Heisenbyte: Thwarting Memory Disclosure Attacks using Destructive Code Reads

Adrian Tang Simha Sethumadhavan Salvatore Stolfo

Motivation

- 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.

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

Taxonomy of Approaches

Dynamic Code Reuse Attack

Memory disclosure
+
1. Scan memory at runtime for gadgets
+
2. Chain gadgets to generate shellcode
+
3. Redirect control flow

Prior Defenses

Memory disclosure
+
XnR (CCS'14)
HideM (CODASPY'15)
Readactor (Oakland'15)
Execute-only Mem
+
2. Chain gadgets to generate shellcode
+
3. Redirect control flow

Our Work

Memory disclosure
+
1. Scan memory at runtime for gadgets
+
2. Chain gadgets to generate shellcode
+
HEISENBYTE (This talk)
Destructive CRs

Key Insights

Werner Heisenberg, in 1933 (German theoretical physicist)

Image credits: Wikipedia

Observer Effect:

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

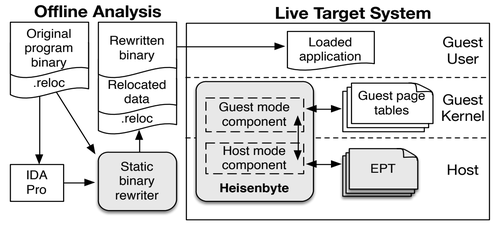
HEISENBYTE's destructive code reads:

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

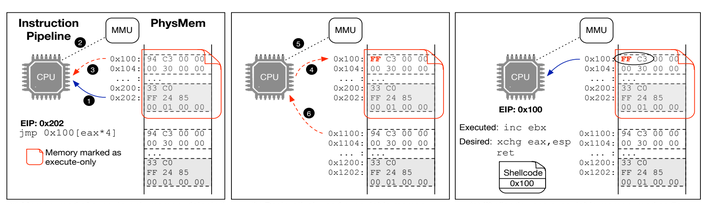
Executing memory after reading it yields unpredictable behavior

Mechanism

Architecture

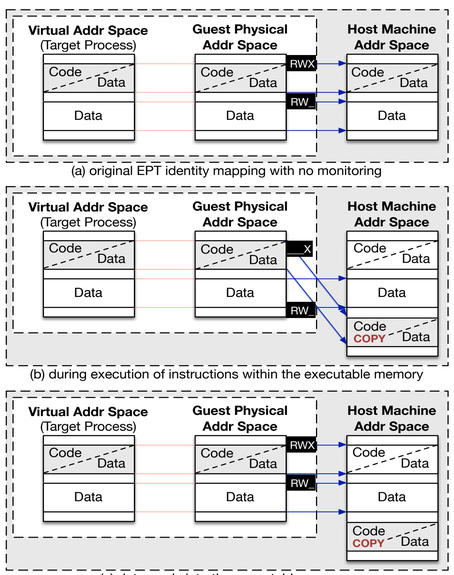


Destructive Code Reads



(a) Memory read of execute-only memory detected (b) Destructively read executable memory (c) Shellcode uses executable memory read earlier

Hardware Virtualization Support



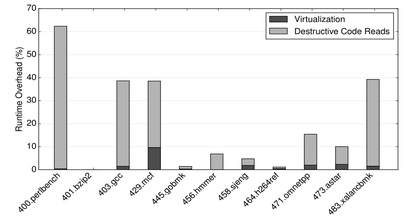
(a) original EPT identity mapping with no monitoring

(b) during execution of instructions within the executable memory

(c) data reads into the executable memory

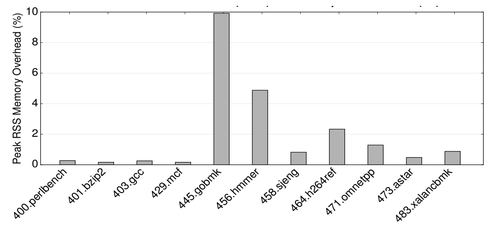
Results

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

HEISENBYTE corrupts code with debug trap code `0xCC`

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