

Intro to Pwn

Stack smashing 101

how2pwn

1. Find vuln(s)
2. Exploit the vuln(s)

race condition

1. Find the vuln

type confusion

it's probably here



buffer overflow

uninitialized variable $\square p^3_4i$

integer overflow

race condition

w

format string injection

double free
use after free

side channel

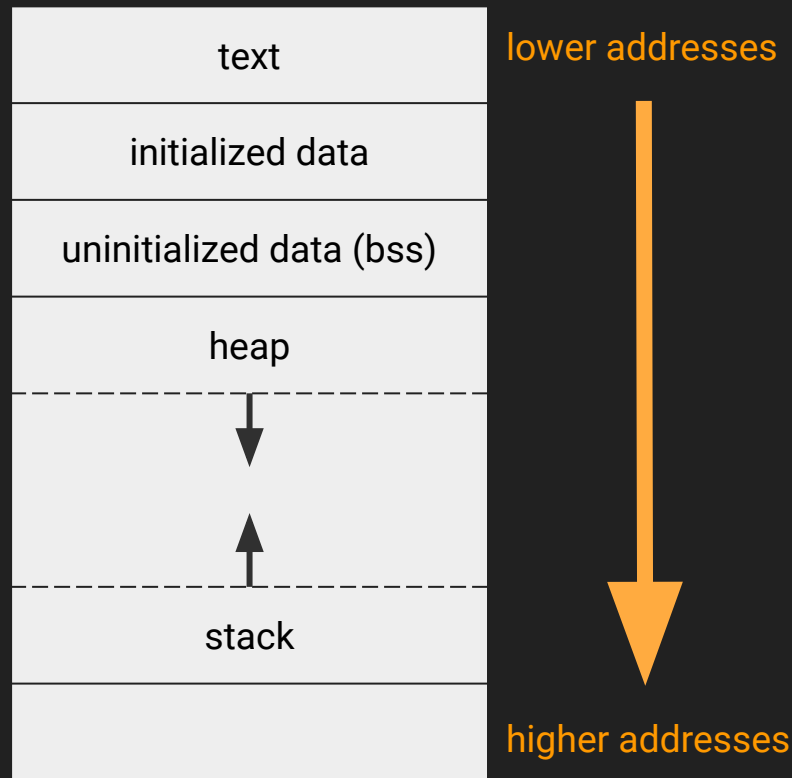
```
void func(int key){
    char overflowme[32];
    printf("overflow me : ");
    gets(overflowme);    // smash me!
    if(key == 0xcafebabe){
        system("/bin/sh");
    }
    else{
        printf("Nah..\n");
    }
}

int main(int argc, char* argv[]){
    func(0xdeadbeef);
    return 0;
}
```

bof.c

Memory layout

- Text region
 - Executable code
 - Other read-only data
- Data region
 - Static and global variables
- Heap
 - Dynamically allocated memory (`malloc`)
 - Grows toward higher addresses
- Stack
 - **Function parameters, return addresses, and local variables**
 - Grows toward lower addresses

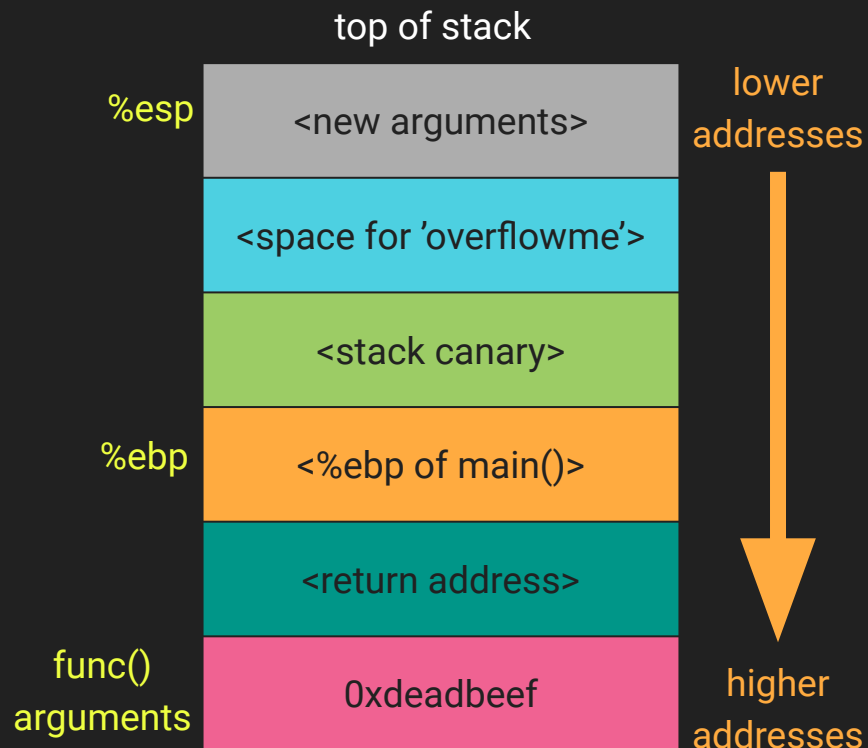


Stack layout in func()

overflowme :

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB

```
ac d3 ff ff 00 00 00 00 c2 00 00 00 e6 1e ea f7
ff ff ff ff ce d3 ff ff f8 8b e1 f7 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 42 00 00 3a 05 24
b0 56 55 56 30 55 55 56 f8 d3 ff ff 9f 56 55 56
ef be ad de 00 d0 ff f7 b9 56 55 56 00 50 fb f7
b0 56 55 56 00 00 00 00 00 00 00 00 63 5a e2 f7
```



Tools

- **gdb** (<https://www.gnu.org/software/gdb/>)
 - Debugger that lets you set breakpoints, inspect process memory, and more
 - Not very user friendly
- **pwndbg** (<https://github.com/pwndbg/pwndbg>)
 - gdb plugin that makes it more hacker friendly and less sucky
- **pwntools** (<https://github.com/Gallopsled/pwntools>)
 - Python library that makes writing and testing exploits much easier
- **Binary Ninja** (<https://binary.ninja>)
 - Scriptable GUI disassembler that's way cheaper than IDA Pro
 - Free demo

2. Exploit the vuln

<http://sigpwny.com/challenges#bof>

`ssh bof@104.248.115.191` (password is "guest")

Useful gdb/pwndbg commands:

- `r` (run the program)
 - `r <<< $(python -c "print '\x41'*31")` (pass 31 A's to standard input of the program)
- `b func` (set a breakpoint on func)
- `stack 32` (dereferences the top 32 words on the stack, pwndbg only)
- `n` (run the next line of code)
- `db $sp 128` (dump 128 bytes starting at the address in %esp)