Address-Oblivious Code Reuse: On the Effectiveness of Leakage-Resilient Diversity

Robert Rudd¹, Richard Skowyra¹, David Bigelow¹, Veer Dedhia¹, Thomas Hobson¹, Stephen Crane², Christopher Liebchen⁴, Per Larsen³, Lucas Davi⁵, Michael Franz³, Ahmad-Reza Sadeghi⁴, Hamed Okhravi¹

1: MIT Lincoln Laboratory  2: Immunant, Inc.  3: UC Irvine  4: TU Darmstadt  5: University of Duisburg-Essen
• Code diversity techniques are vulnerable to information leakage

• Recent leakage-resilient techniques employ “execute-only” memory permissions to prevent information leakage

• We present a generic type of attack called *Address-Oblivious Code Reuse (AOCR)* that can bypass recent leakage-resilient techniques

• We provide 3 real-world exploits
Memory Corruption Attacks

Spatial Memory Violation

Temporal Memory Violation
Code Diversification Techniques

- Address Space Layout Randomization (ASLR)

- Compile-Time Diversity
  - Diverse binaries

- Binary Rewriting
  - Diverse binaries

Information Leakage (Direct Memory Disclosure)

Buffer

Diversified Code

Attacker
Information Leakage (Indirect Memory Disclosure)
Memory Permissions

- Readable
- Writable (RW)
- Readable Executable (RX)

Data

Diversified Code
Leakage Resilient Diversity

Executable Only (X)

Readable Writable (RW)

Data

Diversified Code

Direct Leakage

Attacker
Indirect Leakage Prevention

Executable Only (X)

Readable Writable (RW)

*fptr 1

Stack

Diversified Code

Readable Writable (RW)

Indirect code pointer

Stack

Trampoline

Diversified Code

Trampoline

Readable Writable (RW)

Executable Only (X)
Research Questions

- Indirect code pointers create a surrogate for code
- Can attackers reuse code at the granularity of indirect code pointers?
- Can they accurately identify the corresponding functions?
- Can they chain indirect code pointers together?
• Goal: identify the function corresponding to each indirect code pointer

**Profiling Indirect Code Pointers**

- **icptr**
  - Stack
  - Readable Writable (RW)
  - Trampoline
- **open()**
  - Stack
  - Executable Only (X)
  - Diversified Code

Leakage from Attacker's Local Copy of Target Application to Remote Target Application
Address-Oblivious Code Reuse

Address-Oblivious Code Reuse (AOCR) Gadget

Remote Target Application

Executable Only (X)

Readable Writable (RW)

icptr

Stack

Diversified Code

Trampoline
Accurate Profiling

Remote Target Application

Readable Writable (RW)

icptr 3
icptr 5
icptr 2
icptr 4
Stack

Is it too volatile?
Accurate Profiling using Malicious Thread Blocking (MTB)

- A thread can force another threat to halt by maliciously setting a mutex
- Mutexes are readily accessible is memory
Chaining Gadgets Together using Malicious Loop Redirection (MLR)

```c
while (task) {
    task->fptr(task->arg);
    task = task->next;
}
```

**Normal Loop**

**Maliciously Redirected Loop**

Data

Normal Loop

Func tramp 1

Func tramp 2
Nginx Proof-of-Concept Exploit

1. Locate a mutex for MTB

2. Profile an indirect code pointer for open (1\textsuperscript{st} AOCR gadget)

3. Profile an indirect code pointer for \_IO\_new\_file\_overflow (2\textsuperscript{nd} AOCR gadget)

4. Corrupt Nginx’s task queue to call our profiled trampolines using MLR
Forged Direct Memory Access (FDMA)
- A malicious application forges a software-based DMA call to kernel
- Uses O_DIRECT flag in Linux
- DMA request bypasses memory permissions

Procfs
- Ubiquitous facility in Linux
- Provides memory maps and addresses
- Blocking it breaks many benign applications
- Protections such as GRSecurity’s permissions will not block it
## Impact on Leakage-Resilient Diversity Techniques

<table>
<thead>
<tr>
<th></th>
<th>Direct Leak</th>
<th>Indirect Leak</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TLB-mediated (Buffer Overread)</td>
<td>Non-TLB-mediated (Forged DMA)</td>
</tr>
<tr>
<td>PointGuard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxymoron</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Isomeron</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XnR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HideM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Readactor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heisenbyte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASLR-Guard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Possible Countermeasures

• Complete memory safety

• Data randomization

• Authentication of indirect calls and returns
  – Use HMAC tokens to disallow redirection of indirect code pointers
  – Similar to cryptographically-enforced CFI (CCFI)
Conclusion

• Code pointers pose a major challenge to leakage-resilient diversity

• AOCR attacks bypass code pointer obfuscation by profiling indirect code pointers

• Malicious threat blocking (MTB) allows accurate profiling

• Malicious loop redirection (MLR) allows chaining AOCR gadgets

• Effective defenses should incorporate aspects of *diversification* and *enforcement*
Questions?