Q: Exploit Hardening Made Easy

Edward J. Schwartz, Thanasssis Avgerinos, and David Brumley
Carnegie Mellon University
I found a control flow hijack exploit online!

Exploit

Evil Ed

Windows 7
A problem has been detected and Windows has been shut down to prevent damage to your computer.

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to be sure you have adequate disk space. If a driver is identified in the Stop message, disable the driver or check with the manufacturer for driver updates. Try changing video adapters.

Check with your hardware vendor for any BIOS updates. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000007E (0xc0000005,0x088FF190,0x0x8975BA0,0x89758A0)

*** EPUSBDSK.sys - Address F8FF190 base at FF88FE000, datestamp 3b9f3248

Beginning dump of physical memory
Why didn’t the exploit work?

Evil Ed

Windows 7
Causes of Broken Exploits

1. Exploit used OS/binary-specific tricks/features

2. OS Defenses
OS Defenses

• Modern OS defenses are designed to make exploiting difficult
  – **ASLR**: Address Space Layout Randomization
  – **DEP**: Data Execution Prevention
  – Do not guarantee control flow integrity

• How difficult?
Exploit hardening:
Modifying exploits to bypass defenses
Overview

• Background: Defenses and Return Oriented Programming (ROP)

• Q: ROP + Hardening
  – Automatic ROP
  – Automatic Hardening

• Evaluation

• Limitations

• Conclusion
Simple Exploit

Exploit

Shellcode  Padding  Pointer

Computation  Control
Data Execution Prevention (DEP)

- **Exploit**
  - **Shellcode**
  - **Padding**
  - **Pointer**

**DEP**
- Buffers cannot be writable and executable
- User input is non-executable and non-executable

**Crash**
Bypassing DEP

• **Goal:** Specify exploit computation even when DEP is enabled

• **Return Oriented Programming** [S07]
  – Use existing instructions from program in special order to encode computation
Example: How can we write to memory without shellcode?
Return Oriented Programming

Exploit

<table>
<thead>
<tr>
<th>nextaddr</th>
<th>addr3</th>
<th>address</th>
<th>addr2</th>
<th>value</th>
<th>stack</th>
</tr>
</thead>
</table>

Gadgets

<table>
<thead>
<tr>
<th>addr1</th>
<th>pop %eax</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr2</td>
<td>pop %ebx</td>
<td>ret</td>
</tr>
<tr>
<td>addr3</td>
<td>movl %eax, (%ebx)</td>
<td>ret</td>
</tr>
</tbody>
</table>
Address Space Layout Randomization (ASLR)

ASLR disabled

ASLR enabled

ASLR: Addresses are unpredictable
Return Oriented Programming + ASLR

- **Bad news:** Randomized code can’t be used for ROP

- **Good news:** ASLR implementations leave small amounts of code unrandomized

Evil Ed
Consequences

• **Challenge:** Program image is often the only unrandomized code
  - Small
  - Program-specific

• Prior work on ROP assumes unrandomized large code bases; can’t simply reuse

• We developed new automated ROP techniques for targeting the program image
Overview

• Background: Defenses and Return Oriented Programming (ROP)

• Q: ROP + Hardening
  – Automatic ROP
  – Automatic Hardening

• Evaluation

• Limitations

• Conclusion
Automatic ROP Overview

Source P → Instructions from P

Computation
ROP Overview

Source P → Discovery → Assignment → Arrangement → Computation
Gadget Discovery

• **Gadget Discovery**: Does instruction sequence do something we can use for our computation?

• Fast randomized test for **every program location** (thousands or millions)

```
sbb %eax, %eax;
neg %eax; ret
```
Randomized Testing

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX 0x0298a7bc</td>
<td>EAX 0x1</td>
</tr>
<tr>
<td>CF 0x1</td>
<td>ESP 0x81e4f108</td>
</tr>
<tr>
<td>ESP 0x81e4f104</td>
<td>EBX 0x0298a7bc</td>
</tr>
</tbody>
</table>

sbb %eax, %eax;
```
        neg %eax; ret
```

If 10 random runs satisfy a semantic definition, then Q probably found a gadget of that type.
<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Semantic Definition</th>
<th>Real World Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveRegG</td>
<td>Out &lt;- In</td>
<td>xchg %eax, %ebp; ret</td>
</tr>
<tr>
<td>LoadConstG</td>
<td>Out &lt;- Constant</td>
<td>pop %ebp; ret</td>
</tr>
<tr>
<td>ArithmeticG</td>
<td>Out &lt;- In1 + In2</td>
<td>add %edx, %eax; ret</td>
</tr>
<tr>
<td>LoadMemG</td>
<td>Out &lt;- M[Addr + Offset]</td>
<td>movl 0x60(%eax), %eax; ret</td>
</tr>
<tr>
<td>StoreMemG</td>
<td>M[Addr + Offset] &lt;- In</td>
<td>mov %dl, 0x13(%eax); ret</td>
</tr>
<tr>
<td>ArithmeticLoadG</td>
<td>Out +&lt;- M[Addr + Offset]</td>
<td>add 0x1376dbe4(%ebx), %ecx; (...) ; ret</td>
</tr>
<tr>
<td>ArithmeticStoreG</td>
<td>M[Addr + Offset] +&lt;- In</td>
<td>add %al, 0x5de474c0(%ebp); ret</td>
</tr>
</tbody>
</table>
# Q’s Gadget Types

<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Semantic Definition</th>
<th>Real World Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveRegG</td>
<td>Out &lt;- In</td>
<td>xchg %eax, %ebp; ret</td>
</tr>
<tr>
<td>LoadConstG</td>
<td>Out &lt;- Constant</td>
<td>pop %ebp; ret</td>
</tr>
<tr>
<td>ArithmeticG</td>
<td>Out &lt;- In1 + In2</td>
<td>add %edx, %eax; ret</td>
</tr>
<tr>
<td>LoadMemG</td>
<td>Out &lt;- M[Addr + Offset]</td>
<td>movl 0x60(%eax), %eax; ret</td>
</tr>
<tr>
<td>StoreMemG</td>
<td>M[Addr + Offset] &lt;- In</td>
<td>mov %dl, 0x13(%eax); ret</td>
</tr>
<tr>
<td>ArithmeticLoadG</td>
<td>Out +&lt;- M[Addr + Offset]</td>
<td>add 0x1376dbe4(%ebx), %ecx; (...) ; ret</td>
</tr>
<tr>
<td>ArithmeticStoreG</td>
<td>M[Addr + Offset] +&lt;- In</td>
<td>add %al, 0x5de474c0(%ebp); ret</td>
</tr>
</tbody>
</table>
## Q’s Gadget Types

<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Semantic Definition</th>
<th>Real World Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveRegG</td>
<td>Out &lt;- In</td>
<td><code>xchg %eax, %ebp; ret</code></td>
</tr>
<tr>
<td>LoadConstG</td>
<td>Out &lt;- Constant</td>
<td><code>pop %ebp; ret</code></td>
</tr>
<tr>
<td>ArithmeticG</td>
<td>Out &lt;- In1 + In2</td>
<td><code>add %edx, %eax; ret</code></td>
</tr>
<tr>
<td>LoadMemG</td>
<td>Out &lt;- M[Addr + Offset]</td>
<td><code>movl 0x60(%eax), %eax; ret</code></td>
</tr>
<tr>
<td>StoreMemG</td>
<td>M[Addr + Offset] &lt;- In</td>
<td><code>mov %dl, 0x13(%eax); ret</code></td>
</tr>
<tr>
<td>ArithmeticLoadG</td>
<td>Out +&lt;- M[Addr + Offset]</td>
<td><code>add 0x1376dbe4(%ebx), %ecx; (...); ret</code></td>
</tr>
<tr>
<td>ArithmeticStoreG</td>
<td>M[Addr + Offset] +&lt;- In</td>
<td><code>add %al, 0x5de474c0(%ebp); ret</code></td>
</tr>
</tbody>
</table>
Randomized Testing

• Randomized testing tells us we likely found a gadget
  – Fast; filters out many candidates
  – Enables more expensive second stage

• Second stage: SMT-based gadget discovery
  – Gadget discovery is program verification
SMT-Based Gadget Discovery

sbb %eax, %eax
neg %eax; ret
EAX <- CF

[D76]
Weakest Precondition

F

SMT Validity Check

Valid (Gadget)
Invalid (not Gadget)
**SMT-Based Gadget Discovery**

- Q is better at finding gadgets than I am!

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imul $1, %eax, %ebx</td>
<td>Move %eax to %ebx</td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lea (%ebx,%ecx,1), %eax</td>
<td>Store %ebx+%ecx in %eax</td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbb %eax, %eax; neg %eax</td>
<td>Move carry flag to %eax</td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>
Gadget Arrangement

• **Gadget Arrangement:** How can gadget types be combined to implement a computation?

• Alternate view: Compile user computation for gadget type architecture

• **Example:**
  \[ M[0xcafecafe] := 0xdeadbeef \]
Arrangement: Storing to Memory

T1

LoadConst
deadbeef

Value

T2

LoadConst
cafecafe

Address

T3

StoreMem, u32
How can we write to memory without StoreMem?
Arrangement: Storing to Memory

T1
LoadConst 0

T2
LoadConst cafecafe

T3
Writes zero to M[cafecafe]

Value

Address

ArithmeticStore, u32, Bitwise And
Arrangement: Storing to Memory

T1

LoadConst
deadbeef

Value

T2

Load
cafecafe

T3

ArithmeticStore, u32, Plus

Adds deadbeef to M[cafecafe].

0 + deadbeef = deadbeef

V

Value

0

Dead

Address

Adds
deadbeef
to
M[cafecafe].

0 + deadbeef =
deadbeef
Gadget Arrangement

- Gadgets types are often unavailable
  - Synthesize alternatives on the fly

- Flexible arrangement rules are necessary for small code bases
ROP Overview

Source P → Discovery → Assignment → Arrangement → Computation
Assignment

• **Gadget Assignment**: Assign concrete gadgets found in source program to arrangements

• Assignments must be **compatible**
Assignment: Register Mismatch

CONFLICT
%ebx and %ecx mismatch

LoadConst
deadbeef
pop %eax
ret

LoadConst
cafecafe
pop %ebx
ret

StoreMem, u32
mov %eax, (%ecx)
ret
Gadget Assignment

• Need to search over
  – Gadgets
  – Schedules

• We developed dynamic programming approach to find assignment

• Easy to print payload bytes with assignment
Overview

• Background: Defenses and Return Oriented Programming (ROP)

• Q: ROP + Hardening
  – Automatic ROP
  – **Automatic Hardening**

• Evaluation

• Limitations

• Conclusion
Exploit Hardening

Old Exploit (stopped by DEP+ASLR)

ROP Payload

Hardened Exploit (bypasses DEP+ASLR)
Trace-based Analysis

- Record P on the old exploit

Stop at vulnerability condition
Reasoning about Executions

Symbolic Execution

[SAB10]

Logical Formula For All Inputs On Path
Exploit Constraints

Path

Exploit
Exploit Constraints

How do we ensure the ROP payload gets in the exploit?

M[ESP] = &gadget1
M[ESP+off1] = &gadget2
M[ESP+off2] = &gadget3
Demo!
Overview

• Background: Defenses and Return Oriented Programming (ROP)

• Q: ROP + Hardening
  – Automatic ROP
  – Automatic Hardening

• Evaluation

• Limitations

• Conclusion
Evaluation Questions

1. Can Q harden exploits for real binary programs?

2. How much unrandomized code is sufficient to create ROP payloads?
Real Exploits

- Q was able to **automatically harden** nine exploits downloaded from exploit-db.com

<table>
<thead>
<tr>
<th>Name</th>
<th>Total Time</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free CD to MP3 Converter</td>
<td>130s</td>
<td>Windows 7</td>
</tr>
<tr>
<td>Fatplayer</td>
<td>133s</td>
<td>Windows 7</td>
</tr>
<tr>
<td>A-PDF Converter</td>
<td>378s</td>
<td>Windows 7</td>
</tr>
<tr>
<td>A-PDF Converter (SEH exploit)</td>
<td>357s</td>
<td>Windows 7</td>
</tr>
<tr>
<td>MP3 CD Converter Pro</td>
<td>158s</td>
<td>Windows 7</td>
</tr>
<tr>
<td>rsync</td>
<td>65s</td>
<td>Linux</td>
</tr>
<tr>
<td>opendchub</td>
<td>225s</td>
<td>Linux</td>
</tr>
<tr>
<td>gv</td>
<td>237s</td>
<td>Linux</td>
</tr>
<tr>
<td>Proftpd</td>
<td>44s</td>
<td>Linux</td>
</tr>
</tbody>
</table>
ROP Probability

• Given program size, what is the probability Q can create a payload?
  – Measure over all programs in /usr/bin

• Depends on target computation
  – Call functions statically or dynamically linked by the program (blue on next slide)
  – Call any function in libc (red; harder)
    • system, execv, connect, mprotect, ...
ROP Probability

Call linked functions in 80% of programs \(\geq\) true (20KB)

Call libc functions in 80% of programs \(\geq\) nslookup (100KB)
Overview

• Background: Defenses and Return Oriented Programming (ROP)
• Q: ROP + Hardening
  – Automatic ROP
  – Automatic Hardening
• Evaluation
• Limitations
• Conclusion
Limitations

• Single path (trace-based) analysis
  – restrictive; prevents finding exploits

• Q’s gadgets types are not Turing-complete
  – Calling system(“/bin/sh”) or mprotect() usually enough
  – Comparison with related work

• Q cannot find conditional gadgets
  – Potential automation of interesting work on ROP without Returns [CDSSW10]
Overview

- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - Automatic ROP
  - Automatic Hardening
- Evaluation
- Limitations
- Conclusion
Conclusion

• We built Q, a system that **automatically hardens exploits** to bypass defenses
  – **Challenge:** Reusing small amounts of code

• Q **automatically hardened nine** real exploits found in the wild against latest OS defenses

• **Takeaway:** Unrandomized code is dangerous
  – 20KB makes DEP+ASLR ineffective
Thanks! 😊

• Questions?

• Check out some of the gadgets Q can find at http://plaid.cylab.cmu.edu:8080/~ed/gadgets

Edward J. Schwartz

edmcman@cmu.edu

http://www.ece.cmu.edu/~ejschwar