Challenge:

No Output

Will you be able to retrieve the flag from this land of darkness!!

Author:-Itz_me_strikerr

nc 147.182.172.200 1337

TL-DR:-

Exploiting a huge buffer overflow vulnerability using Ret2Csu to call read second time and Ret2dl_resolve to resolve a symbol of our choosing(system here) and yield a shell.

Writeup:

Only the binary for the challenge is given..

First let's get some basic idea on this binary.

Let's do the basic drill everyone used to do when they get a binary..

```
ajay@kali:~/tamil_ctf_2.0/chall_2$ ./chall2
foobar
ajay@kali:~/tamil_ctf_2.0/chall_2$ ltrace ./chall2
sleep(1)                                                                    = 0
memset(0x404028, '\0', 8)                                                   = 0x404028
memset(0x404018, '\0', 8)                                                   = 0x404018
read(0foobar
, "foobar\n", 512)                                                          = 7
+++ exited (status 0) +++
ajay@kali:~/tamil_ctf_2.0/chall_2$ ./chall2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
ajay@kali:~/tamil_ctf_2.0/chall_2$
```

Let's take a moment and observe the information we got from the opening steps..

The function calls made by this binary are sleep, memset and finally ead.

This binary does not seem to output anything and do not use any function to write to stdout..

This is a huge problem as we have no way to leak memory contents..

For large input, the program seems to crash chances are that overflow vulnerability is present in this binary.

We got a read here to read user input..As this binary only has a main function it has to return to __libc_start_main.. Many will think about the idea of partial overwrite of return address to one gadget.. But there is no libc file given and there is no way to know about the server's libc version..So our partial overwrite of return address plan will never work here..

```
1
2  undefined8 main(void)
3
4  {
5    undefined local_28 [32];
6
7    sleep(1);
8    memset(&PTR_sleep_00404028,0,8);
9    memset(&PTR_memset_00404018,0,8);
10   read(0,local_28,0x200);
11   return 0;
12 }
13
```

By seeing the decompiled code in ghidra for this binary..

We can see that this binary suffers from a huge overflow as the buffer has only size for 32 characters and while the read function can read upto 512 characters..

The memset is used to null out the sleep and memset's .got.plt entry.

Let's see the security mitigations present in this binary

```
[*] '/home/ajay/tamil_ctf_2.0/chall_2/chall2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

From here, We can see that there is a partial RELRO and only NX enabled.

There is no stack canary and PIE here.

Let’s view the symbol table of the binary.

```
pwndbg> info file
Symbols from "/home/ajay/tamil_ctf_2.0/chall_2/chall2".
Local exec file:
        `/home/ajay/tamil_ctf_2.0/chall_2/chall2', file type elf64-x86-64.
        Entry point: 0x401060
        0x00000000004002a8 - 0x00000000004002c4 is .interp
        0x00000000004002c4 - 0x00000000004002e8 is .note.gnu.build-id
        0x00000000004002e8 - 0x0000000000400308 is .note.ABI-tag
        0x0000000000400308 - 0x0000000000400324 is .gnu.hash
        0x0000000000400328 - 0x00000000004003b8 is .dynsym
        0x00000000004003b8 - 0x0000000000400402 is .dynstr
        0x0000000000400402 - 0x000000000040040e is .gnu.version
        0x0000000000400410 - 0x0000000000400430 is .gnu.version_r
        0x0000000000400430 - 0x0000000000400460 is .rela.dyn
        0x0000000000400460 - 0x00000000004004a8 is .rela.plt
        0x0000000000401000 - 0x0000000000401017 is .init
        0x0000000000401020 - 0x0000000000401060 is .plt
        0x0000000000401060 - 0x0000000000401201 is .text
        0x0000000000401204 - 0x000000000040120d is .fini
        0x0000000000402000 - 0x0000000000402004 is .rodata
        0x0000000000402004 - 0x0000000000402040 is .eh_frame_hdr
        0x0000000000402040 - 0x0000000000402140 is .eh_frame
        0x0000000000403e10 - 0x0000000000403e18 is .init_array
        0x0000000000403e18 - 0x0000000000403e20 is .fini_array
        0x0000000000403e20 - 0x0000000000403ff0 is .dynamic
        0x0000000000403ff0 - 0x0000000000404000 is .got
        0x0000000000404000 - 0x0000000000404030 is .got.plt
        0x0000000000404030 - 0x0000000000404040 is .data
        0x0000000000404040 - 0x0000000000404048 is .bss
```

As the location of .bss is seem to be in the range of 0x400000.

There is an attack vector available to exploit this binary.

Ret2dl_resolve

This ret2dl_resolve method allows an attacker to resolve any symbols on their own by taking advantage of the lazy binding method use to resolve symbols in dynamically linked binaries and this method does not need any memory leak..

Attack Plan:

1)There is a huge overflow vulnerability present in this binary but we don’t have any functions that allows us to get a memory leak.

2)We can use the Ret2Dl_Resolve method to exploit this binary.
3)First thing is to get control of the code execution. Overflow of 40 characters will yield us PC Control but where to return tho...

4)We need a fake rel and sym struct to resolve our system function. So for this, let’s use the read function at .bss section to forge our fake rel and sym struct to resolve our system function.

5)For read function, we need to control 3 arguments where we need a pop rdx gadget to setup the size argument.. But there is no pop rdx gadget present in the binary.So, we need to use the Ret2csu method to control the rdx register.

6)After the ret2csu method, Call the read function with inputs as the forged rel and sym structs.

7)And then finally to the .plt section with the fake relocation arguement at the top of the stack.

To perfectly know about this attack method:-

Ret2Dl_resolve: https://syst3mfailure.io/ret2dl_resolve

Ret2Csu: https://guyinatuxedo.github.io/18-ret2_csu_dl/ropemporium_ret2csu/index.html

You can find the exploit for my binary in:-

[Will be uploaded in github and link will posted here during publish time]