M/o/Vfuscator-Be-Gone

Recovering from soul-crushing RE nightmares

Julian Kirsch & Clemens Jonischkeit
Technical University of Munich

June 19, 2016

Disclaimer

1. This talk is **not** an attack on Christopher Domas
2. Our demovfuscator currently performs resubstitution **very sparsely**

About us

PWNING SINCE

HXP

0x7DB

#21 in BkP16, #6 in Insomni'hack16, #12 in pCTF16,
currently #18 in ctftime

Julian

- PhD student at TUM
- program analysis, (de-)obfuscation, malware analysis, ...

Clemens

- Master's student at TUM
- Heap exploitation guru
- Bachelor's thesis on demovfuscation

Chapter 1

The Movfuscator

Movfuscator

Introduction

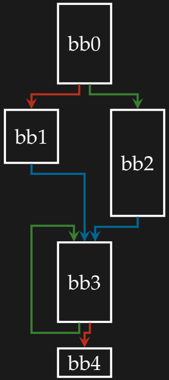
- `mov` is turing complete
- one-instruction compiler (`lcc`)
- VM implemented in `mov`
- two flavours:
  1. `movfuscator1`: BF → `x86/mov`
  2. `movfuscator2`: C → `x86/mov`

`mov` is Turing Complete. Stephen Dolan. White Paper. https://www.cl.cam.ac.uk/~sd601/papers/mov.pdf. 2013.

The M/o/Vfuscator. Christopher Domas. REcon 2015. https://github.com/xoreaxeaxeax/movfuscator. 2015

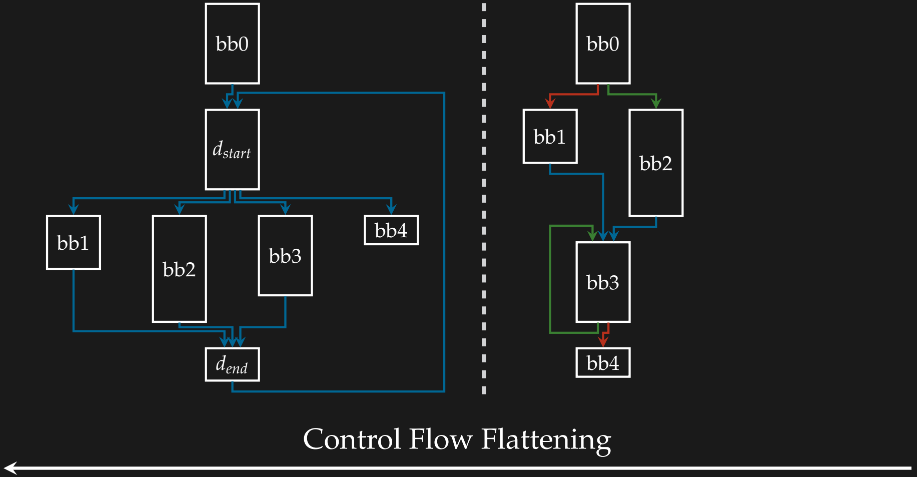
Movfuscator

Introduction



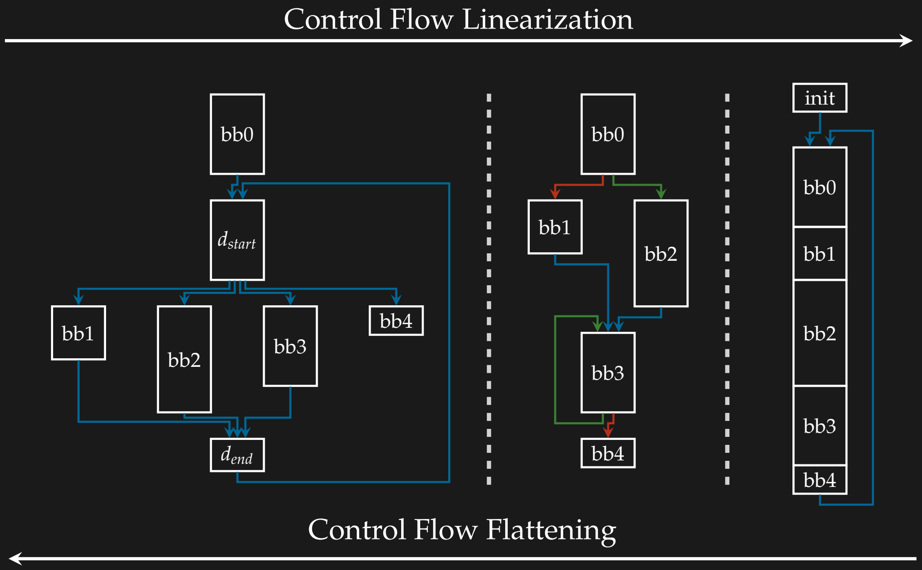
Movfuscator

Introduction



Movfuscator

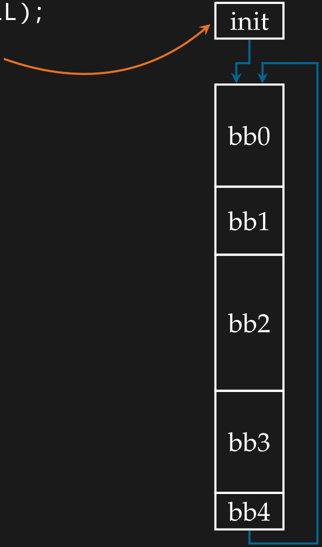
Introduction



Movfuscator

Internals

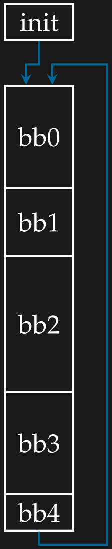
- `sigaction(SIGSEGV, {&dispatch, 0, 0, 0}, NULL);`
  → **external** library calls
- `sigaction(SIGILL, {&bb0, 0, 0, 0}, NULL);`
  → **jump** from bb4 to bb0



Movfuscator

Internals

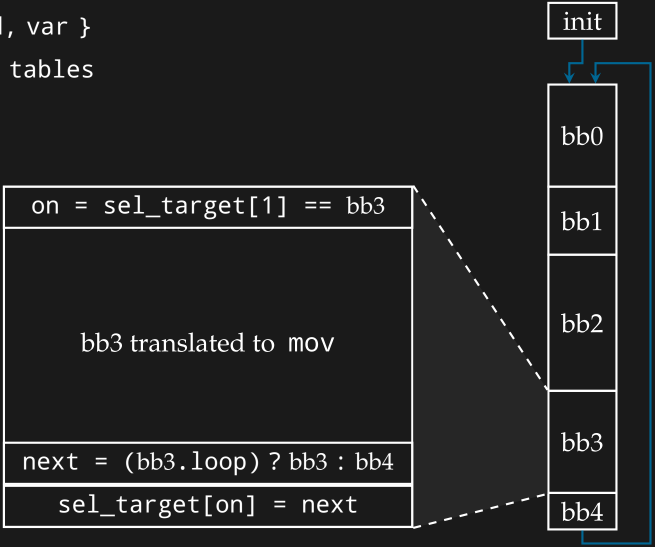
```c
#define DEFVAR(TYPE, NAME)      TYPE NAME[2] = { 0 }
#define ASSIGN(VAR, VAL, CONDVAR, CONDNUM) \
  do { VAR[TRUVAL(CONDVAR) == CONDNUM] = VAL; } while (0)
int main(int argc, char **argv)
{
  DEFVAR(size_t, state); DEFVAR(size_t, cmp);
  DEFVAR(uint64_t, fac);
  DEFVAR(size_t, i); DEFVAR(size_t, j);
  DEFVAR(void *, my_printf); DEFVAR(void *, my_exit);
  RESOLV(my_printf, "printf"); RESOLV(my_exit, "exit");

  do {
    ASSIGN(i,     1,                        state, 0);
    ASSIGN(j,     atoi(argv[1]),            state, 0);
    ASSIGN(fac,   1,                        state, 0);
    ASSIGN(state, 1,                        state, 0);
    ASSIGN(fac,   TRUVAL(fac) * TRUVAL(i), state, 1);
    ASSIGN(i,     TRUVAL(i) + 1,            state, 1);
    ASSIGN(cmp,   TRUVAL(i) > TRUVAL(j),    state, 1);
    ASSIGN(state, 2,                        cmp,   1);

    my_printf[TRUVAL(state) == 2])("%llu\n", TRUVAL(fac));
    my_exit[TRUVAL(state) == 2])(0);
  } while (1);
}
```



Movfuscator

Internals

- `sel_var := { discard, var }`
- `Arithmetic` → `lookup tables`



Movfuscator

The Movfuscator in the Real World

sample program	movfuscated		lcc + as + ld	
	time [ms]	size	time [ms]	size
`for_loop`	0.99	5.67 MiB	0.77	3.05 KiB
`primes (100)`	3.50	5.71 MiB	0.78	3.58 KiB
`tiny-aes128`	2591	6.28 MiB	1.18	12.7 KiB
`sha2-256`	81425	5.87 MiB	28.8	6.13 KiB

Movfuscator

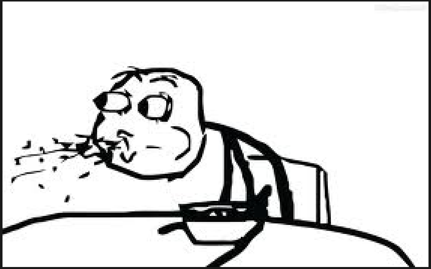
The Movfuscator in the Real World

sample program	movfuscated		lcc + as + ld	
	time [ms]	size	time [ms]	size
`for_loop`	0.99	5.67 MiB	0.77	3.05 KiB
`primes (100)`	3.50	5.71 MiB	0.78	3.58 KiB
`tiny-aes128`	2591	6.28 MiB	1.18	12.7 KiB
`sha2-256`	81425	5.87 MiB	28.8	6.13 KiB

Movfuscator

The Movfuscator in the Real World

CTF	Challenge
Hackover CTF 2015	`move_it`
0ctf 2016	`momo`
Google CTF 2016	`guessme`



Movfuscator

The Movfuscator in the Real World

Technical University of Munich

Movfuscator version	Language	Generated Code	Easy solution?
`movfuscator1`	BF	static	✓
`movfuscator2`	C	static	✓
`movfuscator2` + hardening	C	shuffled	!

Chapter 2
The Demovfuscator

Demovfuscator

Goals

Goals:

1. Recovery of the control flow
2. Recovery of Symbols
3. Substitution of lookups

Demovfuscator

Control Flow Recovery

Technical University of Munich

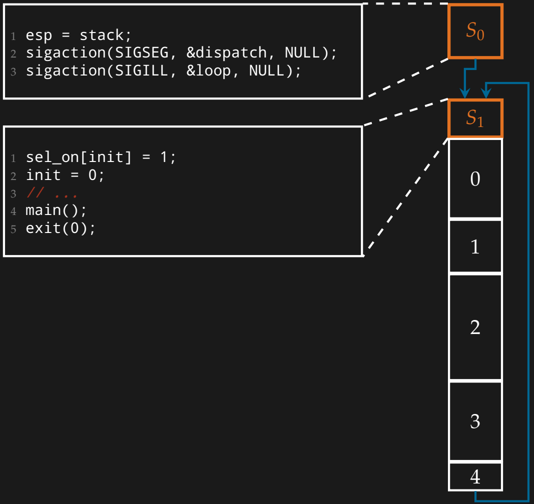
4 Stages:

1. Analyzing the setup
2. Recovering labels
3. Finding jump targets
4. Building the CFG

Technical University of Munich

Analyzing the setup:

- Signal handler - main loop
- Stack setup
- Initializing ON
- excepted from hardening



```
esp = stack;
sigaction(SIGSEG, &dispatch, NULL);
sigaction(SIGILL, &loop, NULL);
```

```
sel_on[init] = 1;
init = 0;
// ...
main();
exit(0);
```

Demovfuscator

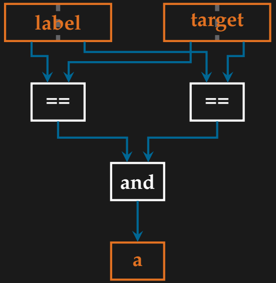
Control Flow Recovery

Technical University of Munich

Recovering the Labels:

Analyzing accesses to sel_on:

```
a = label == target;
sel_on[a] = 1;
```



Demovfuscator

Control Flow Recovery

Beating lookup tables:

Boolean:

- set bits according to the result
- look up the result

operation	value
and	0x8
or	0xE
exclusive or (xor)	0x6
equality (xnor)	0x9

Binary:

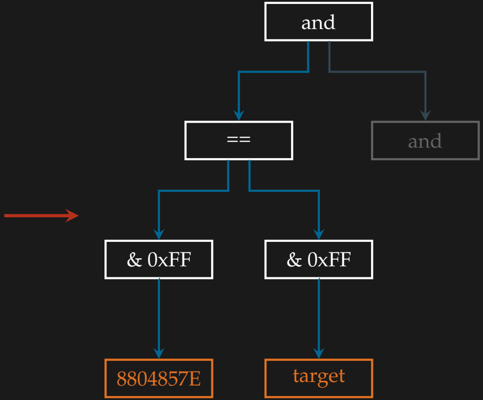
- accessing [0x7][0xCB]
- look up the result

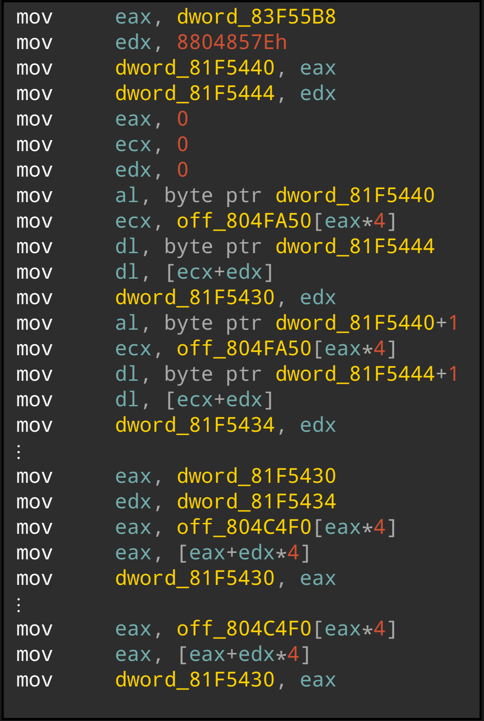
name	value
bit set	0xCB
bit clear	0x4B
and	0x3
or	0xCF
xor	0xCC
mul_l	0x8D
mul_h	0x5

Unary:

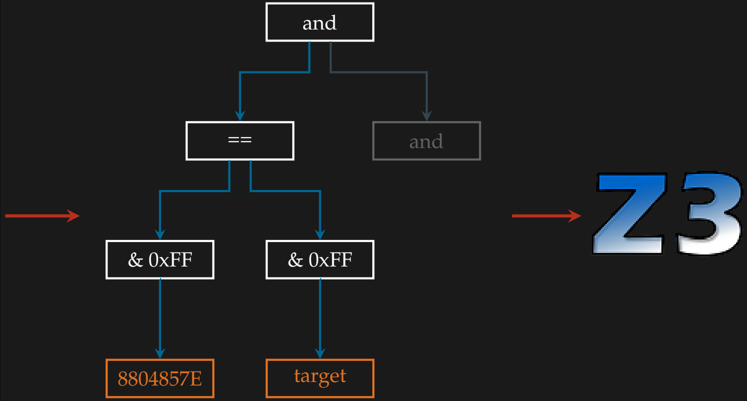
- hashing the tables
- comparing the hashes

```
mov     eax, dword_83F55B8
mov     edx, 8804857Eh
mov     dword_81F5440, eax
mov     dword_81F5444, edx
mov     eax, 0
mov     ecx, 0
mov     edx, 0
mov     al, byte ptr dword_81F5440
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444
mov     dl, [ecx+edx]
mov     dword_81F5430, edx
mov     al, byte ptr dword_81F5440+1
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444+1
mov     dl, [ecx+edx]
mov     dword_81F5434, edx
⋮
mov     eax, dword_81F5430
mov     edx, dword_81F5434
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
⋮
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
mov     eax, dword_83F55B8
mov     edx, 8804857Eh
mov     dword_81F5440, eax
mov     dword_81F5444, edx
mov     eax, 0
mov     ecx, 0
mov     edx, 0
mov     al, byte ptr dword_81F5440
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444
mov     dl, [ecx+edx]
mov     dword_81F5430, edx
mov     al, byte ptr dword_81F5440+1
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444+1
mov     dl, [ecx+edx]
mov     dword_81F5434, edx
⋮
mov     eax, dword_81F5430
mov     edx, dword_81F5434
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
⋮
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
```





```
mov     eax, dword_83F55B8
mov     edx, 8804857Eh
mov     dword_81F5440, eax
mov     dword_81F5444, edx
mov     eax, 0
mov     ecx, 0
mov     edx, 0
mov     al, byte ptr dword_81F5440
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444
mov     dl, [ecx+edx]
mov     dword_81F5430, edx
mov     al, byte ptr dword_81F5440+1
mov     ecx, off_804FA50[eax*4]
mov     dl, byte ptr dword_81F5444+1
mov     dl, [ecx+edx]
mov     dword_81F5434, edx
:
mov     eax, dword_81F5430
mov     edx, dword_81F5434
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
:
mov     eax, off_804C4F0[eax*4]
mov     eax, [eax+edx*4]
mov     dword_81F5430, eax
```



Demovfuscator

Control Flow Recovery

Identifying jump targets:

Jump / Call:

```
tar_ptr = sel_tar[on];
*tar_ptr = label;
```

Return:

```
x = *stack_ptr;
stack_ptr++;
tar_ptr = sel_tar[on];
*tar_ptr = x;
```

Conditional Jump:

```
tar_ptr = sel_tar[condition];
*tar_ptr = label;
```

Indirect jump:

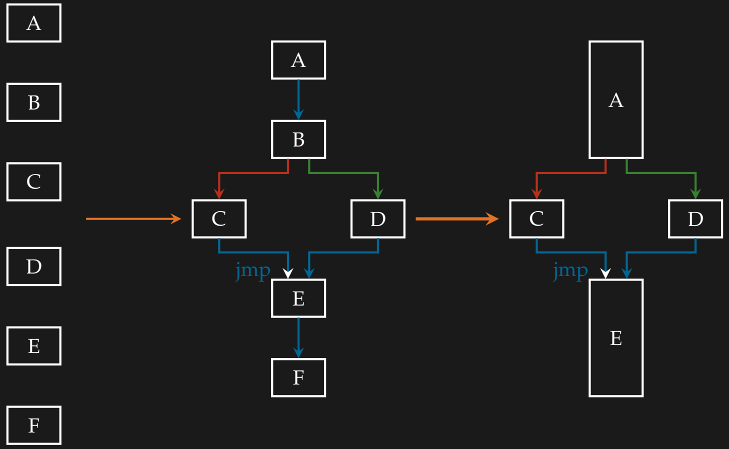
```
x = ...
tar_ptr = sel_tar[on]
*tar_ptr = x
```

remember targets on toggleing execution off

Demovfuscator

Control Flow Recovery

Generating the Graph:



So What?

“You don’t remove all the `movs`!”

Yes, but `demov` is able to . . .

- handle **hardened** executables.
- reconstruct **functions** and their **CFG**s.
- generate a **patched** binary.
- generate **symbols** for IDA.
- perform partial instruction **re-substitution**.

→ Makes reversing `moved` binaries much easier

Demo

Simple crackme:

```c
#include <stdio.h>
#include <stdint.h>
#include <string.h>

int main(int argc, char **argv)
{
  char sol[0x20];
  static int res;

  fgets(sol, sizeof(sol), stdin);
  sol[12] = 0;

  sol[0] ^= 0x01; sol[1] ^= 0x23; sol[2] ^= 0x45; sol[3] ^= 0x67;
  sol[4] ^= 0x89; sol[5] ^= 0xab; sol[6] ^= 0xcd; sol[7] ^= 0xef;
  *(uint32_t *)&sol[8] ^= 0xdeadbeef;
  res = memcmp(sol, "\x31\x5b\x24\x38\xfb\xce\xae\x80\x81\xd3\xd9\xb2", 12);
  if (res)
    puts(":(");
  else
    puts(":)");
}
```

Demo

Simple crackme:

```c
#include <stdio.h>
#include <stdint.h>
#include <string.h>

int main(int argc, char **argv)
{
  char sol[0x20];
  static int res;

  fgets(sol, sizeof(sol), stdin);
  sol[12] = 0;

  sol[0] ^= 0x01; sol[1] ^= 0x23; sol[2] ^= 0x45; sol[3] ^= 0x67;
  sol[4] ^= 0x89; sol[5] ^= 0xab; sol[6] ^= 0xcd; sol[7] ^= 0xef;
  *(uint32_t *)&sol[8] ^= 0xdeadbeef;
  res = memcmp(sol, "\x31\x5b\x24\x38\xfb\xce\xae\x80\x81\xd3\xd9\xb2", 12);
  if (res)
    puts(":(");
  else
    puts(":)");
}
```

http://angr.io

Demo

Simple crackme:

- Time to find correct input to reach the statement in line 20 using `angr`:

Vanilla	Movfuscated	De-Movfuscated
< 1s	$>$ 24h (!)	19s

$\Rightarrow$ Lookup tables are a huge problem for symbolic execution

Contact

- Julian:
  `mail [ at ] kirschju.re`
  `F949 CFBD 140A 6DD0 71E9 0B8C DC24 396B 6D45 1038`
- Clemens:
  `jonischk [ at ] cs.tum.edu`
  `A903 76D1 65F3 25F9 8594 280A 2BA0 1592 EFAC B551`

- **Sources** available – documentation pending :-)
  → Source code:
  `https://github.com/kirschju/demovfuscator`
  → Project website:
  `https://kirschju.re/demov`
  → Clemens' thesis:
  `https://kirschju.re/static/ba_jonischkeit_2016.pdf`

Thanks!