Morpheus

A Vulnerability-Tolerant Secure Architecture

Mark Gallagher, Lauren Biernacki, Shibo Chen, Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Austin Harris, Zhixing Xu, Baris Kasikci, Valeria Bertacco, Sharad Malik, Mohit Tiwari, Todd Austin

Presented by Miles Dai

6.888, Fall 2020

Motivation

A vulnerability-agnostic secure system can be created by defending against exploitation of undefined semantics.

- Control flow exploits are still prevalent!
- New threats use more execution-level information to bypass defenses
- A systematic approach is needed to future-proof processors

Intuition

- We can think of program execution as occurring on multiple levels
  - Language level: we create some pointer to some memory
  - Execution level: where is that memory located? How do I dereference the pointer? What is the memory initialized to? Where is the stack?
- Moving target defenses: since benign programs are (mostly) agnostic to execution-level details, what if we randomize them?
- Strengthen existing moving target defenses through layering defenses and continuous randomization.
  - Ensembles of Moving Target Defenses (EMTD)
  - Churn

Threat Model

- A trusted but vulnerable victim processes untrusted inputs
- Trusted
  - Physical system
  - Boot sequence
  - Random number generator
  - Morpheus hardware
  - Loader and OS scheduler
- Attacker exploits memory errors to hijack control flow
- Currently does not protect against DoS and side-channel attacks

Discussion

What are some strengths and weaknesses of Morpheus?

Evaluation

Strengths

- Systematic approach to memory safety
- Low execution and adoption overhead

Weaknesses

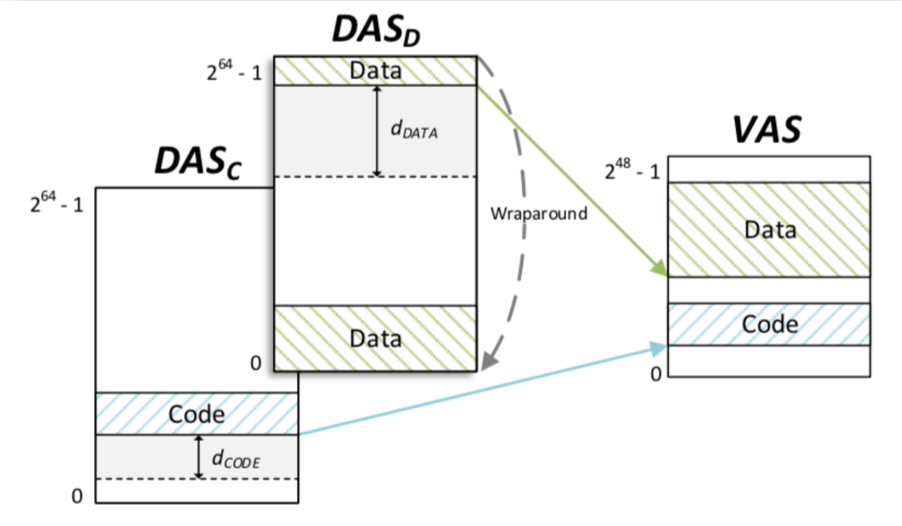
- More discussion on tag propagation and attack detector logic: implementation and area overheads
- Application to non-64-bit architectures might have reduced security guarantees

Domain Tagging

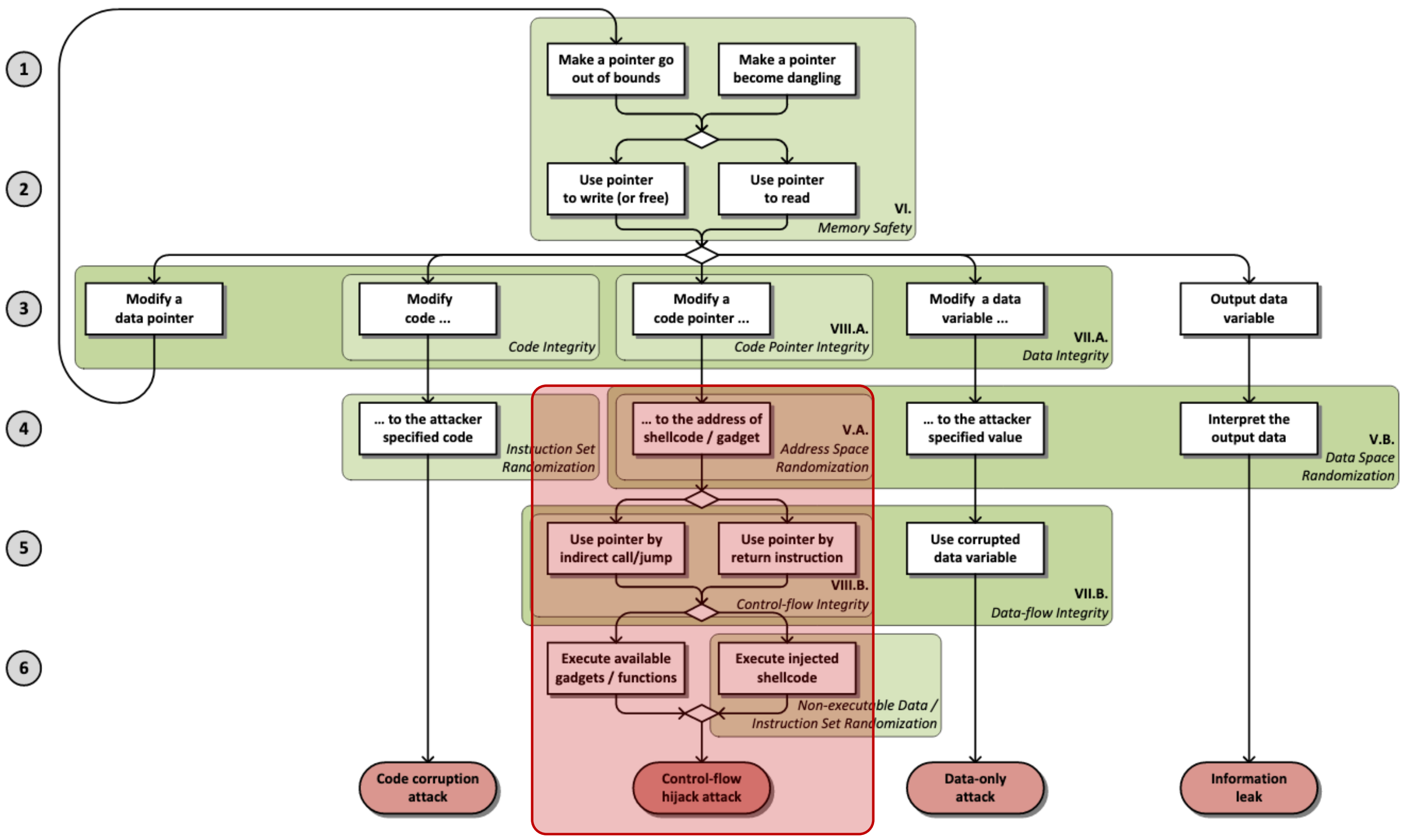
- Memory falls into 4 Domains: code (C), code pointers (CP), data pointers (DP), data (D)
- Compiler tags memory objects in two passes
- Microarchitectural support
  - Each register gets two additional bits
  - All tag information sits together in DRAM and are cached
  - Pipeline propagates tags
- Domain tagging allows for moving target defenses (MTDs)

Pointer Displacement – MTD 1

- Present a Displaced Address Space (DAS) offset by up to 2^{60} to the program
- Code and Data segments receive different offsets

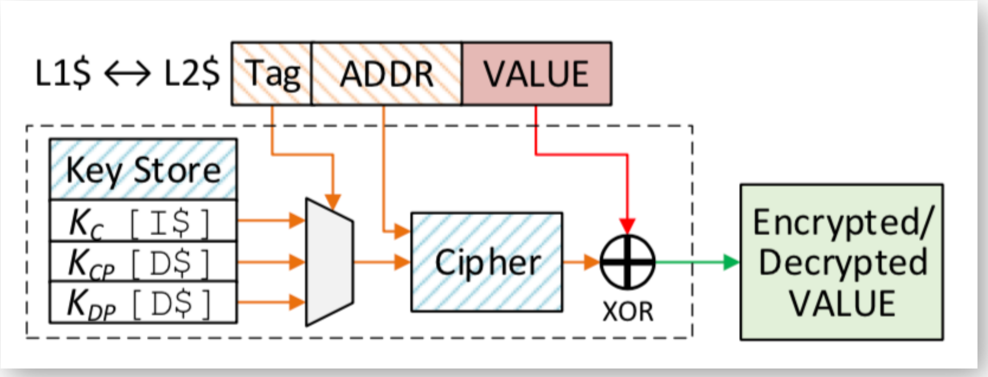


Pointer Displacement Defense

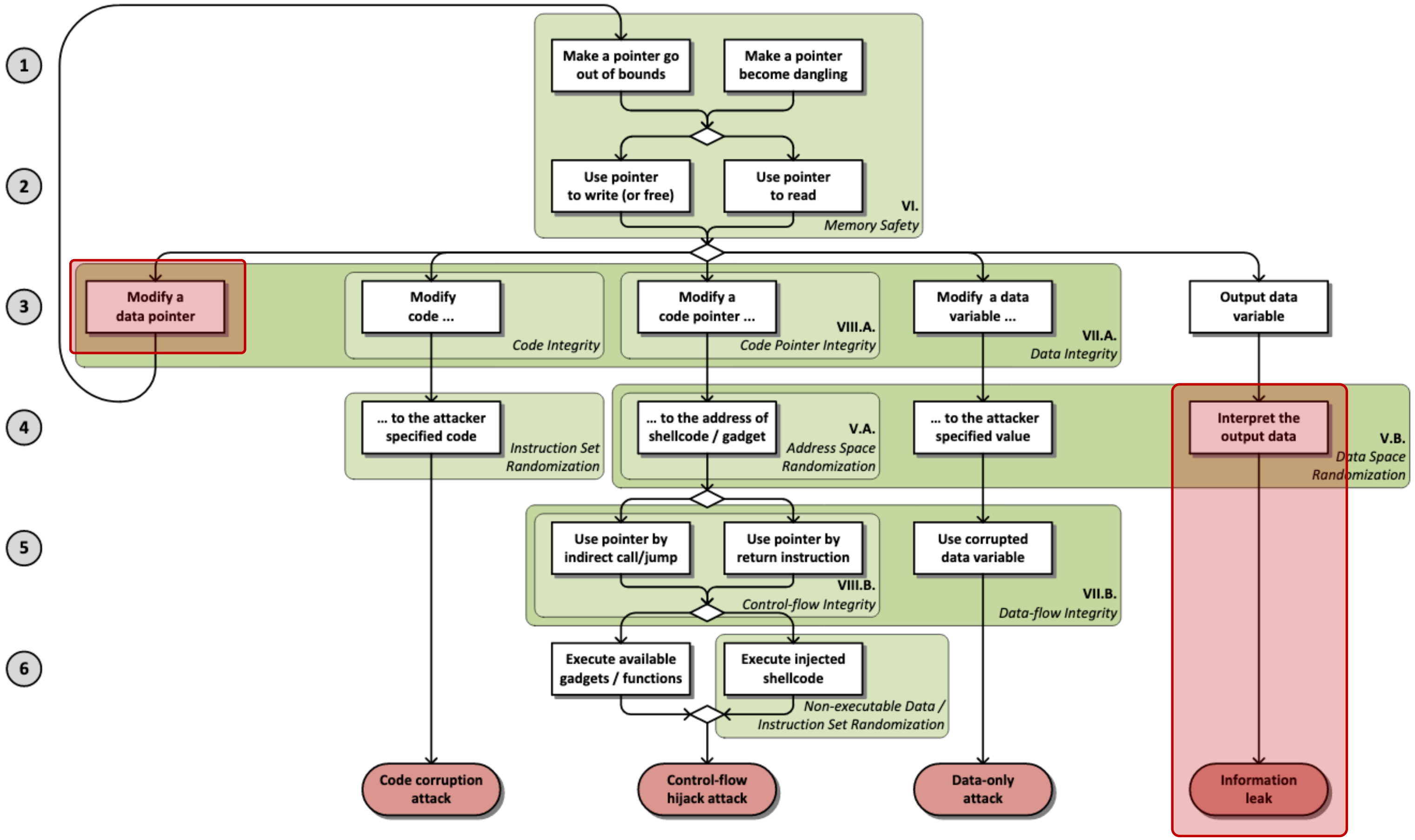


Domain Encryption – MTD 2

- Code, code pointers, and data pointers are encrypted with distinct keys. Variable-sized non-pointer data values are not encrypted.
- Data is encrypted/decrypted on the L1-L2 boundary; L2 and DRAM only contain encrypted information
- Physical address is encrypted with corresponding key and XOR'ed with value

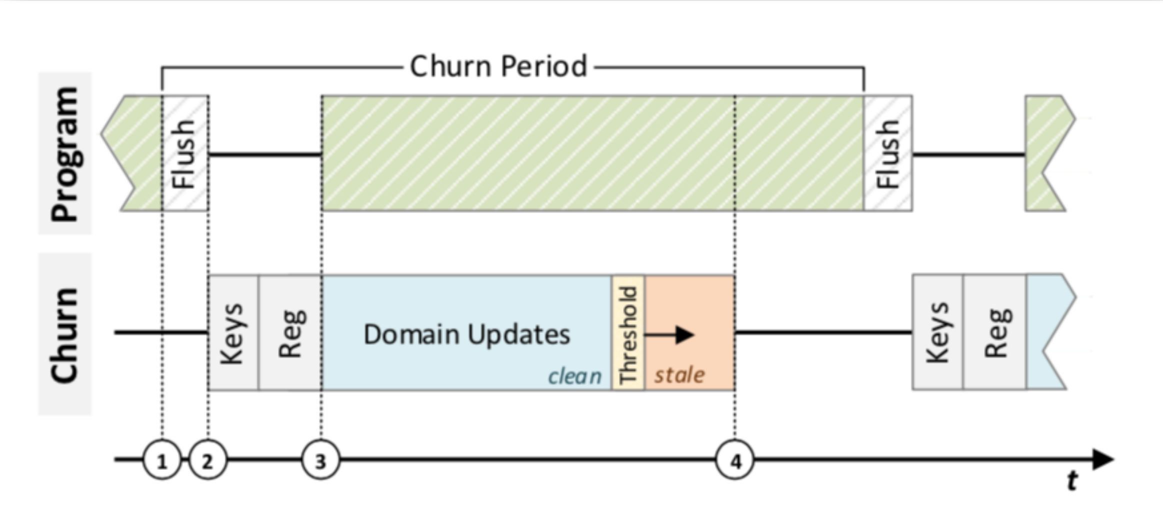


Domain Encryption Defense



Churn

- Creates new offset for data and code segments
- Re-encrypts all encrypted data with new keys
- Steps: Pipeline flush, key generation, register updates, memory update using threshold register

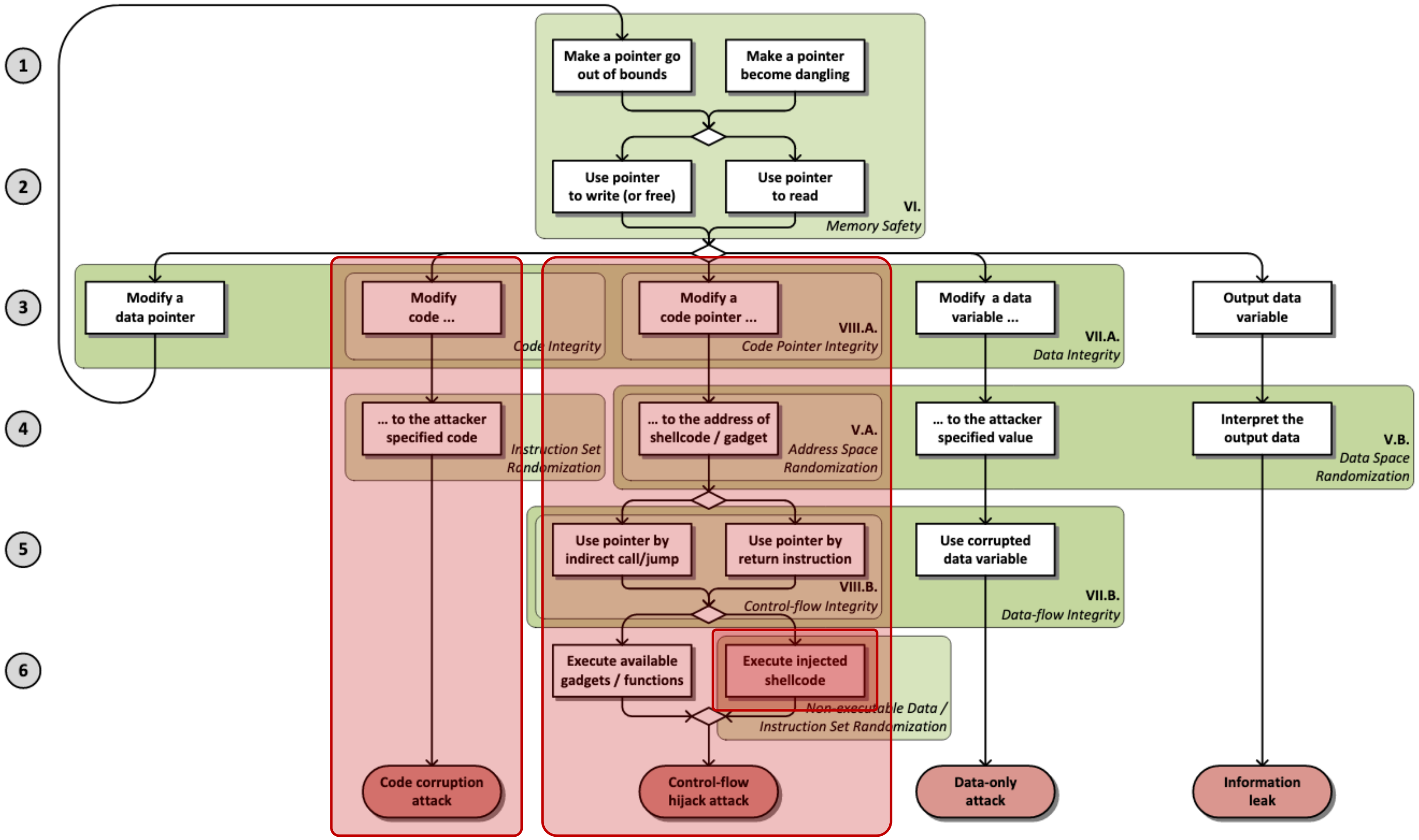


Attack Detector

- Domain tagging allows for policy enforcement
- Program can be ABORTed or suspicious behavior can trigger CHURN

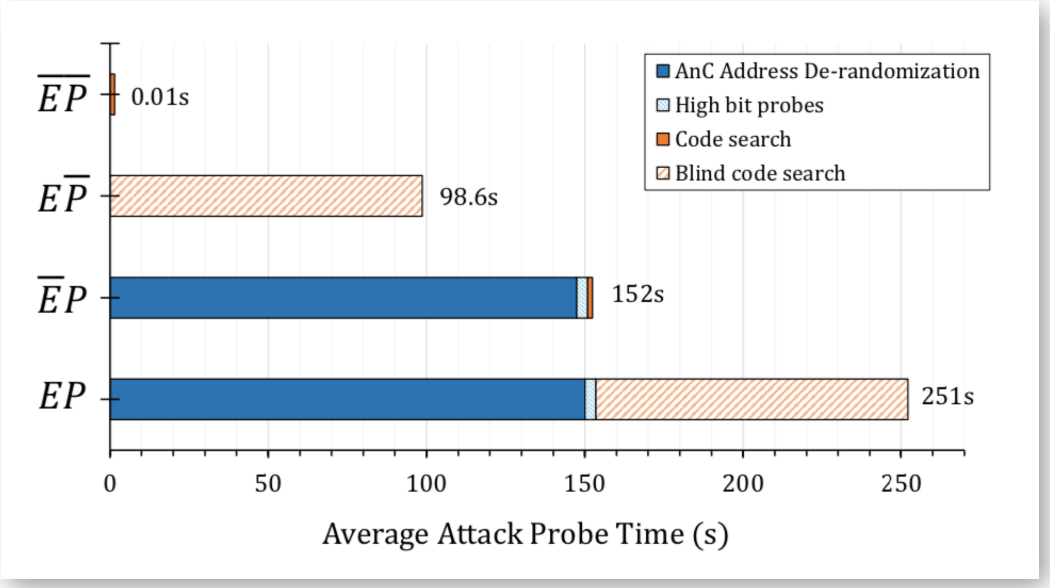
	<OP>	Check Condition	Rule
ABORT	Execute	Insn.tag != C	Only execute C
	ANY	R1/R2.tag == C	No C in the pipeline
	JAL(R)	R1.tag != CP	Jump target must be CP
	LD/ST	R1.tag != DP	Address must be a DP
CHURN	COMPARE	R1.tag != R2.tag	No inter-domain compares
	ANY (not JAL(R))	R1.tag == CP	CP arithmetic suspicious
	ANY (not LD/ST)	R2.tag == DP	DP arithmetic suspicious, except add/sub D
	ANY	Overflow Occurs	Overflows are undefined
	SHIFT	Shift > RegWidth	Invalid shift is undefined

Attack Detector Defense



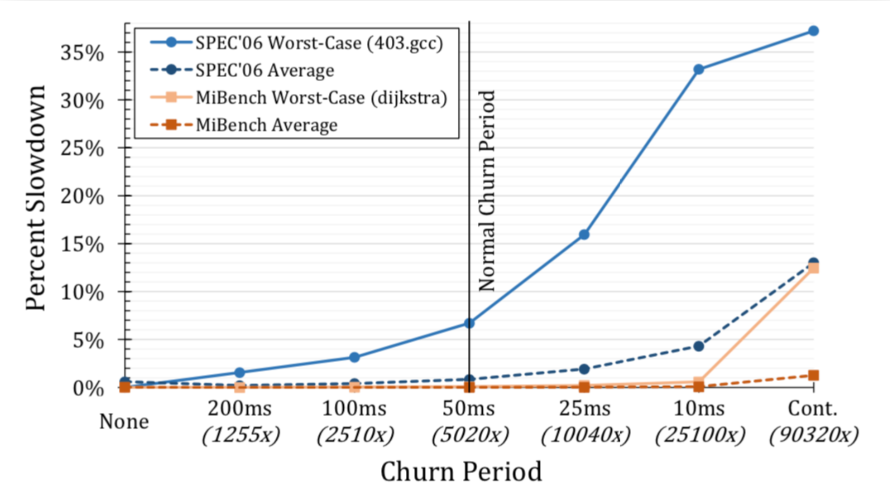
Evaluation – EMTD Effectiveness

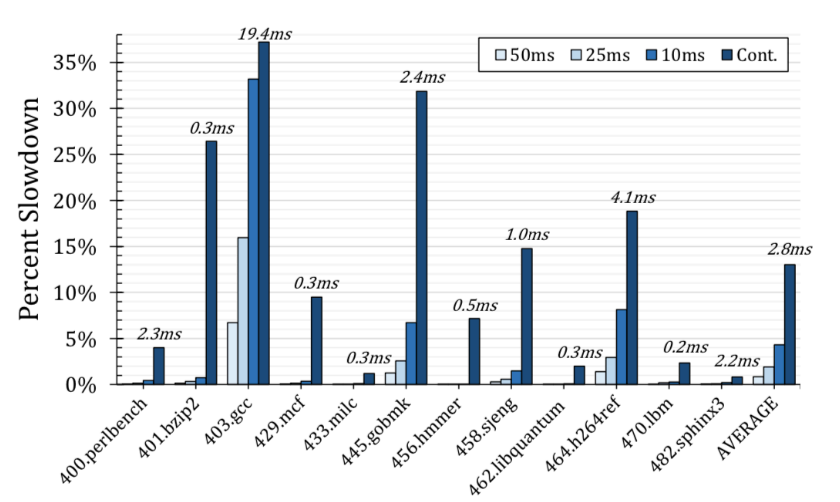
- Stacking defenses (ensemble) has clear benefits
- Attack probe time increases with more defenses applied
  - E = encryption, P = pointer displacement



Performance Impact of Churn

- Large data segments, more pointers, and large codebases cause more work for the churn unit
- For long churn periods (200 ms), churn very slightly improves performance as it acts as a prefetcher





Evaluation – Adoptability

Based on the criteria outlined in the SoK paper[1]:

- Performance overhead
- Compatibility
  - Software toolchain based on LLVM compiler extensions
  - Displacement preserves physical memory alignment
  - Extensive hardware modifications needed
- Robustness
  - More robust than many existing solutions to currently unknown attacks
- Dependencies
  - Toolchain does not appear to be publicly available currently

[1] SoK: Eternal War in Memory. Laszlo Szekeres et al.

Comparison with Existing Solutions

- Displacement (e.g. ASLR)
  - Insufficient randomness
  - Single address leakage discloses all code and data locations
- Encryption
  - Morpheus generally has lower overhead with HW support and stronger encryption
- Software-based MTD (e.g. Shuffler)
  - Morpheus shows lower overhead and more entropy
- Tagged Architectures
  - Full labeling of code is hard and other hardware-based tags lead to high false-positives

Discussion Questions - Security

- Does the fact that pointers are all linearly displaced by a constant amount (rather than being truly shuffled) make this scheme vulnerable to attack?
- Is it possible to avoid triggering churn by exfiltrating data through side channels?
- How could Morpheus be extended to consider DoS attacks in its threat model?

Discussion Questions - Applications

- How does "data" become code safely? (In the sense of a downloaded program being executed for the first time, or really anything being loaded from disk, or JIT programs)
- Are there legitimate uses of reading pointers as data that Morpheus will make impossible? E.g. debugging with stack traces will be very difficult though potentially possible.

Discussion Questions - Performance

- Can continuous churn cause performance issues or battery life reduction (denial of service rather than control flow attack)? Is it just exchanging one attack for another?
- Are there legitimate programs that Morpheus hinders?