Rethinking the use of execute-only memory to closed-source COTS binaries

Dynamic code reuse attacks assemble exploit payload at runtime using memory disclosure attacks. Existing works require source code, or do not support JIT code. Disassembly of binaries is incomplete.

**Motivation**

**Taxonomy of Approaches**

- **Dynamic Code Reuse Attack**
  1. Memory disclosure
  2. Scan memory at runtime for gadgets
  3. Chain gadgets to generate shellcode
  4. Redirect control flow

- **Prior Defenses**
  1. Memory disclosure
  2. Scan memory at runtime for gadgets
  3. Chain gadgets to generate shellcode
  4. Redirect control flow

- **Our Work**

**Key Insights**

**Observer Effect:**

"The act of observing a system inevitably changes the state of the system."

**Heisenbyte’s destructive code reads:**

"Reading executable memory changes the executable state of the read memory."

**Execution Overhead**

Virtualization avg overhead: ~1.8%
Destructive code reads avg overhead: ~16.5%

**Memory Overhead**

Peak RSS memory avg overhead: ~0.8%

**Detection Results**

Crafted dynamic code reuse exploits and monitor for invoked debug trap

1. Dynamic code
   - Self-injected bug in toy program that mimics the creation of a JIT code buffer
2. Static code
   - CVE-2013-2551: Internet Explorer Bug

Exploits on both static programs and dynamic JIT code triggered debug traps