Can DREs Provide Long-Lasting Security?  

The Case of Return-Oriented Programming and the AVC Advantage

Stephen Checkoway,* Ariel J. Feldman,† Brian Kantor,* J.Alex Halderman,‡ Edward W. Felten,† Hovav Shacham*

*UCSD, †Princeton, ‡U Michigan
## Voting System Studies

<table>
<thead>
<tr>
<th>Study</th>
<th>Vendors</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appel et al.</td>
<td>Sequoia</td>
<td>2008</td>
</tr>
<tr>
<td>EVEREST</td>
<td>ES&amp;S, Hart, Premier</td>
<td>2007</td>
</tr>
<tr>
<td>California TTBR</td>
<td>Hart, Premier, Sequoia</td>
<td>2007</td>
</tr>
<tr>
<td>Feldman et al.</td>
<td>Diebold</td>
<td>2006</td>
</tr>
<tr>
<td>Hursti</td>
<td>Diebold</td>
<td>2006</td>
</tr>
<tr>
<td>Kohno et al.</td>
<td>Diebold</td>
<td>2003</td>
</tr>
</tbody>
</table>
The proposed 'red team' concept also contemplates giving attackers access to source code, which is unrealistic and dangerous if not strictly controlled by test protocols. It is the considered opinion of election officials and information technology professionals that ANY system can be attacked if source code is made available. We urge the Secretary of State not to engage in any practice that will jeopardize the integrity of our voting systems.

— California Association of Clerks and Election Officials, 2007
Response

No computer system could pass the assault made by your team of computer scientists. In fact, I think my 9 and 12-year-old kids could find ways to break into the voting equipment if they had unfettered access.
– Santa Cruz County Clerk Gail Pellerin, 2007

Letting the hackers have the source codes, operating manuals and unlimited access to the voting machines “is like giving a burglar the keys to your house.”
– Contra Costa County Clerk-recorder and head of the state Association of Clerks and Election Officials Steve Weir, quoted by sfgate.com, 2007

Your guidelines suggest that you will provide concept also source code to an expert and ask that person access to source code to subvert the system, which is almost certain and dangerous if not strictly controlled by test protocols.
– California Association of Clerks and Election Officials, 2007

By any standard – academic or common sense – the study is unrealistic and inaccurate.
– Diebold Election Systems, 2006

Company officials have said the researchers were given unusual access to the machines that real-world hackers could never gain.
– Mercury News, 2007

Putting isolated technology in the hands of computer experts in order to engage in unrestricted, calculated, advanced and malicious attacks is highly improbable in a real-world election.
– Hart InterCivic, 2007

Response

In short, the Red Team was able to, using a financial institution as an example, take away the locked front door of the bank branch, remove the security guard, remove the bank tellers, remove the panic alarm that notifies law enforcement, and have only slightly limited resources (particularly time and knowledge) to pick the lock on the bank vault.

Company officials have said the researchers were given unusual access to the machines that real-world hackers could never gain.
– Mercury News, 2007

No computer system could pass the assault made by your team of computer scientists. In fact, I think my 9 and 12-year-old kids could find ways to break into the voting equipment if they had unfettered access.
– Santa Cruz County Clerk Gail Pellerin, 2007

Company officials have said the researchers were given unusual access to the machines that real-world hackers could never gain.
– Mercury News, 2007

Putting isolated technology in the hands of computer experts in order to engage in unrestricted, calculated, advanced and malicious attacks is highly improbable in a real-world election.
– Hart InterCivic, 2007

No computer system could pass the assault made by your team of computer scientists. In fact, I think my 9 and 12-year-old kids could find ways to break into the voting equipment if they had unfettered access.
– Santa Cruz County Clerk Gail Pellerin, 2007

Company officials have said the researchers were given unusual access to the machines that real-world hackers could never gain.
– Mercury News, 2007

Putting isolated technology in the hands of computer experts in order to engage in unrestricted, calculated, advanced and malicious attacks is highly improbable in a real-world election.
– Hart InterCivic, 2007

Long Lasting Security: EVT’09
Is it practical to hack a voting machine without “unreasonable” access?

Hint: Yes
AVC Advantage

- Best-case to study
- Only does one thing: count votes
- Defenses against code injection
Challenges

1. Understand how the machine works without source code or documentation by reverse-engineering

2. Find an exploitable bug

3. Defeat code-injection defense using recently developed techniques from system security
Reverse-Engineering

Z80

ROMs

Long Lasting Security: EVT’09

Monday, August 10, 2009

; = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

; SUBROUTINE

; memcopy( from, to, size )
; returns 1 in bc on success and 0 if size = 0

memcopy:

1d h1, 2
add h1, sp
1d e, (h1)
inc h1
1d d, (h1)
push de
inc h1
1d e, (h1)
inc h1
1d d, (h1)
inc h1
1d c, (h1)
inc h1
1d b, (h1)
pop h1
1d a, b
or c
jr z, zero_copy
ld a, bc

zero_copy:
ret

; End of function memcopy

; CODE XREF:
Artifacts Produced

- Hardware Functional Specifications
- Hardware Simulator
  - Initial version by Joshua Herbach
  - Exploit developed on the simulator — tested on machine, worked first try
Exploit

- Classic stack-smashing buffer overflow
- Roughly a dozen bytes overwritten
- Exploit code needs to be in memory
- For now, assume we can inject code
Vote-Stealing Attack

- Gain physical access
- Malicious auxiliary cartridge
- Trigger exploitable bug
- Follow instructions
Vote-Stealing Attack
Gain physical access
Malicious auxiliary cartridge
Trigger exploitable bug
Follow instructions

Remove sploit cart
Turn Power off
Vote-Stealing Program

- Survives turning power switch to off
- Runs election as normal
- Silently shifts votes

---

Long Lasting Security: EVT’09

Monday, August 10, 2009
Vote-Stealing Program

President

E1

Benedict Arnold V

George Washington

In Votes

In Votes in Memory

Total

***

(1)

2

1

Long Lasting Security: EVT’09
Code Injection?

- Earlier, we assumed we could inject code
- Hardware interlock prevents fetching instructions from RAM
- Program code in read-only memory
Harvard Architecture

- Program in read-only memory
- Nonexecutable, writable data memory

No code injection
Return-Oriented Programming
Return-Oriented Programming

- Arbitrary behavior without code injection
- Combine snippets of existing code
- Requires control of the call stack
- Processor/program specific
Return-Oriented Programming

- Arbitrary behavior without code injection
- Combine snippets of existing code
- Requires control of the call stack
- Processor/program specific

Instructions
Return-Oriented Programming

- Arbitrary behavior without code injection
- Combine snippets of existing code
- Requires control of the call stack
- Processor/program specific

Instructions

```
movl    $0x006f6d2e,(%eax,%ebx)
movl    0xd4(%ebp),%eax
movl    %eax,(%esp)
calll   0x0008ba11
addl    $0x1f,%eax
andl    $0xf0,%eax
subl    %eax,%esp
leal    0x20(%esp),%edx
movl    %edx,0xb4(%ebp)
jmp     0x0006d8b4
incl    0xd4(%ebp)
movl    0xd4(%ebp),%eax
movzbl  (%eax),%ecx
cmpb    $0x3a,%cl
je      0x0006d8b1
testb   %cl,%cl
movl    0xb4(%ebp),%ebx
jne     0x0006d8db
movb    $0x43,(%ebx)
movb    $0x00,0x01(%ebx)
jmp     0x0006d90d
movb    %cl,(%ebx)
incl    %ebx
incl    0xd4(%ebp)
movl    0xd4(%ebp),%eax
movzbl  (%eax),%ecx
testb   %cl,%cl
setne   %dl
cmpb    $0x3a,%cl
setne   %al
testb   %al,%dl
jne     0x0006d8cf
movb    $0x00,(%ebx)
cmpl    $0x01,0x0008a780
jne     0x0006d90d
movl    0xb4(%ebp),%edx
movl    $0x0000002f,0x04(%esp)
movl    %edx,(%esp)
calll   0x0008b9e9
testl   %eax,%eax
jne     0x0006d8b4
movl    0xb4(%ebp),%esi
movl    $0x00000002,%ecx
movl    $0x0007e270,%edi

cld
repz/cmpsb      (%esi),(%edi)
movl    $0x00000000,%eax
je      0x0006d92e
movzbl  0xff(%esi),%eax
movzbl  0xff(%edi),%ecx
subl    %ecx,%eax
testl   %eax,%eax
jel     0x0006da53
movl    0xb4(%ebp),%esi
movl    $0x00070bbb,%edi
movl    $0x00000006,%ecx

cld
repz/cmpsb      (%esi),(%edi)
movl    $0x00000000,%edx
je      0x0006d956
movzbl  0xff(%esi),%edx
movzbl  0xff(%edi),%ecx
subl    %ecx,%edx
testl   %edx,%edx
```
Return-Oriented Programming

- Arbitrary behavior without code injection
- Combine snippets of existing code
- Requires control of the call stack
- Processor/program specific

Instructions

Stack
The Usual Method

High-level specification

```plaintext
if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
    arnold = arnold + amount
    washington = washington - amount
```
The Usual Method

High-level specification

if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
    arnold = arnold + amount
    washington = washington - amount

Assembly

movl ..., %edx
movl ..., %ecx
compl %ecx, %edx
jg winning
movl %ecx, %eax
subl %edx, %eax
shrl %eax
incl %eax
addl %eax, %edx
movl %edx, ...
subl %eax, %ecx
movl %ecx, ...
winning:
The Usual Method

High-level specification

if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
    arnold = arnold + amount
    washington = washington - amount

Assembly

movl ..., %edx
movl ..., %ecx
compl %ecx, %edx
jg winning
movl %ecx, %eax
subl %edx, %eax
shrl %eax
incl %eax
addl %eax, %edx
movl %edx, ...
subl %eax, %ecx
movl %ecx, ...

Binary

00000000  55 89 e5 53 e8 00 00 00 00 5b 8b 93 2f 00 00 00
00000010  8b 8b 2b 00 00 00 39 ca 77 17 89 c8 29 d0 d1 e8
00000020  40 01 c2 89 93 2f 00 00 00 29 c1 89 8b 2b 00 00
00000030  00 5b c9 c3

Long Lasting Security: EVT’09
The ROP Method

High-level specification

if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
    arnold = arnold + amount
    washington = washington - amount
The ROP Method

High-level specification

if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
    arnold = arnold + amount
    washington = washington - amount

Pseudo-assembly

ld t1, 0(A)
l0 t2, 2(A)
s1t t3, t2, t1
btr t3, winning
sub amt, t2, t1
srl amt, amt, 1
inc amt
sub t2, t2, amt
add t1, t1, amt
st t1, 0(A)
St t2, 2(A)
winning:
The ROP Method

High-level specification

if arnold ≤ washington:
    amount = (washington - arnold)/2 + 1
arnold = arnold + amount
washington = washington - amount

Pseudo-assembly

```
ld t1, 0(A)
ld t2, 2(A)
slt t3, t2, t1
btr t3, winning
sub amt, t2, t1
srl amt, amt, 1
inc amt
sub t2, t2, amt
add t1, t1, amt
st t1, 0(A)
st t2, 2(A)
```

winning:
The Usual Method

Sequence of instructions: %eip

Execute instruction, update %eip

Control flow by changing %eip

%eip → movl ..., %edx
    movl ..., %ecx
    movl %ecx, %edx
    compl %ecx, %edx
    jg winning
    movl %ecx, %eax
    subl %edx, %eax
    shrl %eax
    incl %eax
    addl %eax, %edx
    movl %edx, ...
    subl %eax, %ecx
    movl %ecx, ...

winning:
## The Usual Method

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl</td>
<td>move long word to register</td>
</tr>
<tr>
<td>movl</td>
<td>move long word to memory</td>
</tr>
<tr>
<td>compl</td>
<td>complement register</td>
</tr>
<tr>
<td>jg</td>
<td>jump if greater than</td>
</tr>
<tr>
<td>movl</td>
<td>move long word to register</td>
</tr>
<tr>
<td>subl</td>
<td>subtract long word</td>
</tr>
<tr>
<td>shrl</td>
<td>shift right long word</td>
</tr>
<tr>
<td>incl</td>
<td>increment register</td>
</tr>
<tr>
<td>addl</td>
<td>add long word</td>
</tr>
<tr>
<td>movl</td>
<td>move long word to register</td>
</tr>
<tr>
<td>subl</td>
<td>subtract long word</td>
</tr>
<tr>
<td>movl</td>
<td>move long word to register</td>
</tr>
<tr>
<td>winning</td>
<td>label for control flow</td>
</tr>
</tbody>
</table>

---

**Sequence of instructions:** %eip

- Execute instruction, update %eip
- Control flow by changing %eip
The Usual Method

- Sequence of instructions: %eip
- Execute instruction, update %eip
- Control flow by changing %eip

movl ..., %edx
movl ..., %ecx
compl %ecx, %edx
jg winning
movl %ecx, %eax
subl %edx, %eax
shrl %eax
incl %eax
addl %eax, %edx
movl %edx, ...
subl %eax, %ecx
movl %ecx, ...

%eip → winning:
The ROP Method

Sequence of Gadgets: %esp

Pointers to instructions

Data

Execute Gadget

ret increments %esp

Control flow by changing %esp

Long Lasting Security: EVT’09
The ROP Method

- Sequence of Gadgets: \%esp
- Pointers to instructions
- Data
- Execute Gadget
- \texttt{ret} increments \%esp
- Control flow by changing \%esp

![Diagram showing sequence of gadgets and control flow]

Long Lasting Security: EVT'09

Monday, August 10, 2009
The ROP Method

- Sequence of Gadgets: %esp
- Pointers to instructions
- Data
- Execute Gadget
- ret increments %esp
- Control flow by changing %esp

Long Lasting Security: EVT’09
ROP Example 1: No-op

Usual

%eip → nop

Just advances %eip

ROP

%esp → ret

Just advances %esp

Pointer to ret instruction
ROP Example 2: Immediate Constants

**Usual**

- `%eip` → `movl $0xdeadbeef, %eax`
- `movl $0xcafebabe, %ebx`

**ROP**

- Set `%eax` to 0xdeadbeef
- Set `%ebx` to 0xcafebabe
- Put constants on stack
- Pop them into registers

---

Long Lasting Security: EVT'09
ROP Example 3: Control Flow

Usual
%eip $\rightarrow$ jmp +16

Update %eip

ROP

%esp $\rightarrow$

Update %esp

Conditional branch possible

... ret
popl %esp
ret

Long Lasting Security: EVT’09
ROP Wrap-Up

- Use stack for program (%esp vs. %eip)
- Gadgets
  - Multiple instruction sequences & data
  - Chained together by ret
- Turing-complete
- No code injection!
ROP On The AVC Advantage

- Extended ROP to Z80
- 16 kB instruction corpus
- Turing-complete gadget set
- Some automation

```
pop hl, de
bc ← (hl)
SP
SP

sp ← sp + hl
sp

0xFFF
pop hl
sp ← sp + hl

0x0000
pop hl
(d) ← hl + bc

0x000C
pop hl
(de) ← hl + bc

d
pop hl
```

Long Lasting Security: EVT’09

Monday, August 10, 2009
Challenges Overcome

1. Reverse-engineered hardware and software
2. Found an exploitable bug in the code
3. Defeated code-injection defense using return-oriented programming
Thank you