SAND2019-3837PE

PROTECTING BARE-METAL SYSTEMS FROM REMOTE EXPLOITATION

Abraham A. Clements

Final Exam

April 5th 2019

Supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND - XXXX

PURDUE UNIVERSITY.

THESIS STATEMENT

Using static and dynamic analysis, modern micro-controllers can:

- Employ defenses against memory corruption and control-flow hijack attacks that match or exceed those deployed on desktop systems, and
- Utilize state-of-the-art testing techniques to identify and prevent memory corruption errors

OUTLINE

- Background Information
  - Bare-metal Systems
  - Memory Corruption and Control-Flow Hijack Attacks
  - Static and Dynamic Analysis
- Thesis Statement
  - Thesis Impact
- Case Studies to Demonstrate Thesis Statement
  1. EPOXY: Protecting Bare-metal Applications with Privilege Overlays
  2. ACES: Automatic Compartments for Embedded Systems
  3. HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation

BACKGROUND

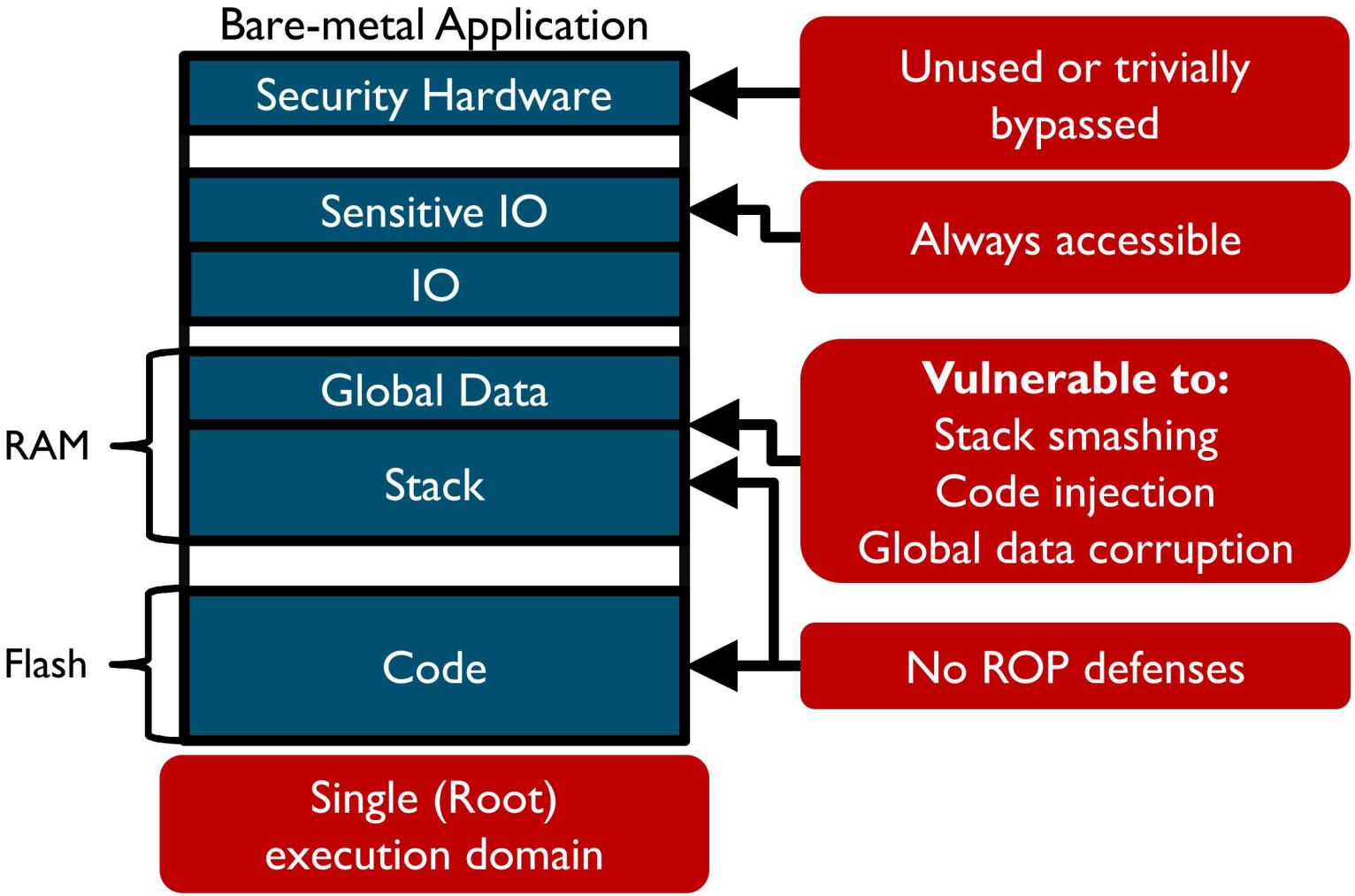
BARE-METAL SYSTEMS

- Systems without an OS
- Constraints
  - Small memory sizes
  - Tight run-time constraints
  - Low power requirements
- Single Application
  - No kernel/user space separation
  - Tightly coupled to hardware
  - Limited execution observability

Examples

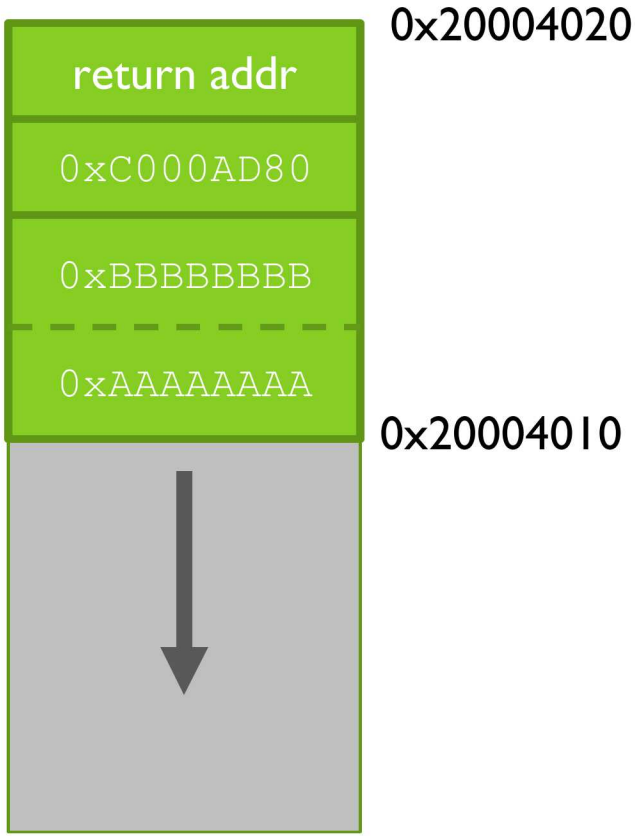
Security left out and testing limited by coupling to hardware

DEFAULT: NO DEFENSES



MEMORY CORRUPTION

```
#define UART_RX 0x40000100

void get_uints(){
  uint32_t  i = 0;
  uint32_t  buf[2];

  buf[i] = *(UART_RX)
  while (buf[i] != 0){
    ++i;
    buf[i] = *(UART_RX)
  }
}
```

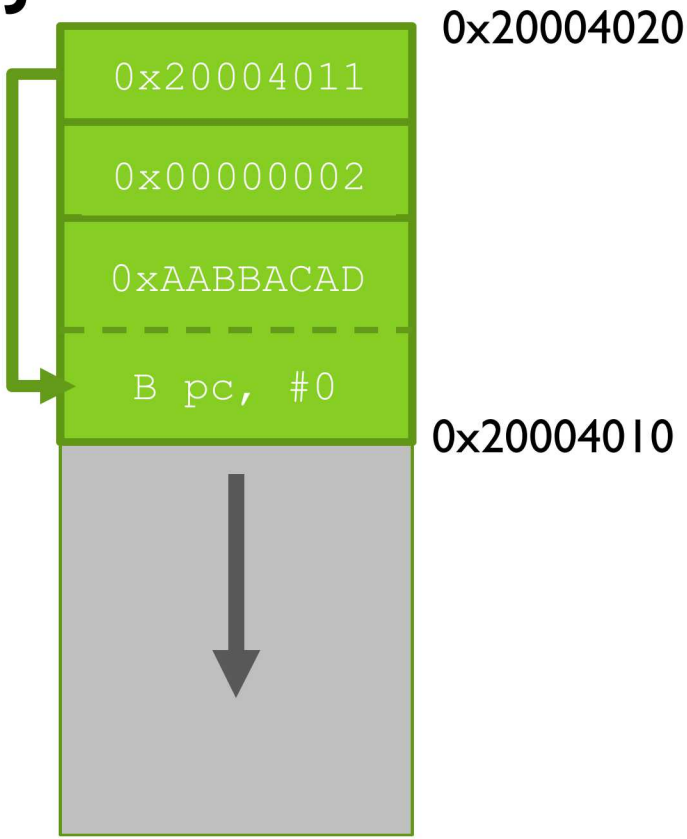


Input: 0xAAAAAAAA: 0xBBBBBBBB: 0xC000AD80: 0xFF000000:

Writes 0xFF000000 to 0xE000ED94
Disabling Memory Protection

CONTROL-FLOW HIJACK

```
#define UART_RX 0x40000100

void get_uints(){
  uint32_t  i = 0;
  uint32_t  buf[2];

  buf[i] = *(UART_RX)
  while (buf[i] != 0){
    ++i;
    buf[i] = *(UART_RX)
  }
}
```




```
Input: 0xAAABACAD: 0xAABBACAD: 0x00000002:
       0x20004011: 0x00000000:
```

STATIC AND DYNAMIC ANALYSIS

- Static Analysis
  - Examining application without executing it
  - Used to determine control and data flow through a program
  - Precise static analysis is intractable due to aliasing
    - Approximations are used
- Dynamic Analysis
  - Examining the execution of the program
  - Is precise – observed control and data flow actually occur
  - Incomplete – have to observe every possible input to be complete

THESIS STATEMENT

Using static and dynamic analysis, modern micro-controllers can:

- Employ defenses against memory corruption and control-flow hijack attacks that match or exceed those deployed on desktop systems, and
- Utilize state-of-the-art testing techniques to identify and prevent memory corruption errors

THESIS IMPACT

- Advance bare-metal security several decades
  - Enable protections beyond those currently used on desktop system
  - Force attacks to be individually tailored per device
- Single vulnerability does not compromise entire system
- Enables state-of-the-art testing of bare-metal systems

THESIS CASE STUDIES

- EPOXY: Protecting Bare-metal Systems with Privilege Overlays
  - Presented at IEEE Security and Privacy 2017
- ACES: Automatic Compartments for Embedded Systems
  - Presented at Usenix Security 2018
- HALucinator:
  - Under review at Usenix Security 2019

EPOXY: PROTECTING BARE-METAL APPLICATIONS WITH PRIVILEGE OVERLAYS

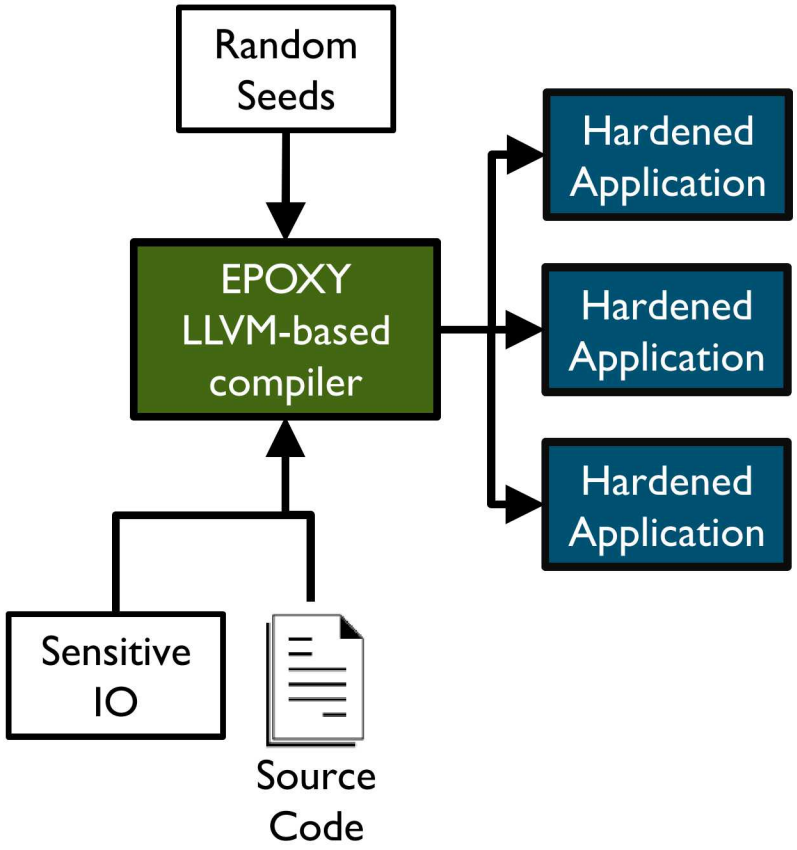
Presented at IEEE Security and Privacy 2017

Authors: Abraham Clements, Naif Almakhdhub, Khaled Saab, Jinkyu Koo, Saurabh Bagchi and Mathias Payer

EPOXY

Embedded Privilege Overlay across X hardware for Y software

- LLVM based compiler
- Protects against
  - Code injection
  - Control flow hijacking
  - Data corruption
  - Direct manipulation of IO
- Privilege Overlays
  - Creates two privilege levels
  - Created using static analysis
  - Foundation for other defenses
- Low overhead
  - 1.8% Runtime
  - 0.5% Energy



THREAT MODEL AND REQUIREMENTS

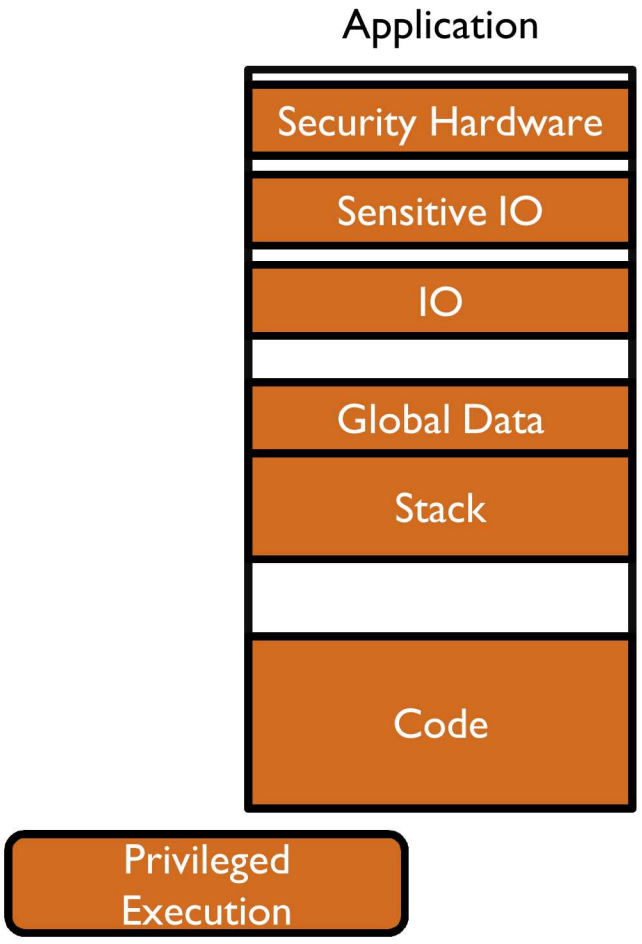
Threat Model

- Arbitrary memory corruption
- Attacker goals:
  - Control-flow hijacking
  - Corrupt specific global data
- Does ***not*** have physical access

Requirements

- Hardware support for two execution privilege modes
- Memory Protection Unit (MPU)
  - Hardware that enforces access permissions on physical memory
- Memory usage determined a priori

BEFORE EPOXY



PRIVILEGE OVERLAY EXAMPLE

Default

```
#define UART_RX 0xdeadbeef
char  menu_option;
...

menu_option = *(UART_RX)

switch (menu_option):
case '1':
  handle_case_1;
  break;
...
```

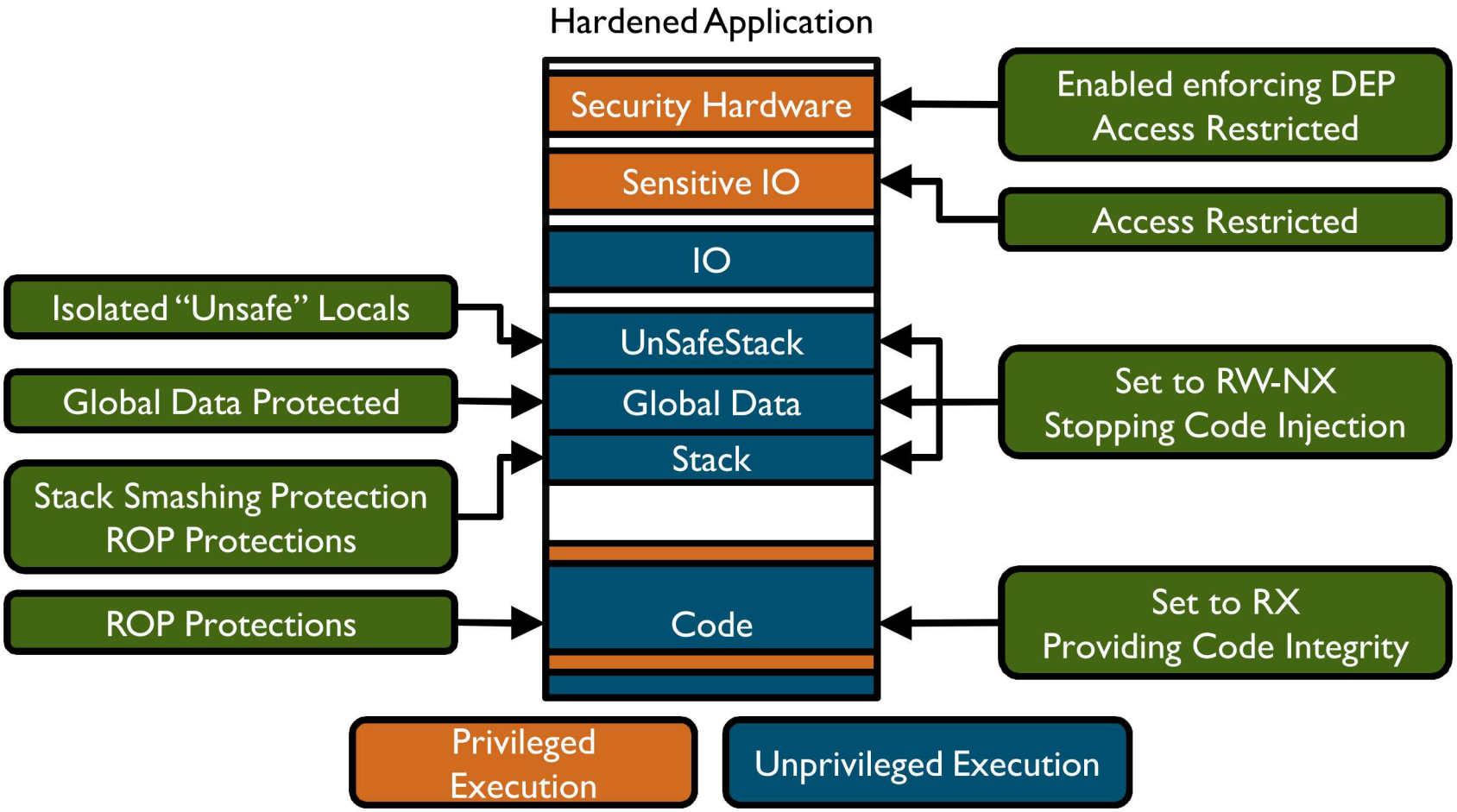
Privileged Execution

Privilege Overlay

```
#define UART_RX 0xdeadbeef
char  menu_option;
...
request privileges;
menu_option = *(UART_RX)
drop privileges;
switch (menu_option):
case '1':
  handle_case_1;
  break;
...
```

Unprivileged Execution

EPOXY – ALL PROTECTIONS



ACES: AUTOMATIC COMPARTMENTS FOR EMBEDDED SYSTEMS

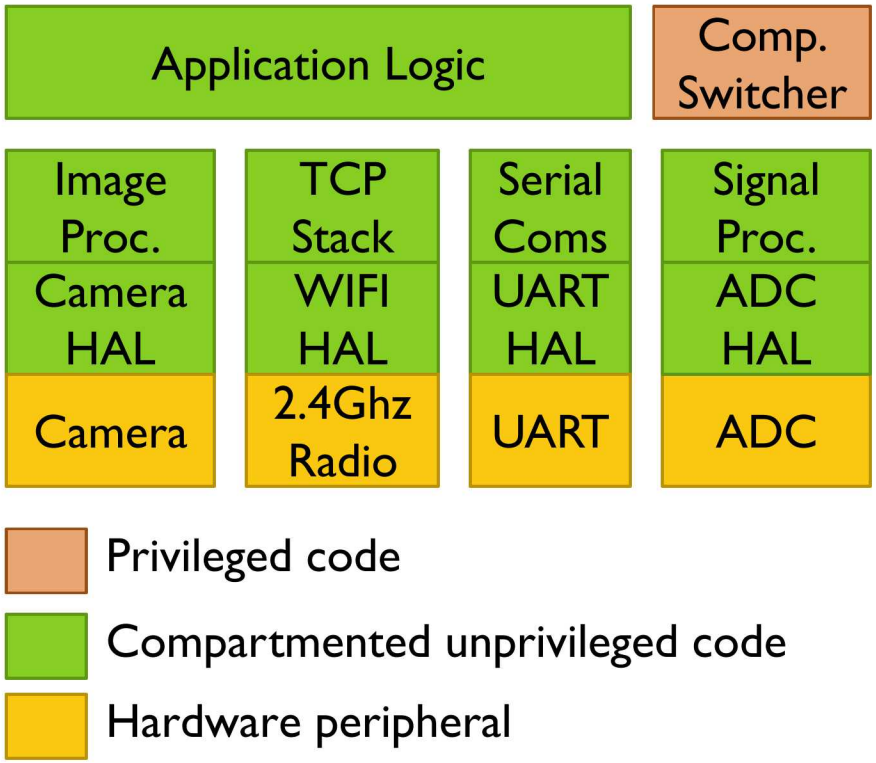
Presented at Usenix Security 2018

Authors: Abraham Clements, Naif Almakhdhub, Saurabh Bagchi, and Mathias Payer

ACES INTRODUCTION

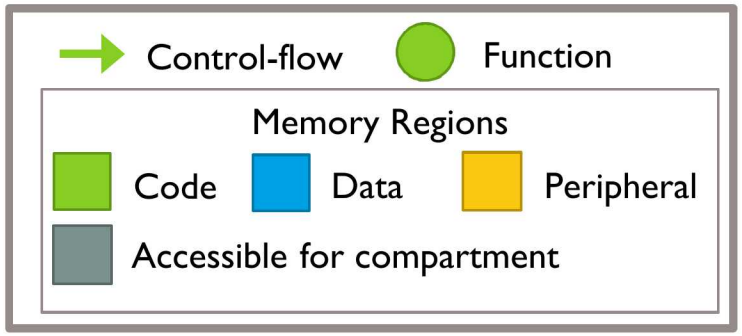
- ACES creates many compartments
  - Applies least privileges
  - Creates sub-thread compartments
  - Protects **integrity** of sensitive data and peripherals
- Uses static analysis to automatically infer compartments using a policy
- Separates compartmentalization from application development
- Each compartment has associated data/peripherals

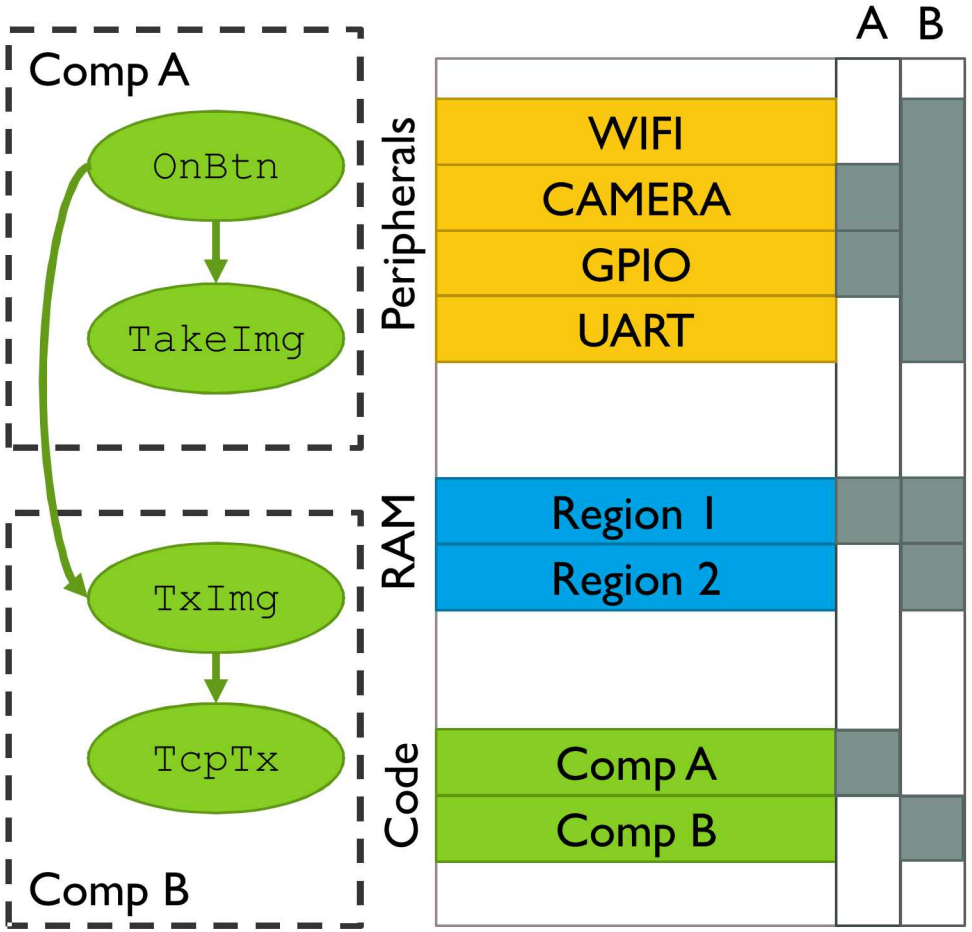
Compartmented Application



COMPARTMENTS

- Set of concurrently accessible memory regions and authorized control-flows between them
- Compartments restrict
  - Accessible memory
  - Control-flow between compartments



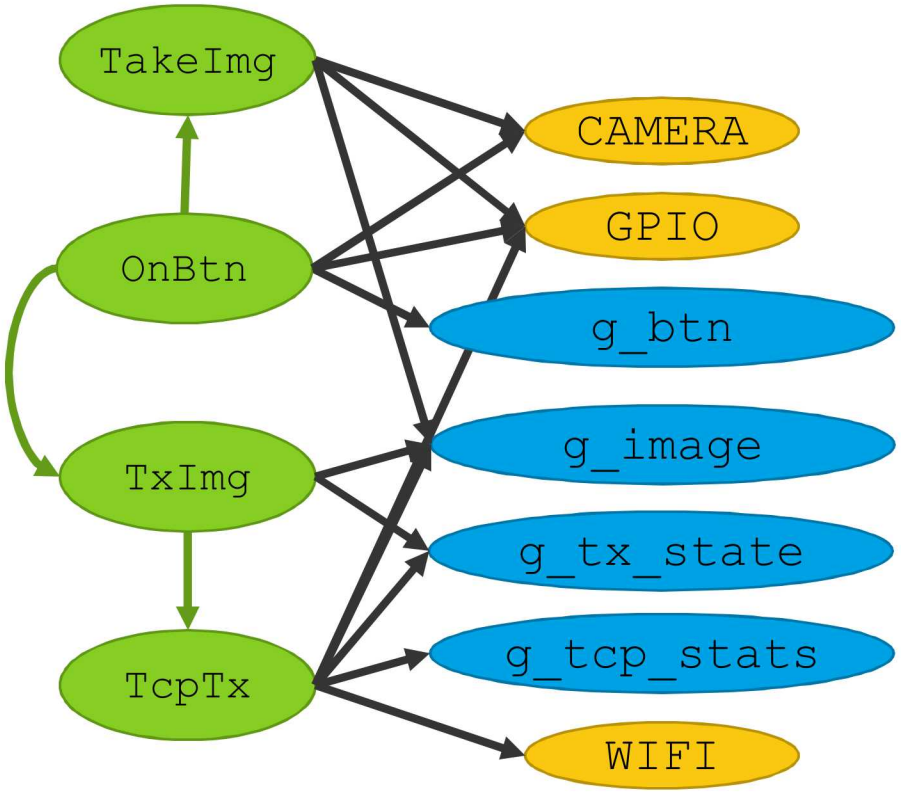


CREATING COMPARTMENT

- Static analysis identifies code, data, and peripheral dependencies
  - ACES ensures compartments can access all required data and peripherals
  - Mirco-Emulator used to dynamically identify missed dependencies due to aliasing
- Compartments are code centric
  - i.e. code belongs to only one compartment
- Policy determines how functions, global variables, and peripherals are grouped to create compartments
- Different policies possible
  - We evaluate: Naïve Filename, Optimize Filename, and Peripheral policies
- MPU is used to restrict access to memory regions

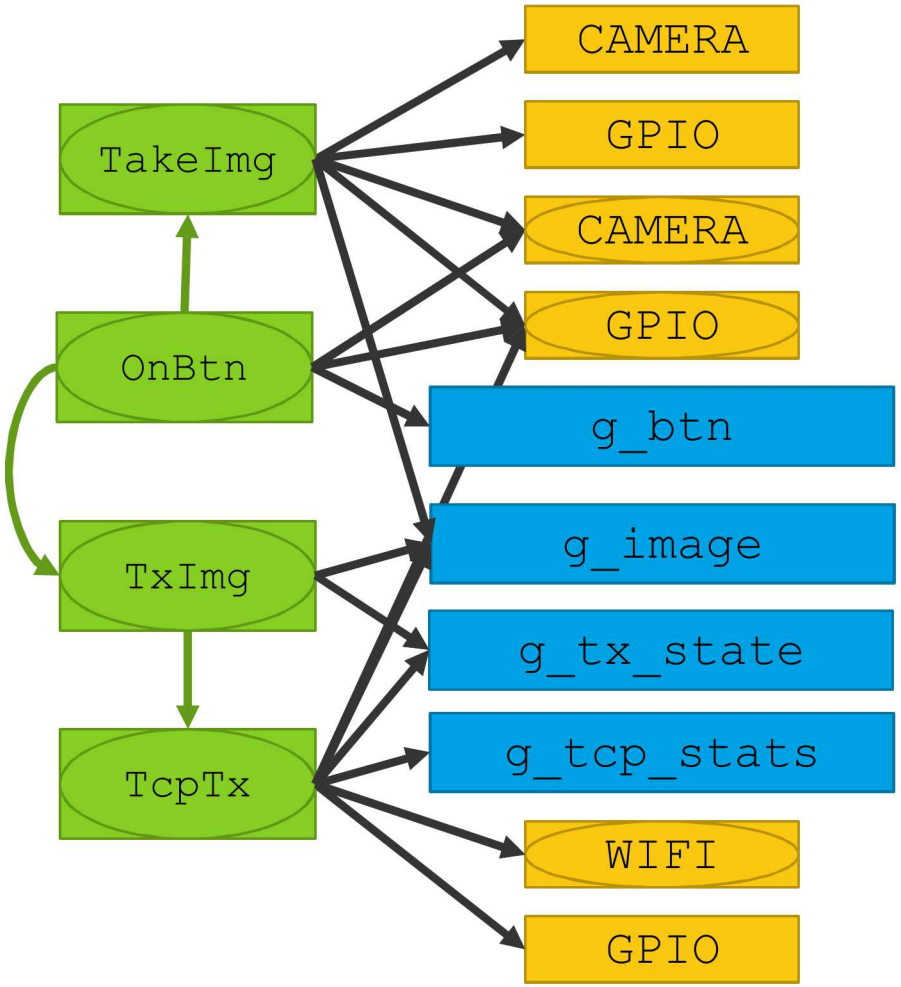
PROGRAM DEPENDENCY GRAPH

```
void TakeImg()
  CAMERA.take_pic = 1
  g_image.buf = CAMERA.rx_reg
  GPIO.led = OFF

void OnBtn()
  GPIO.led = ON
  CAMERA.pwr = ON
  g_btn = PUSHED
  TakeImg()
  TxImg()

void TxImg()
   g_tx_state = ACTIVE
   TcpTx(g_image.buf)
   g_image.sent = TRUE

void TcpTx(*buf)
  GPIO.led_tx = ON
  WIFI.tx_reg = buf
  g_tcp_stats.tx_count++
  g_tx_state = IDLE
```



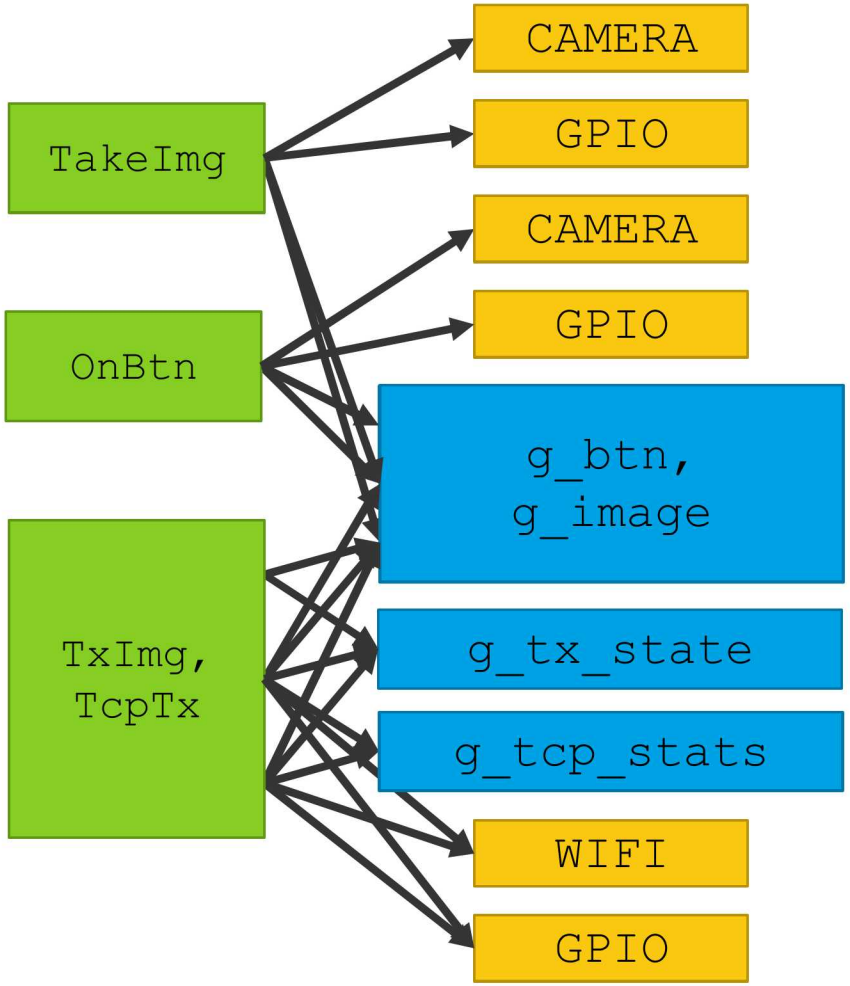
REGION GRAPH

- PDG mapped to a Region Graph
- Functions 1:1
  - Control edges **not** transferred
- Data 1:1
  - Data edges transferred
- Peripherals 1:Many
  - Unique region created per dependency edge



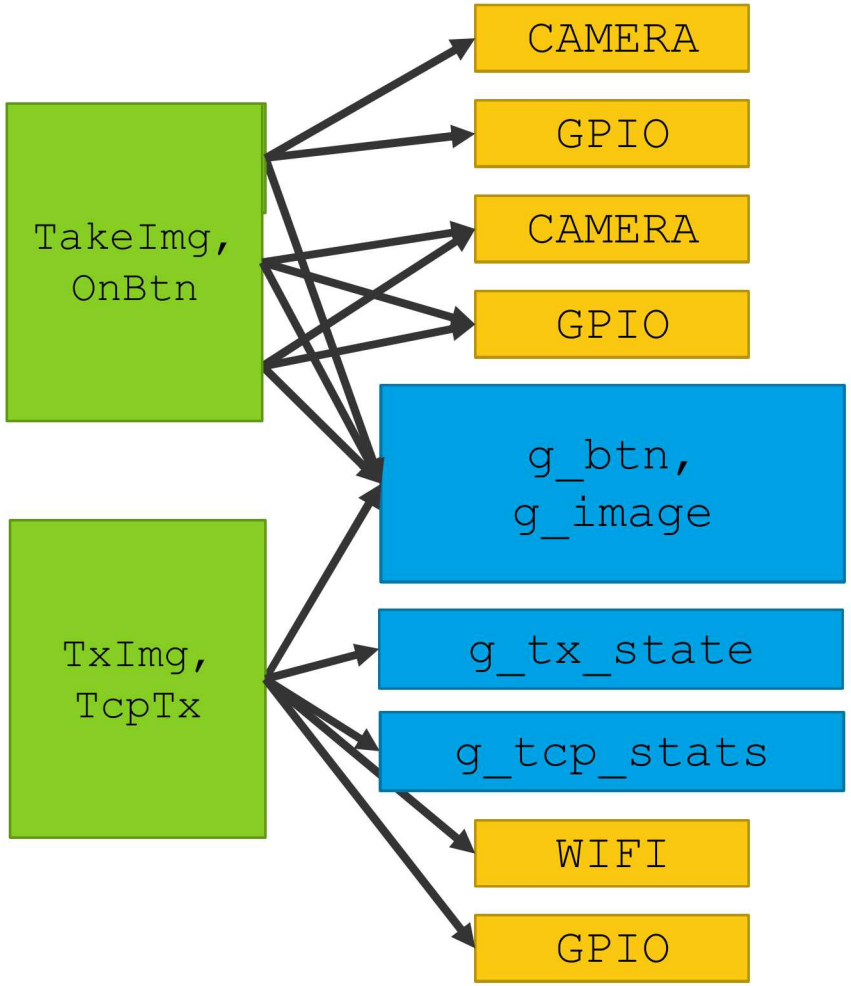
COMPARTMENTALIZATION POLICY

- Defines what should be grouped to form compartments
- Implemented policies
  - Naïve Filename
  - Optimized Filename
  - Peripheral
  - Many more are possible



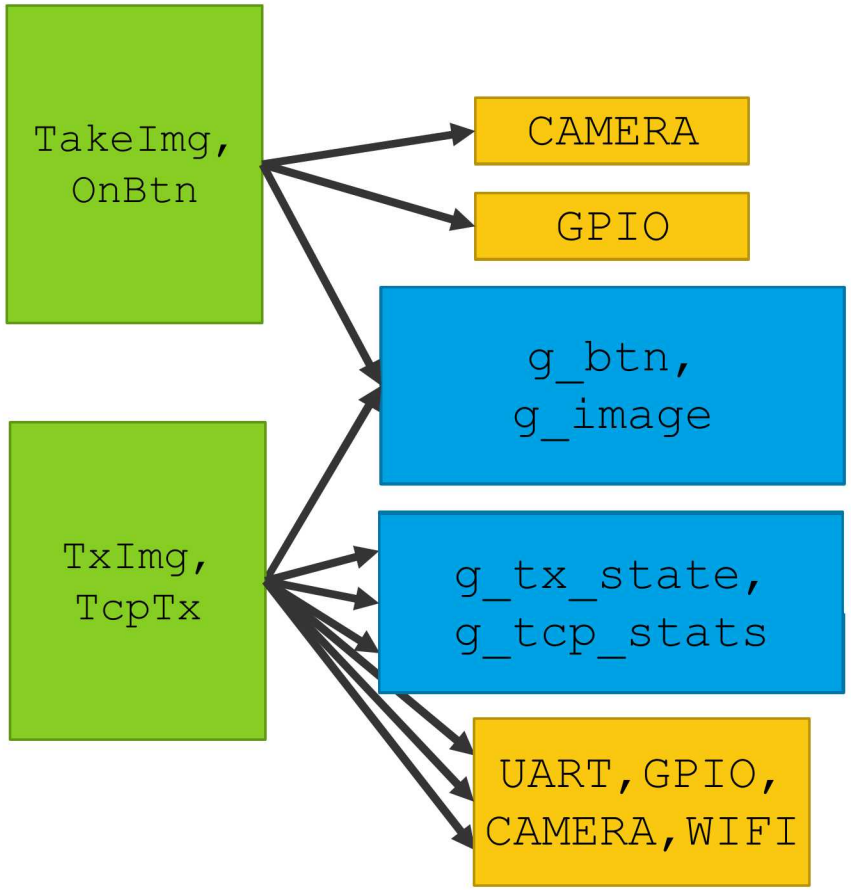
OPTIMIZE

- Improve security
- Improve performance

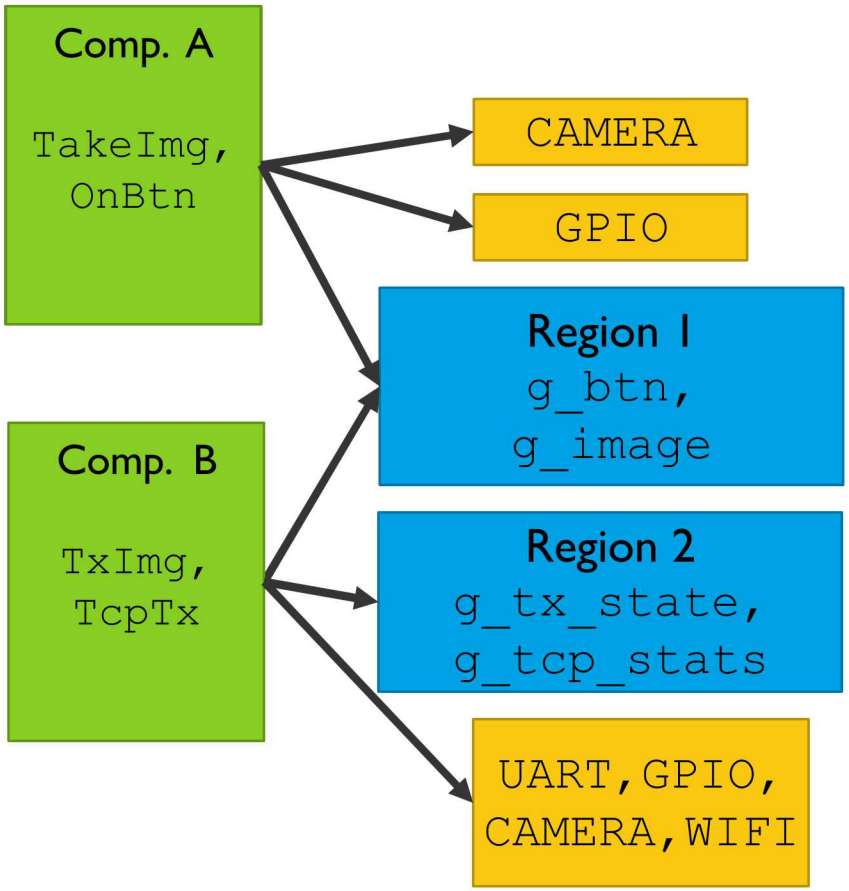


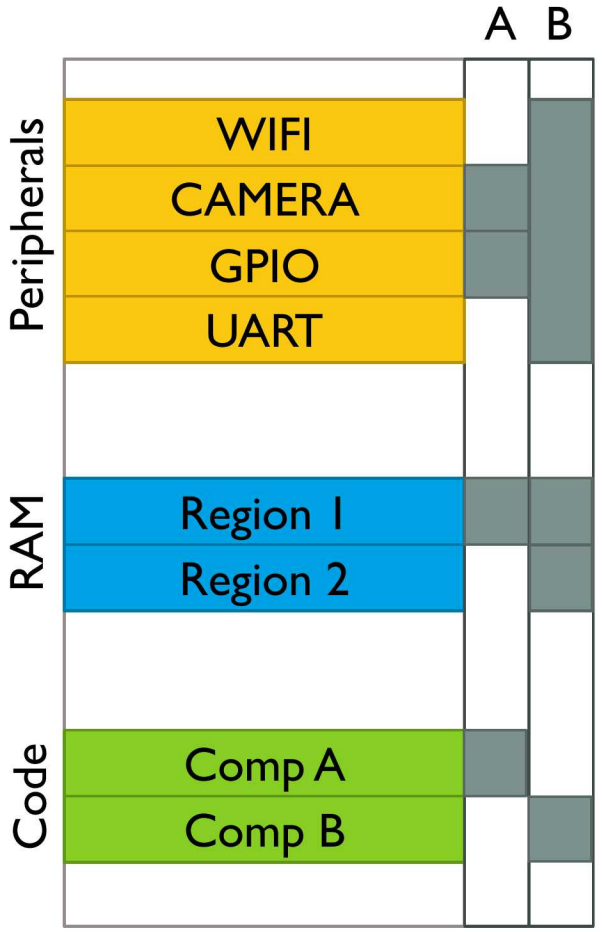
LOWERING

- Reduces graph to meet HW constraints
  - Degree of each code regions must be less than the number of MPU regions
- Lowest cost regions merged until constraints are met
- Lowing may increases permissions of compartments
- Merging peripherals may capture additional peripherals



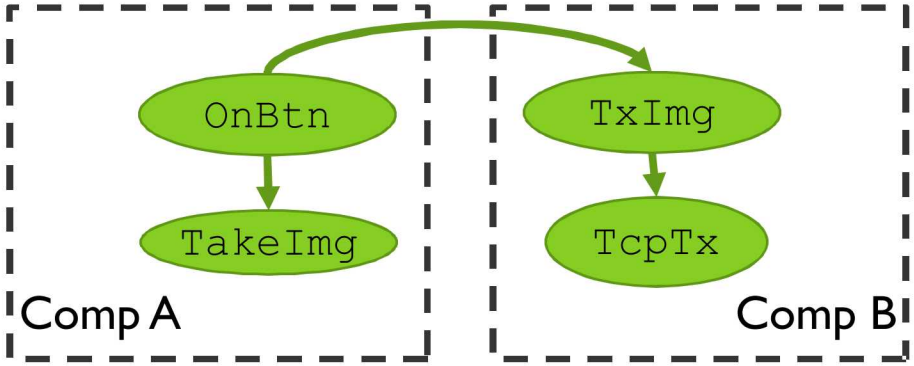
MAPPING TO MEMORY

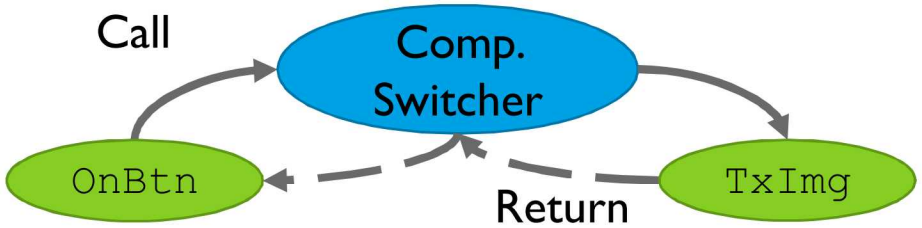




RESTRICTING CONTROL-FLOW

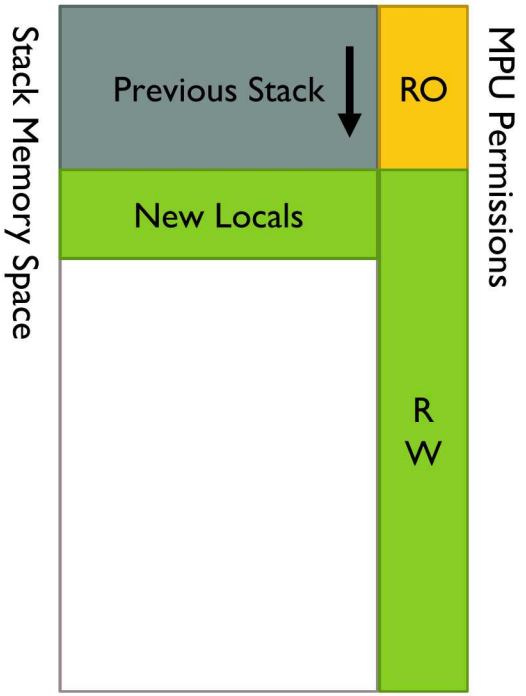
- Instrument calls and returns crossing compartment boundaries
  - Invoke compartment switcher
- Compartment switcher authenticates both directions
  - Memory permissions only changed for valid transitions





MICRO-EMULATOR

- Emulates store instructions in software
  - Overcomes limitations of static analysis in generating PDG
  - Authenticated writes outside a compartment's regions
- Dynamic profiling run creates white-list of accesses per compartment
- Used for stack protection



EVALUATION

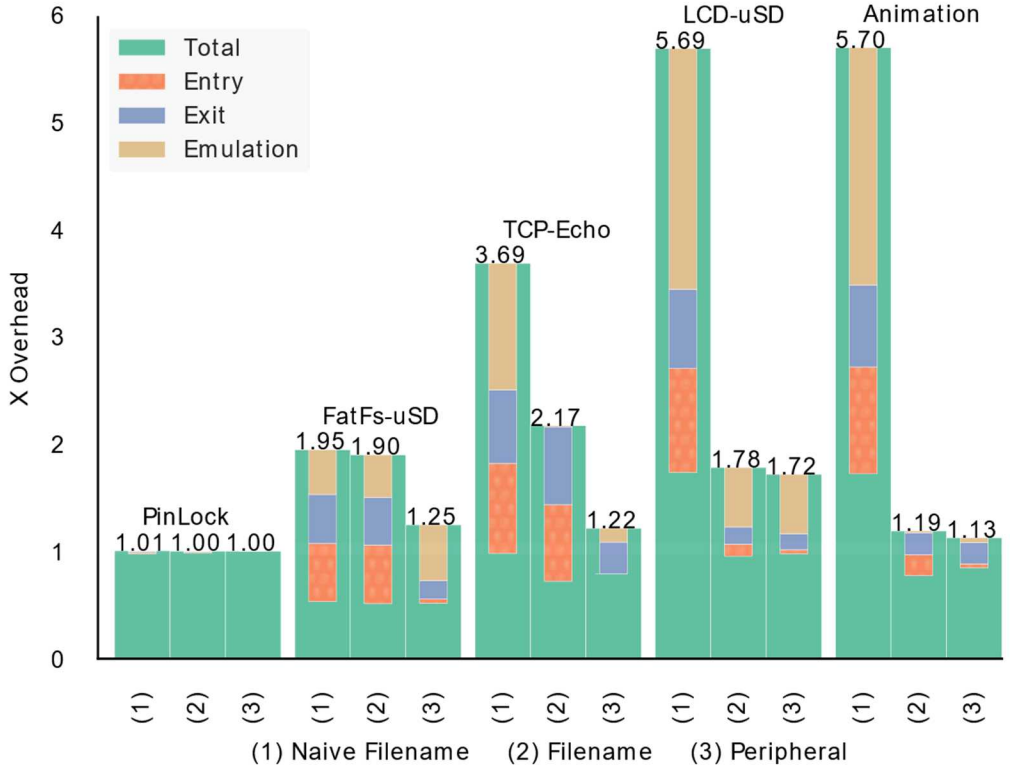
- Evaluated policies for security, runtime, resource usage
  - Naïve Filename
  - Optimized Filename
  - Peripheral
- Five applications run on Cortex-M4
  - PinLock
  - FatFs-uSD
  - TCP-Echo
  - LCD-uSD
  - Animation

PINLOCK CASE STUDY

- Attacker trying to unlock lock
- Assume write-what-where vulnerability in HAL_UART_Receive_IT

	Overwrite		**Control Hijack**	
Policy	Global	GPIO	Direct	Deputy
Naïve Filename	✓	✓	✓	✓
Opt. Filename	✓	✓	✓	✓
Peripheral	✗	✓	✗	✗

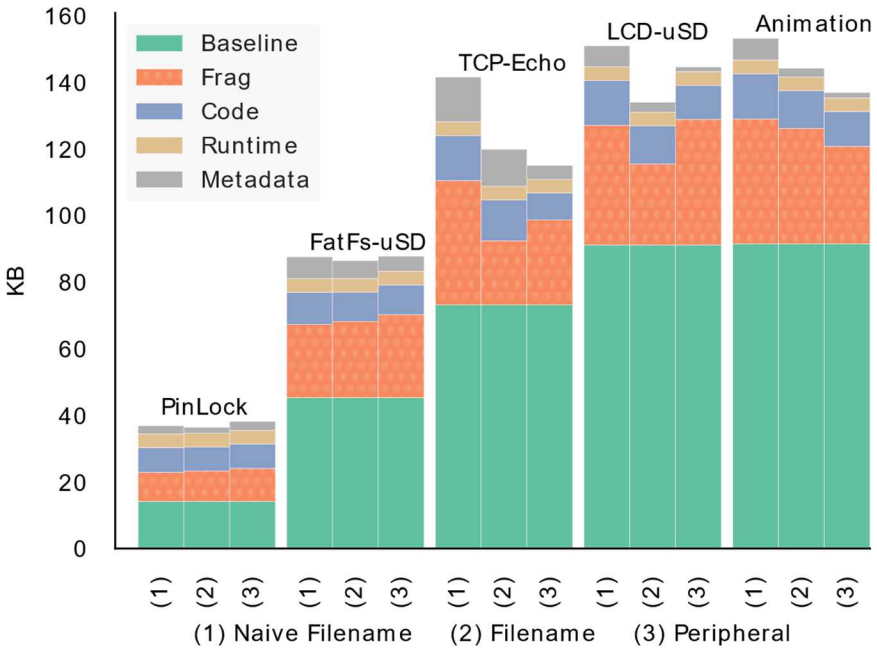
RUNTIME EVALUATION



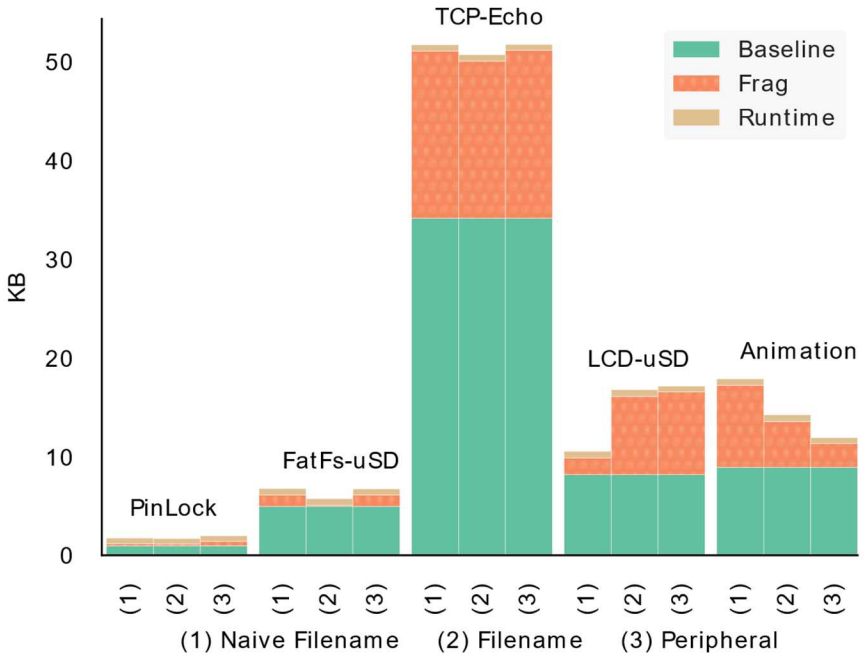
- Can have moderate runtime impact
- Emulating instructions accounts for largest increase in execution time

MEMORY OVERHEAD

Flash Overhead



RAM Overhead



- Largest impact from fragmentation

ACES CONCLUSION

- Applies least privileges to bare-metal IoT devices
  - Does not require changes to application logic
  - Uses existing hardware
- ACES automatically creates and enforces sub-thread
  - Decouples security policy from application
  - Frees developer from having to manage underlying security hardware
  - Enables research in creating compartmentalization policies
- Code available at:

  https://github.com/embedded-sec/ACES

HALUCINATOR: FIRMWARE RE-HOSTING THROUGH ABSTRACTION LAYER EMULATION

Under review at Usenix Security 2019

Authors: Abraham Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer

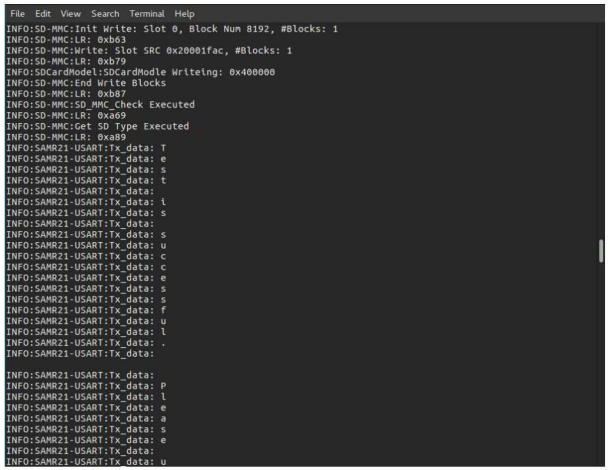
CONTRIBUTIONS

- Equal collaboration with Eric Gustafson
- My contributions:
  - Identification that HAL's provide decoupling point suitable for emulation
  - Development of models for replacing HALs
- Eric's contributions:
  - Development of LibMatch
  - Implementation of Fuzzer
- Primary contribution of HALucinator lies is in combination of these

GOAL

Re-host micro-controller firmware in generic system emulator

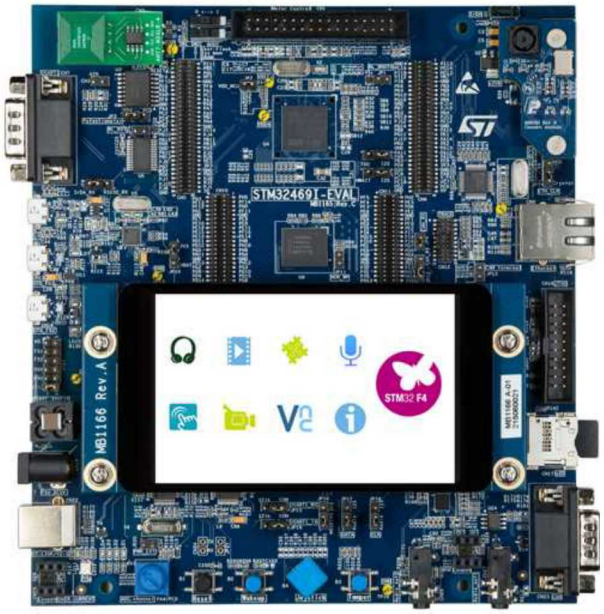
- Smart Watches
- AMI (Smart Meters)
- Wireless Network SoC



WHY RE-HOSTING

- Enables dynamic analysis of bare-metal systems
  - Vulnerability Analysis
  - Fuzzing
  - Automated testing
- Remove dependency on hardware
  - Enables large scale analysis
  - Simplifies test setup and tear down
- We use HALucinator to emulate 12 firmware from 2 manufactures
  - Found: buffer overflows, buffer overreads, double frees, and use-after- free bugs

RE-HOSTING CHALLENGE



Emulation Requires

Internal
- CPU
- AES Accelerator
- IAP
- DMA

External
- Ethernet
- SD-MMC
- IO
- mera
- D
- uch Screen
- reless EEPROM
- Serial
- CAN
- Analog IO
- USB

This is one chip
Mouser lists:
41K micro-controllers with
3,100 unique data sheets from
33 manufactures

Providing proper values for large variety of peripherals is primary challenge of re-hosting these systems

CURRENT APPROACHES

- Manually implement all required hardware in emulator
  - 3,100 datasheets
- Forward hardware accesses to real hardware (AVATAR2[1], Surrogates[2])
  - Require
  - High lat
- Record an
  - Restricts emulation to previously executed code paths[3]

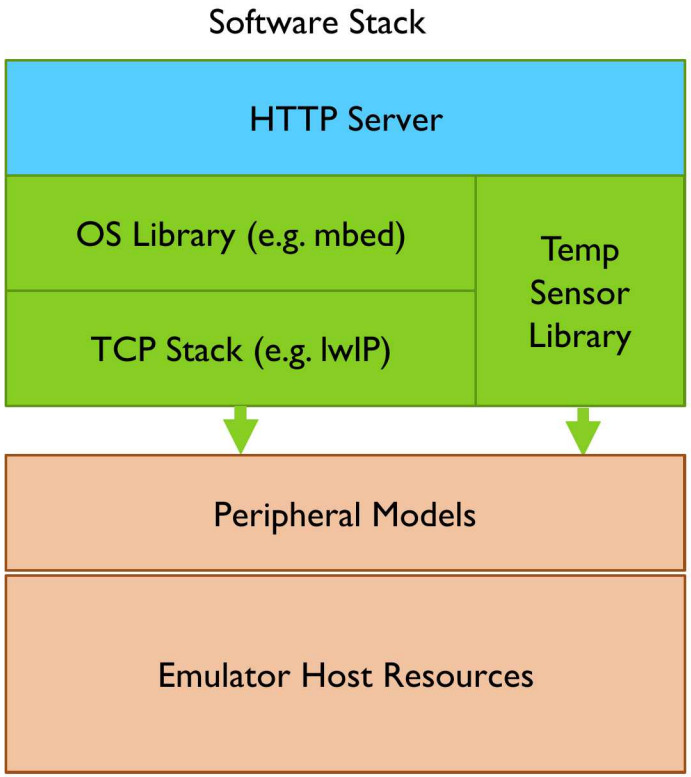
None allow scalable emulation without hardware

[1] Muench, Marius, et al. "Avatar 2: A Multi-target Orchestration Platform." *Workshop on Binary Analysis Research*. 2018

[2] Koscher, Karl, et al. "SURROGATES: Enabling Near-Real-Time Dynamic Analyses of Embedded Systems." *Workshop on Offensive Technologies*. 2015

[3] Tancreti, Matthew, et al. "TARDIS: software-only system-level record and replay in wireless sensor networks." *Information Processing in Sensor Networks*. 2015.

HALUCINATOR CONCEPT



- Application Logic
  - E.g. HTTP Server
  - Developer Implemented
- Middleware

For this to scalably we need:
1. To automatically identify abstraction in firmware
2. A scalable way to develop peripheral models

- Hardware Abstraction Libraries (HAL)
  - Abstracts hardware
  - Provided by micro-controller manufacture
  - Source commonly available
- Peripherals
  - Hardware performs IO

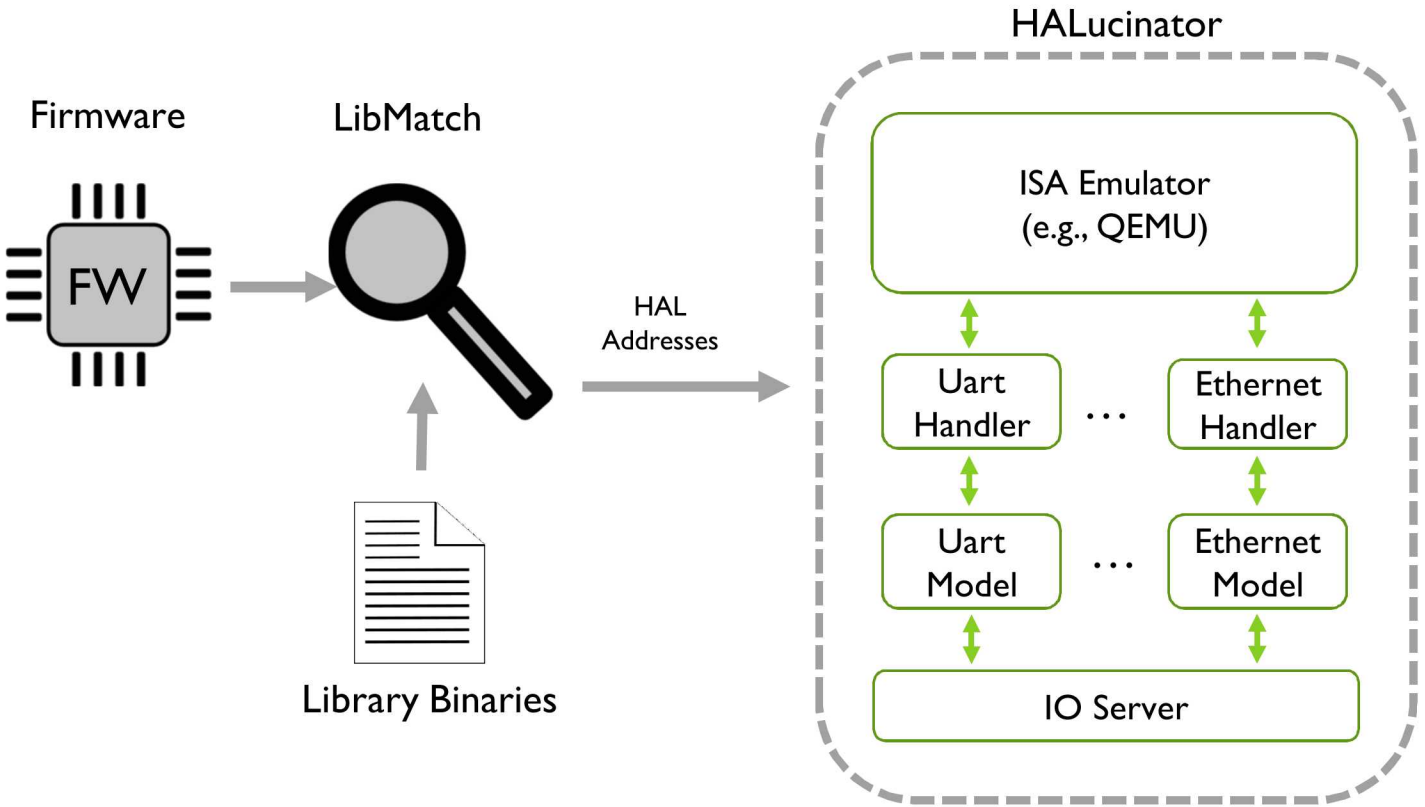
HALUCINATOR OVERVIEW

- Intercept abstraction library functions
  - Use peripheral models to enable I/O
  - Fix up state so firmware executes as if function executed
- Scalable
  - Changes supporting number of devices to number of HALs
    - e.g., 33 Manufactures vs 3,100 parts
  - Peripheral Models
    - Core functionality of peripherals is independent of micro-controller
    - Implement once per peripheral type (e.g. Ethernet)
  - Intercept Handlers
    - Unique handler to map each HAL to peripheral model
    - Implement once per HAL

REQUIREMENTS

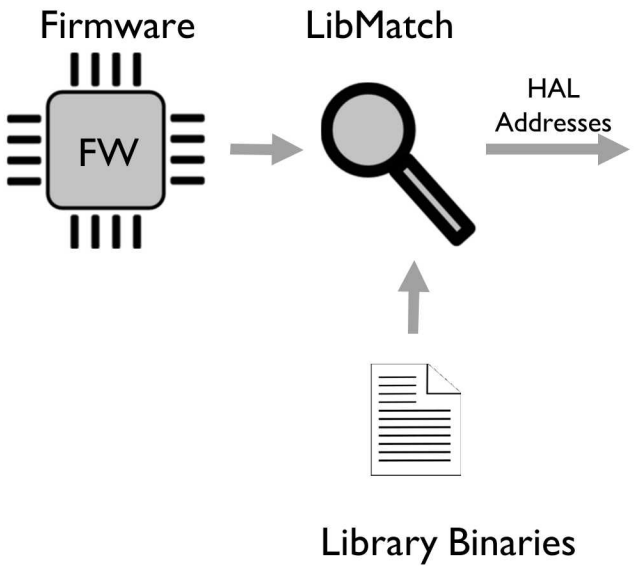
- Firmware to emulate
- Emulator for ISA
  - Instruction Set Architecture
  - General memory layout of device
- Libraries with symbols to replace
  - Compiled with same flags as firmware
- Ability to hook execution specific addresses
  - Read/Write processor state within the hook

DESIGN


Firmware
LibMatch
FW
HAL
Addresses
Library Binaries
HALucinator
ISA Emulator
(e.g., QEMU)
Uart
Handler
Ethernet
Handler
Uart
Model
Ethernet
Model
IO Server

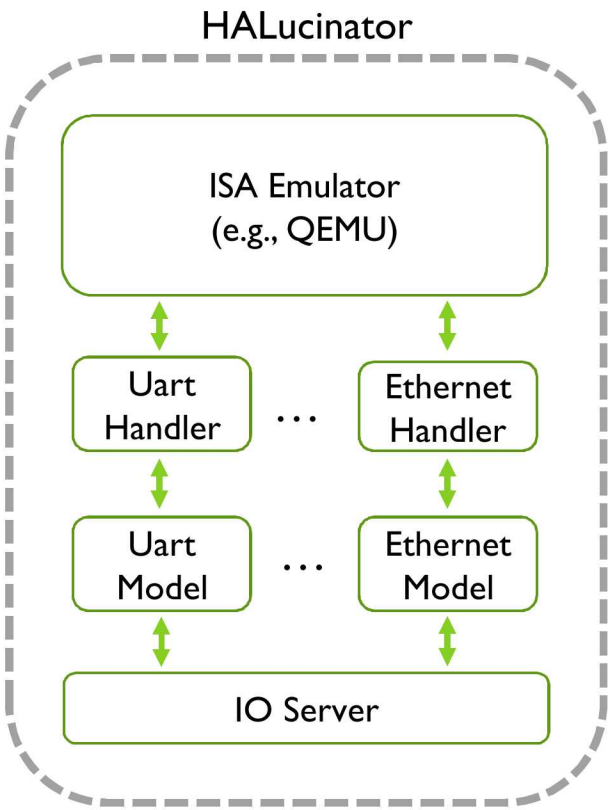
LIBMATCH CONTRIBUTIONS

- Identifies location of abstraction functions in firmware
- Resolve binary equivalent functions
  - E.g Getters and setters on MMIO registers
- Callbacks/Override function
  - Identify code statically called by HAL (e.g. weak functions)



HALUCINATOR

- Input
  - HAL addresses from LibMatch
  - Mapping of HAL function to intercept handlers
- Intercept handler
  - Maps HAL to peripheral model
- Peripheral Model
  - Abstracts peripheral
  - Common for peripheral type
- IO Server
  - Enables flexible external communication
  - Sends and receives tagged messages



INTERCEPT HANDLERS

STM32Cube Ethernet TX Frame

```
HAL_StatusTypeDef
HAL_ETH_TransmitFrame(
   ETH_HandleTypeDef *heth,
   uint32_t FrameLength);
```

Intercept Handler

```
f_ptr = heth->TxDesc->buf
frame = read_memory(f_ptr,
                    FrameLength)
model.send_frame(frame)
return 0 #(HAL_OK)
```

ATMEL Ethernet TX Frame

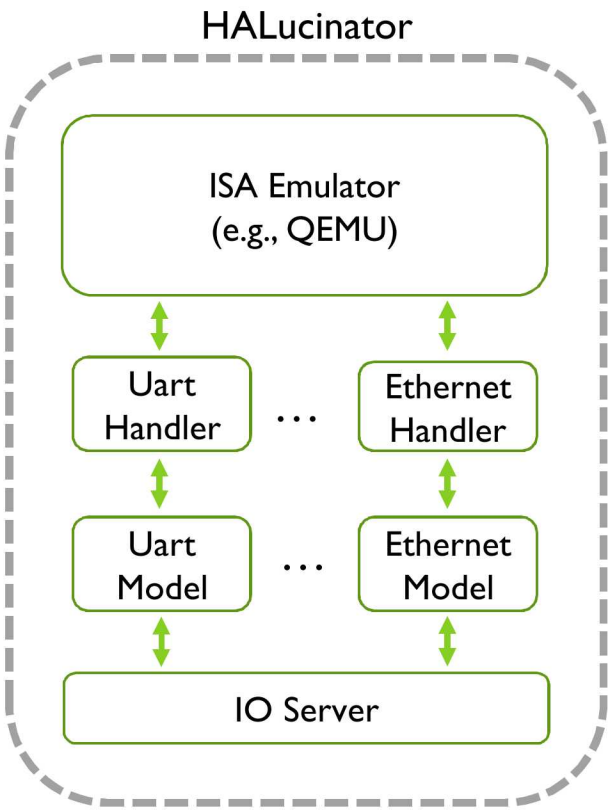
```
void ksz8851_fifo_write(
   uint8_t *buf,
   uint32_t len);
```

Intercept Handler

```
frame = read_memory(buf,len)
model.send_frame(frame)
```

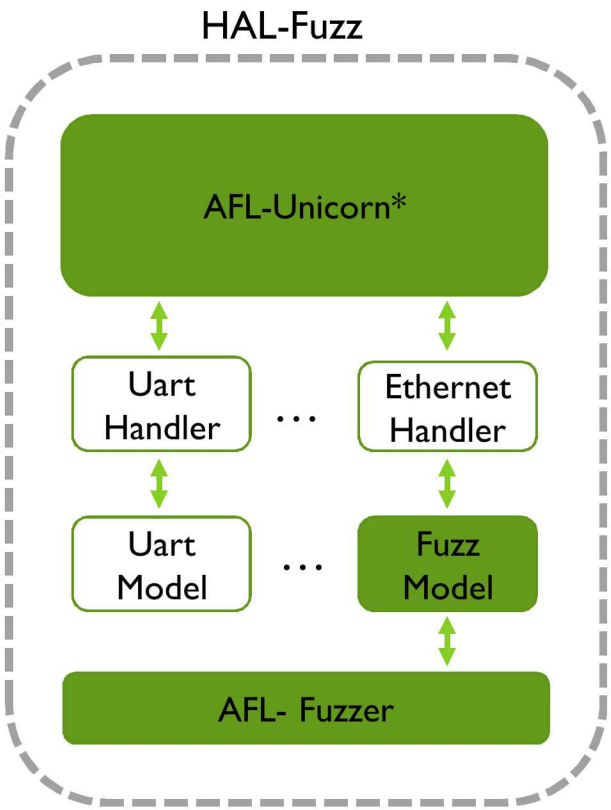
HALUCINATOR IMPLEMENTATION

- Build on Avatar2
  - Provides python API to instrument QEMU, GDB, and angr.
- ISA Emulator is QEMU
- Interception done using GDB
- Everything written in Python



HAL-FUZZ

- Replaced QEMU with AFL-Unicorn*
  - Stripped down QEMU integrated with AFL
- Use Fuzz model to replace input with fuzz output from AFL
- Control sources of non-determinism
  - Timers tied to basic blocks executed
  - Interrupt execution tied to basic blocks
  - Rand() deterministic



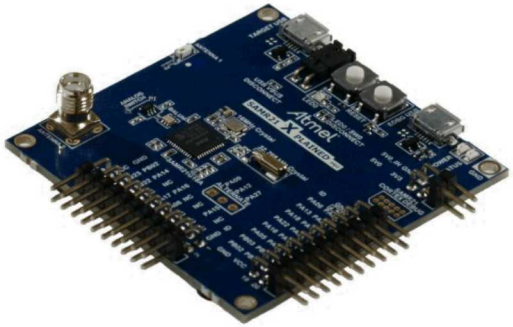
*https://github.com/Battelle/afl-unicorn

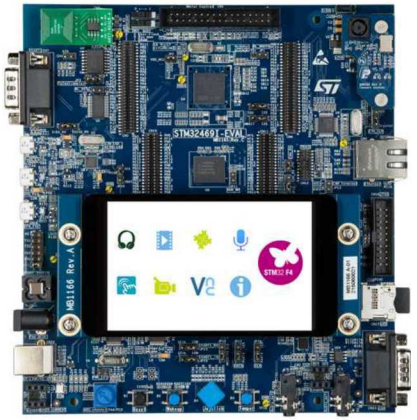
HALUCINATOR EVALUATION

- Determine effectiveness of LibMatch
- Evaluate interactive emulation
  - Validate Black Box behavior with external interaction
- Assess complexity of implementing handlers and models
- Demonstrate usefulness of HALucinator by fuzzing
  - Execution not constrained to single path

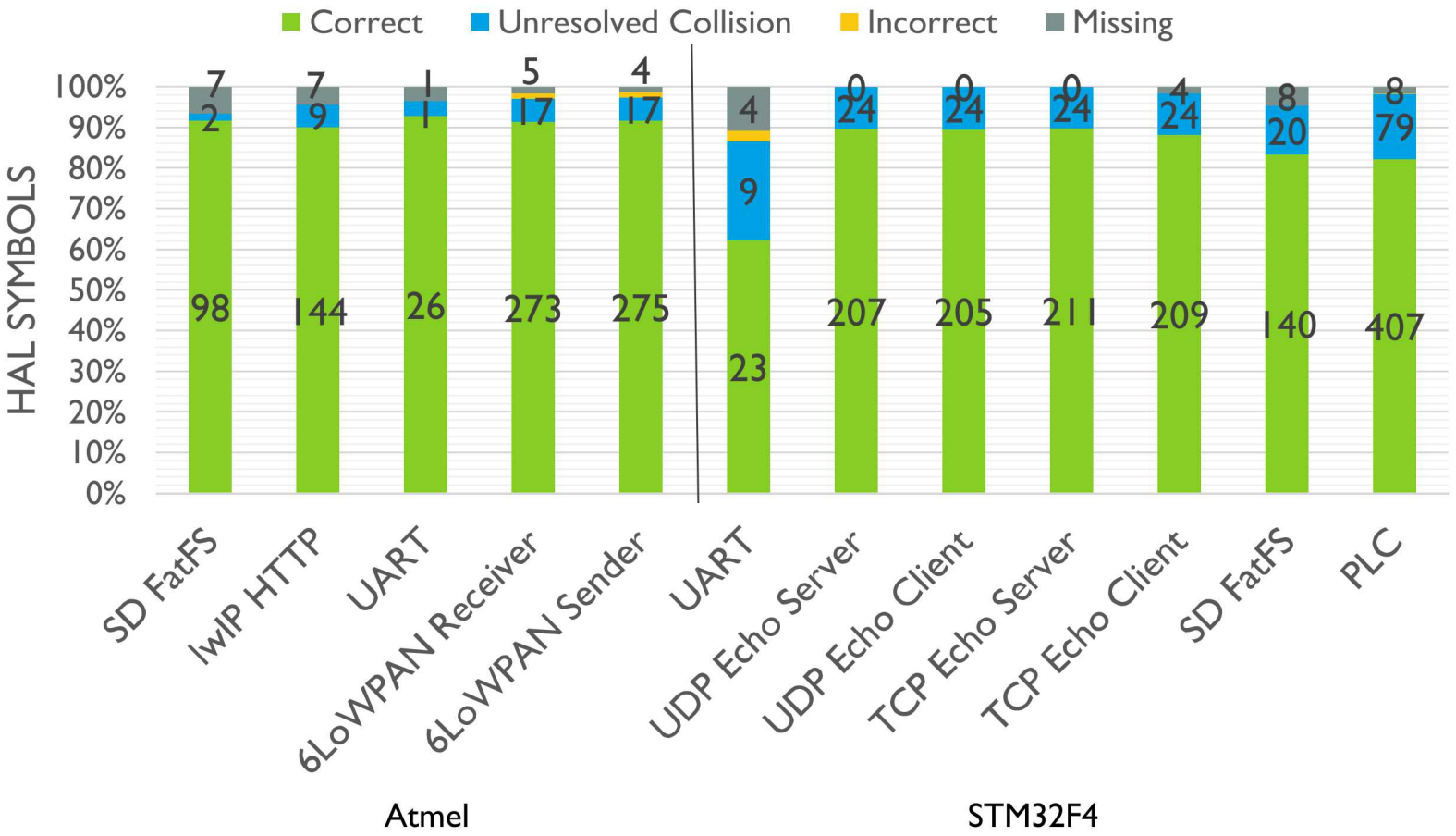
EXAMPLE APPLICATIONS

- ATMEL SAMR21
  - USART-Terminal
  - SD-Card
  - HTTP-Server
  - 6LoWPAN Sender and Receiver
- STM32F479 Eval Board
  - UART
  - SD-Card
  - UDP-Echo Server and Client
  - TCP-Echo Server and Client
  - PLC

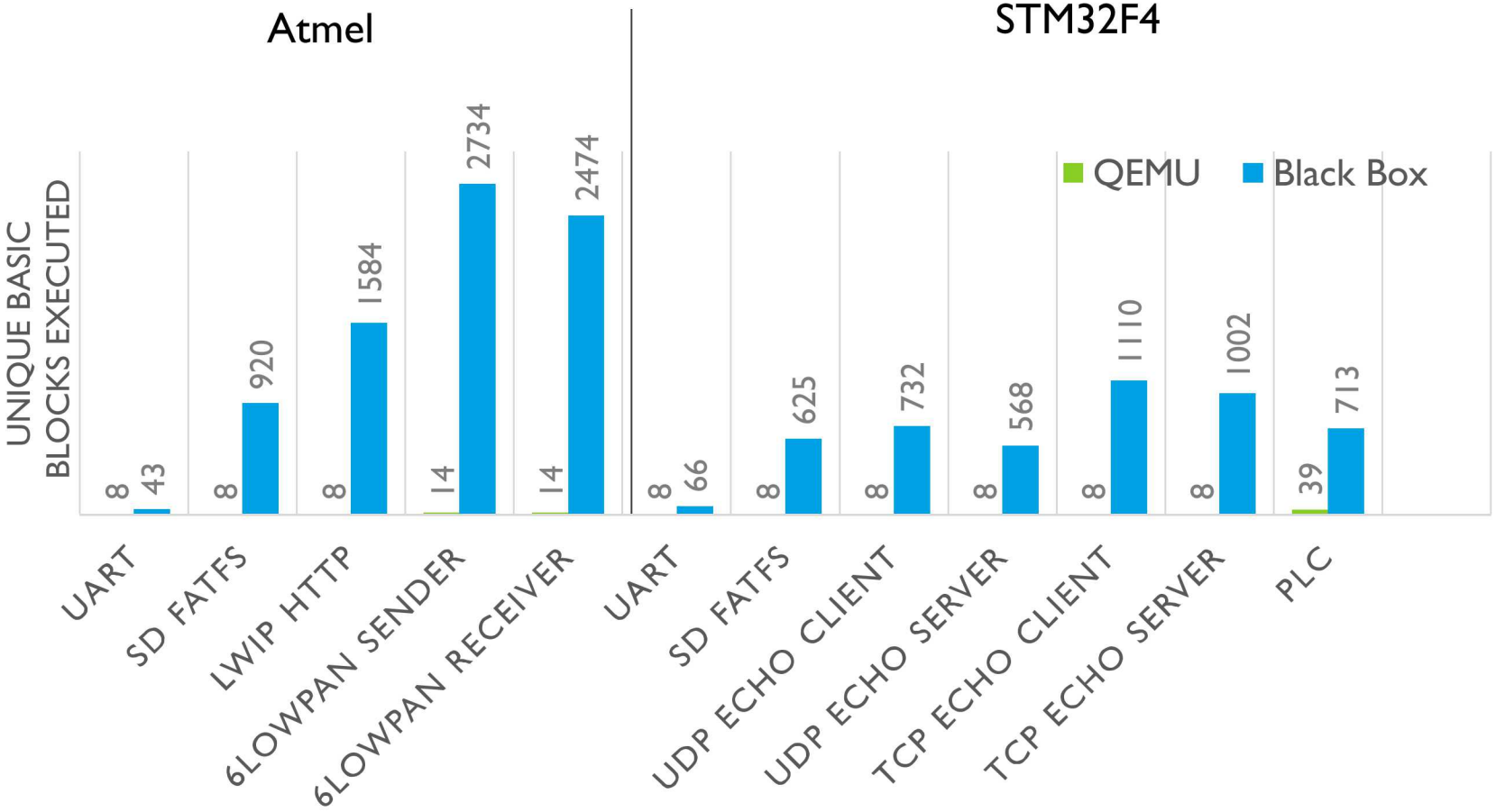




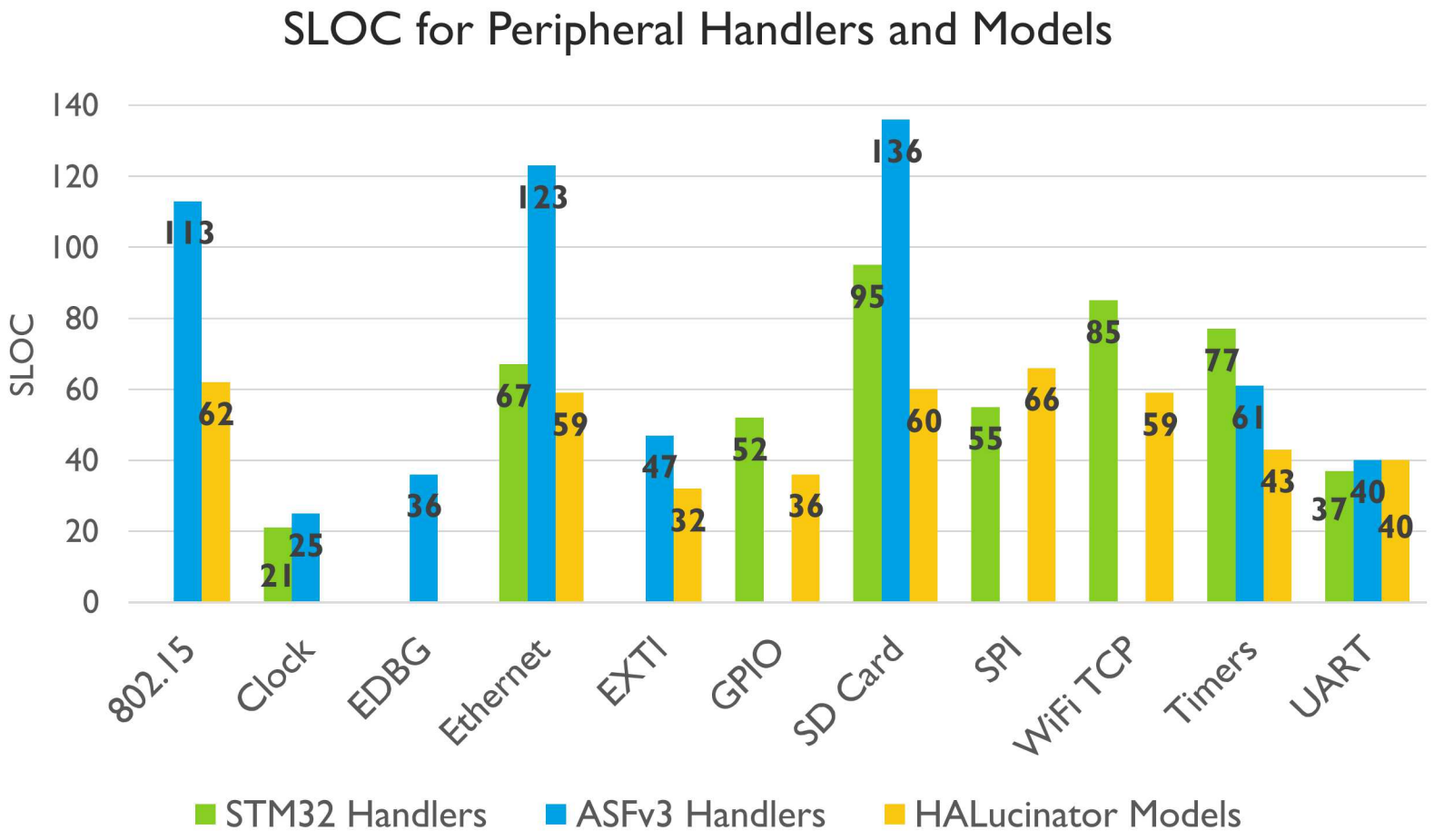
LIBMATCH ANALYSIS



INTERACTIVE EMULATION



PERIPHERAL MODEL COMPLEXITY



KEY FUZZING RESULTS

- Fuzzed the nine network connected firmware samples
- Bugs either fixed or reported to application vendor

Application	Bugs	Possible Attack	Fuzzing Layer
LWIP HTTP	Buffer Overread	Memory Leak	TCP
LWIP HTTP	Double free, Use-after free	Denial of Service	Ethernet
6LoWPAN	Buffer Overflow	Remote Code Exe	802.15.4
PLC	Buffer Overflow	Remote Code Exe	TCP

HALUCINATOR CONCLUSION

- Presents novel concept of high-level emulation to enable re-hosting bare-metal firmware
- Enables interactive emulation and fuzzing of firmware in scalable way

THESIS CONCLUSION

Case studies show that using static and dynamic analysis, we can automatically enable protections against memory corruption and control-flow hijack attacks on bare-metal systems

- EPOXY
  - Demonstrates bare-metal applications can enjoy protections as strong as desktop computers with negligible overhead
- ACES
  - Shows many small compartments can be automatically formed within a single application reducing the impact a single vulnerability has on entire system
- HALucinator
  - Scalably emulates bare-metal systems enabling dynamic analysis
  - Enabling state-of-the-art coverage based fuzzing of bare-metal systems

PUBLICATIONS

1. **A. A. Clements**, N. S. Almakhdhub, S. Bagchi, M. Payer, *"ACES: Automatic Compartments for Embedded Systems"*, Usenix Security 2018.
2. **A. A. Clements**, N. S. Almakhdhub, K. Saab, J. Koo S. Bagchi, M. Payer, *"Protecting Bare-metal Systems with Privilege Overlays"*, IEEE Security & Privacy 2017.
3. N. S. Almakhdub, **A. A. Clements**, M. Payer, and S. Bagchi, *"IoT2: A benchmark for the things in the Internet of Things"*, To Appear at Dependable Systems and Networks 2019.
4. A. R. Hota, **A. A. Clements**, S. Bagchi, and S. Sundaram, *"A Game-Theoretic Framework for Securing Interdependent Assets in Networks,"* (To Appear) Game Theory for Security Risk Management – From Theory to Practice, Springer/Birkhauser's series on "Static & Dynamic Game Theory: Foundations and Applications." editors: Stefan Rass, Stefan Schauer, pp. 1-28, 2018.
5. A. R. Hota, **A. A. Clements**, S. Sundaram, and S. Bagchi, *"Optimal and Game-Theoretic Deployment of Security Investments in Interdependent Assets."* GameSec 2016.

Under Review

1. **A. A. Clements**, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Kruegel, G. Vigna, S. Bagchi, and M. Payer *"Halucinator: Firmware Re-hosting Through Abstraction Layer Emulation"*

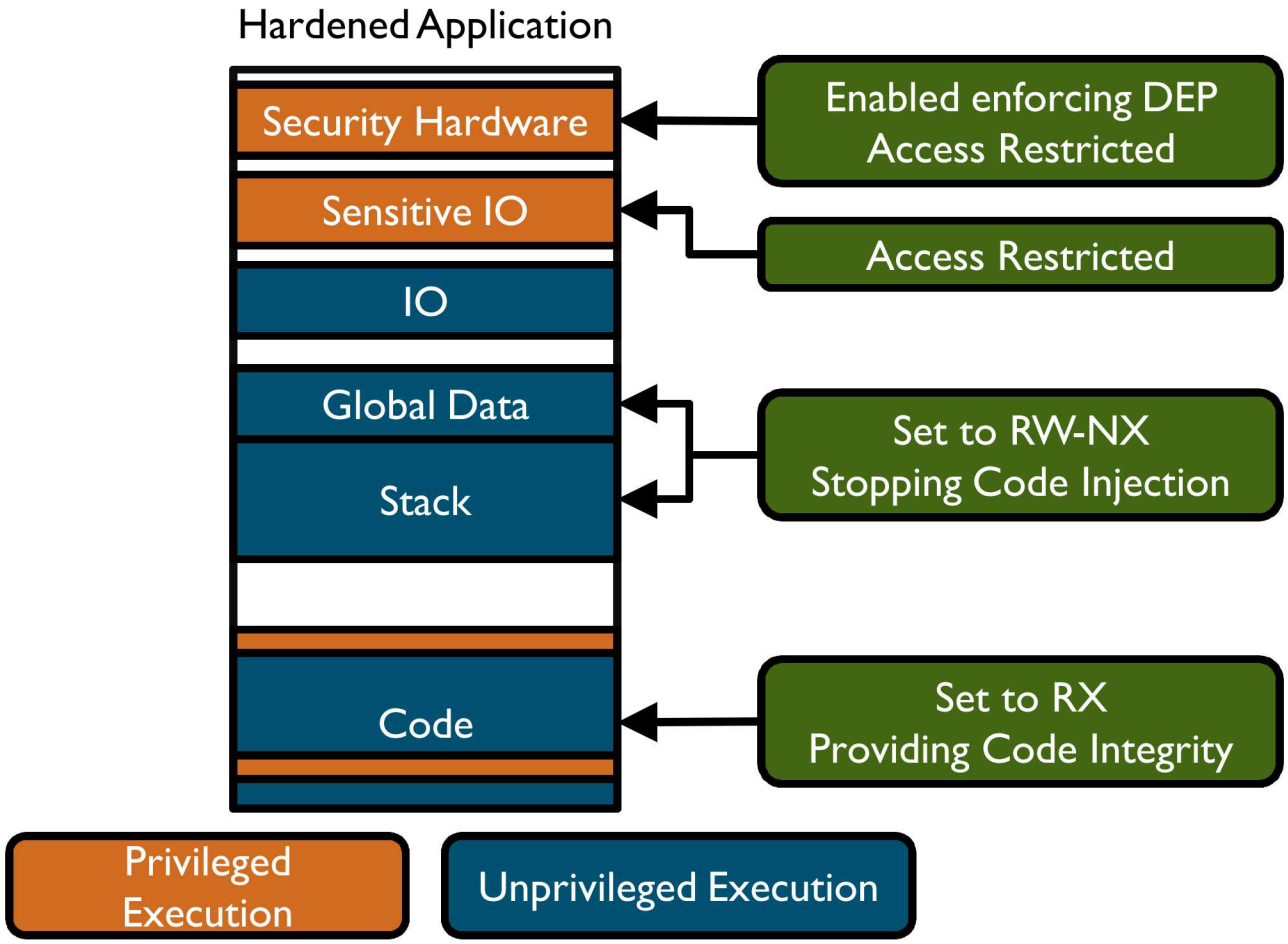
QUESTIONS

PURDUE
UNIVERSITY.

BACKUP

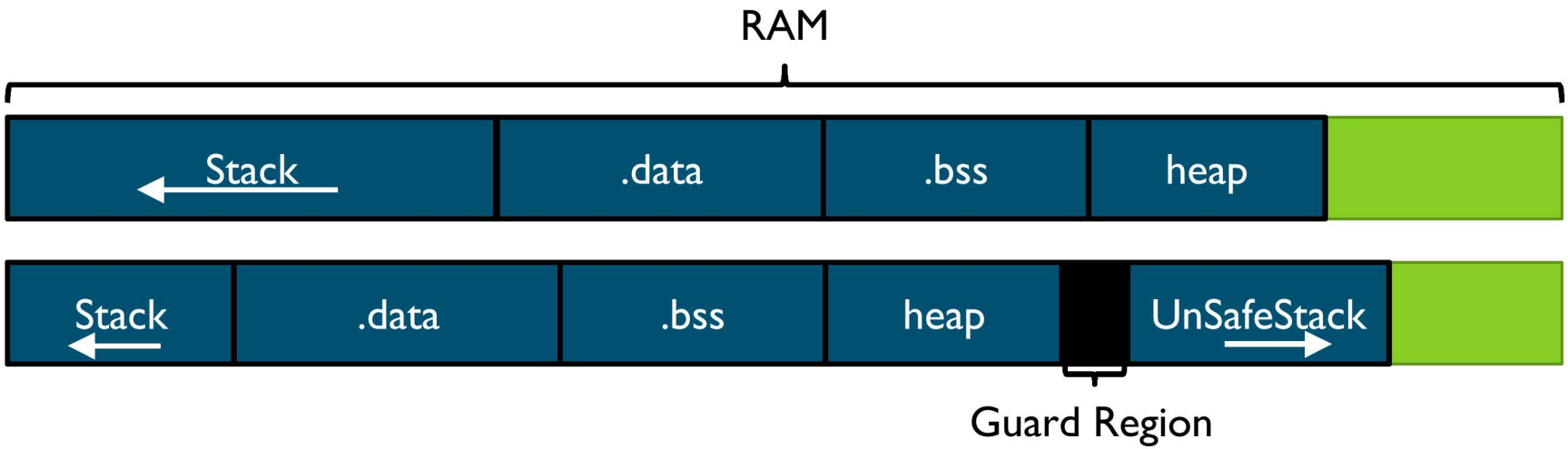
EPOXY BACKUP

EPOXY – AFTER PRIVILEGE OVERLAY



SAFESTACK

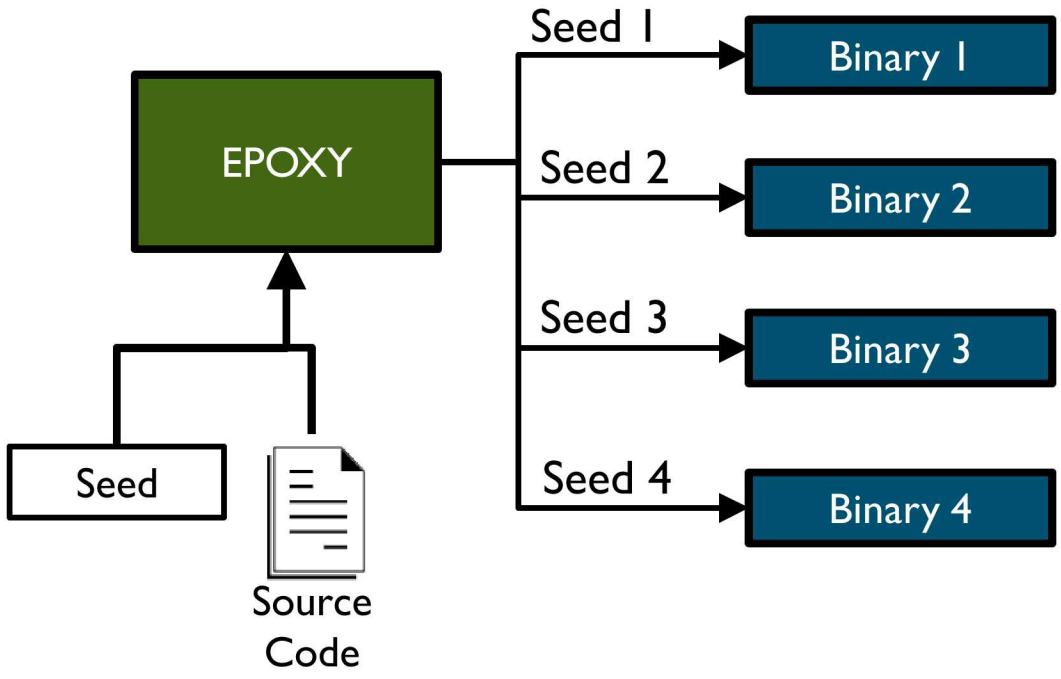
- SafeStack from Code Pointer Integrity *
  - Protects against stack smashing
  - "Unsafe" variables moved to separate stack
  - We adapted to bare-metal systems



*V. Kuznetsov *et al.*, Code Pointer Integrity, OSDI 2014

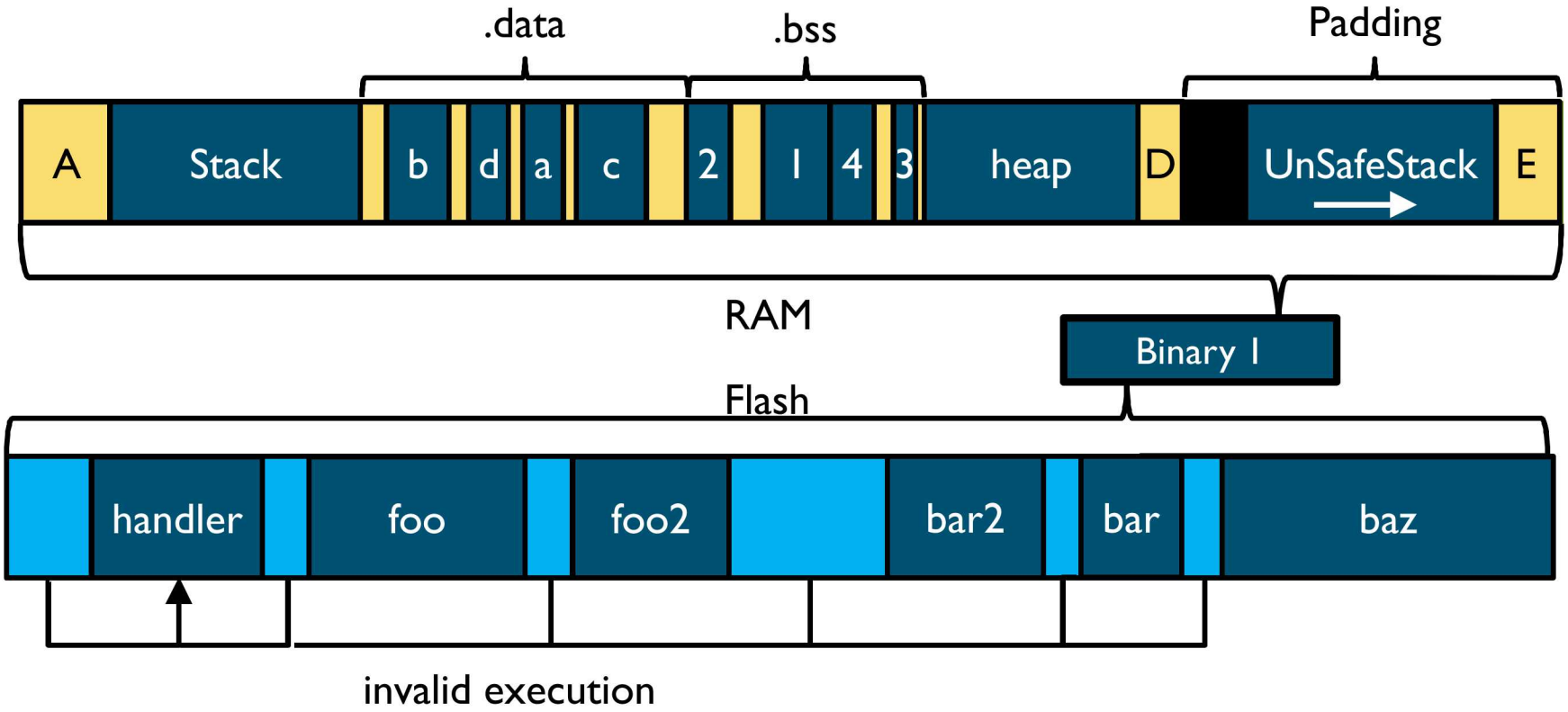
DIVERSIFICATION

- Further protects against ROP attacks
- Corruption of specific global data



DIVERSIFICATION

- Further protects against ROP attacks
- Corruption of specific global data



MEMORY USAGE

Test App	Code	Global Data	Stack
PinLock	3,390 (29%)	14.6 (1%)	104 (25%)
FatFs-uSD	2,839 (12%)	18.2 (1%)	164 (4%)
TCP-Echo	3,249 (8%)	7.2 (0%)	128 (29%)

Absolute units are bytes

- Stack increase because two stacks used
  - Stack size is max size of deepest regular stack and deepest SafeStack

ROP COMPILER

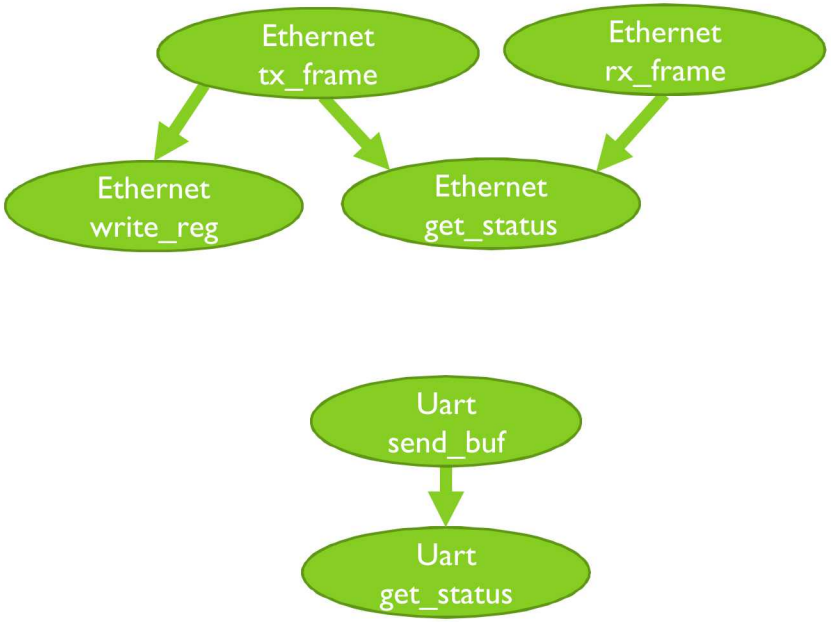
- Used ROPgadget compiler* to identify gadgets across 1000 variants
  - Gadget survives if same instructions (ending with a branch) at same address

App	Total	# Surviving Across 2	5	25	50	Last
PinLock	294K	14K	8K	313	0	48
FatFS-uSD	1,009K	39K	9K	39	0	32
TCP-Echo	676K	22K	9K	985	700	107

* J. Salwan, ROPgadget, http://shell-storm.org/project/ROPgadget/

LIBMATCH ALGORITHM

- Statistical Comparison
  - Number BB
  - Number unique functions called
- Basic Block Comparison
  - Does VEX IR match
- Call Graph Contextual Matching
  - Resolves conflicts
  - Gives names to callback functions

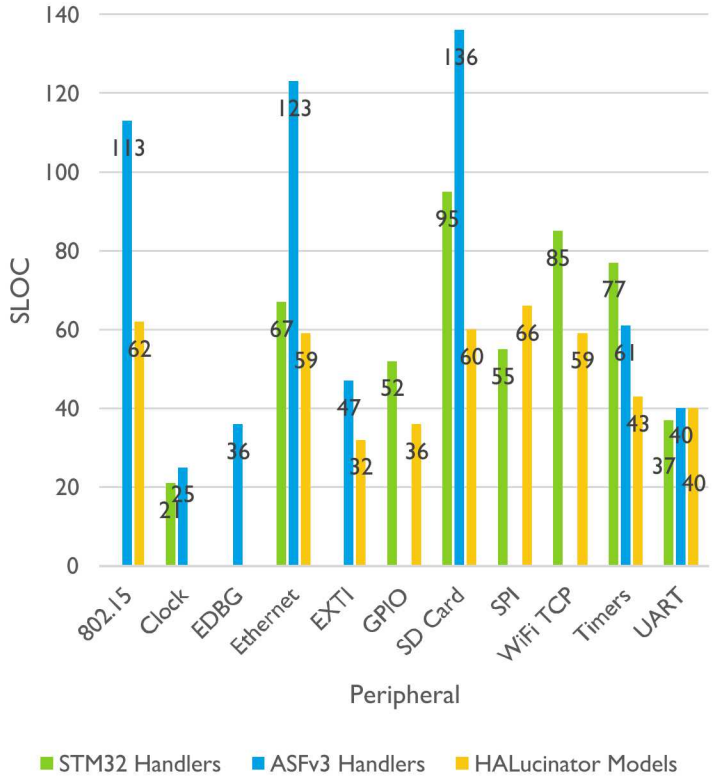


RELATED WORK

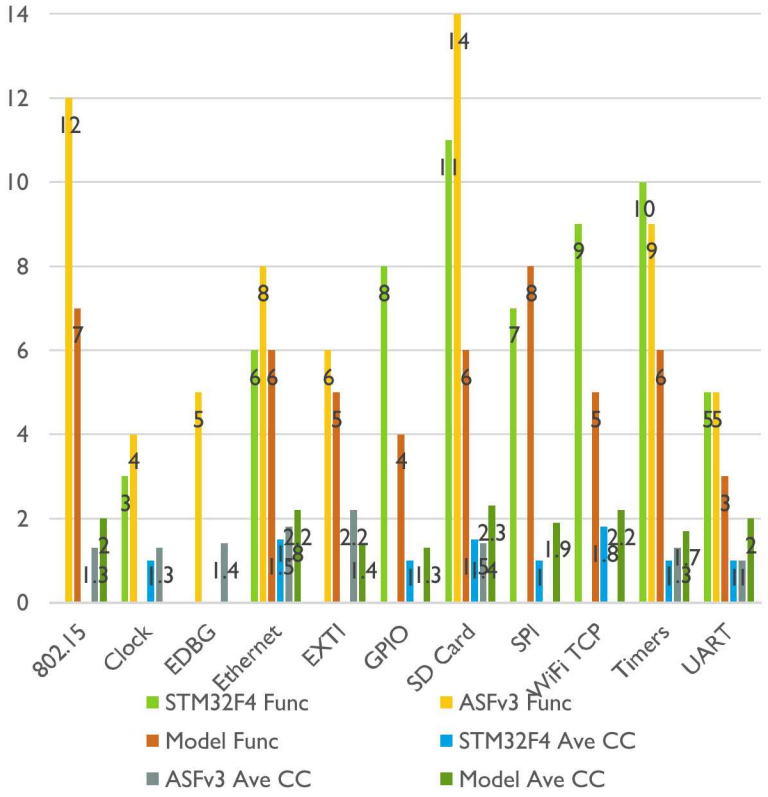
- General Computing Defenses
  - Artificial Diversity (Larsen et al, SoK IEEE S&P 2014)
    - ASLR (Pax Team 2003)
  - Code Pointer Integrity (Kuznetsov et al., OSDI 2014)
  - Control Flow Integrity (Burow et al., ACM Computing Surveys 2018)
- Embedded Systems
  - Cui and Stolfo Symbiotes (RAID 2011)
  - FreeRTOS-MPU
  - Firmware Attestation
    - Eldefrawy NDSS 2012, Francillon EDAA 2014, Abera CSS 2016, Li CCS 2011
  - ARM-Trust Zone

PERIPHERAL MODEL COMPLEXITY

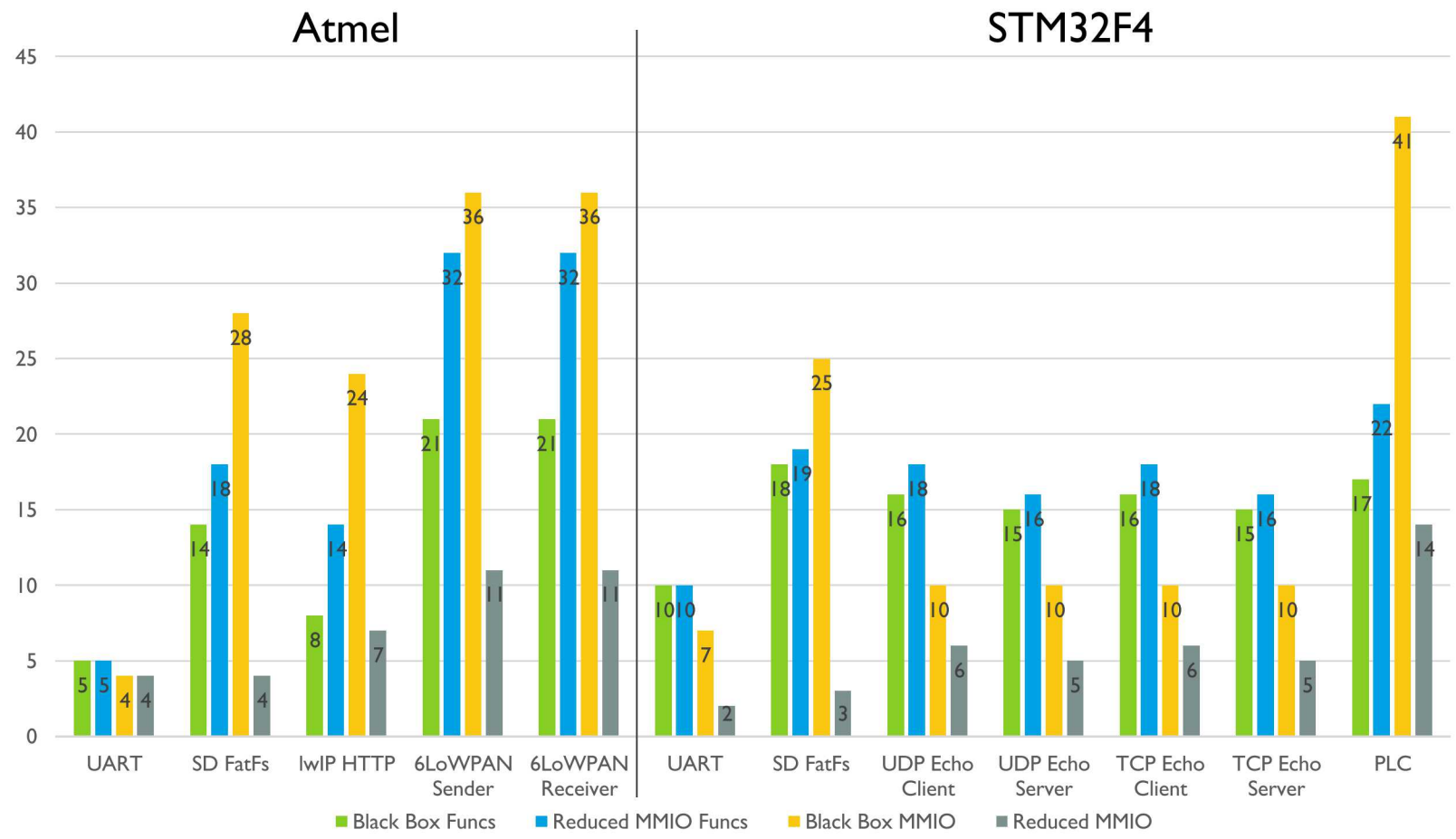
SLOC handlers and models



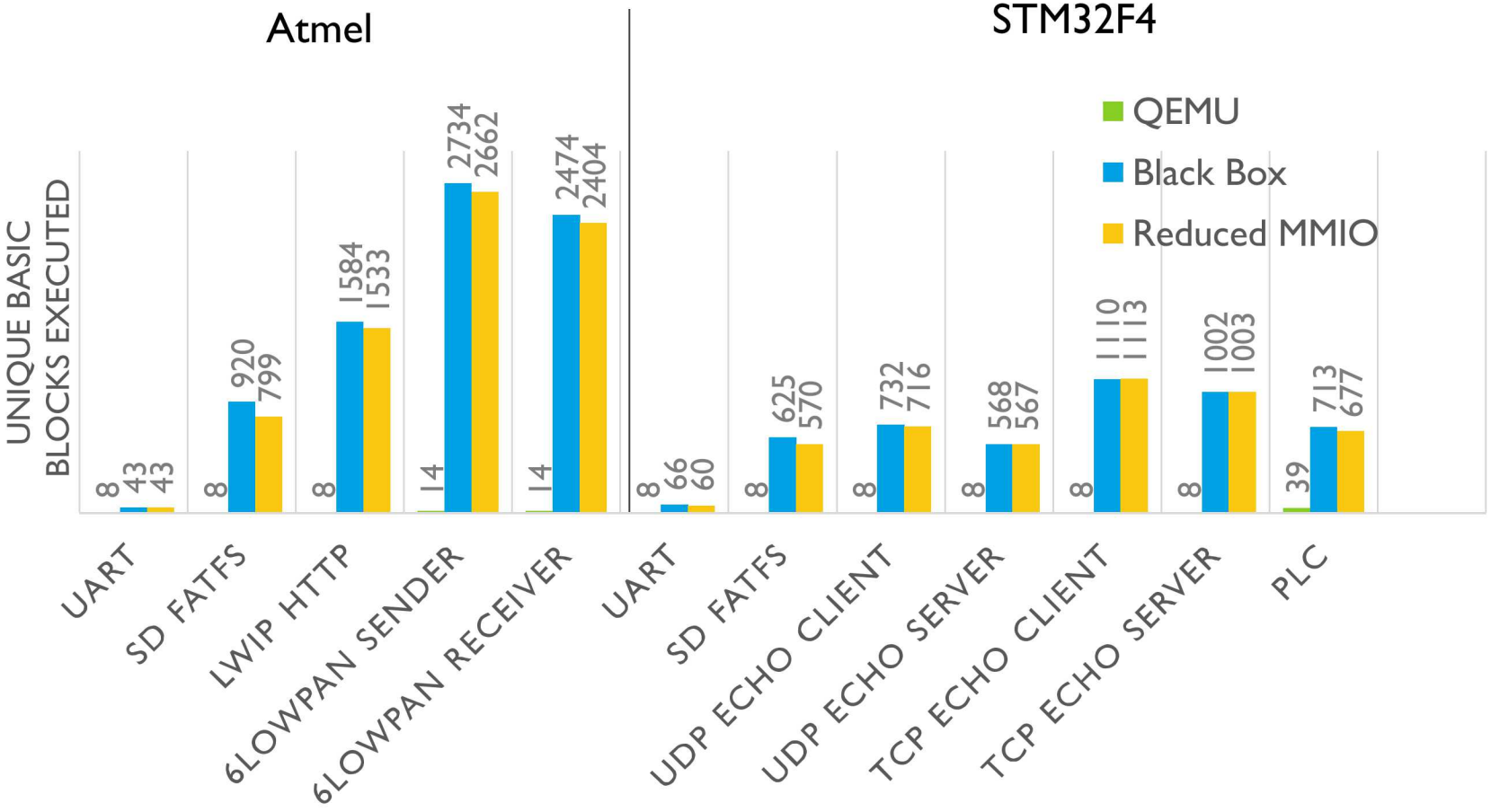
Functions and Average CC



FUNCTIONS INTERCEPTED AND MMIO



INTERACTIVE EMULATION



PERFORMANCE

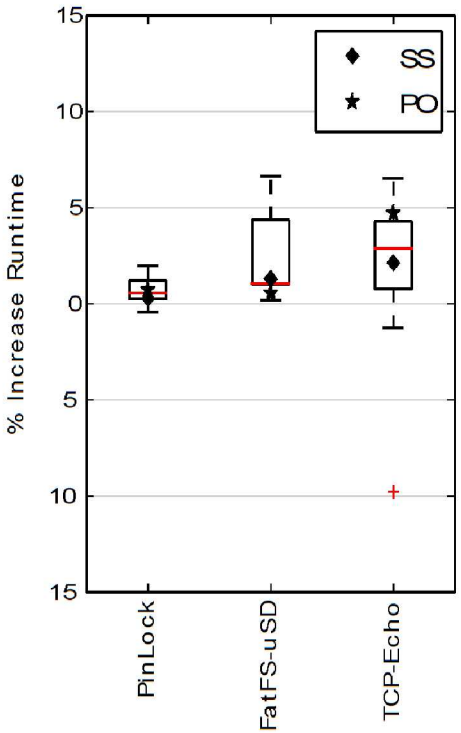
BEEBs Runtime

	SS	PO	All
Min	-7.3%	-1.3%	-11.7%
Ave	-3.5%	0.1%	**1.1%**
Max	4.4%	2.1%	14.2%

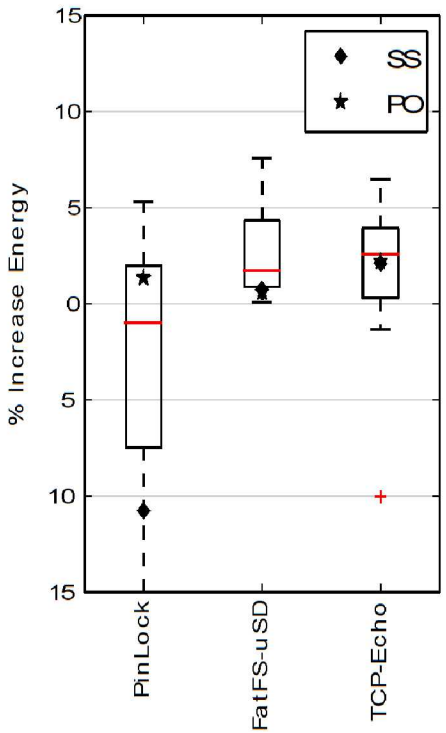
BEEBs Power

	SS	PO	All
Min	-4.2%	-10.3%	-10.2%
Ave	0.2%	-0.2%	**2.5%**
Max	7.3%	2.8%	17.9%

IoT Apps Runtime



IoT Apps Energy



SS - SafeStack Only, PO - Privilege Overlay Only