Detecting Code Reuse Attacks Using Dyninst Components

Emily Jacobson, Drew Bernat, and Bill Williams
Paradyn Project

Paradyn / Dyninst Week
Madison, Wisconsin
April 29-May 1, 2013
Threat Model

- Attack goal: effect some malicious intent by hijacking program control flow

- Historically, accomplished via code injection
Threat Model

- \( \mathcal{W} \oplus \mathcal{X} \) prevents code injection

- Alternative: construct an exploit using code that already exists within the program’s address space

“code reuse attacks”
Anatomy of a Code Reuse Attack

Select gadgets from within the address space of the process

Chain gadgets together with indirect control flow

Usually a short attack with the goal of escaping the confining W⊕X environment

code

xchg %eax,%ecx
fdiv %st(3),%st
jmp *-0xf(%esi)

add %edi,%ebp
jmp *-0x39(%ebp)

mov 0xc(%esi),%eax
mov %eax,(%esp)
call *0x4(%esi)

add %edi,%ebp
jmp *-0x39(%ebp)

sysenter
...
pop %ebx

exec(“/bin/sh”)
Anatomy of a Code Reuse Attack

**process address space**
- stack
- heap
- code
  - entry point

**code injection attack**

**process address space**
- stack
- heap
- code
  - entry point

**code reuse attack**

Detecting Code Reuse Attacks Using Dyninst Components
Previous Code Reuse Attack Defenses

- **Detect using heuristics based on attack behaviors**
  

- **Enforce control flow integrity at runtime**

  [Abadi et al. 2009], [Bletsch et al. 2011], [Zhang et al. 2013]

*In the next talk, Tugrul will talk about another interesting defense technique.*
Our Approach

- Define *conformant program execution* (CPE) as a set of requirements on program states
  - Valid program counter
  - Valid callstack

- Enforce CPE by monitoring program at runtime
Valid program counter (PC):

PC must point to instruction in the original program

PC: 0xb7fe3424

b7fe3424: pop %ebp
b7fe3425: pop %edx
b7fe3426: pop %ecx
b7fe3427: ret
Valid callstack:

For each frame:

1. frame must have valid stack frame height
2. (caller $\rightarrow$ current frame) must represent a valid control flow transfer in the program
Program Validation

Design decision: when do we validate?

- **Option 1**: At all instructions
  - “Conformant program execution”
  - Disadvantage: inefficient
- **Option 2**: At system calls
  - “Observed conformant program execution” (OCPE)
  - Effective because attacks must use the system call interface to modify overall machine state
ROPStop Implementation

Initialization
1. Attach to running process or launch new process from binary
2. Parse program binary
3. Register callbacks at system calls
4. Continue process

At each system call
1. Validate current program counter
2. Perform robust stackwalk and validate the current callstack
Evaluation

Accuracy

- Real code reuse attacks

Overhead

- SPEC CPU2006
- SPEC CPU2006
## Results: Real Code Reuse Attacks

<table>
<thead>
<tr>
<th>Exploit</th>
<th>Type</th>
<th>Detected</th>
<th>Detection Component</th>
<th>Why Invalid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>17286(a)</td>
<td>ROP</td>
<td>✓</td>
<td>Invalid stack frame height</td>
<td>Overwritten return address</td>
</tr>
<tr>
<td>17286(b)</td>
<td>ROP</td>
<td>✓</td>
<td>Invalid stack frame height</td>
<td>Overwritten return address</td>
</tr>
<tr>
<td>Rsync</td>
<td>ROP</td>
<td>✓</td>
<td>Invalid stack frame height</td>
<td>Overwritten return address</td>
</tr>
<tr>
<td>Bletsch</td>
<td>JOP</td>
<td>✓</td>
<td>Invalid stack frame height</td>
<td>Gadget executing</td>
</tr>
<tr>
<td>Stack-smash</td>
<td>Stack-smash</td>
<td>✓</td>
<td>Invalid stack frame height</td>
<td>Overwritten return address</td>
</tr>
</tbody>
</table>

100% accuracy using real ROP and JOP exploits
Results: SPEC CPU2006

100% accuracy (0 false positives),
5.42% overhead (geometric mean)
Open Questions

- **Data-driven attacks (orthogonal type of attack)**
  [Chen et al. 2005], [Demay et al. 2011]

- **Mimicry/evasion attacks (do not exist as code reuse attacks)**
  [Giffin et al. 2006], [Wagner and Soto 2002]
Conclusion

- We defined *conformant program execution* and an efficient approximation, observed *conformant program execution*.

- We built a tool to enforce OCPE, ROPStop, on top of Dyninst components.
Questions?


- Come see the demo on Tuesday