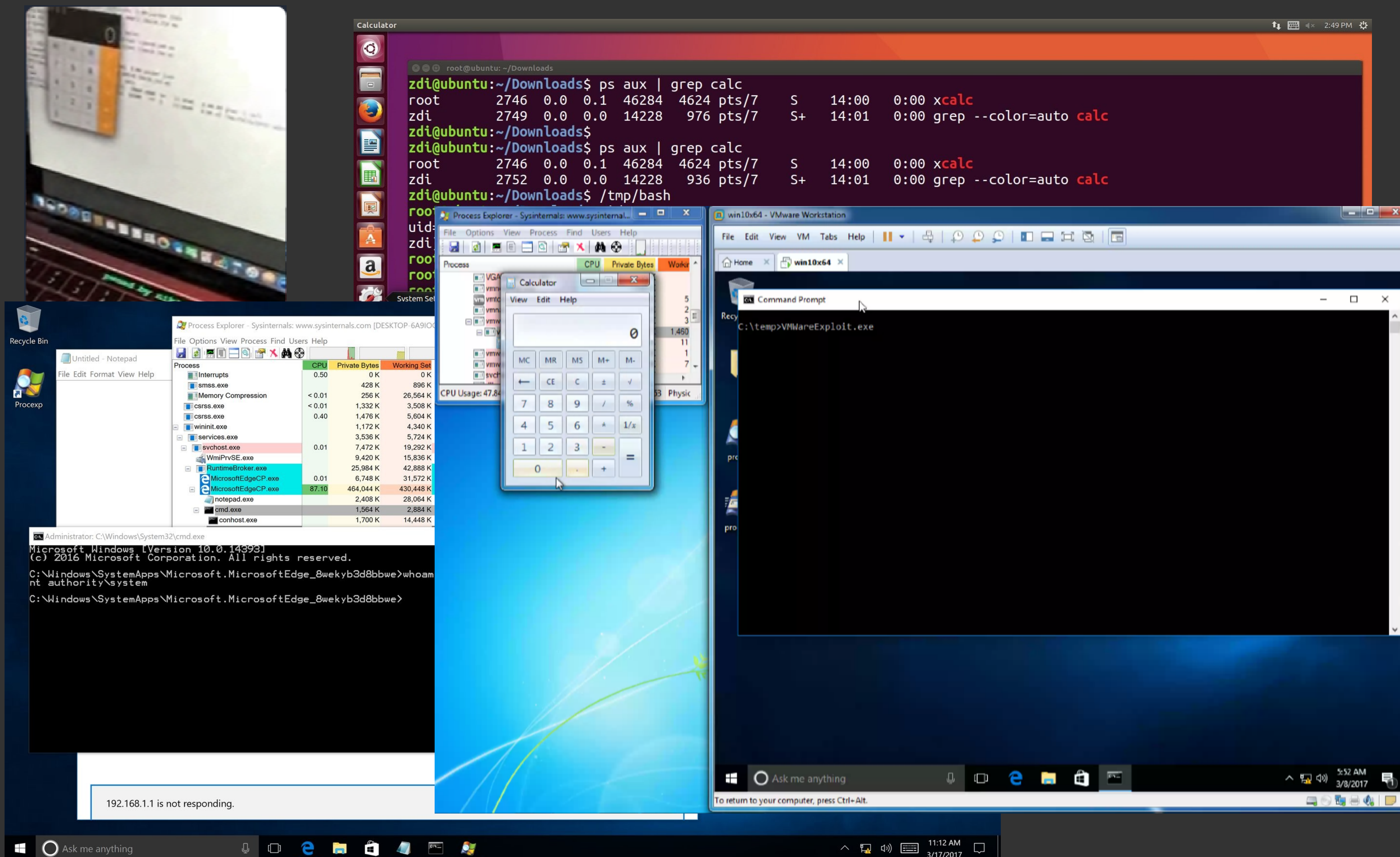
hexhive

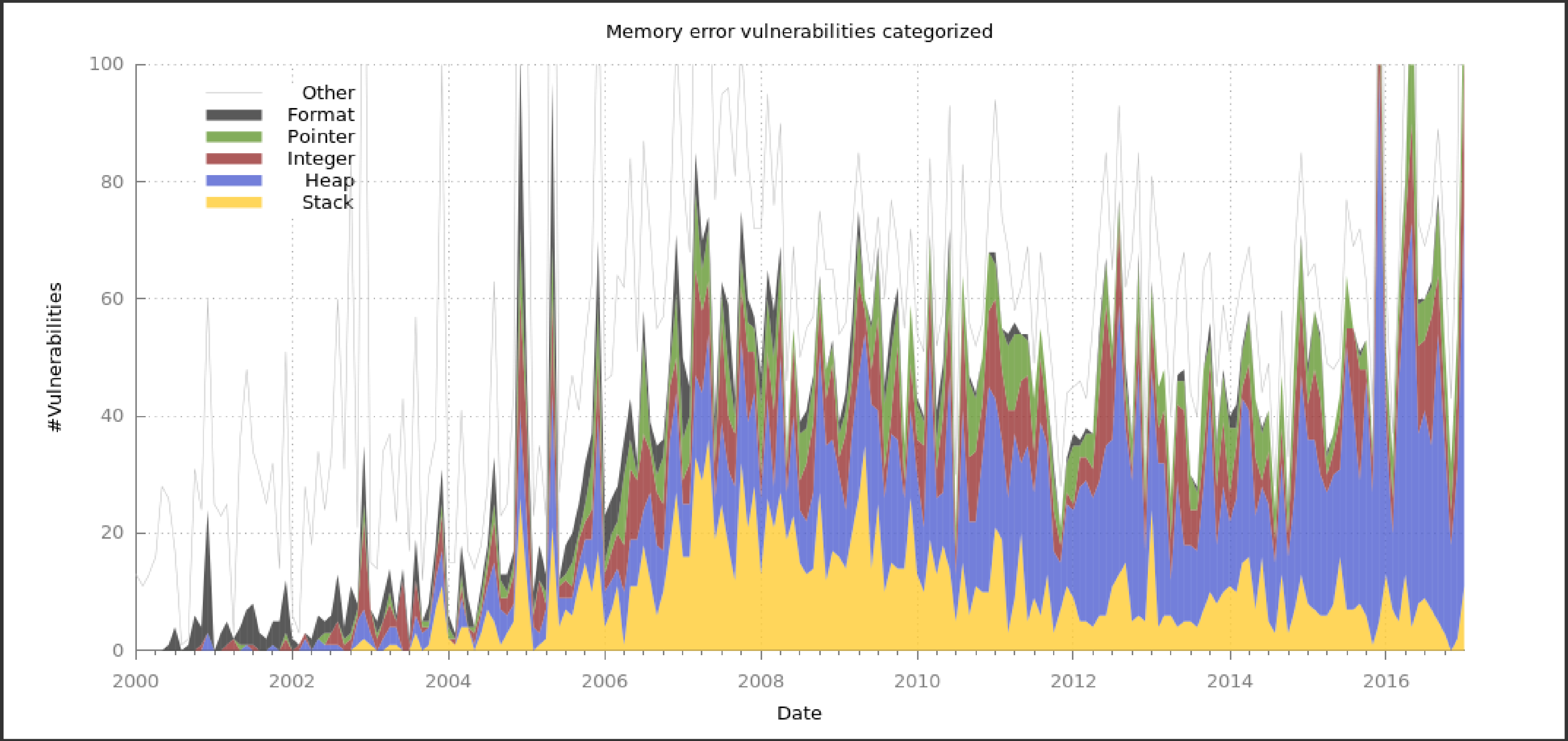
EPOXY: Shielding Bare-Metal Embedded Systems

Mathias Payer (@gannimo), Purdue University
Jointly with Abraham Clements and Saurabh Bagchi
http://hexhive.github.io

Bugs are everywhere?



Trends in Memory Errors*



* Victor van der Veen, https://www.vvdveen.com/memory-errors/, updated Feb. 2017

Software is unsafe and insecure*

- Low-level languages (C/C++) trade type safety and memory safety for performance
  - Our systems are implemented in C/C++
  - Too many bugs to find and fix manually

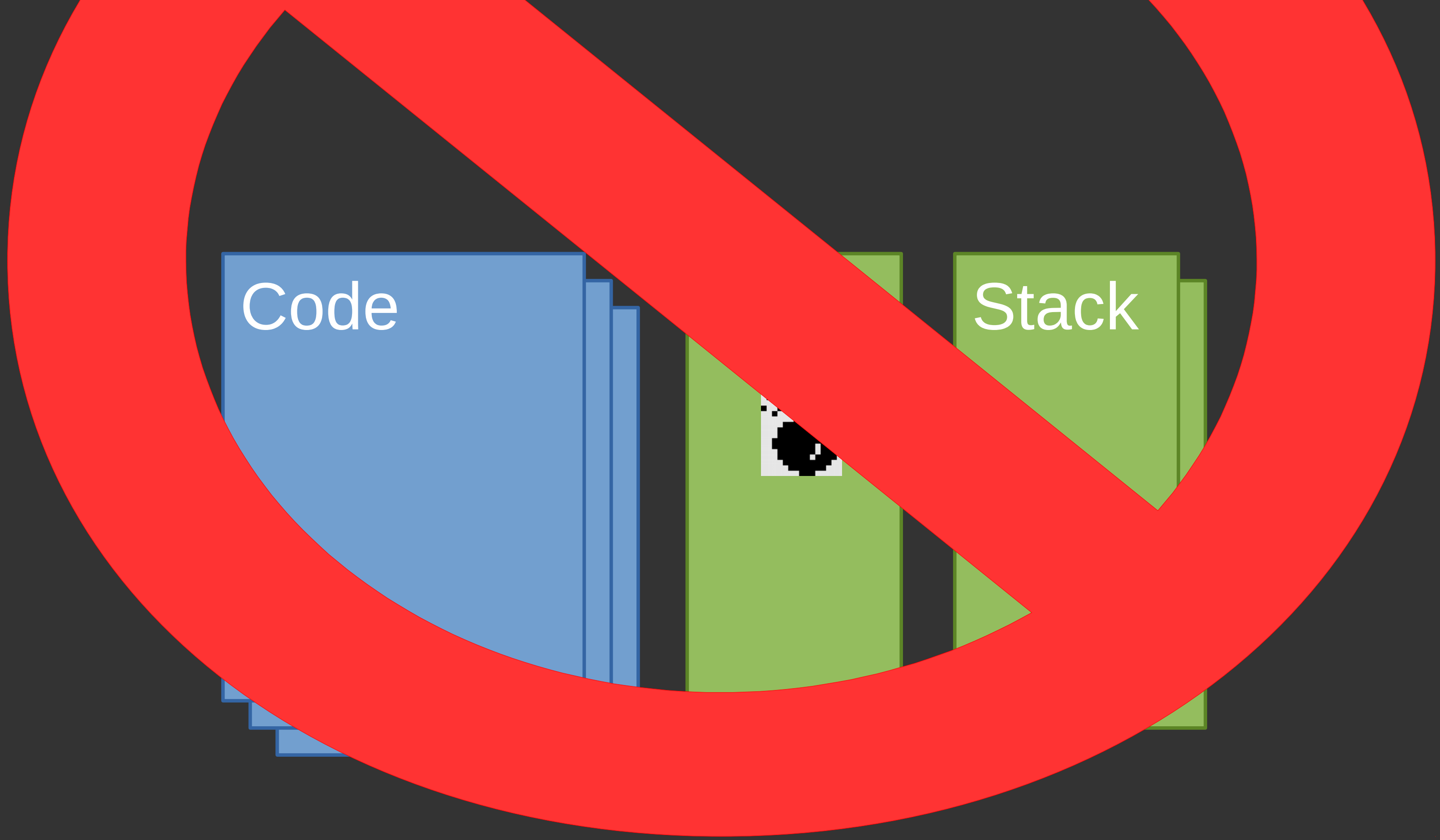
Google Chrome:	**76 MLoC**
glibc:	**2 MLoC**
Linux kernel:	**14 MLoC**

* SoK: Eternal War in Memory. Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. In IEEE S&P'13

Control-Flow Hijack Attack

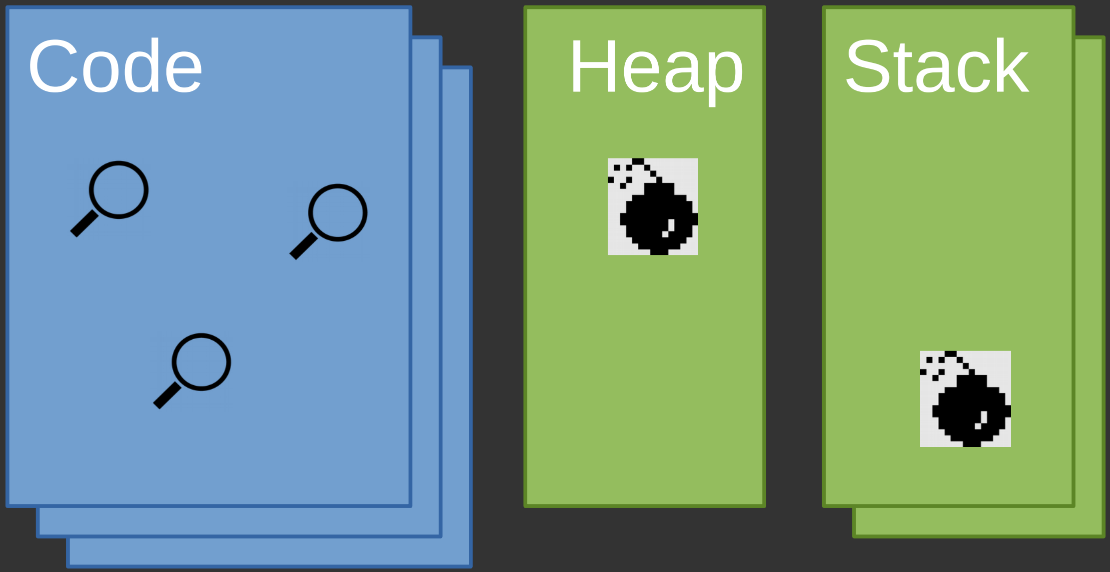
Attack scenario: code injection

- Force me ... uption ... attack
- Redir ... ol-flow to injected co



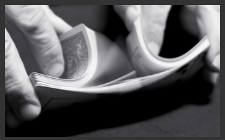
Attack scenario: code reuse

- Find addresses of gadgets
- Force memory corruption to set up attack
- Redirect control-flow to gadget chain



Defenses protect desktops/servers

- Address Space Layout Randomization



  - Shuffles address space, requires information leak

- Data Execution Prevention

  - Prohibits code injection, requires ROP

- Stack Canaries

  - Prohibits stack smashing, requires direct write

Control-Flow Integrity

Control-Flow Integrity (CFI)*

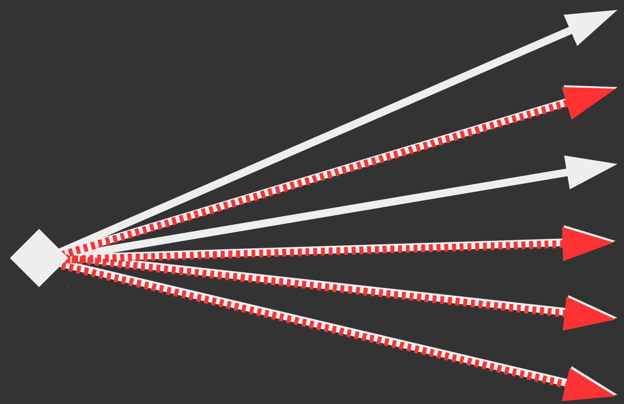
- Restrict a program's dynamic control-flow to the static control-flow graph
  - Requires static analysis
  - Dynamic enforcement mechanism

- Forward edge: virtual calls, function pointers
- Backward edge: function returns

* ***Control-Flow Integrity.*** Martin Abadi, Mihai Budiu, Ulfar Erlingsson, Jay Ligatti. CCS '05
**** Control-Flow Integrity: Protection, Security, and Performance.*** Nathan Burow, Scott A. Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, Mathias Payer. ACM CSUR '18, preprint: https://nebelwelt.net/publications/files/18CSUR.pdf

Control-Flow Integrity (CFI)

```
CHECK(fn);
(*fn)(x);
```



Attacker may corrupt memory, code ptrs. verified when used

CFI: limitations

- CFI provides incremental security
  - Attacker can choose between valid targets
  - Data-flow attacks are out of scope
- Strength of CFI depends on static analysis
  - Coarse-grained: all functions are allowed
  - Fine-grained: arity or function prototype

Are we making progress?

2007

2017

The State of the IoT

Defenses deployed on IoT devices

Bare-metal devices

(c) dekuNukem, hackaday.io


om

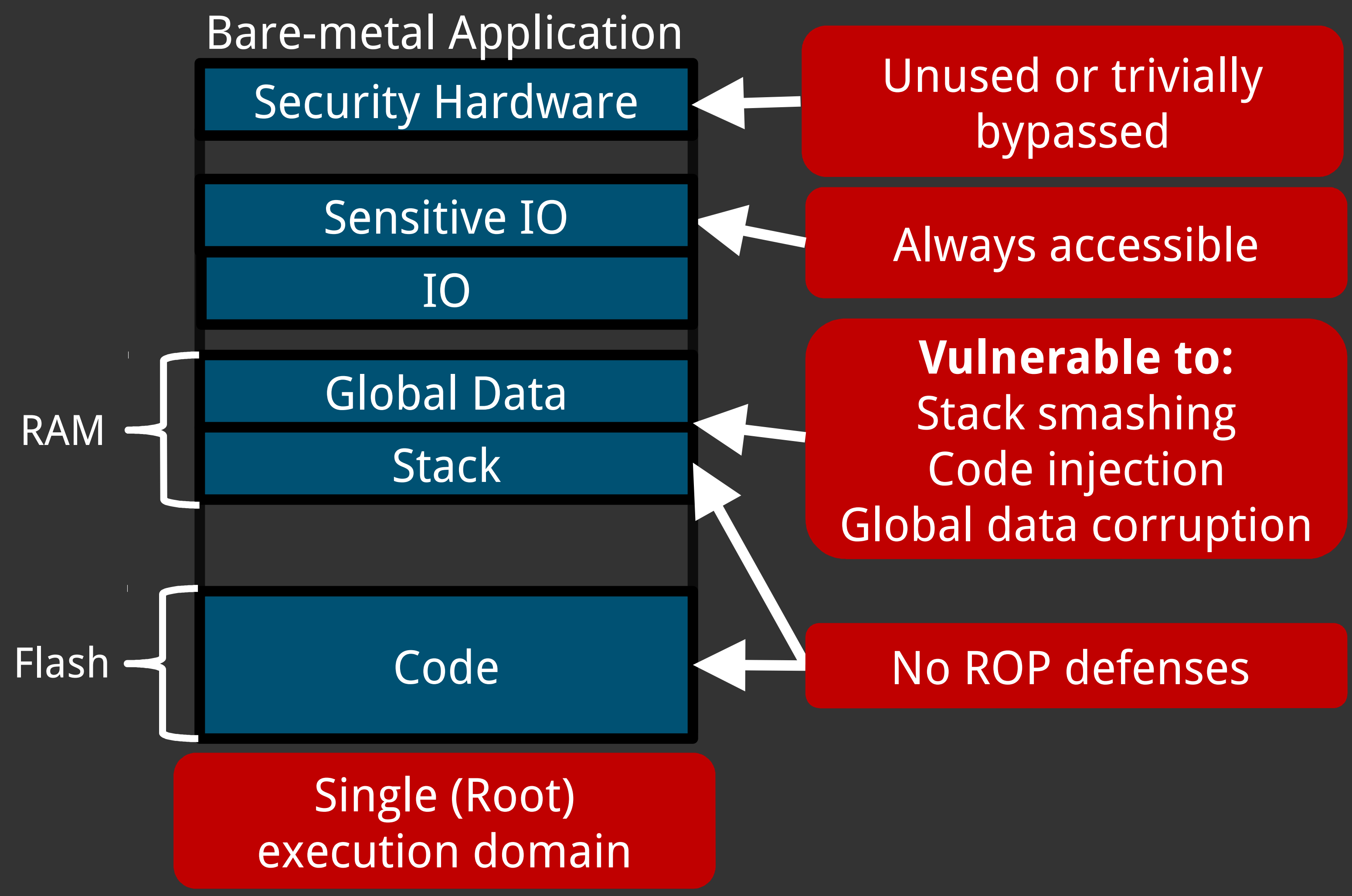
(c) yenra


(c) bunnie, bunniestudios.com

Security challenges

- Single application
  - No separate privilege levels (kernel/user)
- No MMU (virtual memory)
  - Defenses limited to physical memory space
- Tight constraints
  - Runtime, memory, battery

IoT security stack



Let’s exploit like in ‘99

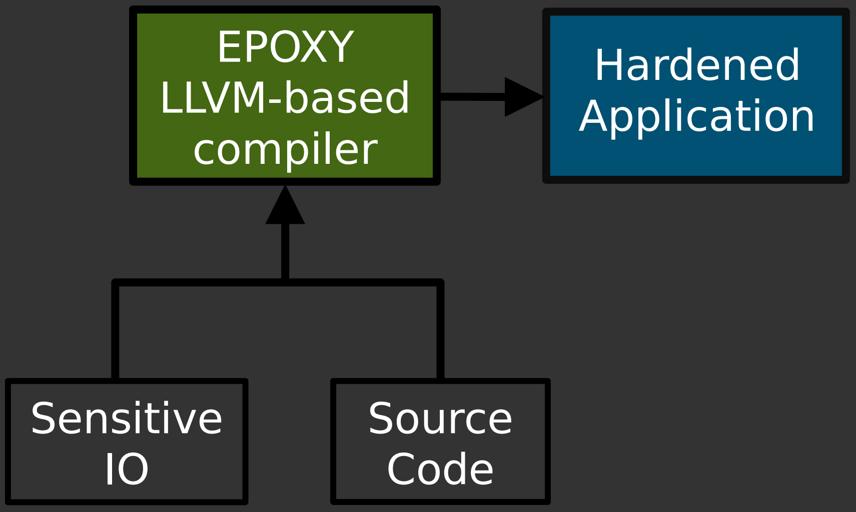
(c) MGM

EPOXY*

*** Embedded Privilege Overlay across (X) hardware for anY software**

EPOXY design

- LLVM-based compiler
- Protects against
  - Code injection
  - IO manipulation
  - Control-flow hijack*
  - Data corruption*



* Probabilistic, strength may vary (tm)

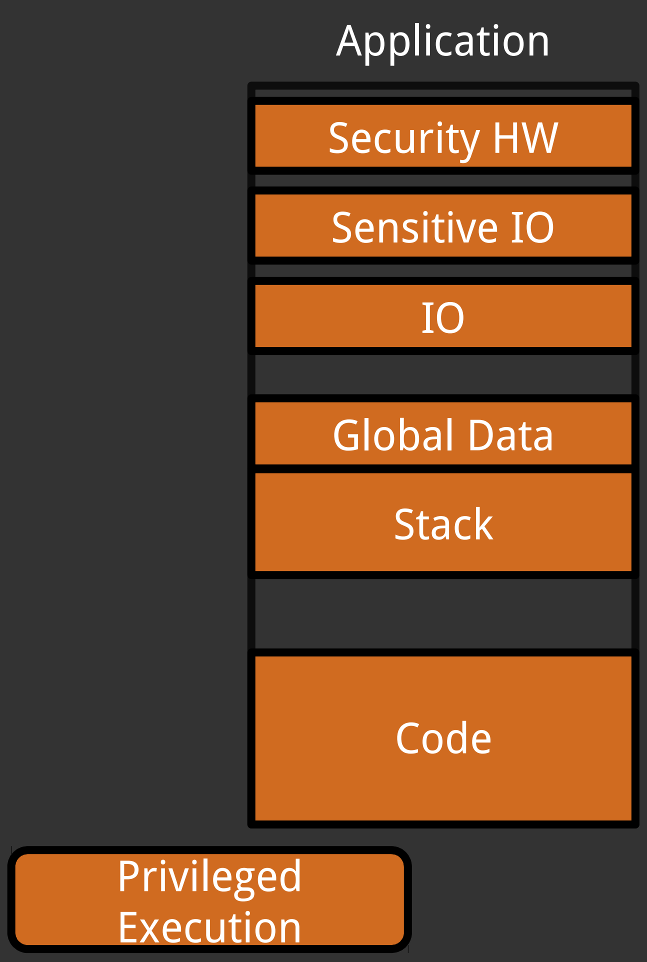
Embedded systems: opportunities

- No separation between “apps” or user/kernel
  - Only few instructions require privileges
- Small memory size: MBs of Flash, KBs of RAM
  - Memory is dedicated, may reuse all slack space
- Tight runtime constraints
  - Execution is interrupt driven, use slack
- Low power requirements
  - Limit overhead to few instructions

Mission 1: privilege separation

(c) AMC, Walking Dead

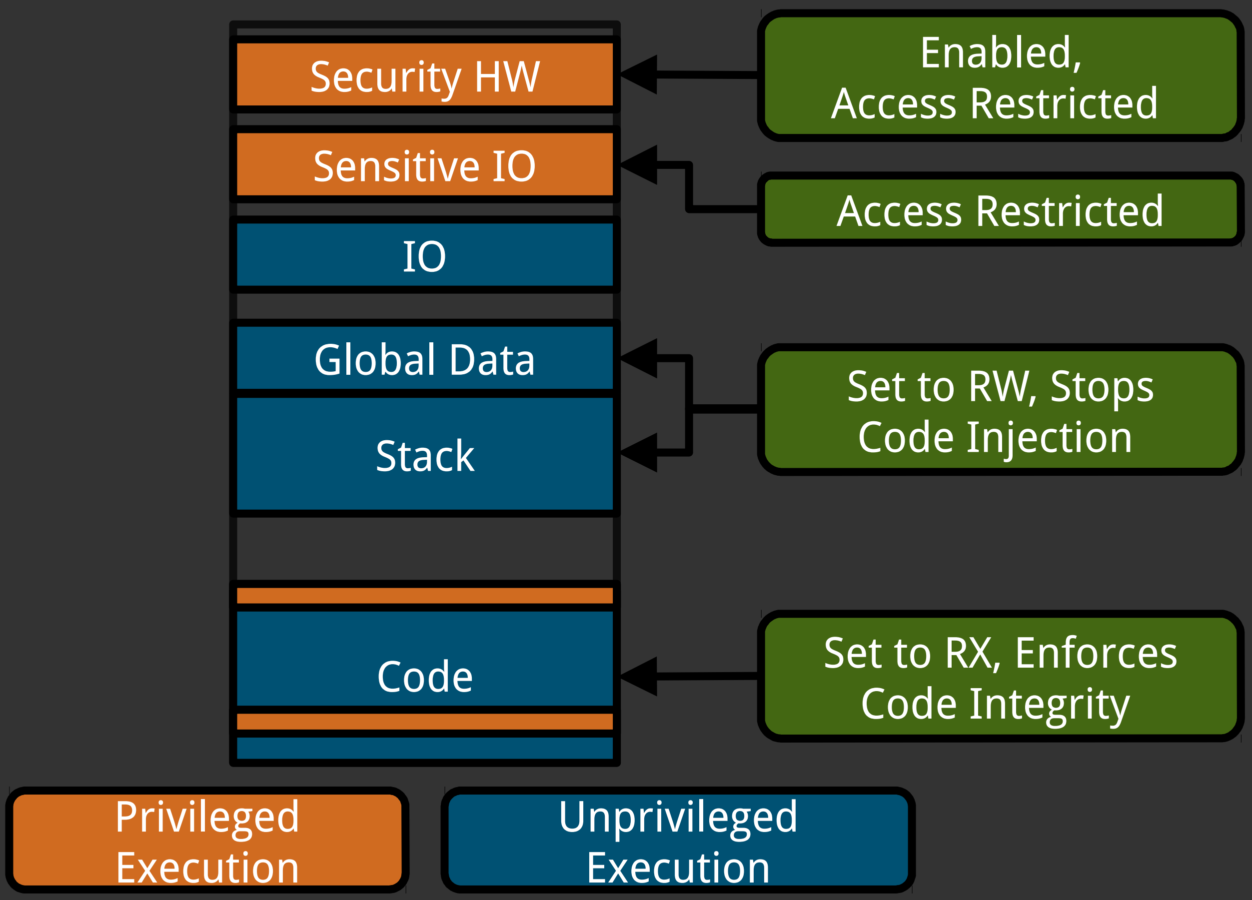
Before EPOXY



Privilege separation

- Static analysis identifies restricted operations
  - Specific instructions per ISA
  - Sensitive memory-mapped registers (MPU, IO)
- Instrumentation to
  - Configure MPU to drop privileges
  - Raise privileges selectively
- Enable security hardware
  - Enforce W^X code, RW data
  - Protect access to security hardware, I/O

Privilege overlay: benefits



Evaluation: privileged instructions

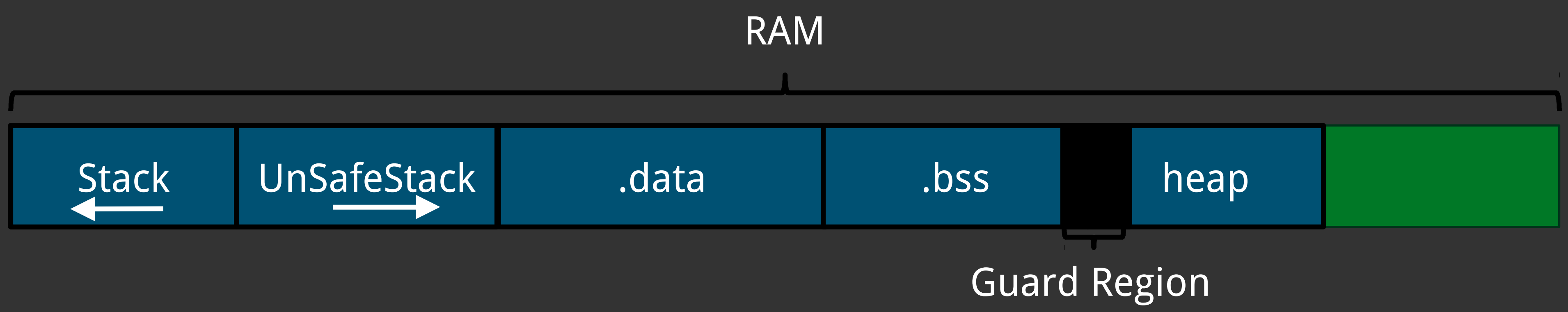
Application	Tool	Exe	Priv	Priv %
PinLock	EPOXY	823K	1.4K	0.17%
	FreeRTOS-MPU	823K	813K	98.78%
FatFS-uSD	EPOXY	33.3M	3.9K	0.01%
	FreeRTOS-MPU	34.1M	33.0M	96.77%
TCP-Echo	EPOXY	310M	1.5K	<0.001%
	FreeRTOS-MPU	322M	307.0M	95.34%

Mission 2: stop stack smashing

(c) Nintendo

Stack integrity through SafeStack

- Split stack into safe stack and unsafe stack*
- Move unsafe objects to unsafe stack
- Protects against stack smashing

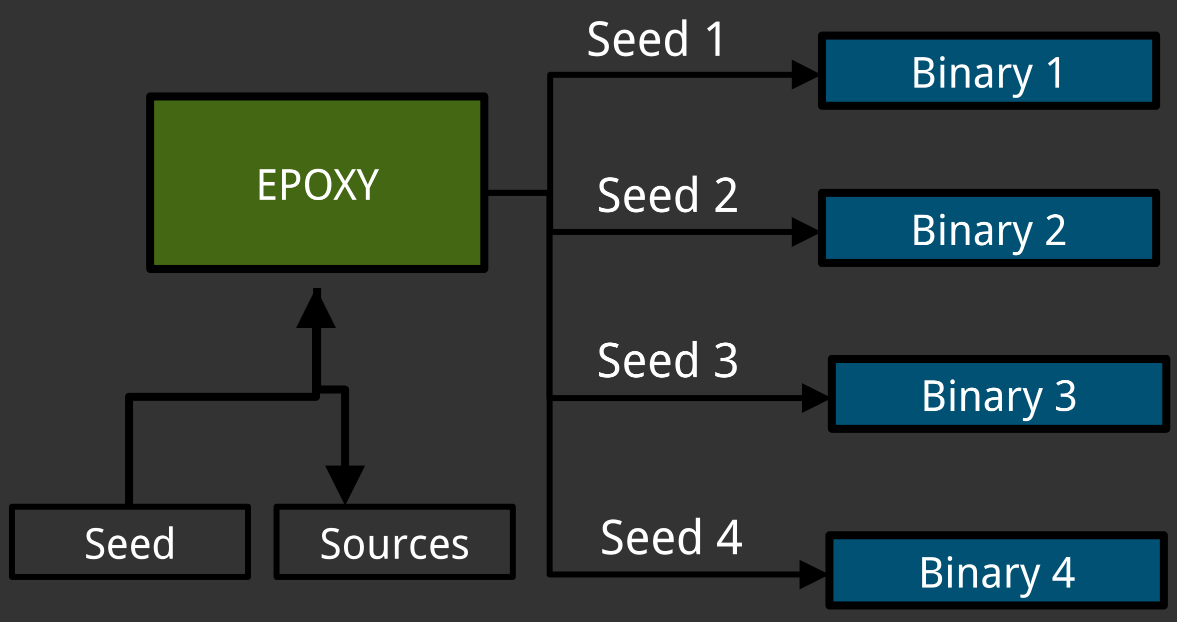


* V. Kuznetsov *et al.*, Code Pointer Integrity, OSDI 2014

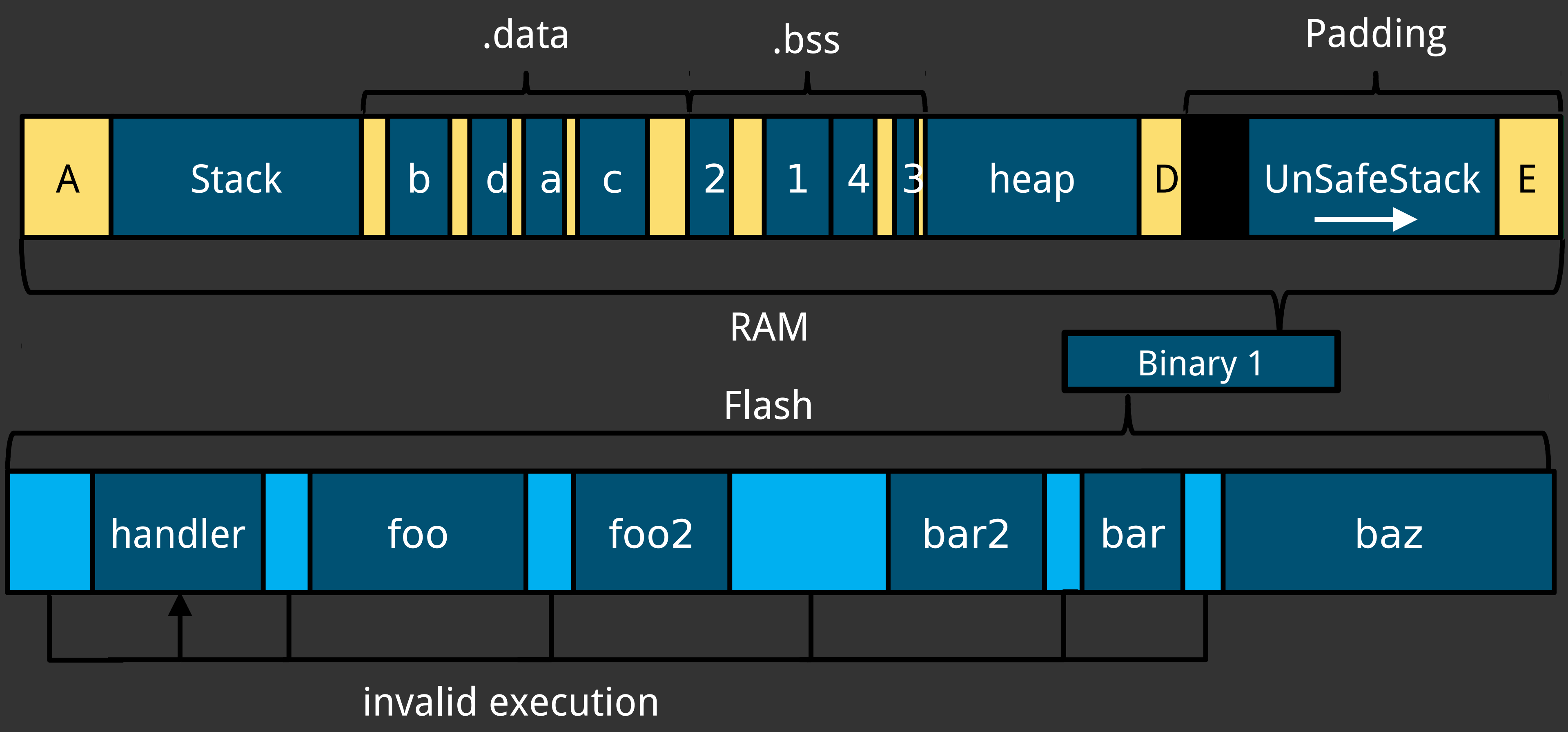
Mission 3: shuffle

Diversification

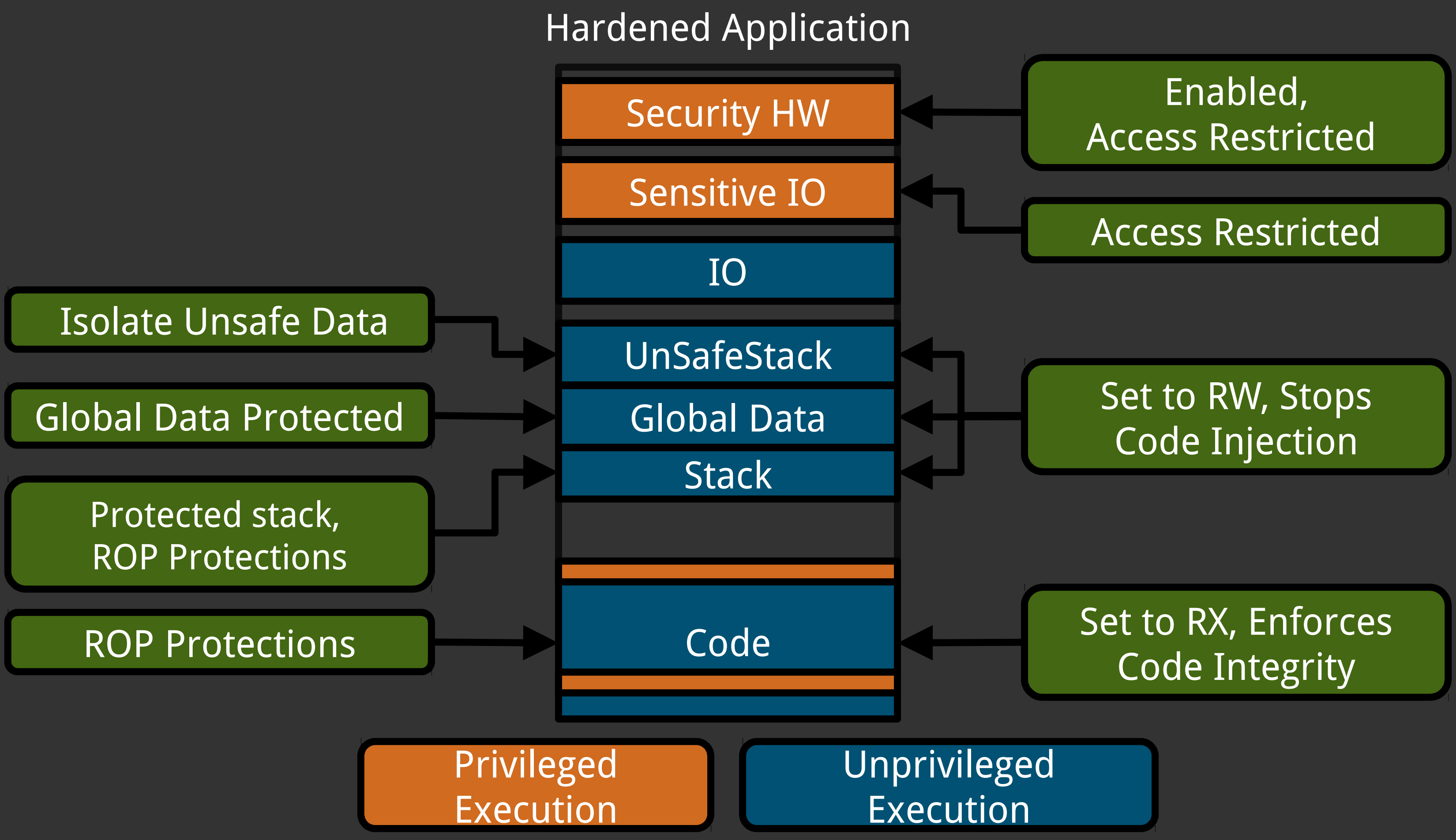
- Shuffle globals, stack, and code
  - Protects against ROP
  - Protects against global data corruption



Diversification



EPOXY: full feature set



Evaluation: ROP gadgets

		# Surviving Across				
App	**Total**	**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

Using ROPgadget compiler to identify surviving gadgets across # diversified binaries

Performance impact (BEEP)

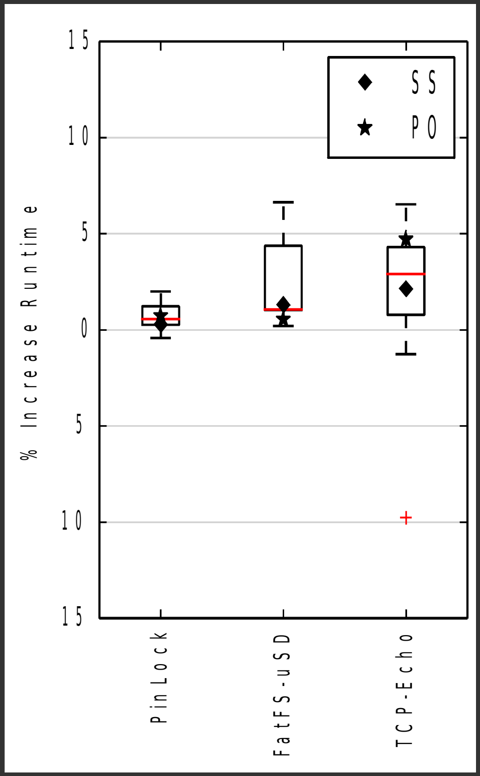
Runtime	SS	PO	All
Min	-7.3%	-1.3%	-11.7%
Ave	-3.5%	0.1%	1.1%
Max	4.4%	2.1%	14.2%

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

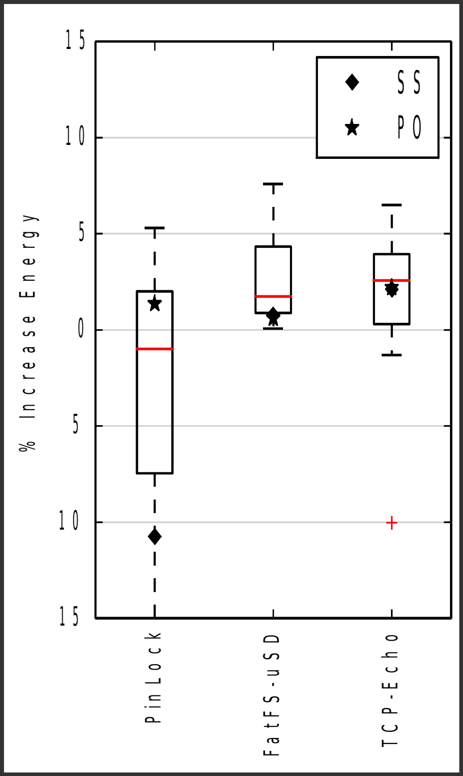
SS: SafeStack, PO: Privilege Overlay

Performance impact

IoT Apps Runtime



IoT Apps Energy



Conclusion

Conclusion

- Embedded systems need protection
  - Currently no defenses, easy target
- Fast forward embedded security by 3 decades
  - Privilege separation, mitigate code injection
  - Safe stack protects against stack smashing
  - Diversification instead of ASLR
- Meets runtime, memory, energy requirements

Source: https://github.com/HexHive/EPOXY hexhive

Thank you!

Questions?

Mathias Payer (@gannimo), Purdue University
http://hexhive.github.io