EVANDRIX .

evanc
writing (
SVBTLE ne

@

facebook.co

abo

NOVEMBER 6, 2013

OSX 10.9 Intel Reverse Engineering Tutorial (adapted)

Toolkit to get started

1. gcc 4.9 + gdb 7.6 w/ ~/.gdbinit (or clang 5.0/llvm 3.3 + lldb 300.2.51)
2. decent hex editor: 0xEd/Hex Fiend (gui) or od, chex, hexdump/hexedit (cli)
3. diagnostics: otool/XCode, file, strings; inspect universal binary - moatool, lip OSX - class-dump v3.4
4. Hopper Disassembler/IDA Pro

Mac OSX Application Architecture

Challenge.app package content structure

- Contents
  - Info.plist
  - MacOS - contains main binary (also look for ‘Frameworks’ folder)
  - PkgInfo
  - Resources ### Inspect if fat/universal binary (>1 arch): `file <binary>` or `otool -h <binary>` ### Check .gdbinit via `help user` within gd
- debug by `attach`ing to `<pid>`, or running live using `exec-file <binar`
- command-line arguments: `set args` or `r`/`run < <input>`

Basic gdb command

77
KUDOS

- breakpoint w/ condition
  - set: `b`/`bp` on mem loc, eg. `bp 0x1234`, or symbol, eg. `bp [NSContr stringValue]`
  - list all: `bpl`
  - enable/disable: `bpe/bpd` or `delete #`
- step(i)

  - `n` / `next`, `ni` / `next(i)[nstr]`, `step(i)[nstr],` step(o)[ver fn call]` - can bypass obj_msgSend()

- modify flag/mem
  - `cfX`: X={a,c,d,i,o,p,s,t,z,s} (gdbinit) - invert current cpu register flag state, eg. JE followed iff Zero flag (Z/ZF) == 1 (ie. take the jump); `cfz` to update

List

1. cfa - Auxiliary carry
2. cfc - Carry flag
3. cfd - Direction flag
4. cfi - Interrupt flag
5. cfo - Overflow flag
6. cfp - Parity flag
7. cfs - Sign flag
8. cft - Trap flag
9. cfz - Zero flag

- dump/eval mem loc
  - help x/FMT ADDR: examine memory
  - Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal), t(binary), f(float), a(address), i(instruction), c(char) and s(string), T(OSType)
  - Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).
  - (default fmt/sz is last used; cnt = 1; addr: last `print`ed or `x`-ed)
  - eg. `x/2s $eax`

`example.c`

```
#include <stdio.h>
main(int argc, char *argv[])
{
 printf("Hello GDB!\n");
 printf("Argument is: %s\n", argv[1]);
}
```

```
0x00001fb9 in main ()
-----------------------------------------------------------------------------[regs]
EAX: 00001FE1  EBX: 00001FB2  ECX: BFFFF848  EDX: 00000000  o d I t S z a p c
ESI: 00000000  EDI: 00000000  EBP: BFFFF828  ESP: BFFFF810  EIP: 00001FB9
CS: 0017  DS: 001F  ES: 001F  FS: 0000  GS: 0037  SS: 001F
[001F:BFFFF810]-------------------------------------------------------------[stack]
BFFFF860 : D2 F9 FF BF  F1 F9 FF BF - 01 FA FF BF  3B FA FF BF ............;...
```

```
BFFFF850 : 33 F9 FF BF  6C F9 FF BF - 88 F9 FF BF  C1 F9 FF BF 3...l...........
BFFFF840 : 00 00 00 00  01 00 00 00 - FC F8 FF BF  00 00 00 00 ................
BFFFF830 : 01 00 00 00  48 F8 FF BF - 50 F8 FF BF  BC F8 FF BF ....H...P.......
BFFFF820 : 00 10 00 00  BC F8 FF BF - 40 F8 FF BF  7A 1F 00 00 ........@...z...
BFFFF810 : 00 00 00 00  00 00 00 00 - 3C F8 FF BF  37 10 E0 8F ........:    mov
DWORD PTR [esp],eax
0x1fbc :    call   0x300a
0x1fc1 :    mov    eax,DWORD PTR [ebp+0xc]
0x1fc4 :    add    eax,0x4
0x1fc7 :    mov    eax,DWORD PTR [eax]
0x1fc9 :    mov    DWORD PTR [esp+0x4],eax
0x1fcd :    lea    eax,[ebx+0x3a]
0x1fd3 :    mov    DWORD PTR [esp],eax
--------------------------------------------------------------------------------
gdb$ x/s $eax
0x1fe1 :     "Hello GDB!"
```

- `print <program var>`
- `po $eax`: request object to print itself
- `set`: change mem `set *0xaddr = newval`/reg `set $eax = newval` contents
  - type casting: `set $eax = (char) 0x12345` - $eax = 0x45
  - `set $eax = (int) 0x12345` - $eax = 0x12345

```
...
_main:
...
7:    +13  00001fb3  8d832f000000 leal 0x0000002f(%ebx),%eax  Hello GDB!
...
9:    +22  00001fbc  e849100000   calll 0x0000300a            _puts
...
14:   +39  00001fcd  8d833a000000 leal 0x0000003a(%ebx),%eax  Argument is: %s\n
...
16:   +48  00001fd6  e82a100000   calll 0x00003005            _printf
...
```

- note compiler optimization: first `printf` -> `puts`, since no format string supplied
- `#9: +22 0000**1fbc** e849100000 calll 0x0000300a _puts`
- +22: local offset, ie. offset within this function
- 00001fbc: code addr - used for breakpoint
- e8 49 10 00 00: opcodes - **note: endian-ness**
- calll 0x0000300a: corresponding ASM mnemonic
- _puts: additional information identified
- set breakpoint here: `bp *0x1fbc`; `bpl` to check
- [regs]: current state of cpu registers and flags

- [code]: disasm output for current mem addr (1st line is next to exec)
- display [regs] and [code] section with `context` in gdb
- **note: Intel vs AT&T syntax*
- can `c` / `continue` @ breakpoint
- `printf("Argument is: %s\n", argv[1]);` - 2 args passed onto stack (ESP) in reverse order
- EAX holds supplied arg(s)

Get cracking!

1. Reconnaissance: program limitation & error/info messages - anything interesting?
    1. are they contained in the binary itself, or external resources/assets?
    2. refer to Apple's documentation for methods, eg. 'applicationDidFinishLaunching' may contain method 'isRegistered'
    3. look for program control flow - "good" vs. "bad" paths
2. Solutions
    1. JE [0x74 0x2b] -> NOP [0x90]
    2. patch return value of method to always be true

original

```
-(BOOL)[Level1 isRegistered]
    +0  00002a88  55        pushl  %ebp
    +1  00002a89  89e5      movl   %esp,%ebp
    +3  00002a8b  8b4508    movl   0x08(%ebp),%eax
...
```

`gdb$ assemble`

```
xor eax, eax
inc eax
ret
```

```
00000000  31C0  xor eax,eax
00000002  40    inc eax
00000003  C3    ret
```

- **set *0x2a88 = 0xc340c031** -little endian
- patch single byte `set *(char *) address = 0x90`
- TODO: 4 bytes starting @ 0x00002a88, can optionally NOP the remaining 2

bytes of original 3rd instr

1. also can patch code before method invocation

```
...
   +22  00002534  e8252b0000 calll   0x0000505e -
isRegistered
   +27  00002539  84c0       testb   %al,%al
   +29  0000253b  742b       je      0x00002568 -if not
reg, show bad msg
...
```

`gdb$ assemble` @ 0x00002534

```
xor eax, eax
inc eax
```

- ...and NOP remaining bytes as before

Patching

- if universal binary, have to first strip out x86 component
- `otool -f <binary>` -> nfat_arch = 2 -> cputype 7 ie. Intel x86 (cputype 18 is PPC)
- formula = offset (from ottol output in dec) + offset_to_patch
- patch @ this calculated addr in favourite hex editor
- (if PPC had been first, formula for x86 = offset - 0x1000 + offset_to_patch; 0x1000h=4096d is header for PPC part)
- tools exist to derive addr formula: offset1.3, ocalc.c

Typical valid serial # gen/verify routine

`-(BOOL)[Level1 validateSerial:forName:]`

1. Verify if user serial number length is ok. If ok continue, else give an error.
2. Compute the good serial number.
3. Compare the user serial number with the good serial number.

Serial should be 8 chars in length, as this piece of code shows: `+29 00002abb 83f808 cmpl $0x08,%eax`

A quick look at the whole method and we find the piece of code we are interested in:

```
...
  +369   00002c0f   891c24     movl   %ebx,(%esp)
```

- Instead of a comparison with a single serial number, 2 comparisons are made, the 1st half and then the 2nd half
- Merge these pieces to get the complete valid serial #

```
/* Keygen for Macserialjunkies Challenge '09
   The serial algorithm has a bug because the format string has no
zero padding.
   For example with the following name "zeparreco" the valid
serial is 39940081
   but since there is lack of padding, the algorithm generates
399481.
   Serial length must be equal to 8 so this username is impossible
to keygen due to this small bug.
*/
#include <cstdio>
#include <cstdlib>
#include <cstring>

int main(int argc, char *argv[])
{
 char name[256], *pname;
 printf("Macserialjunkies.com challenge #1 Keygen v0.1\n\n");
 printf("name:\n");
 fflush(stdout);
 fgets(name, 256, stdin);
 if ((pname = strchr(name, '\n')) != NULL)
 {
  *pname = '\0';
 }
/* serial number is composed by 8 digits
   there are two algorithms, one for the first 4 digits and the
other for the remaining
*/
// first block of four digits
 int i=0;
 int digit,multiplier=4;
// mov    eax,0x68db8bad
 int wtf = 0x68db8bad;
 int accumulator=0;
 int ecx=0;
 unsigned long long temp1;
 int temp2, temp3, temp4;
 int x=0;
 int stringsize = strlen(name);
 int firstblock,secondblock;
for (x=0; x < stringsize ; x++)
{
// movsx  eax,BYTE PTR [edi+ebx]
 digit = name[i];
// inc    ebx
 i++;
// imul   eax,esi
 digit = digit * multiplier;
// add    esi,0x4
 multiplier += 4;
// shl    edx,0x4
// sub    edx,eax
 digit = (digit << 4 ) - digit;
// lea    ecx,[edx+ecx+0x29a]
 ecx = digit + ecx + 0x29a;
// imul   ecx
 temp1 = (unsigned long long) ecx * wtf;
// this grabs the ecx value since long multiplication the result
goes to EDX:EAX
 temp1 = temp1 >> 32;
// mov    eax,ecx
// sar    eax,0x1f
 temp2 = ecx >> 0x1f;
// sar    edx,0xc
 temp1 = temp1 >> 0xc;
// sub    edx,eax
 temp3 = temp1 - temp2;
// imul   edx,edx,0x2710
 temp4 = temp1 * 0x2710;
// sub    eax,edx
 ecx = ecx - temp4;
}
// the last ecx is the good first serial part
firstblock = ecx;
// second block
 i=0;
 multiplier = 4;
 int edx;
 x=0;
 ecx=0;
 int firstsar, secondsar;
for (x=0; x < stringsize ; x++)
{
//movsx  eax,BYTE PTR [edi+ebx]
 digit = name[i];
// inc    ebx
 i++;
// imul   eax,esi
 digit = digit * multiplier;
// add    esi,0x8
 multiplier += 8;
// lea    edx,[eax+eax*4]
 edx = digit + digit * 4;
// lea    edx,[eax+edx*2+0x2d]
 edx = digit + edx*2 + 0x2d;
 ecx = ecx + edx;
 temp1 = (unsigned long long) ecx * wtf;
 edx = temp1 >> 32;
 firstsar = ecx >> 0x1f;
 secondsar = edx >> 0xc;
 temp2 = secondsar - firstsar;
 edx = secondsar * 0x2710;
 temp3 = ecx - edx;
}
// 2nd part of good serial
 secondblock = temp3;
// convert to decimal and print
printf("Serial number is: %04d%04d\n", firstblock, secondblock);
}
```

Future sections

- Intel x86 opcode map reference
- Mac OSX Apps: codesign - check status & disable
- unpack .pkg
- mount/install .dmg from CLI

- XCode building from CLI & debugging

More coming soon, stay tuned!

77 KUDOS

NOW READ THIS

How I gained access to Amazon EC2 servers from Github Search (adapted)

Github Search allows advanced filters that allow us to search for these private keys @ link. This looks for: private keys with a .pem extension “BEGIN RSA PRIVATE KEY” text that marks the beginning of a private key sorted by most… **Continue →**

@evandrix facebook.com/lwy08

SVBTLE