-constrained IoT
- **Young (2016) but promising**
  Input from major chipmakers

Explicit focus on security

```
349 FUNC_NORETURN void _Cstart(void)
350 {
351 #ifdef CONFIG_ARCH_HAS_CUSTOM_SWAP_TO_MAIN
352         struct k_thread *dummy_thread = NULL;
353 #else
354         struct k_thread dummy_thread_memory;
355         struct k_thread *dummy_thread = &dummy_thread_memory;
356 #endif
357
358         /*
359          * Initialize kernel data structures. This step includes
360          * initializing the interrupt subsystem, which must be performed
361          * before the hardware initialization phase.
362          */
363
364         prepare_multithreading(dummy_thread);
365
366         /* perform basic hardware initialization */
367         _sys_device_do_config_level(_SYS_INIT_LEVEL_PRE_KERNEL_1);
368         _sys_device_do_config_level(_SYS_INIT_LEVEL_PRE_KERNEL_2);
369
370         /* initialize stack canaries */
371 #ifdef CONFIG_STACK_CANARIES
372         __stack_chk_guard = (void *)sys_rand32_get();
373 #endif
374
375         /* display boot banner */
376
377         switch_to_main_thread();
378
379         /*
380          * Compiler can't tell that the above routines won't return and issues
381          * a warning unless we explicitly tell it that control never gets this
382          * far.
383          */
384
385         CODE_UNREACHABLE;
386 }
```

Zephyr
Stack Canaries

- **Based on Clang/GCC SSP**
- **Canary Failure -> System Error**
- **One master canary for entire address space**
  Generated once at system boot
  No refreshing
- **Generated using RNG API**
  Implementation depends on chosen *random* driver

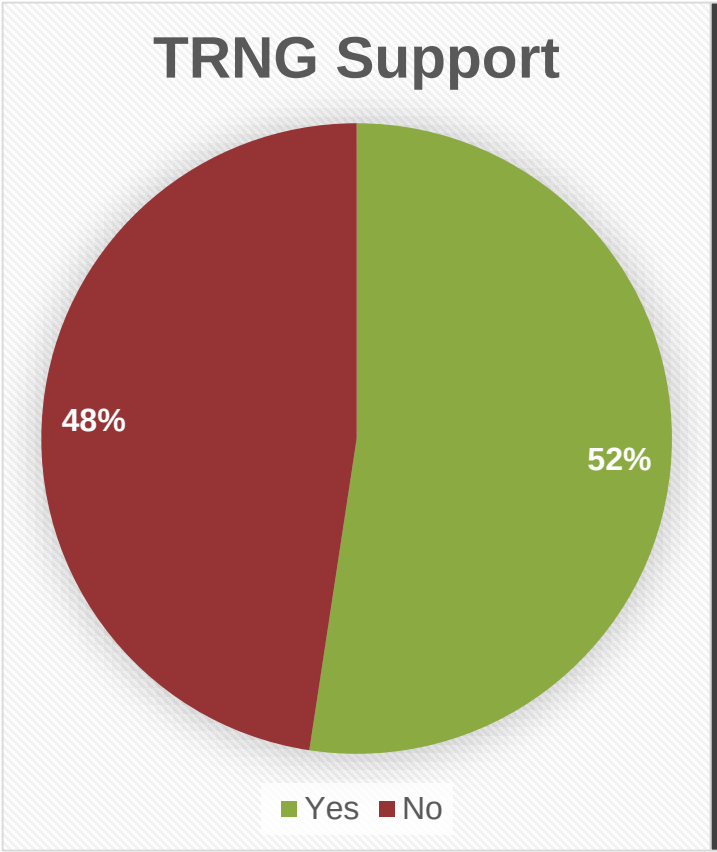


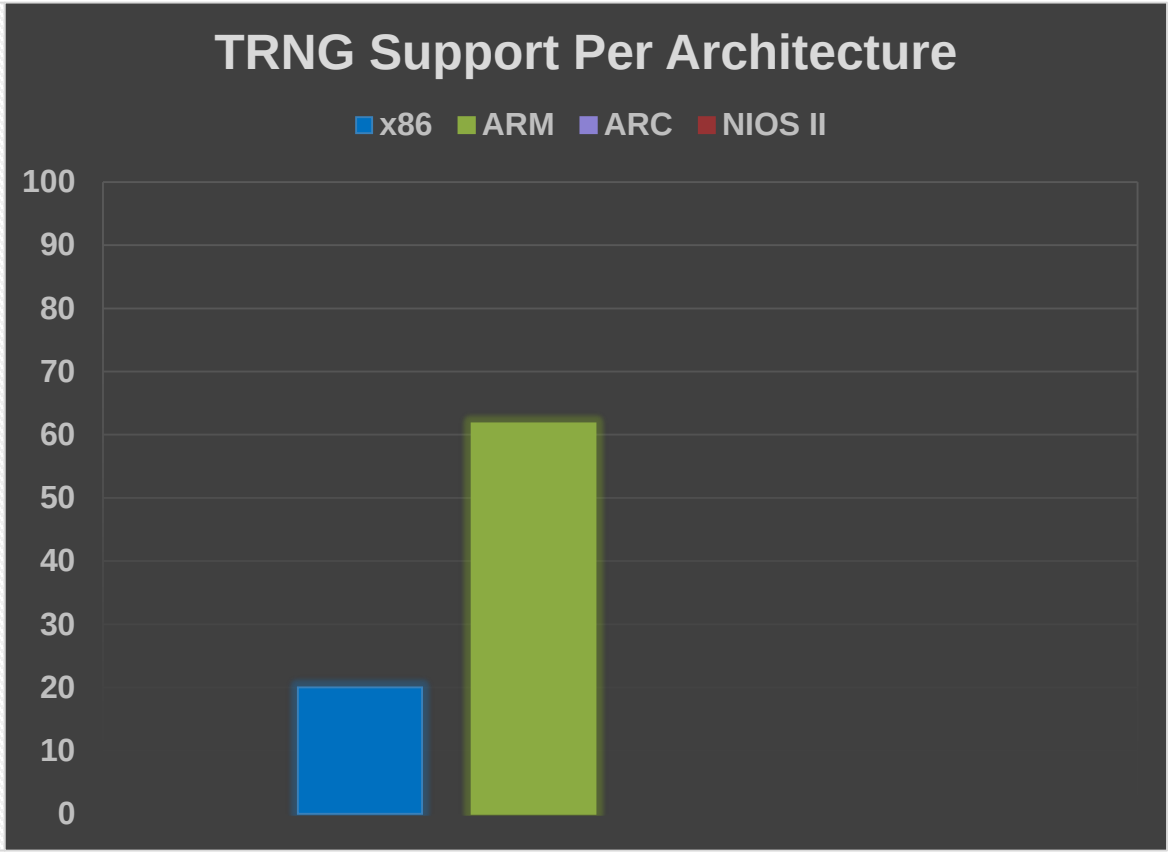
Zephyr Random API

RANDOM_HAS_DRIVER (TRNG)

- **RANDOM_MCUX_RNGA**
  NXP Kinetis K64F
- **RANDOM_MCUX_TRNG**
  NXP Kinetis KW40Z & KW41Z
- **RANDOM_STM32_RNG**
  STM32 Boards
- **RANDOM_ESP32_RNG**
  ESP32 Boards
  (requires Wi-Fi & Bluetooth enabled)

TRNGs Among Zephyr 1.8 Supported Boards





Zephyr Random API
TEST_RANDOM_DRIVER (PRNG)

- **X86_TSC_RANDOM_GENERATOR**
  Uses x86 RDTSC

  Canary 1st value drawn from PRNG
  little difference between bootruns

  Infoleaks everywhere

- **TIMER_RANDOM_GENERATOR**
  Suffers from similar issues

- **Had contact with Zephyr team, plans to integrate OS CSPRNG**
  Most likely NIST SP800-90A based

Fuchsia OS

- **Under-development Google RTOS**
  Pushed to github w/o official statement in August 2016
- **Seems intended for IoT to smartphones**
- **Based on Magenta microkernel**
  Derived from LK (Little Kernel)
  Virtual Memory / MMU support

  Kernel- / Userspace Separation

```
} regions[] = {
    {
        .name = "kernel_code",
        .base = (vaddr_t)&__code_start,
        .size = ROUNDUP((size_t)&__code_end - (size_t)&__code_start, PAGE_SIZE),
        .arch_mmu_flags = ARCH_MMU_FLAG_PERM_READ | ARCH_MMU_FLAG_PERM_EXECUTE,
    },
    {
        .name = "kernel_rodata",
        .base = (vaddr_t)&__rodata_start,
        .size = ROUNDUP((size_t)&__rodata_end - (size_t)&__rodata_start, PAGE_SIZE),
        .arch_mmu_flags = ARCH_MMU_FLAG_PERM_READ,
    },
    {
        .name = "kernel_data",
        .base = (vaddr_t)&__data_start,
        .size = ROUNDUP((size_t)&__data_end - (size_t)&__data_start, PAGE_SIZE),
        .arch_mmu_flags = ARCH_MMU_FLAG_PERM_READ | ARCH_MMU_FLAG_PERM_WRITE,
    },
```

```
590  void VmAspace::InitializeAslr() {
591      aslr_enabled_ = is_user() && !cmdline_get_bool("aslr.disable", false);
592
593      crypto::GlobalPRNG::GetInstance()->Draw(aslr_seed_, sizeof(aslr_seed_));
594      aslr_prng_.AddEntropy(aslr_seed_, sizeof(aslr_seed_));
595  }
```

```
52  uintptr_t choose_stack_guard(void) {
53      uintptr_t guard;
54      if (hw_rng_get_entropy(&guard, sizeof(guard), true) != sizeof(guard)) {
55          // We can't get a random value, so use a randomish value.
56          guard = 0xdeadbeef00ff00ffUL ^ (uintptr_t)&guard;
57      }
58      return guard;
```

Fuchsia OS

Exploit Mitigations

- **ESP / DEP**
  x86 MMU-based DEP policies

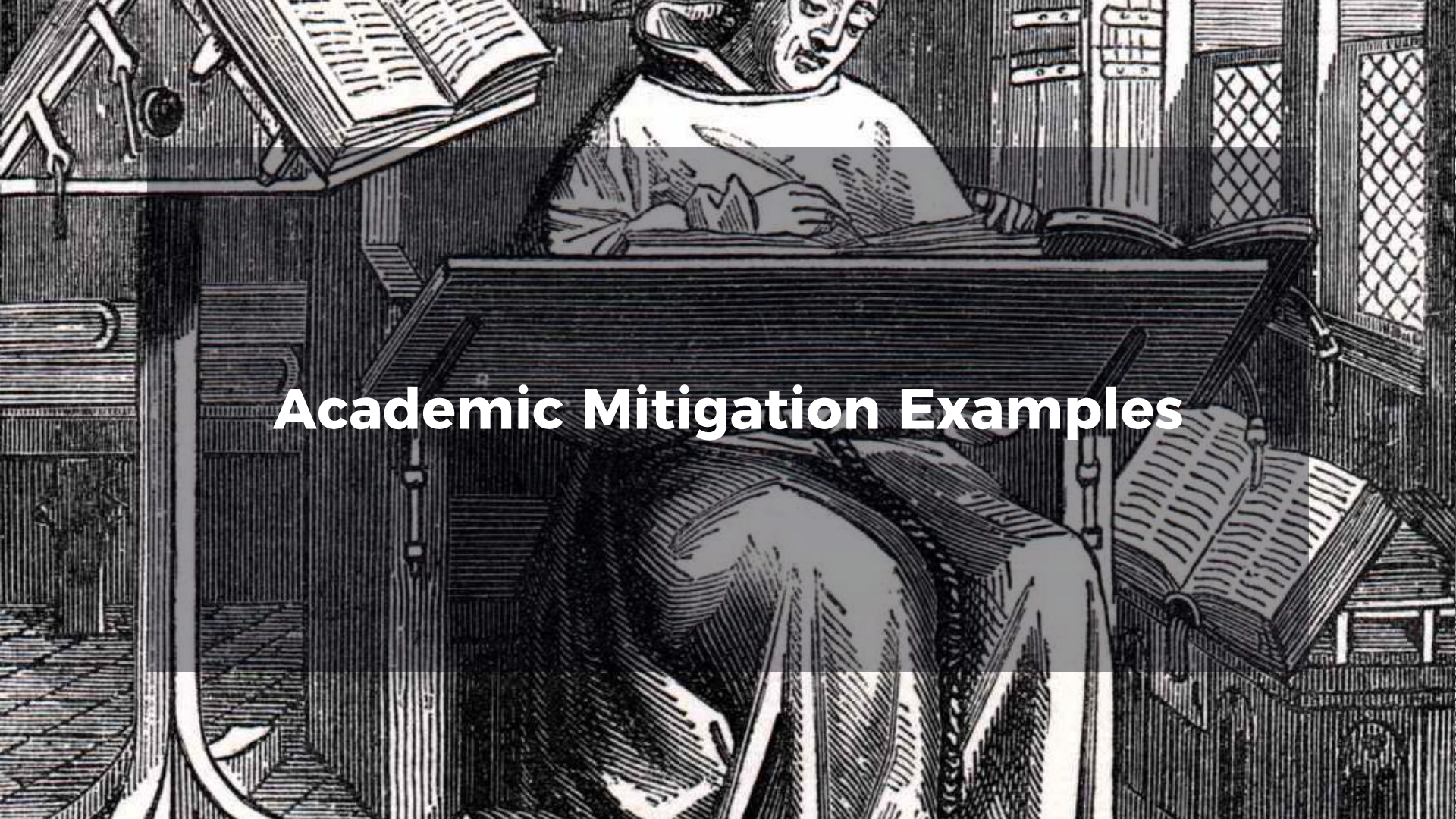
- **ASLR**
  All executables PIE by default

  Randomization draws on OS CSPRNG

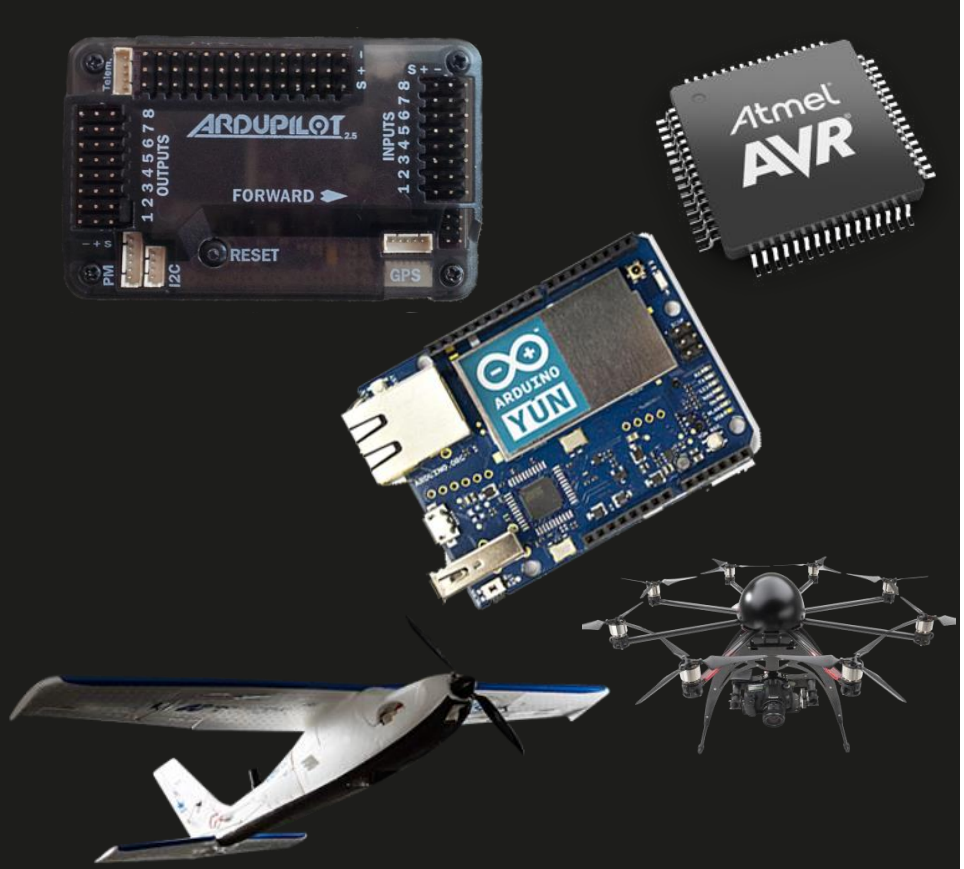
- **Stack Canaries (GCC SSP Model)**
  Single early-boot global canary

  Canary generated using HW RNG API
  Static value fallback

  Canary failure -> kernel panic



Academic Mitigation Examples



** MAVR: Code Reuse Stealthy Attacks and Mitigation on Unmanned Aerial Vehicles – Habibi et al. (2015)*

*** AVRAND: A Software-Based Defense Against Code Reuse Attacks for AVR Embedded Devices – Pastrana et al. (2016)*

MAVR* & AVRAND**

- **Boot-Time Code Diversification**
- **MAVR: Ardupilot UAV system**
- **AVRAND: AVR chip in Arduino Yun**
- **MAVR requires hardware mod**
  Additional master processor + flash chip
- **(Re)Randomize by re-flashing**
  Every N restarts or after crash
- **Attacker can reduce device lifespan by forcing re-flash to exhaust program / erase cycles**

Deeply Embedded Exploit Mitigation Criteria

Embedded Exploit Mitigation Criteria

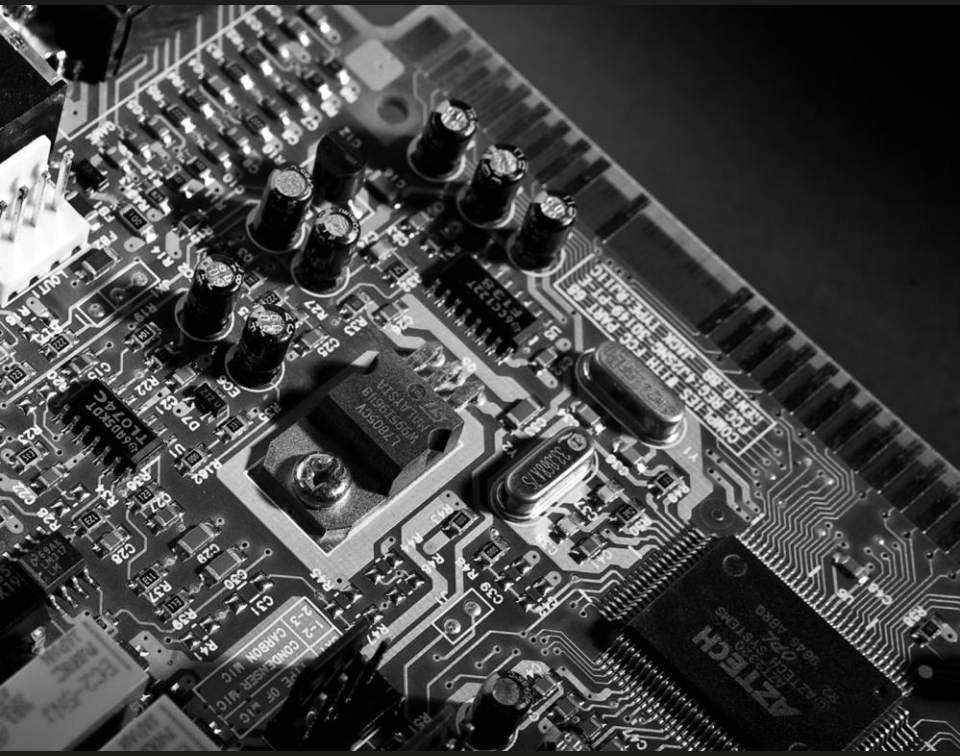
- **Low Resource Pressure**
  Max 5% Overhead*
- **Hardware Agnostic**
- **Availability Preservation**
- **‘Real-Time Friendly’**
- **Easy RTOS Integration**

** Microsoft BlueHat Prize Rules (2012),*
‘SoK: Eternal War in Memory’ – Szekeres et al. (2013)

Embedded OS CSPRNG Criteria

- **Low Resource Pressure**
  Use lightweight crypto
- **Entropy Gathering Limitations**
  Rapidly available upon boot
  No constant runtime polling
- **Non-Domain Specific Entropy Sources**
  No reliance on sensor values, accelerometers, etc.


µArmor



µArmor

- **Deeply Embedded Mitigation Baseline**
  µESP + µScramble + µSSP + µRNG
- **Target Systems**
  Harvard or VNA CPU w. HW ESP
  Single-address space (RT)OS or bare metal
- **Attacker Model**
  Remote (eg. WiFi, Bluetooth, RF) control-flow hijacking attacks

Representative Platform

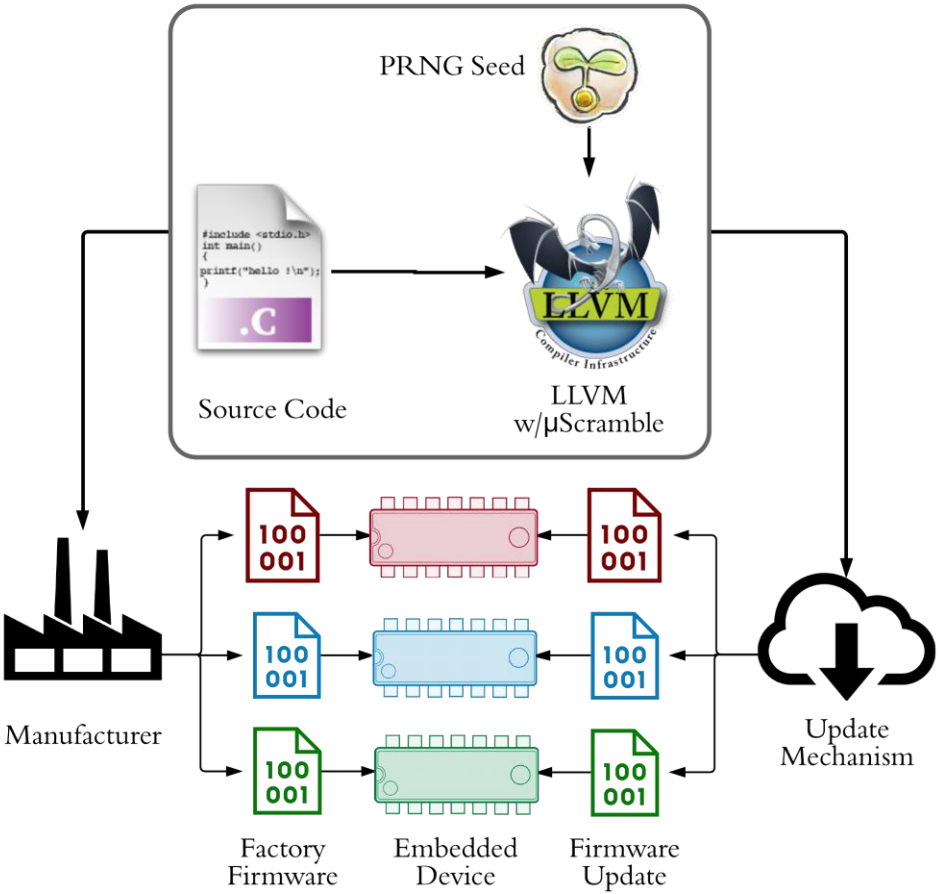
Memory	Permissions
Code (sensitive)	RO+XN
Code (other)	RO+X
Data	RW+XN
Peripherals	RW+XN
SCB Config	RO+XN
MPU Config	RO+XN

Table 14: µESP Memory Permission Policies

µESP

- **Use MPU to Enforce ESP**
- **Stack Grows Away From Data**
- **Limit Code-Reuse Surface**
  Prevent ret2bootloader style attacks
- **'Lock Down' MPU & SCB**
  Prevent MPU disabling

µScramble



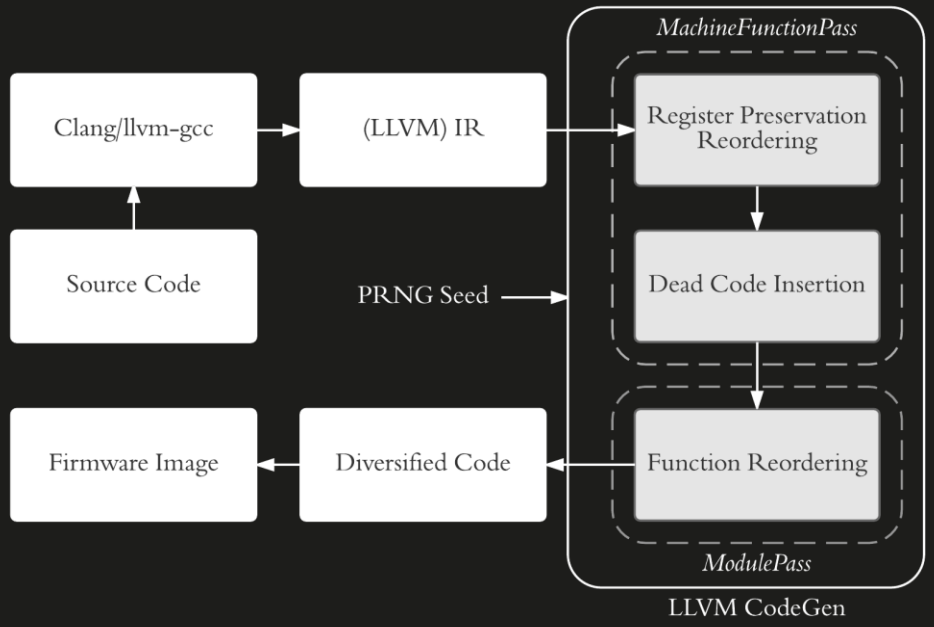
- **Need Alternative for ASLR**

- **Software Diversification**
  When: Compilation, Boot, Runtime, Etc.
  What: Code, Data

- **Hybrid Compile/Update Solution**
  Integrated in firmware generation

  Per-device diff firmware image

- **Benefits**
  Limited Overhead
  No virtual memory required
  Works on source: platform independent



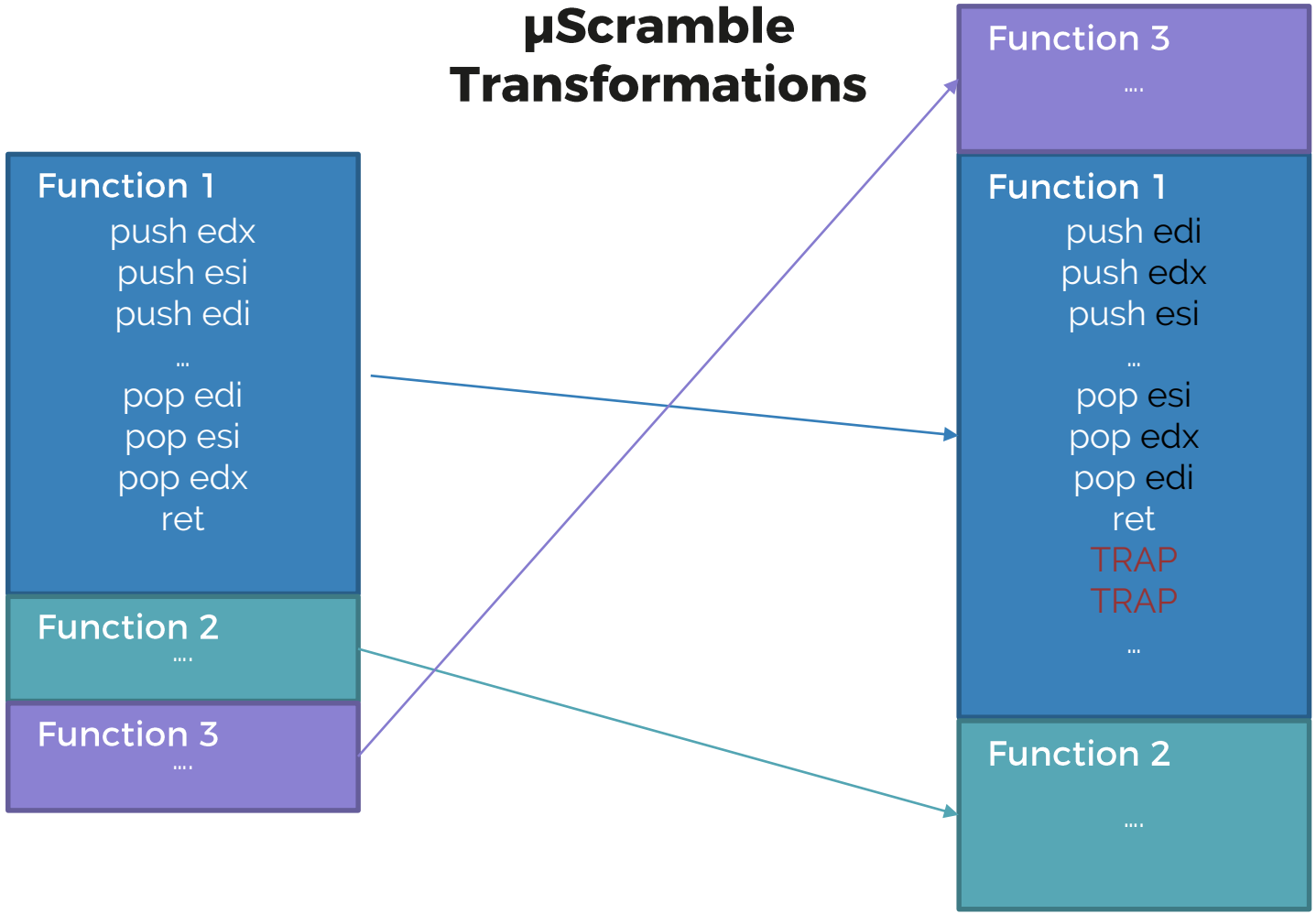
µScramble
Code Diversification

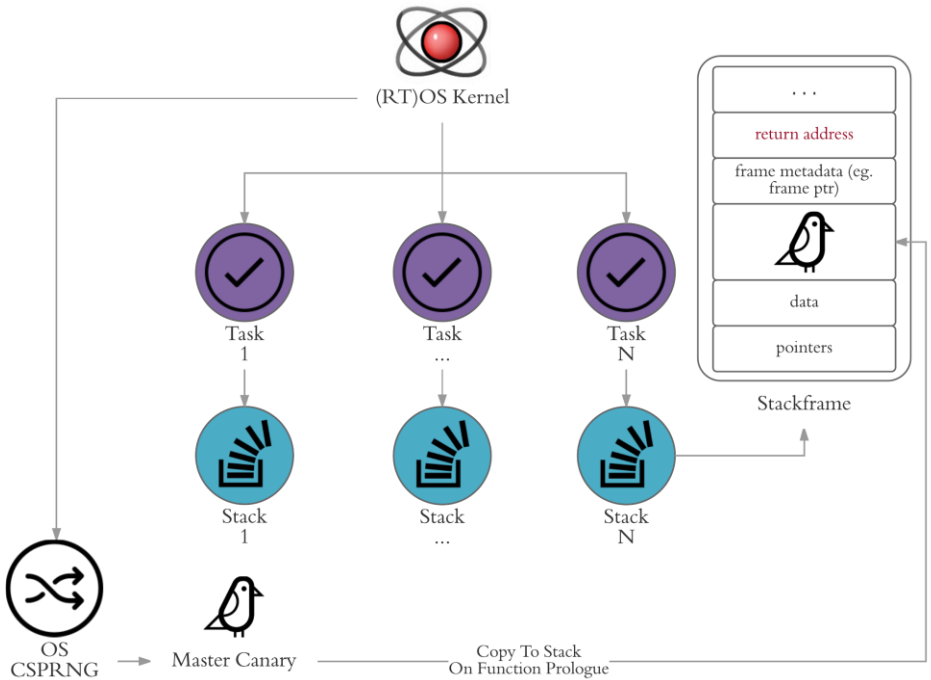
- **Code Transformations**
  Function Reordering
  Dead Code Insertion
  Register Preservation Reordering

- **Fine-Grained Randomization**
  Wrt. image base
  Wrt. function base
  Wrt. inter-function distance
  Order of register preservation code

- **Decorrelation reduces infoleak & bruteforce impact**

µScramble Transformations





µSSP

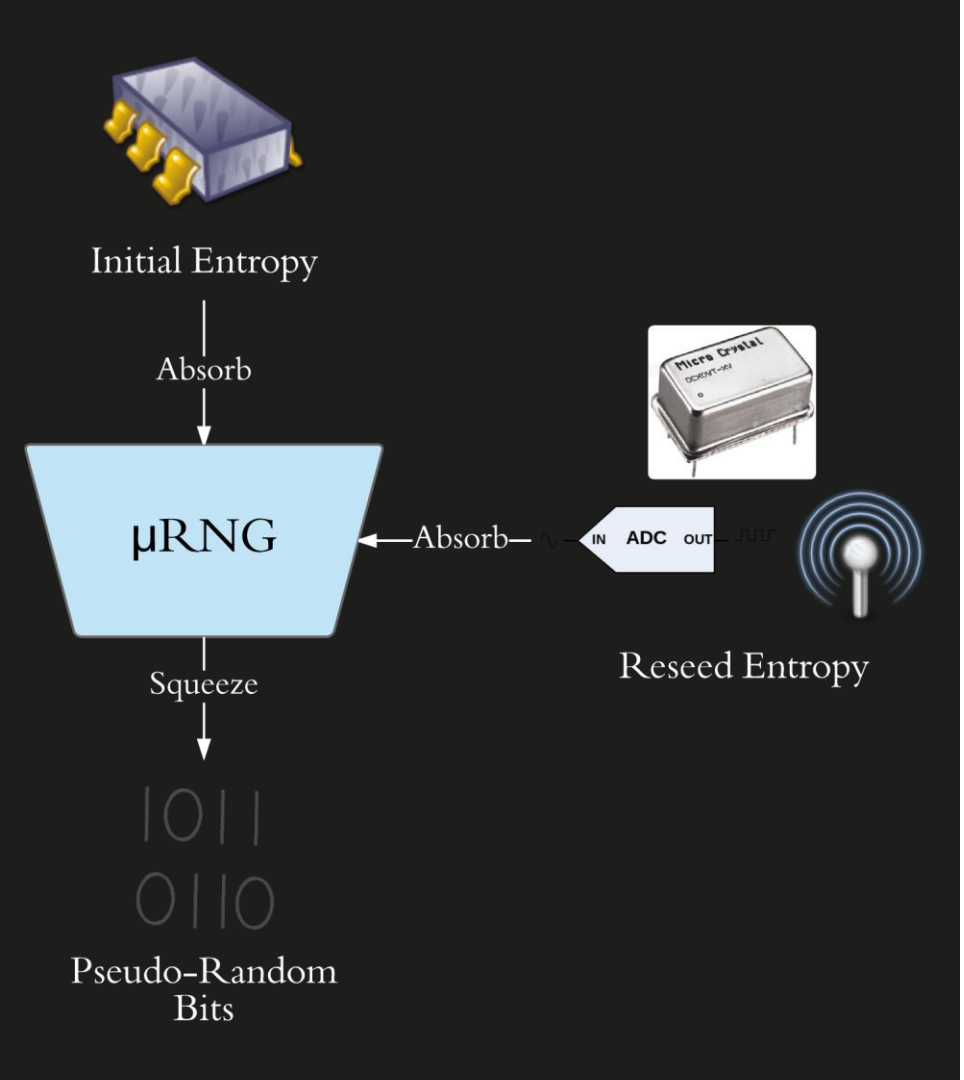
- **Based on Zephyr SSP**
- **Master canary drawn from OS CSPRNG**
- **Modular violation handler**

  Passive

  Fatal

  Thread Terminate / Restart

  System Terminate / Restart



µRNG

- **Keccak-based OS CSPRNG**
  128-bit security

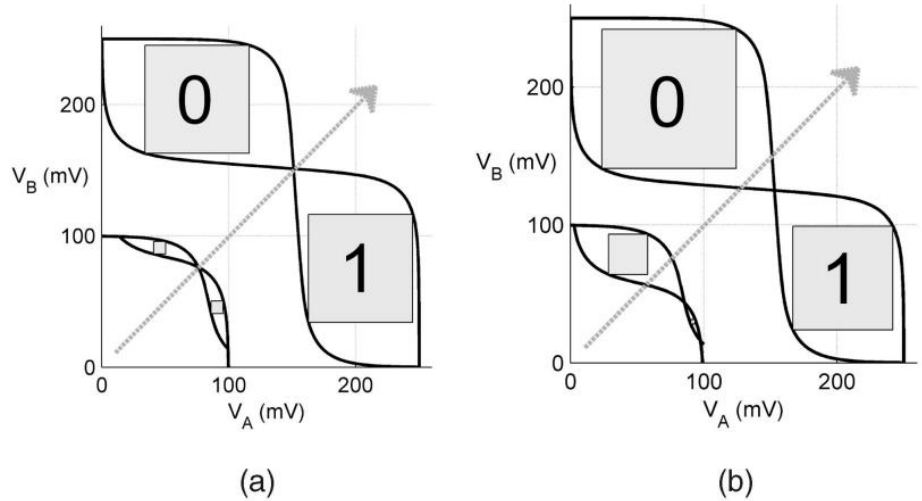
  25 byte internal state

  1 algorithm for entropy gather & RNG

  Integrated as Zephyr random driver

- **Avoid Reseed Polling**
  Reseed Control linked to PRNG output

µRNG
Initial Entropy Source



- **SRAM Startup Values (SUVs)**
  Two stable states
  Converge after initial instability
- **Manufacturing Differences**
  Biased cells -> PUFs
  Random cells -> PRNGs
- **Many Benefits**
  Instantly available at very early boot

  Differs between devices & bootruns

  SRAM omnipresent on MCUs

** Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers – Holcomb et al. (2009)*

µArmor
Overhead & Security Evaluation

µArmor
Overhead Evaluation

RNG	ROM	RAM
µRNG [1]	709	81
TinyRNG [302][2]	11086	471
TinyKey [428][3]	5990	790

- **Metrics**
  Code Size, Data Size, Memory Usage, Runtime Overhead
- **Evaluated Against**
  IoT sample applications

  TACLeBench suite
- **Overhead**
  Code Size ≤ 5%

  Others ≤ 1%

Set	Avg. GS[1]	Max. GS[1]
Sequential		
adpcm_dec	0.5	4
adpcm_enc	0.6	5
ammunition	3.6	147
anagram	0.6	4
audiobeam	4	79
cjpeg_transupp	32	96
cjpeg_wrbmp	1.4	3
dijkstra	1.3	5
epic	0	0
fmref	0.5	3
gsm_dec	1.3	43

µScramble Security Evaluation

- **Coverage Analysis**
  1000 firmware variants per benchmark

  Harvest ROP gadgets

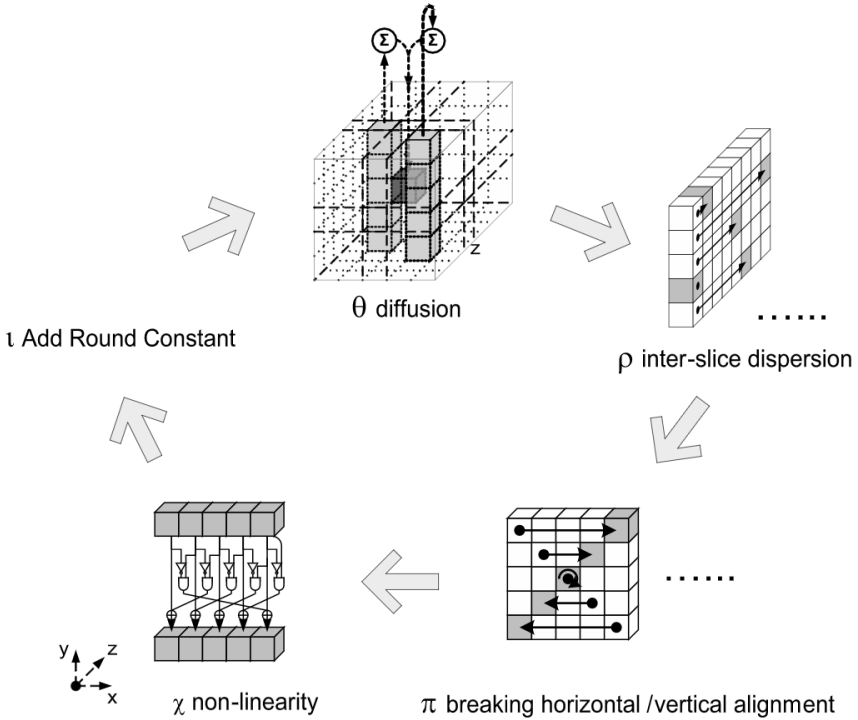
  Max avg. gadget survival rate: 10.15%

- **Complicating Attacker Factors**
  Payload length

  Firmware complexity

  Fine-Grained Randomization

μRNG
Security Evaluation



- **Keccak is well-analyzed**
- **Passive State Recovery Attacks**
  Reseed at least every 1GB of output
  well within 32GB bound*
- **No '*Boot Time Entropy Hole*'**
  All initial entropy instantly available

* *Sponge-based pseudo-random number generators - Guido Bertoni et al. (2010)*

µRNG
Security Evaluation

Table 1. Min-entropy results for STM32F100R8 SRAMs at different temperatures. Min-entropy denoted as percentage of total available SRAM.

Temp.	Microcontroller ID									
[°C]	1	2	3	4	5	6	7	8	9	10
−30	5.3%	5.3%	5.4%	5.5%	5.4%	5.3%	5.4%	5.2%	5.3%	5.8%
+25	6.6%	6.6%	6.7%	6.8%	6.7%	6.5%	6.8%	6.5%	6.7%	6.7%
+90	6.3%	6.5%	6.5%	6.6%	6.2%	6.5%	6.5%	6.2%	6.5%	6.5%

- **SRAM Suitability Depends on Chip**
  Need Entropy ≥ 2 * security strength

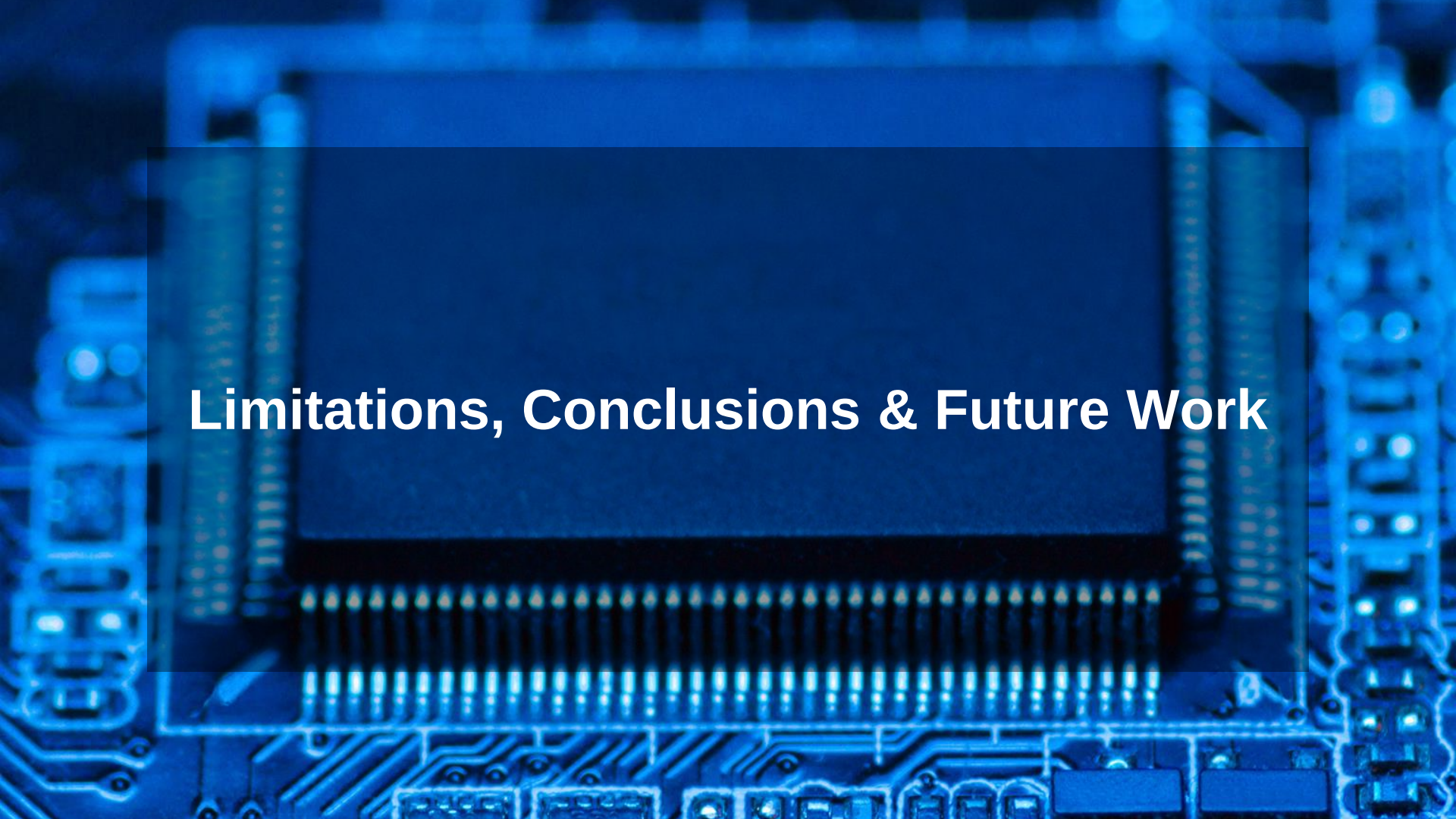
  Avg. Entropy: 5% of SRAM size*
  Worst-case: 1.2% (100 bits) for PIC16F

- **Reality**
  SRAM slightly contaminated
  Harvest in v. early boot & limit stack use

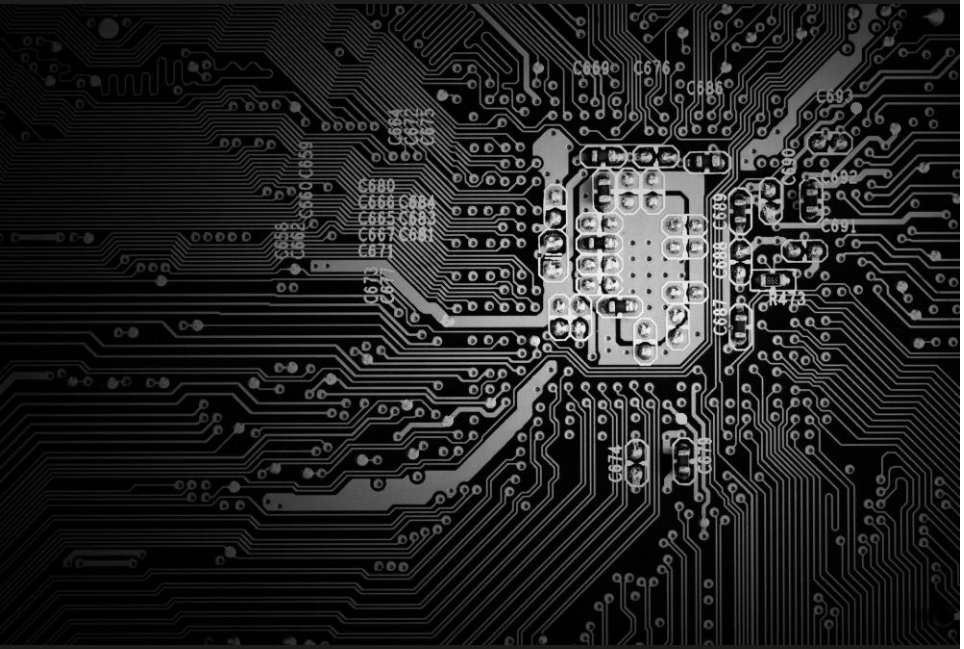
- **Memory Retention**
  Minimal at normal ambient temps
  Clear loss in cold (-20C) for some chips

  Full SRAM reset between power cycles

** Secure PRNG Seeding on Commercial Off-the-Shelf Microcontrollers - Van Herrewege et al. (2013)*

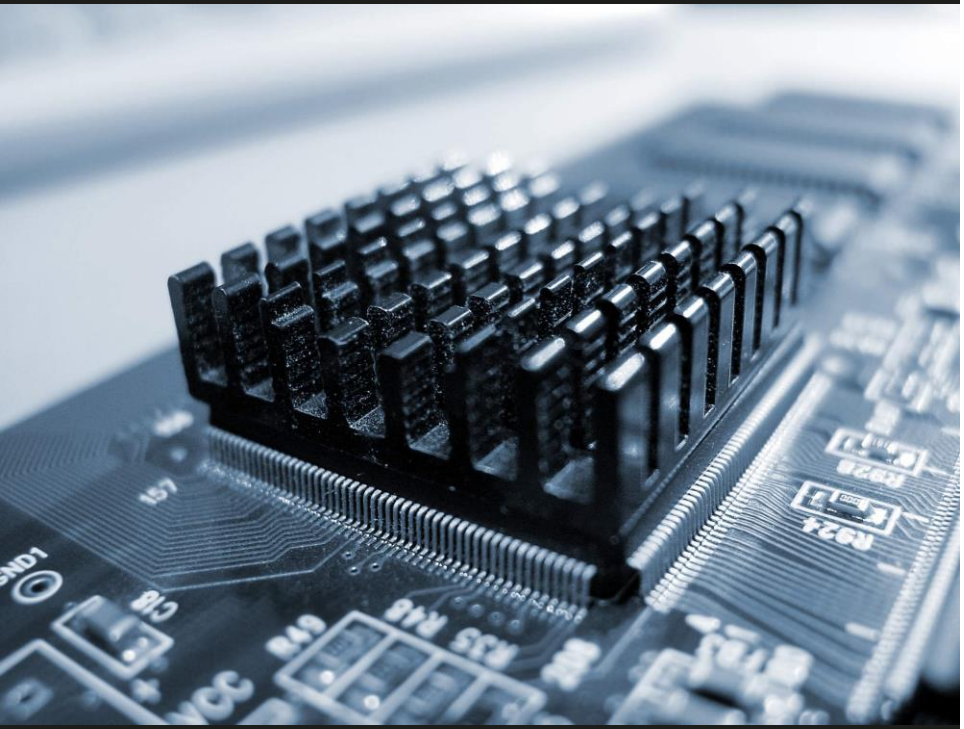


Limitations, Conclusions & Future Work



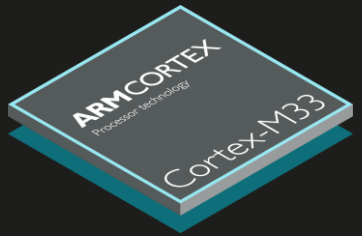
µArmor Limitations

- **Platform**
  Harvard CPU or VNA with hardware ESP
- **Attacker Model**
  Remote Control-Flow Hijacking Only
- **Diversification Relocation Frequency**
  Compile- / Update-Time Only
- **Diversification Coverage**
  Code Memory only



Conclusions

- **Embedded memory Corruption issues warrant serious attention**
  Smallest of devices are connected
  Unsafe languages are here to stay
- **Current state of embedded mitigations is terrible**
  Esp. in deeply embedded / RTOS
  Not easily fixed either
- **µArmor is intended as small step forward**



Future Work

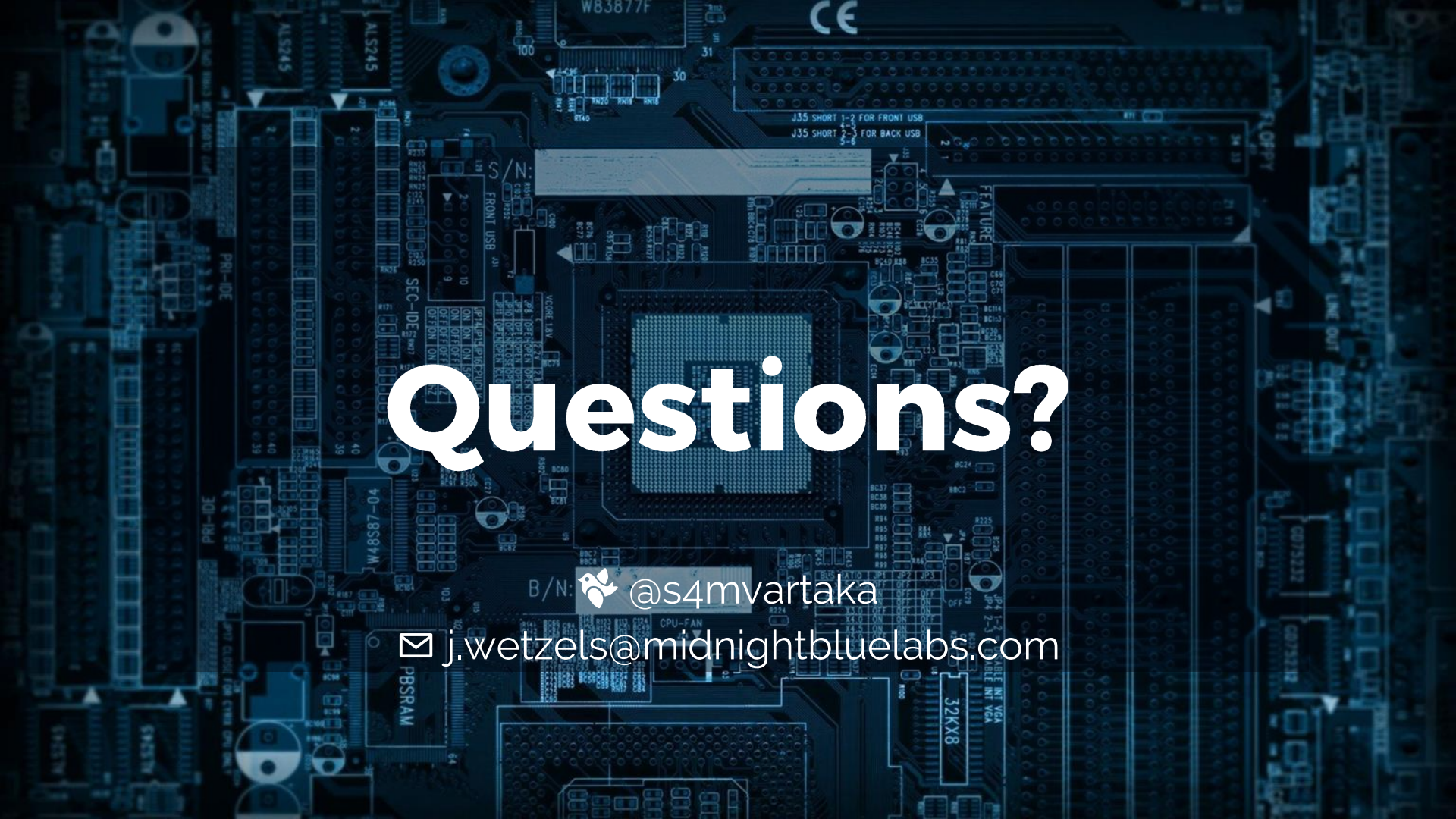
- **Long Term**
  Embedded Safe Language Adoption (Rust Embedded, Tock OS, etc.)

  Secure & Scalable Patching

- **Short Term**
  Mitigation Baseline Adoption in RTOS

  Hardware Dependency Support

  Advanced Embedded Mitigations (CFI, XoM, etc.)


Questions?
@s4mvartaka
j.wetzels@midnightbluelabs.com