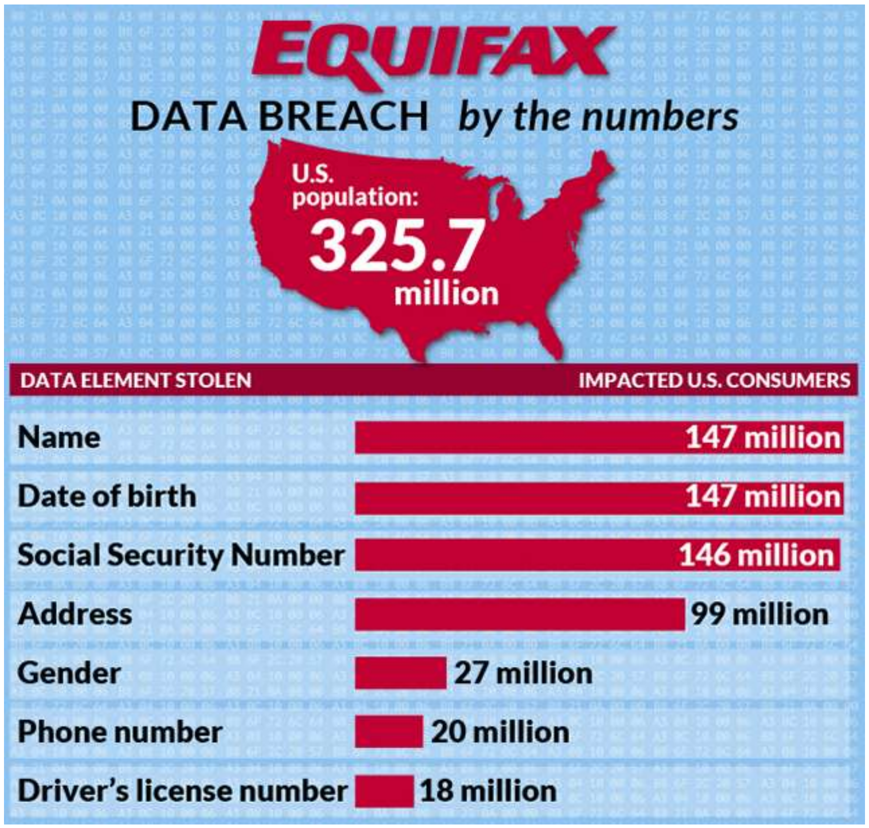
Morpheus II: A RISC-V Security Extension for Protecting Vulnerable Software and Hardware

Todd Austin
University of Michigan / Agita Labs
austin@umich.edu

Joint work with:
Austin Harris (UT), Tarunesh Verma (UM), Shijia Wei (UT),
Lauren Biernacki (UM), Alex Kisil (Agita Labs), Misiker Aga (Agita Labs),
Valeria Bertacco (UM), Baris Kasikci (UM), Mohit Tiwari (UM)

Assessing the (Dour) State of Today's Security Defenses

Who Can We Trust? Attackers Within and Without

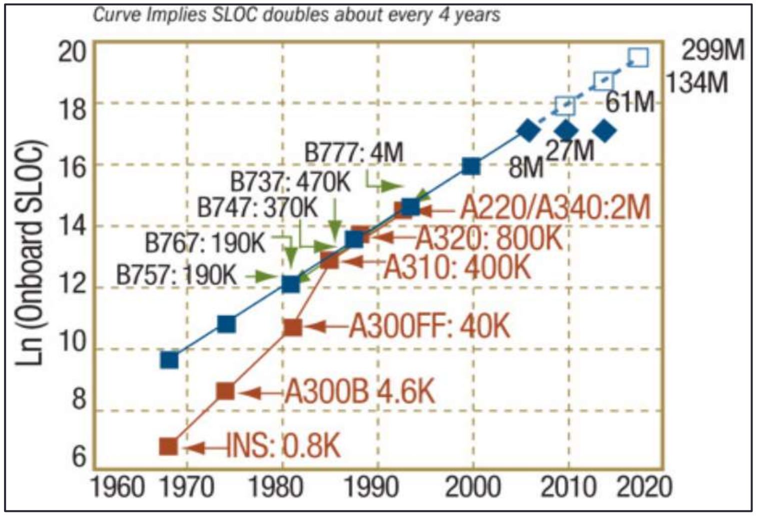


facebook

Cambridge Analytica

Because Here There Be Two Powerful Dragons

- Software protects data
  - All software is (eventually) hackable
  - Finding/fixing vulnerabilities doesn't scale
  - E.g., Malicious 7: buffer errors, code injection, numeric errors, permissions, resource mgt
- Side channels abound
  - Control, memory, timing, cache, speculative
  - Performance-centric design creates side channels
  - E.g., Malicious 7: crypto errors, information leakage, resource mgt



Assessing Today's Security Capabilities

- What we do well:
  - Finding and fixing vulnerabilities
  - Deploying system protections that stop well-known attacks
- Where we fail: ***identifying and stopping emergent attacks***

Valgrind

Synopsys' Coverity Tools

ARM's TrustZone

Intel's Control-Flow Enforcement

Can hardware security defenses be built to be more durable?

Morpheus’ Unique Approach to Security

Vulnerabilities + Implementation Assets = Exploit

Attack Detector

- Buffer overflow
- Code pointer arithmetic
- Data pointer logical operation
- Code forgery
- Pointer forgery
- Uninitialized variable access
- Mem permission violation
- Integer overflow
- Shift overflow
- Code read
- Cyclic interference

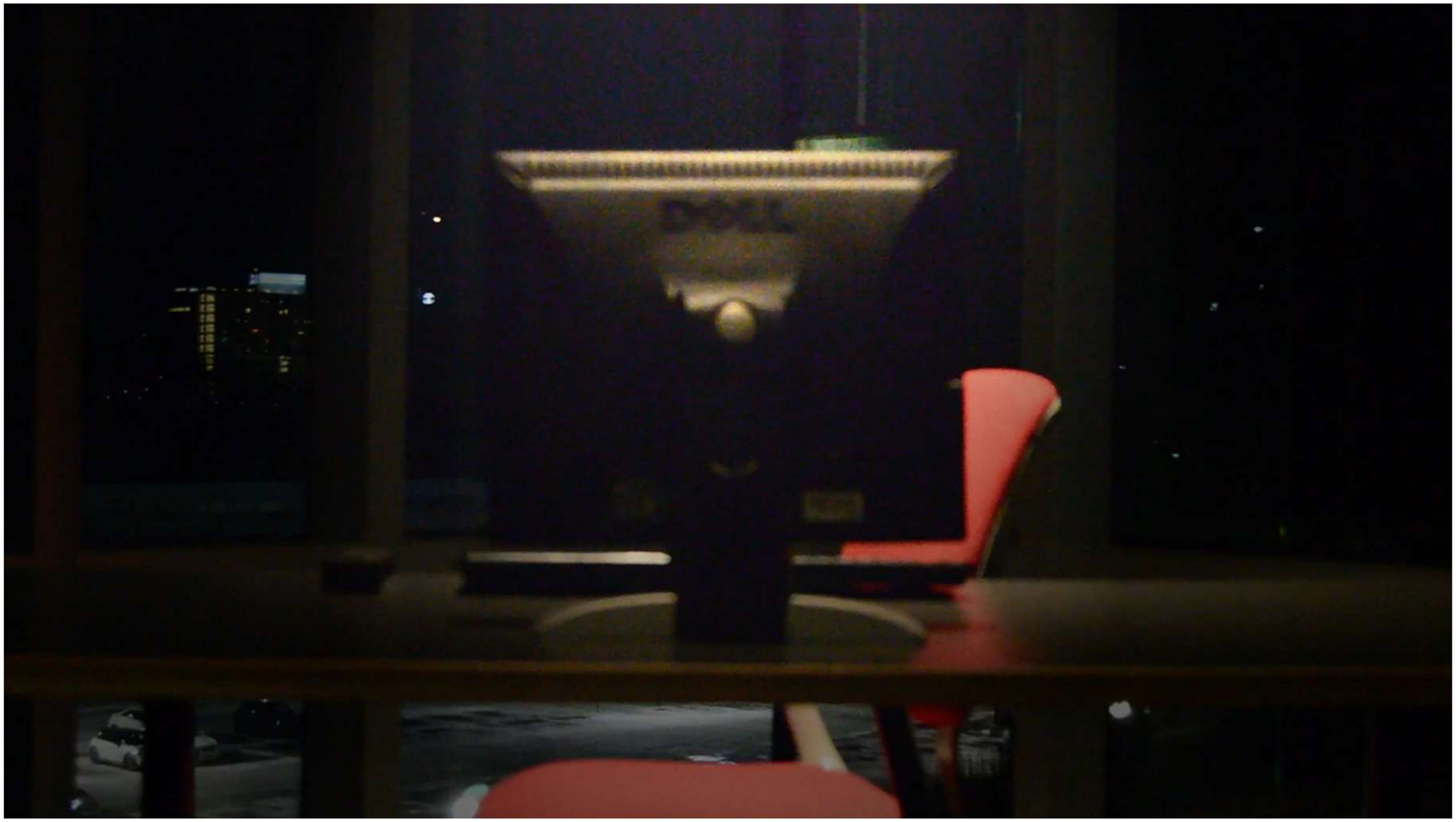
or every 50 ms

Randomization Defenses (w/Churn)

- Code representation
- Code layout (absolute and relative)
- Code pointer representation
- Function pointer representation
- Return pointer representation
- Data pointer representation
- Data layout (absolute and relative)
- Microarchitectural mappings

504 bits of true random entropy

Morpheus: A Puzzle that Computes



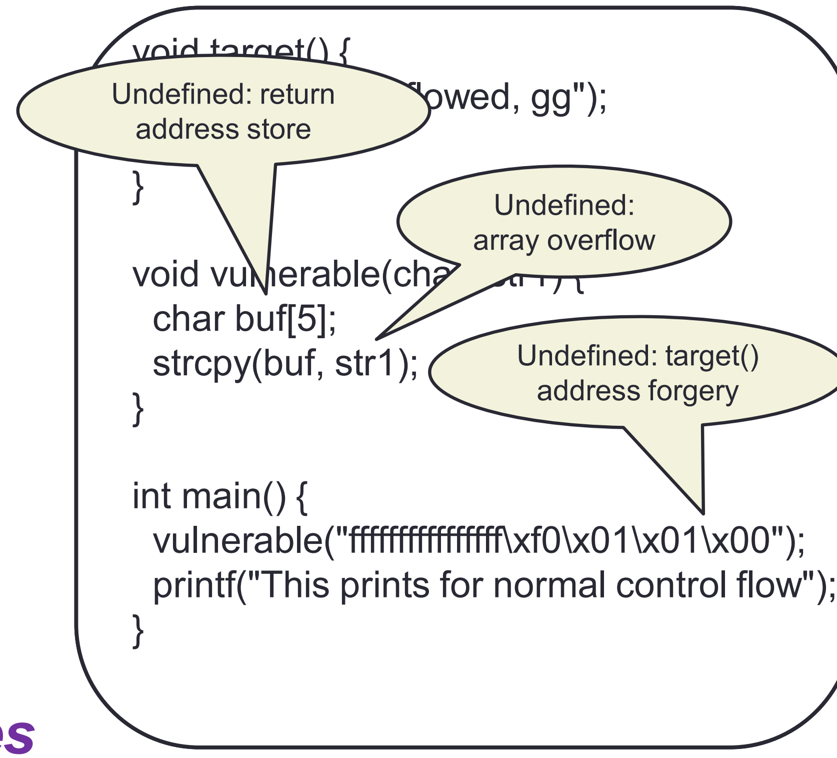
Mark Gallagher

Lauren Biernack

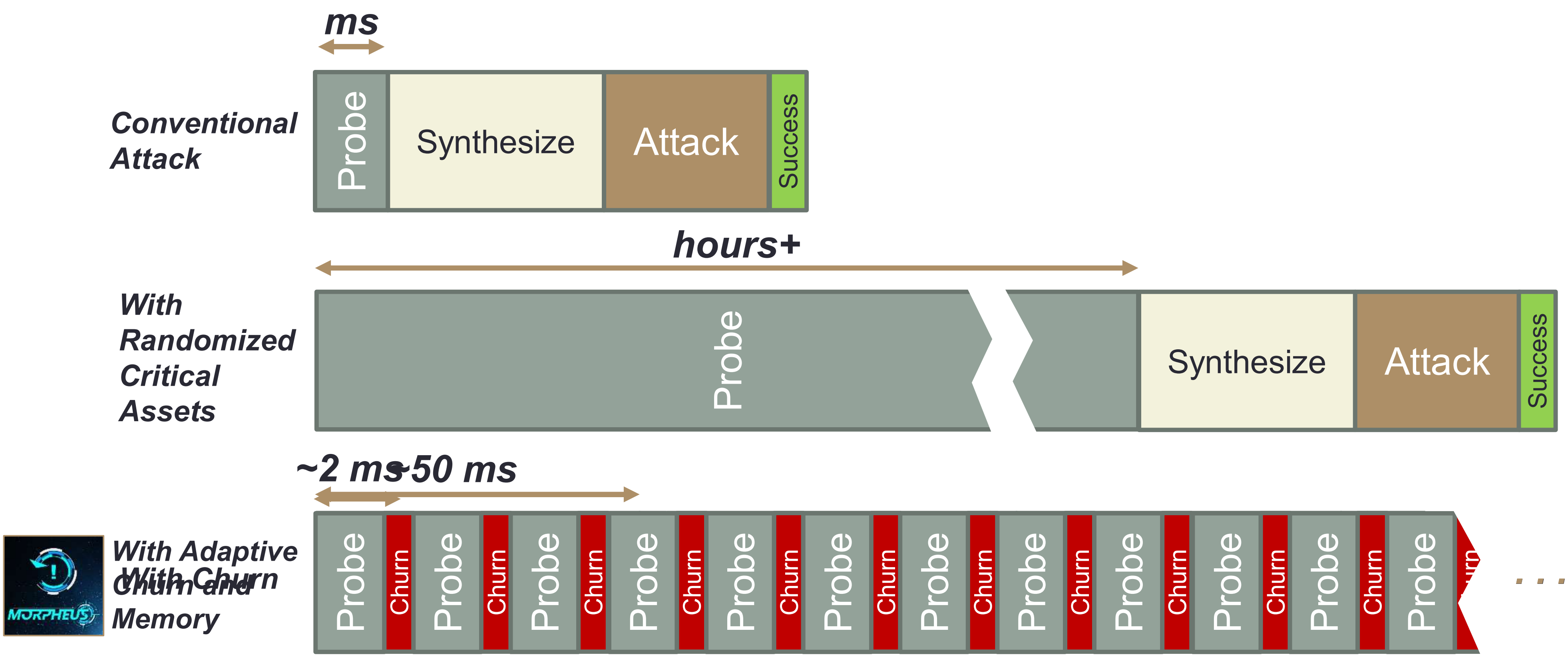
Alex Kisil

Morpheus Deploys Encryption and Churn

- Morpheus attack detectors discern ***normal*** code from ***malicious*** code, via ***undefined semantics***
- To stop unknown attacks, Morpheus continuously ***encrypts*** undefined program assets, a process called "***churn***"
- Churning undefined assets ***breaks malicious security attacks***, but has no effect on normal software
- Learning mechanisms can ***record and prioritize successful defense strategies*** to speed up protections



Morpheus Breaks Emergent Attacks



Morpheus II RISC-V Extensions and Microarchitecture

Morpheus Code and Pointer Defenses

- ***Always-encrypted code*** is ***physically isolated*** when decrypted

Opcode	Semantics
dst := ptr1 <op> ptr2	Pointer arithmetic: +,-
dst := ptr1 <rel> ptr2	Pointer test: <,>,==, !=,…
dst := load/jump (ptr)	Dereference: ->, *

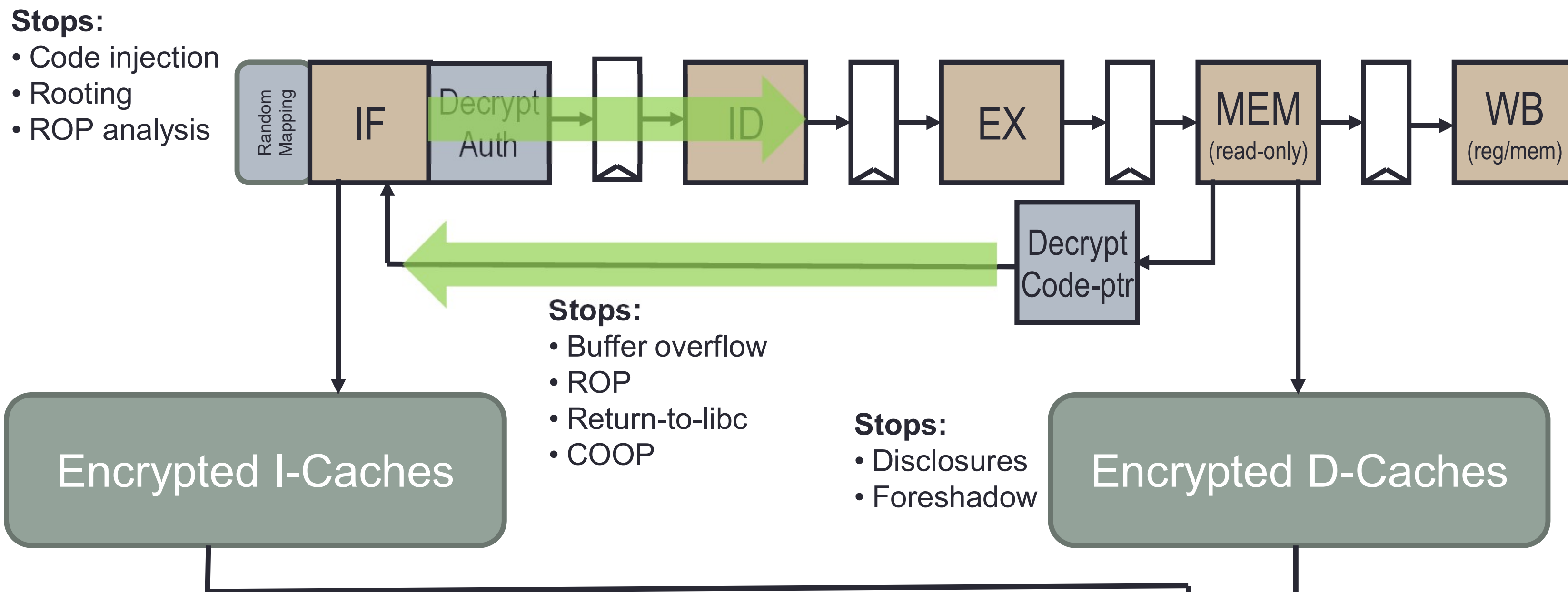
Legend:
Green = decrypted
Red = encrypted

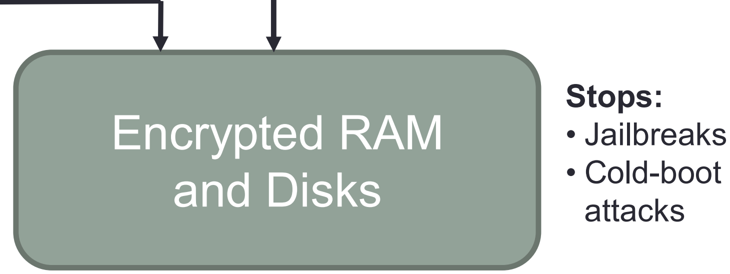
- ***Always-encrypted pointers*** are ***physically isolated*** when decrypted
  - Pointers are accessed with RISC-V instruction set extension
- ***No tagging required*** because we universally change code/pointer format
  - This is ***not*** a problem for normal software
- Pointer tests are leaky, so use ***churn*** to limit utility of side channels
  - Churn re-encrypts program assets while the system is running

Morpheus RISC-V Microarchitecture

Austin Harris

Tarunesh Verma



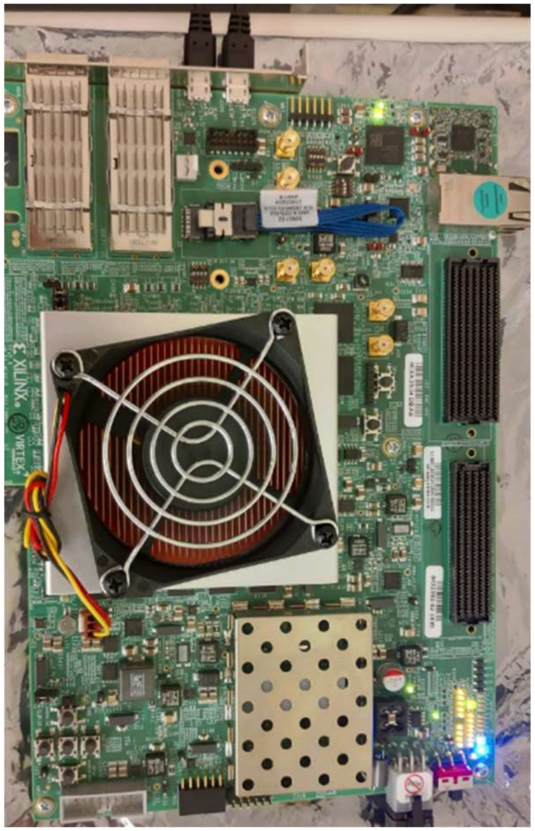


- Built to stop ***remote code execution (RCE)***
  - Built on the RISC-V Rocket Core
  - Always-encrypted code
  - Always-encrypted code pointers

Morpheus II Performance, Area and Security Analysis

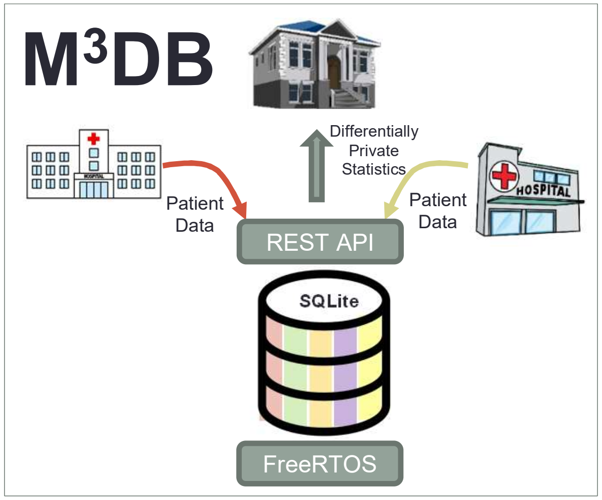
Morpheus Design Overheads

- Integrated into the RISC-V Rocket Core
  - Only 369 lines of Chisel code added
- Deployed in a Xilinx UltraScale+ FPGA
  - Utilized a 12-round Simon cipher
  - < 1% performance overhead
  - 0.2% power overhead
  - 1.3% area overhead
  - Negligible impact to network apps



Putting Morpheus to the Test

- 32-bit Morpheus entered FETT
  - Deployed on AWS F1 FPGAs
  - 535 attackers were recruited for 3 months
  - Worked for sizeable bug bounties
- Running a mock medical DB
  - Only 3 lines of code changes required!
  - Attackers had to penetrate the target (RCE)
- Toward the end of the program, a “high-value payout” was created
  - For a Morpheus SQLite-to-RCE attack
- Morpheus was the second-most engaged target in FETT
- ***Morpheus was penetrated ZERO times***



Morpheus' Evolution and Beyond

- Why is Morpheus hard to hack?
  - Always-encrypted pointers deny attackers ability to forge/analyze code/pointers
  - Churn places a time-limit on replay attacks and probing results
  - Morpheus attacks must be bespoke and lightning-fast (stochastic attacks)
- Lean into secure systems with durable security mechanisms
  - Avoid non-durable mechanisms: software, resource sharing, leaky operations
  - ***Time-Tested Cryptography***, examples: RSA, AES, SHA-2
  - ***Physical Isolation***, examples: TPMs, Intel CAT
- Next-generation Morpheus-derived technology is being deployed
  - Provides highly secure ***secret computation***
  - Based on ***cryptography*** and ***physical isolation*** based defenses
  - Deployed in the Microsoft Azure and Amazon AWS clouds

Shibo Chen

AGITA LABS

Questions?

? ? ? ?

? ? ? ?

?

? ? ?

austin@umich.edu