Architectural Support for Security

Tim Loffredo
Branch Regulation: Low-Overhead Protection from Code Reuse

- Prevents "Code Reuse Attacks"
  - CRAs are a BIG PROBLEM!
- New Architectural Component: Secure Call Stack
- Good Performance (2% Overhead)
- But First...
A Little History

- Code Injection

![Memory Diagram]

- Stack
- Heap
- BSS (uninitialized)
- Data (initialized)
- Text (Code)

Attacker's Code

Buffer Overflow
A Little History

- Code Injection
- DEP ~2004
A Little History

- "The Geometry of Innocent Flesh on the Bone" by Hovav Shacham, 2007
  - Return to Libc by Solar Designer, 1997
- Code Reuse Attack, aka ROP

Programming in ROP is like writing a ransom note
Return Oriented Programming

- ROP "borrows" code from the exploited application to create the attacker code.
Return Oriented Programming

Attacker Stack Data

Borrowed Code

Resulting Attacker Code

Stack Pointer
Return Oriented Programming

- Takeaway Point:
  Attackers reuse code to circumvent DEP
  - Usually with ROP
  - Also circumvents code signing
Back to Branch Regulation

• Prevents code reuse attacks
• Hardware performs a check on every indirect branch
  • "ret" instructions
  • "jmp <blah>" instructions
  • "call <blah>" instructions
Branch Regulation

- Call and ret are simple cases
- On "call <blah>":
  - Verify that <blah> is a valid function entry point
  - Record next instr address (we will return there)
- On "ret":
  - Verify that we are returning to an address recorded by a previous valid call
Branch Regulation

- General indirect jump is hard to regulate
  - Compilers do weird things...
  - Authors chose an OK heuristic
- On "jmp <blah>":

```
.global f1
.type f1, @function
<br-annotation>
...
mov eax, ecx
...
jmp edx
...
.global f2
.type f2, @function
<br-annotation>
...
add ecx, 04h
...
```

1. JMP inside of the function
   - Function Base < Target < Function Bound

2. JMP to a new function
   - (Function Bound < Target Address)
   - Target = <br-annotation>

3. JMP to middle of another function
   - (Function Bound < Target Address)
   - Target Address ≠ <br-annotation>
Putting It Together

- Need Secure Call Stack
- Need Function Boundary Annotations
How?

- Special Hardware in Pipeline

Function Bounds Stack = Secure Call Stack + Function Start / End
Performance

- Performance Overhead: 1-2%
  - Cuz it executes in parallel!
  - Measured by simulation
- Foundation Bound Stack Size: Only 16 Entries
Effectiveness

- Constrain RET targets: gadgets can't chain
  - 99% reduction in available gadgets
- Effectively stops ROP
  - ... in the 5 binaries the authors looked at
- Should slow ROP regardless
  - ROP programming goes from Hard to Infeasible
Security Analysis

- Paper Makes Assumptions
  - All Exploits Use a Syscall
  - All Exploits Need a "Dispatcher" Gadget
- Not as good as full Control Flow Integrity (CFI)
  - Why not take that extra step?
    - Because CFI requires compiler-level static analysis

- Extra Note:
  The "security" of a system is difficult to measure
Other Paper: kBouncer

- Uses Last Branch Recording (LBR) Registers
  - Existing Hardware
- Checks LBR for ROP on Syscalls
- Runtime Overhead: ~1%
- Limitations:
  - User Space Unprotected
  - Syscall Boundary Can Be Fooled
Other Paper: CFIMon

- Uses Branch Trace Store (BTS)
  - Existing Hardware
- Trains branch data based on normal application runs
- Flags branches taken as "suspicious" when witnessing abnormal behavior
- Runtime Overhead: ~6%
- Limitations:
  - Some False Positives
CFIMon Example

ROP Attack

```
stack
... 
ADDR7
ADDR6
ADDR5
ADDR4
ADDR3
ADDR2
ADDR1
old ebp
password

Inst1->inst2->inst3->inst4->...
```

CFIMon Checks

```
samples

(ntim_check_auth)
ret

insn4
ret

insn3
ret

insn2
ret

insn1
ret

ADDR5
```

- <source> is direct call/jump
  - <target> in ret_set? [yes → Legal, no → Illegal]
- <source> is return
  - <target> in ret_set? [yes → Legal, no → Illegal]
- <source> is indirect call
  - <target> in call_set? [yes → Legal, no → Illegal]
- <source> is indirect jump
  - <target> in train_set? [yes → Legal, no → Illegal]
- <source> is unknown
  - Illegal

• CFIMon catches this attack at the first "ret"
Done

- Any Questions?