Subversive-C: Abusing and Protecting Dynamic Message Dispatch

Julian Lettner, Benjamin Kollenda, Andrei Homescu, Per Larsen, Felix Schuster, Lucas Davi, Ahmad-Reza Sadeghi, Thorsten Holz, Michael Franz
Control Flows

- return
- jump
- ical
- vcall
- msgSend

- canaries
- shadow stacks
- safestack

- virtual table randomization
- control-flow integrity
Control Flow Hijacking

Variable A → Function foo

fn ptr

addr = Load(A);
Goto(addr);
Control Flow Hijacking

```
addr = Load(A);
Goto(addr);
```

```
function foo

system("/bin/sh")
```

variable \texttt{A} \quad \text{fn ptr}
Objective-C

Smalltalk-style object orientation

Good ‘ol C
Message Dispatch

**C++**

A *obj = new A;
obj->foo();

- Caller “calls a method” in object
- Resolved using vtables
- Static class structure

**Objective-C**

A *obj = [[A alloc] init];
[obj foo];

- Caller “sends a message” to object
- Resolved dynamically at run-time
- Dynamic class structure
Class Mutability

```c
void fooIMP(id self, SEL _cmd) {}

A *obj = [[A alloc] init];
class_addMethod([obj class], @selector(foo),
                (IMP) fooIMP, "v@:`);
[obj foo];
```
Object Layout

- **object**
  - class
  - var 1
  - var n

- **class**
  - isa
  - super
  - cache
  - flags
  - methods

- **super class**

- **cache**
  - sel
  - impl
  - nil
  - nil

- **methods**
  - sel
  - impl
  - nil
  - nil

- **user forwarder**
  - fwd

- **super class**
Attacker Model

• Arbitrary memory read (information disclosure)

• Arbitrary memory write

• No other control flow hijacking
  • No code injection
  • No code reuse (ROP, COOP, etc.)
Previously: COOP

• COOP: Counterfeit Object-Oriented Programming

• Counterfeit objects attack for C++

• Reuses existing **vtables** (fully or partially)

• Reuses whole C++ functions

Subversive-C

fake object
- class
- var 1
- var n

fake class
- isa
- super
- cache
- flags
- methods

super class

cache
- sel
- impl
- nil
- nil

methods
- sel
- impl
- sel
- impl

user forwarder
- fwd

- gadget 1
- gadget ...
- gadget ...
- gadget N
Subversive-C

• What we have
  • Arbitrary counterfeit Objective-C objects
  • Control flow hijacking

• What we want
  • Call malicious system call, e.g., `system("/bin/sh")`
Calling `system("/bin/sh")`

1. Find the address of `system()` in GOT

2. Set up function call arguments
   - Store "/bin/sh" in memory
   - Set up argument registers/stack

3. Invoke `system()` via computed address
## Gadgets

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML-G</td>
<td>Dispatch execution to other gadgets</td>
</tr>
<tr>
<td>LOAD-R64-G</td>
<td>Load register from Objective-C object</td>
</tr>
<tr>
<td>R-G</td>
<td>Load register from memory</td>
</tr>
<tr>
<td>ARITH-G</td>
<td>Add two registers</td>
</tr>
<tr>
<td>W-G</td>
<td>Write result to Objective-C object</td>
</tr>
<tr>
<td>INV-G</td>
<td>Call function pointer from object</td>
</tr>
</tbody>
</table>
Example: Main Loop Gadget

• Used to invoke other gadgets repeatedly (gadget loops)
• Code from dealloc in NSTextReplacementNode

```c
children = self->children;
counter = 0;
while (children[counter] != 0 && counter < 28) {
    [children[counter] release];
    counter++;
}
```
Results

• Successfully applied attack to AppKit on vulnerable PoC program

• AppKit is used by many popular Mac OS X apps
Defense: Object Layout Integrity

- No slow path attack
- No fast path attack

Classes:
- class
  - super
    - cache
      - methods

Methods:
- user forwarder
  - fwd
  - fwd hmac

No forwarder attack
No slow path attack
No fast path attack
Securing the Slow Path

- \( \text{HMAC}(K, m) = \text{HMAC-MD5}(K, m) \)

- Checked on every slow path lookup

- \( K \) is a random 64-bit key stored in execute-only memory

- \( m = \&\text{class} \parallel \text{isa} \parallel \text{superclass} \parallel \text{flags} \parallel \text{method list elements} \)
Securing the Fast Path

- **UMAC**
  - \( H_1(K, m) = \sum_{i=0}^{i<3} (m_L[i] + KL[i]) \times (m_H[i] + KH[i]) \)

- **K** is a 192-bit random number stored in execute-only memory

- **m** = `&class || sel || impl`

\[ H_I(K, m) \]
eXecute-only Memory

- Crucial defense against information leaks
- Store HMAC keys in XoM (write-once or constant data)
- Access via execution
- Can be implemented in hardware or software
  - `mprotect()`-based mechanism
  - TLB splitting
  - EPT on x86
  - ARMv8 native support
Performance Evaluation

• Drop-in replacement for Objective-C runtime shipped by Apple!
  • Micro-benchmarks
  • iTunes, Pages, etc.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>msgSend calls</th>
<th>Calls/ms</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatch</td>
<td>10,000,000,215</td>
<td>190583</td>
<td>106.46 %</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>2,986,070,515</td>
<td>173527</td>
<td>88.66 %</td>
</tr>
<tr>
<td>Sorts</td>
<td>13,329,480,611</td>
<td>82597</td>
<td>34.54 %</td>
</tr>
<tr>
<td><strong>Average</strong> (micro)</td>
<td>148902</td>
<td></td>
<td><strong>76.55 %</strong></td>
</tr>
<tr>
<td>XML-100</td>
<td>7,940,898</td>
<td>6475</td>
<td>2.81 %</td>
</tr>
<tr>
<td>XML-1000</td>
<td>78,119,698</td>
<td>6386</td>
<td>1.97 %</td>
</tr>
<tr>
<td>iTunes play</td>
<td>8,592,257</td>
<td>1667</td>
<td>0.37 %</td>
</tr>
<tr>
<td>iTunes enc.</td>
<td>114,948</td>
<td>29</td>
<td>1.82 %</td>
</tr>
<tr>
<td>Pages PDF</td>
<td>78,691</td>
<td>46</td>
<td>0.75 %</td>
</tr>
<tr>
<td><strong>Average</strong> (application)</td>
<td>2921</td>
<td></td>
<td><strong>1.54 %</strong></td>
</tr>
</tbody>
</table>
Summary

• Control flow hijacking attack on Objective-C message dispatch

• HMAC-based object integrity defense for Apple Objective-C runtime

• Low performance overhead (1.54% on real-world applications)
Questions?

Previous joint work open sourced and released into
Hardened Tor Browser for Linux
https://github.com/immunant/selfrando