Morpheus
Adaptive Defenses for Tomorrow’s Secure Systems

Todd Austin
University of Michigan
austin@umich.edu

Joint work with:
Valeria Bertacco (UM)
Sharad Malik (Princeton)
Mohit Tiwari (UT-Austin)
Assessing the State of Security

- Jeep hacked remotely while driving
- DHS attacks Boeing 757, details classified
- Pacemaker wirelessly infiltrated
- Mirai botnet disables DynDNS
- Entire baby monitor market hacked
- Atrium fish tank thermometer hacked
Why is Security So Hard to Get Right?

• Currently, a patch-based approach
  • Find and fix vulnerabilities
  • Complexity growth *far outstrips* security
  • Manual testing & analyses don’t scale

• Endless *security arms race*
  • Patch and pray…

• How do we protect against *unknown (0-day) attacks*?
  • Anticipate the “unknown unknowns”
Attacking is Easy, Protecting is HARD

- **Attacking is easier than protecting**
  - Attackers needs only *one* vulnerability
  - Protecting requires *100% coverage*

- Related software growth rates:
  - Protections: doubles every 2 years
  - Malware: 40% growth in 30 years

- Vulnerabilities are on the rise
  - Rate of attacks is exploding
Durable Security: the Big Unsolved Challenge

- What we do well:
  - Finding and fixing vulnerabilities
  - Deploying system protections that stop well-known attacks

- Where we fail: *identifying and stopping emergent attacks*
What If a Secure System Could…

- Respond lightning-fast against common attacks
- Self-adapt quickly to unknown emerging threats
- Learn and prioritize the most successful defense strategies
- Utilize a self-protecting distributed implementation

T-Cell Adaptive Immunity
Human Adaptive Immunity Primer

- T-cells receptors discern **normal** cells from **malicious** cells, via genetic markers

- To stop an unknown disease, T-cells undergo hypermutation that **randomizes** T-cell defense capabilities

- Boosted T-cell diversity will likely **stop the pathogen attack**

- **Immunological memory records successful T-cell variants** to speed future recoveries
Morpheus Mimics Adaptive Immunity

- Morpheus attack detectors discern **normal** code from **malicious** code, via undefined semantics.

- To stop an unknown attack, Morpheus **randomizes** a system’s undefined semantics, a process called “churn”.

- Churning undefined semantics **stops security attacks**

- **Learning mechanisms record successful defenses** and stop future attacks quicker.
Morpheus’ Unique Approach to Security

Vulnerabilities + Implementation Assets = Exploit

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

**Randomization Defenses (w/Churn)**
- Code representation
- Code layout (absolute and relative)
- Code pointer representation
- Data pointer representation
- Data layout (absolute and relative)
- Function pointer representation
- Return pointer representation
- User enclave data representation
- Microarchitectural mappings

or every 50 ms

504 bits of true random entropy
Protecting Critical Assets with Encryption

• Critical program assets are encrypted under their domain keys
  • Code, code pointers, data pointers
  • Decrypted at fetch, jumps and load/stores
  • Tracked at runtime using dynamic tagging

• Assets remain encrypted in registers, memory, buses, I/O
  • Requires strong ciphers in the pipeline

• Churn re-encrypts a domain under a new random key
  • Places a time limit on penetrating encryption
Morpheus Breaks Emergent Attacks

Conventional Attack

With Randomized Critical Assets

With Adaptive Churn and Memory

ms

hours+

~2 ms-50 ms

Success
Fast Churn Defeats Probing

- Blind call attack example
  - Attacker attempts to call `syscall()`

- Attack success rate dependent on **churn rate** and degree of **entropy**
  - State-of-the-art: no churn and low/high entropy
  - Morpheus: frequent churn and high entropy

- H/W churn makes probes no more powerful than **random guesses**
  - Impractically difficult with high entropy
Morpheus Platform Details

Morpheus Secure Platform

S/W Ecosystem
- LLVM
- GCC/Binutils
- Type Analysis
- Backend Metadata Emitter
- FreeRTOS

H/W Architecture
- 32/64-bit RISC-V Rocket Core
- Morpheus Defense Layers
- Domain Encryption
- Pointer Locking
- Hard NULLs
- Tagged Memory
- Churn Unit

Morpheus Defense Layers
- Encryption
- Pointer Locking
- Tagged NULLs
- Churn Unit
Tagging & Attack Detection

- Tags enable behavior tracking
- Illegal Ops
  - Clearly dangerous
- Suspicious Ops
  - Normal programs may perform
  - May be probes or attacks

```
<table>
<thead>
<tr>
<th>Operand Tags</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illegal Ops</td>
<td>Suspicious Ops</td>
</tr>
<tr>
<td>Illegal</td>
<td>Suspicious</td>
</tr>
<tr>
<td>• Executing non-code</td>
<td>• CP arithmetic</td>
</tr>
<tr>
<td>• Jump to non-CP</td>
<td>• Arith. overflow</td>
</tr>
<tr>
<td>• ...</td>
<td>• ...</td>
</tr>
</tbody>
</table>

Terminate Program
Churn

Otherwise, churn every 50ms
**Morpheus Microarchitecture**

**Key Hardware Advantages**
- Power efficiency and speed
- Strong root-of-trust
- Randomization via strong ciphers (not XOR or CTR!)

**Stops:**
- Code injection
- Buffer overfl
- Heartbleed
- Meltdown
- Spectre
- AnC de-random
- Rowhammer

- Rooting
- ROP
- Return-to-libc
- COOP
- ROP analysis
- Jailbreaks
- Cold-boot attacks
- Disclosures
- Foreshadow
- Heartbleed
- AnC de-random
- Rowhammer
- Spectre
- Meltdown
- Fallout
- Flush+Reload
- Jailbreaks
- Cold-boot attacks
Churning Keys at Runtime

**Churn Period**

- **Stale:** Under OLD key
- **Clean:** Updated to NEW key
Assessing the Security of Morpheus

How long does it take to penetrate Morpheus defenses?

• Difficult to attack a system that is
  • Constantly changing
  • Has high entropy

• Approach: Attack a *weaker* Morpheus

De-featured Morpheus
Churn Disabled
Shared Key for Defenses
Morpheus--Penetration Testing Results

E == Domain encryption on (E) or off (E)
P == Pointer displacement on (P) or off (P)

Average Attack Probe Time (s)

0 50 100 150 200 250

EP

0.01s

98.6s

152s

251s

Ep

AnC Address De-randomization
High bit probes
Code search
Blind code search
How Effective is Morpheus? Early Results

Analysis: RISC-V Morpheus on Gem5 simulated system

Early results:

- Performance cost: **2% average slowdown** with 504-bits of entropy and 50ms churn
- Power cost: **2.5% power**
- Area cost: **8% area** increase
- Developer cost: **No impact on normal applications**
Why: We want to build strong confidence in our security
How: Provide RISC-V based H/W to attacker community

Demo 1: Voting machine at DEFCON – by Dec 2019
  Goal: Validate security claims with black-hat community

Demo 2: Network-facing website – by Feb 2020
  Goal: Deploy a long-term world-attackable platform with bounty
  Runs a subset of Wikipedia, includes an interface to inject code

Demo 3: Secure avionics demonstration – by Jun 2020
  Goal: Excise developer issues via engagement with defense contractors
Morpheus’ Evolution and Beyond

- Originally Morpheus had decrypted caches
  - Foreshadow taught us that was a potential vulnerability
- Today’s Morpheus has encrypted memory, caches, registers
  - And more encryption domains: data pointer, code pointer, return pointer, user data, etc…
- Observation: to build security, we deploy two durable mechanisms
  - Isolation and encryption
  - History: physical memory begat virtual memory begat virtualization begat containers begat TEEs begat Morpheus…
  - Each step, we accomplish the important goal of putting less trust in software

- What is the endgame of security?
  - Total isolation and total encryption… and zero trust in software?
  - This is where I want to go next… let’s work together!
Toward Zero Trust in Software

Unprotected

Apps/OS
CPU/Mem

0%

Homomorphic Encryption
HE App/OS
CPU/Mem
100,000-1,000,000%

More S/W Trust

More H/W Trust

Overheads

Unprotected

Apps/OS
CPU/Mem

0%

Container’ized

App
OS
App
CPU/Mem

5-10%

SGX Enclave

Apps/OS
CPU/Mem

SGX

10-15%

Morpheus

CP
DP
S/W
CPU/Mem

2%

Homomorphic Encryption
HE App/OS
CPU/Mem
100,000-1,000,000%

Less Trust in Software
Homomorphic Encryption Minimizes Trust

- HE advances privacy
  - No trust in S/W
  - No trust in H/W
  - Only trust in (immature) crypto

- What is the cost?
  - $10^5$ – $10^6$ times slower than comparable unencrypted computation
  - Can be parallelized extensively, and a focus of accelerator designers
  - Is it safe? Is it economical?

The Cost of Data Breaches

Varonis.com:
• 1 in 4 chance of experiencing data breach in a given year

IBM:
• Average cost per data breach in 2018: $3.86 million

Cybersecurity Ventures:
• Global cybersecurity market >$120 B in 2017
• Typical S&P 500 bank spends $500 M/year on cybersecurity

<table>
<thead>
<tr>
<th>AWS Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yearly revenue</td>
</tr>
<tr>
<td>Expected total cost of data breaches for AWS user base</td>
</tr>
</tbody>
</table>
Questions?

We demand rigidly defined areas of doubt and uncertainty!