Shuntaint: Emulation-based Security Testing for Formal Verification

Bruno Luiz
ramosblc@gmail.com

Black Hat Europe 2009

Overview

- Give a brief overview of the emulation-based security testing
- Introduction to VEX
- Formalism
- Implementation details
- Benchmark programs

What is it?

- A automated proof in an error or a pattern we are looking for
  - Detected violation of range or bondary limits
  - Convergence to an inappropriate point
- Methods for modeling of symbolic memory
  - It periodically determines checks for specific user data or functionality can be bypassed
- Bug-finding during simulated execution of the computer program
  - Using valgrind “tool plug-in”

Example: bounds checking on static array

```
8048387 sub    $0x18,%esp
804838a sub    $0x4,%esp
804838d push   $0xf
804838f push   $0x0
8048391 lea    -0xf(%ebp),%eax
8048394 push   %eax
8048395 call   80482cc <memset@plt>
804839a add    $0x10,%esp
804839d sub    $0x4,%esp
80483a0 push   $0xf
80483a2 pushl  0x8(%ebp)
80483a5 lea    -0xf(%ebp),%eax
80483a8 push   %eax
80483a9 call   80482ec <strncat@plt>
80483ae add    $0x10,%esp
```

Problem Statement

- Flaws may pass through the software checks
- Error-checking tool detects something bad happening, but not how error can be triggered
- Memory-to-Memory propagation cannot address some situations
- Doesn't perform automatic crafted manipulations trying to replace legitimate memory to trigger bug

Emulation-based Method

- Abusing of self-modifying code
  - Translation, instrumentation and compilation to machine code
- Math background
  - Very useful in computing sets of states
  - Ensuring correctness of the model that leads to the error
- Error trace that leads to an error state
  - More precise understanding of entry points

Dealing with VEX:intermediate representation

- Library for instrumentation or translation
- Converts blocks of machine code to an intermediate representation
- Provides usefull operations for low-level memory manager
- Architecture-neutral intermediate representation

VEX interface overview

- Instrumentation suports: VgCallbackClosure:
  - Thread requesting the translation
  - Guest address: redirected and non-redirected
- Superblocks represents instructions
- Guest state layout contains stack pointer and program counter
- Byte ranges of original code is available
- Native word of Simulated/Real CPU have easy control

VEX interface, main fuctions

- VG_(basic_tool_funcs)
  - This is enough for initialisation
- VG_(needs_client_requests)
  - Trapdoor mechanism
- VG_(needs_syscall_wrapper)
  - Trackable events before and/or after system calls
- VG_(needs_malloc_replacement)
  - Replace behaviour of friends fuctions

VEX interface, some more functions

- VG_(track_new_mem_startup)
  - Memory events notified to the appropriate function
- VG_(track_new_mem_stack)
  - Track start of stack
- VG_(track_pre_mem_write)
  - Called before an event of memory write
- Plus fuctions, read/writer register events, thread events, client requests, etc.

VEX IR description

- Super blocks (IRSB) are blocks of simulated instruction
- Each IRSB contains a list of statements (IRStmt) with side effects
  - storing a value to memory
  - assigning to a temporary variable
- IRStmt may have expressions (IRExpr) without side effects
  - arithmetic expressions
  - loads from memory

Guest code addresses

8048384	55	push %ebp	------ IMark(0x8048384, 1) ------ t0 = GET:I32(20) t19 = GET:I32(16) t18 = Sub32(t19,0x4:I32) PUT(16) = t18 STle(t18) = t0
8048385	89 e5	mov %esp,%ebp	------ IMark(0x8048385, 2) ------ PUT(20) = t18
8048387	83 ec 18	sub $0x18,%esp	------ IMark(0x8048387, 3) ------ t2 = Sub32(t18,0x18:I32)
804838a	83 ec 04	sub $0x4,%esp	------ IMark(0x804838A, 3) ------ t5 = Sub32(t2,0x4:I32) PUT(32) = 0x6:I32 PUT(36) = t2 PUT(40) = 0x4:I32 PUT(44) = 0x0:I32

How does this approach work?

- Analyse programs at run-time at the level of intermediate representation
  - Modeling, Specification and Verification
    - State transition system
    - Temporal Logic
    - Algorithm

Model Checking

- Converts a design into a formalism: Memory Graph
- Find a set of states that satisfy a temporal logic formula
  - Reverse Tainting Analysis

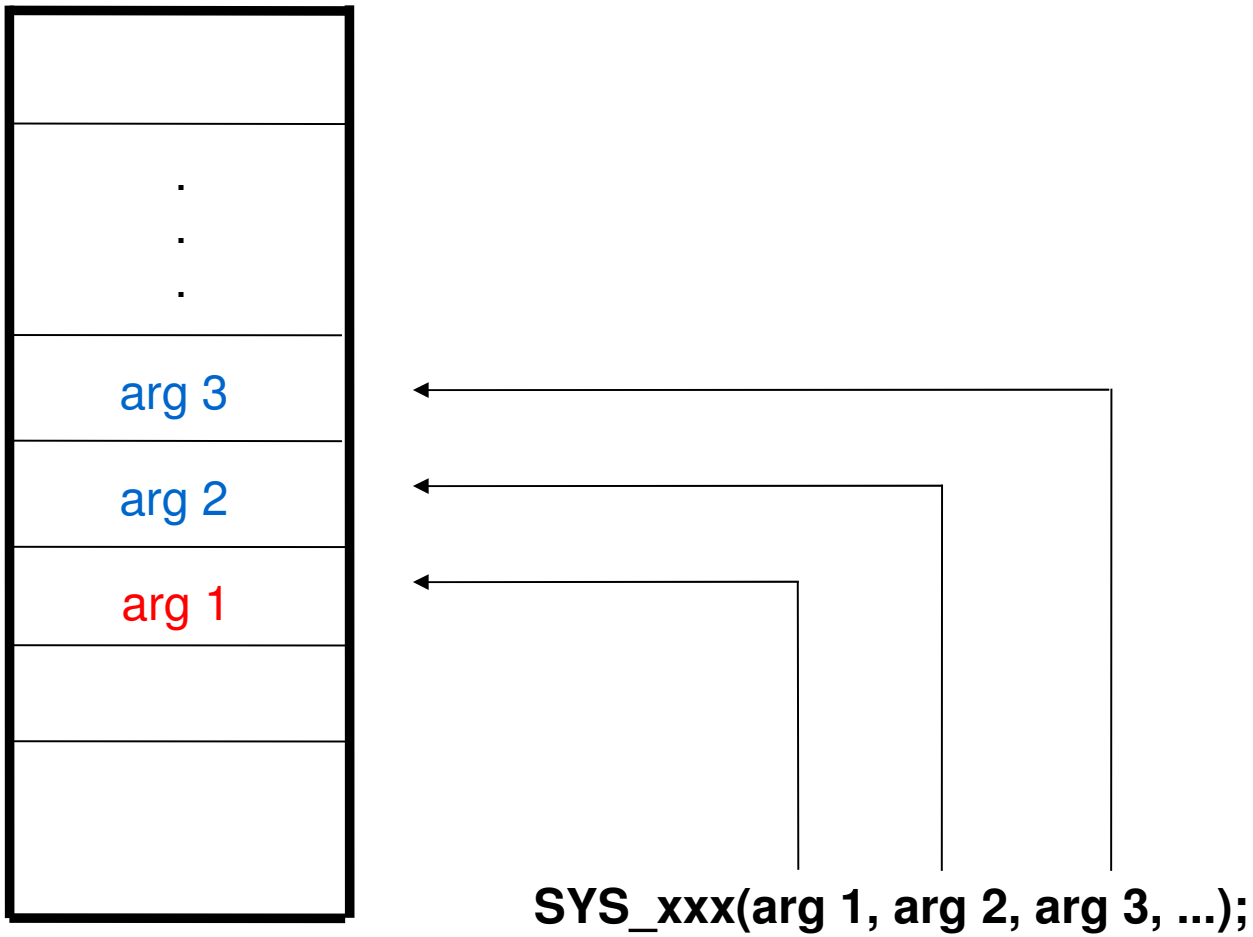
Network Tainting

- Network is the most likely vector of attack
- Data from network
- File descriptions tracking
  - Trace all inputs from open file description
    - open, socket, connect, accept, socketpair, and close

Locating Potential Manipulation

- Look at chunks of guest state
- Parameter error is written
  - VG_(get_ThreadState)
- Mark shadow area as valid
- Characteristics:
  - To set these events VG_(track_pre_reg_read) and VG_(track_post_reg_write) are called
  - Access area of guest's shadow state using VG_(set_shadow_state_area)() and VG_(get_shadow_state_area)()
  - Record definedness at [offset, offset+len)

Manipulation Layout



Locating roots (CWE)

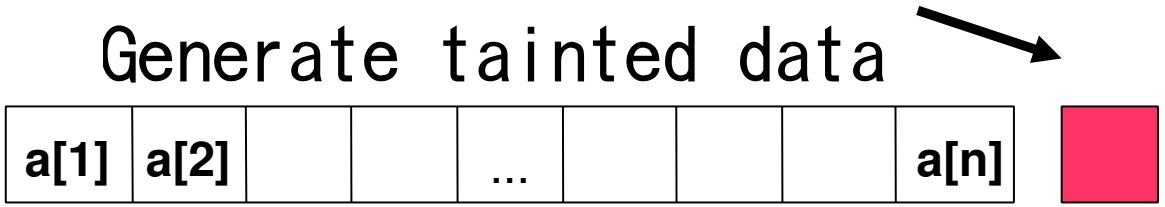
- Argument Injection or Modification (ID: 88)
- Return of Wrong Status Code (ID: 393)
- NULL Pointer Dereference (ID: 476)

Scanning invalid operation

- Hacking pointer check
  - Checks accesses to generate a set of tainted value
    - Write-what-where conditions
  - Adding instructions to VEX IR translated back to machine code
- How to we get their contents/location?
  - Pointer: for each possible pointer in memory
  - LOAD, STORE: interact with memory
  - Syscalls: memory accesses

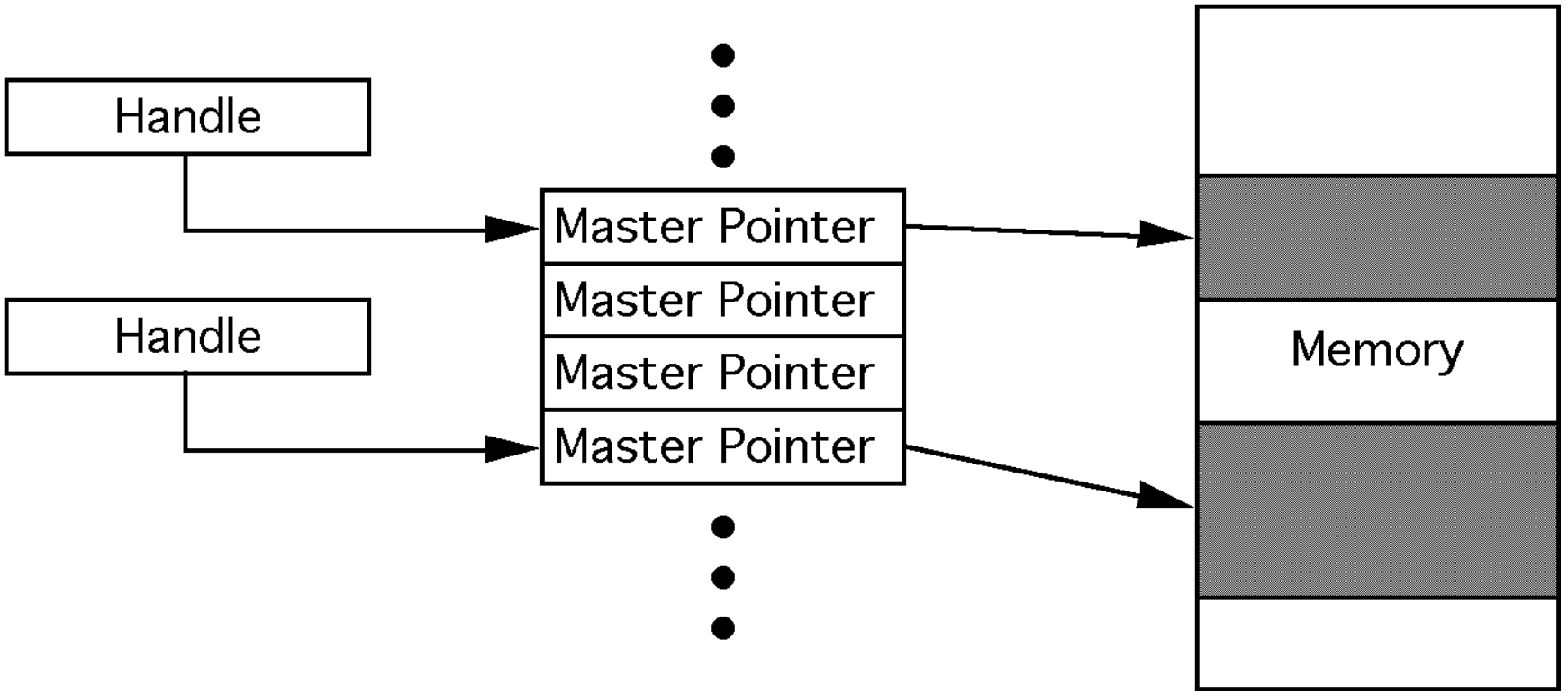
Meta-data

- Mark bit for segment ranges
- Range check possible pointers for extra space



- Instrumentation deals with shadow value
  - Generate instrumentation

Meta-data lookup



Locating root (CWE)

- Unchecked Array Indexing (ID: 129)
  - ST<end>(<addr>) = <data>
  - PUT(16) = <data>
- Incorrect Pointer Scaling (ID: 468)
  - Add32(GET:I32(16),<con>)
- Failure to Handle Length Parameter Inconsistency (ID: 130)
  - <op>(<arg1>, <arg2>)
  - ST<end>(<addr>) = <data>

Locating root (CWE)

- Incorrect Calculation of Buffer Size (ID: 131)
  - t<tmp> = <data>
  - <op>(<arg1>, <arg2>)
- Integer Overflow or Wraparound (ID: 190)
  - <op>(<arg1>, <arg2>)
  - ST<end>(<addr>) = <data>
- Off-by-one Error (ID: 193)
  - ST<end>(<addr>) = <data>
  - PUT(16) = <data>

Locating root (CWE)

- Use of sizeof() on a Pointer Type (ID: 467)
- Assignment of a Fixed Address to a Pointer (ID: 587)
- Attempt to Access Child of a Non-structure Pointer (ID: 588)

Future improvements

- Efficient search procedure
- Use logical formalism

Thank you!

Questions?
ramosblc@gmail.com