Patching Executable Code

Andy Goth

andrew.m.goth@gmail.com
http://www.facebook.com/andygoth
http://andy.junkdrome.org/

25 March 2013

Patching Executable Code

- Binary code and data can be modified by hand
  - No source code required!
- Need a strong understanding of assembly language
  - Learn as you go by examining objdump output
- Several powerful command line tools
  - objdump: analyze executable object files
  - xxd: dump and patch binary files
  - gcc -S: compile test programs to assembly language
  - as: convert assembly language to machine code

Demonstration

- Example will consist of mixed Ada and C code
- Takes one command line argument, 0 through 99
- Prints it sometimes, but with a twist
- Prints a counter that's sometimes incremented
- Also included: a Makefile so you can follow along
- Will show how to make a few changes by hand

program.adb, 1/2

```
with Ada.Command_Line;

procedure Program is
   procedure C_Print(Arg: Integer);
   pragma Import(C, C_Print, "print");

   Arg: Integer range 0 .. 99 :=
       Integer'Value(
       Ada.Command_Line.Argument(1));

   Count: Integer := 10;
```

program.adb, 2/2

```
begin
   case Arg is
      when 1 .. 5 | 7 | 9 | 75 .. 80 =>
         C_Print(Arg);
         Count := Count + 1;
      when 8 | 10 .. 20 | 30 | 81 .. 99 =>
         C_Print(0);
      when others =>
         null;
   end case;
   C_Print(Count);
end Program;
```

print.c

```
#include <stdio.h>

void print(int arg)
{
    printf("%d\n", arg * 2);
}
```

Makefile

```
all: program program.objdump

CFLAGS := -ggdb3 -O0
ADAFLAGS := -ggdb3 -O0

program: program.ali program.o print.o
	gnatbind program.ali
	gnatlink program.ali print.o

%.o %.ali: %.adb
	$(CC) -c $(ADAFLAGS) $<

%.o: %.c
	$(CC) -c $(CFLAGS) $<

program.objdump: program
	objdump -sxS $< > $@

clean:
	$(RM) program program.ali program.o print.o program.objdump

.PHONY: all clean
```

Running the Program

- ```
  $ ./program 1
  2
  22
  ```
- ```
  $ ./program 6
  20
  ```
- ```
  $ ./program 80
  160
  22
  ```
- ```
  $ ./program 99
  0
  20
  ```

First Change

- All the printed numbers are doubled
- Let's make them not be doubled
- Need to find and remove the MUL instruction
  - Could instead SHL 1 or ADD the number to itself
  - Compiler versions and optimization flags cause wide variation in the actual machine code
- Impossibly difficult to move anything around
  - Will need to overwrite removed code with NOPs

program.objdump

- Made using gcc 4.4.4 on a 350MHz Pentium II
  - Yes, this is what I have at home, please don't laugh
- 29,174 lines long, so search skills are crucial
  - Recommend ":sp", ":vsp", and "/" in Vim
- Everything about the program is in this file
  - Could also have dumped the *.o files, but since linking hasn't happened, wouldn't know the final virtual memory addresses of anything

Finding the Code

- Typed "/<print>:" in Vim to find the print() function
- Argument is passed on the stack
  - Found relative to %esp register
  - Code saves %esp to %ebp so it can put more arguments on the stack when it calls printf()
  - Argument is copied from stack to %eax for processing
- Abuses LEA (Load Effective Address) to double number
  - Compilers can get disgustingly creative, even with -O0
  - Doubled number is put in %edx

Assembly Dump for print()

```
08049620 <print>:
void print(int arg)
{
 8049620: 55                    push   %ebp
 8049621: 89 e5                 mov    %esp,%ebp
 8049623: 83 ec 08              sub    $0x8,%esp
    printf("%d\n", arg * 2);
 8049626: 8b 45 08              mov    0x8(%ebp),%eax
 8049629: 8d 14 00              lea    (%eax,%eax,1),%edx
 804962c: b8 6a 70 05 08        mov    $0x805706a,%eax
 8049631: 83 ec 08              sub    $0x8,%esp
 8049634: 52                    push   %edx
 8049635: 50                    push   %eax
 8049636: e8 8d fc ff ff        call   80492c8 <printf@plt>
 804963b: 83 c4 10              add    $0x10,%esp
}
 804963e: c9                    leave
 804963f: c3                    ret
```

Simple Fix

- The goal isn't to optimize, merely to patch
  - Find "lea (%eax,%eax,1),%edx" at address 0x8049629
  - Overwrite with "mov %eax,%edx"
- Assemble the new machine code
  - `echo "mov %eax,%edx" | as -al -o /dev/null`
  - `1 0000 89C2 mov %eax,%edx`
- The machine code bytes are "89 c2"
- Pad with NOP to get three bytes, so append "90"

Performing the Patch

- Need to find the right file offset
  - According to objdump, .text segment is at virtual memory address 0x8049430 and file offset 0x1430
  - Simple math says address 0x8049629 is at offset 0x1629
- Confirm three bytes at 0x1629 are currently "8d 14 00"
  - `xxd -s 0x1629 -l 3 program`
- Replace these bytes with "89 c2 90"
  - `echo 89c290 | xxd -p -r -s 0x1629 - program`
- Can rerun objdump to make sure change is as expected

Running the Program Again

- ```
  $ ./program 1
  1
  11
  ```
- ```
  $ ./program 6
  10
  ```
- ```
  $ ./program 80
  80
  11
  ```
- ```
  $ ./program 99
  0
  10
  ```

Second Change

- Program prints “10” when run with argument “6”
- Let’s make it print “9” instead
  - Design approach: decrement Count before it’s printed
- Need to find room to shoehorn in the decrement
- This case statement uses a jump table
  - Find in Vim with “/<_ada_program>:”, “/jmp **”
  - Not all case statements use jump tables
- Must update jump table slot 6 to point to new code

Assembly Dump for Case Statement

```
  case Arg is
80496c7: 8b 45 f0                   mov    -0x10(%ebp),%eax
80496ca: 83 f8 63                   cmp    $0x63,%eax
80496cd: 77 2e                      ja     80496fd <_ada_program+0xa1>
80496cf: 8b 04 85 80 70 05 08       mov    0x8057080(,%eax,4),%eax
80496d6: ff e0                      jmp    *%eax
    when 1 .. 5 | 7 | 9 | 75 .. 80 =>
      C_Print(Arg);
80496d8: 8b 45 f0                   mov    -0x10(%ebp),%eax
80496db: 83 ec 0c                   sub    $0xc,%esp
80496de: 50                         push   %eax
80496df: e8 3c ff ff ff             call   8049620 <print>
80496e4: 83 c4 10                   add    $0x10,%esp
      Count := Count + 1;
80496e7: 8b 45 f4                   mov    -0xc(%ebp),%eax
80496ea: 40                         inc    %eax
80496eb: 89 45 f4                   mov    %eax,-0xc(%ebp)
80496ee: eb 0d                      jmp    80496fd <_ada_program+0xa1>
```

Examining the Jump Table

- `mov 0x8057080(,%eax,4),%eax`
  `jmp *%eax`
  - The jump table is located at 0x8057080
  - Each slot is four bytes long, starting with slot #0
  - Interested in slot #6 at address 0x8057098
- `8057090 d8960408 d8960408 fd960408 d8960408`
  - Type "/8057090" in Vim to find this in program.objdump
  - Jump target address is 0x80496fd (mind the endianness!)
  - Will need to change the jump table to point to new code

Modifying the Count

- Look at the existing code to see how to access variables
  -
```
8b 45 f4    mov    -0xc(%ebp),%eax
40          inc    %eax
89 45 f4    mov    %eax,-0xc(%ebp)
eb 0d       jmp    80496fd <_ada_program+0xa1>
```
- Code to load, increment, and store Count is 7 bytes long
  - Older objdump prints 0xfffffff4 instead of -0xc
- Jump to end of case statement is 2 bytes long in this situation
  - 5 bytes if jump displacement is outside range -128...127
- Always remember, jumps are relative to address of next instruction
  - "eb 00" is a no op, but "eb fe" is an infinite loop (0xfe = -2)

Making Room for New Code

- Need 9 or 12 free bytes to patch in new code to decrement Count and jump to the end of the case statement
- Older gcc generates startlingly inefficient Ada which can be hand-optimized to make plenty of room for new code
  - No such luck with gcc 4.4.4, even with -O0
- Sometimes can remove redundant error checking code
- Preferably, find and overwrite dead code or NOPs
- Not always possible to find enough room to work
- Found 14 consecutive NOPs between <_start> and <__do_global_dtors_aux> starting at 0x8049452

Assembling the New Code, 1/2

- Put your assembly code into a file called "test.s"
  - ```
    .org   0x8049452
    mov    -0xc(%ebp),%eax
    dec    %eax
    mov    %eax,-0xc(%ebp)
    jmp    0
    nop
    ```
- `as -al test.s -o /dev/null`
  - ```
    1 0000000 00000000   .org 0x8049452
    2 8049452 8B45F4     mov -0xc(%ebp),%eax
    3 8049455 48         dec %eax
    4 8049456 8945F4     mov %eax,-0xc(%ebp)
    5 8049459 E9FCFFFFFF jmp 0
    6 804945e 90         nop
    ```

Assembling the New Code, 2/2

- Gather the instruction bytes, ignoring the dummy NOP (90)
  - 8B45F4488945F4E9FCFFFFFF
- Compute the correct jump displacement
  - Jump is relative to next instruction address (0x804945e)
  - Want to jump to end of case statement (0x80496fd)
  - Displacement is 0x29f = 0x80496fd – 0x804945e
  - Rewrite as 32-bit little endian value to get 9F020000
- Replace dummy displacement of FCFFFFFF with 9F020000
  - 8B45F4488945F4E99F020000

Performing the Second Patch, 1/2

- ```
  Idx Name      Size      VMA       File off
   11 .text     0000dc0c  08049430  00001430
   13 .rodata   000018b0  08057060  0000f060
  ```
- New code belongs at 0x8049452 virtual memory address and 0x1452 file offset
  - 0x1452 = 0x8049452 – 0x8049430 + 0x1430
- Patch code using this command, all on one line:
  - ```
    echo 8B45F4488945F4E99F020000 |
    xxd -p -r -s 0x1452 - program
    ```

Performing the Second Patch, 2/2

- ```
  Idx Name      Size      VMA       File off
   11 .text     0000dc0c  08049430  00001430
   13 .rodata   000018b0  08057060  0000f060
  ```
- Jump table slot #6 is at 0x8057098 virtual memory address and 0xf098 file offset
  - 0xf098 = 0x8057098 − 0x8057060 + 0xf060
- Patch table to point to new code (0x8049452):
  - Swap endianness to Intel convention for 52940408
  - `echo 52940408 | xxd -p -r -s 0xf098 - program`
  ```

Running the Program Yet Again

- ```
  $ ./program 1
  1
  11
  ```
- ```
  $ ./program 6
  9
  ```
- ```
  $ ./program 80
  80
  11
  ```
- ```
  $ ./program 99
  0
  10
  ```

Advice

- Go slowly, and recheck your work continually
- Take copious notes
  - Very easy to lose track of which address is which
  - Changes will need to be put under configuration management
- Learn assembly language
- Look at the output of "gcc -S" and "objdump -S" to see how the compiler does things
- Be lucky
  - Creativity is an excellent substitute for luck