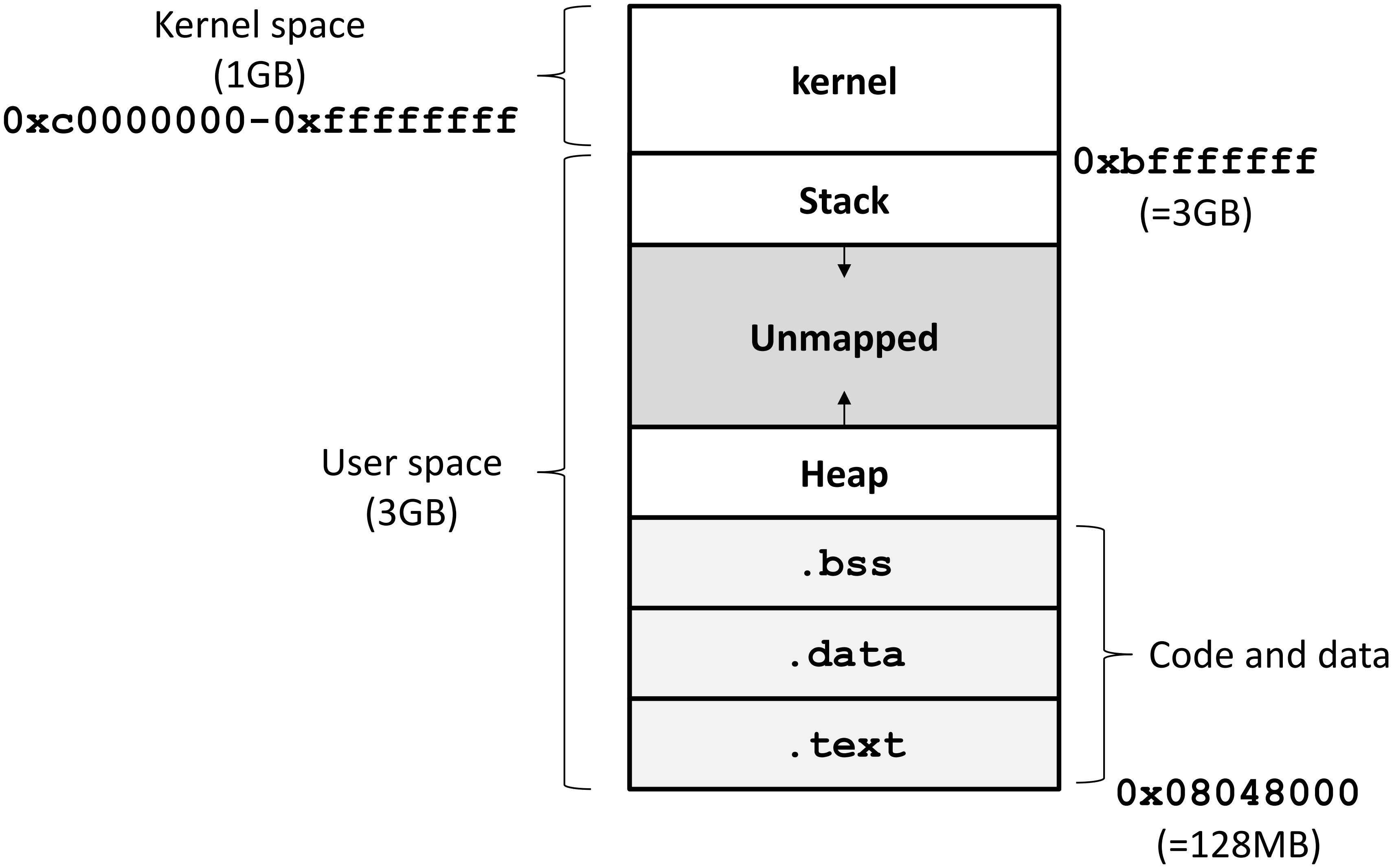
CS326 – Systems Security

Lecture 13

Code Injection

Elias Athanasopoulos
athnasopoulos.elias@ucy.ac.cy

Process Memory Layout
32-bit, Linux, no defenses



Stack

- Contains control data
  - Return address
  - Function pointers and VTable pointers allocated locally
- Overflows in local buffers (*buffer overflows*) can overwrite control data
  - Control-flow attack

Example

```
int password_valid = 0;

void authenticate_root(char *passwd) {
    unsigned long marker = 0xdeadbeef;
    char password[16];

    strcpy(password, passwd);
    fprintf(stderr, "%p\n", &marker);
    fprintf(stderr, "Validating password: %s\n", password);

    if (!strcmp(password, "e5ce4db216329f4f"))
        password_valid = marker;
}

int main(int argc, char *argv[]) {

    authenticate_root(argv[1]);

    if (password_valid != 0) {
        printf("Welcome administrator.\n");
    } else {
         printf("Access denied.\n");
    }
    return 1;
}
```

Stack layout

```
(gdb) x/32x $ebp-32
0xffffdb08: 0x08048034 0x41414141 0xf7004141 0xf7fe3230
0xffffdb18: 0x00000000 0xdeadbeef 0xf7fbf3fc 0xffffffff
0xffffdb28: 0xffffdb48 0x0804922c 0xffffdd70 0xffffdc04
0xffffdb38: 0xffffdc10 0x0804928d 0xf7fe3230 0xffffdb60
0xffffdb48: 0x00000000 0xf7df8e46 0xf7fbf000 0xf7fbf000
0xffffdb58: 0x00000000 0xf7df8e46 0x00000002 0xffffdc04
0xffffdb68: 0xffffdc10 0xffffdb94 0xffffdba4 0xf7ffdb40
0xffffdb78: 0xf7fca410 0xf7fbf000 0x00000001 0x00000000
```

password buffer
local variable (marker)
saved frame pointer (%ebp)
return address of current frame

Beyond Return Addresses

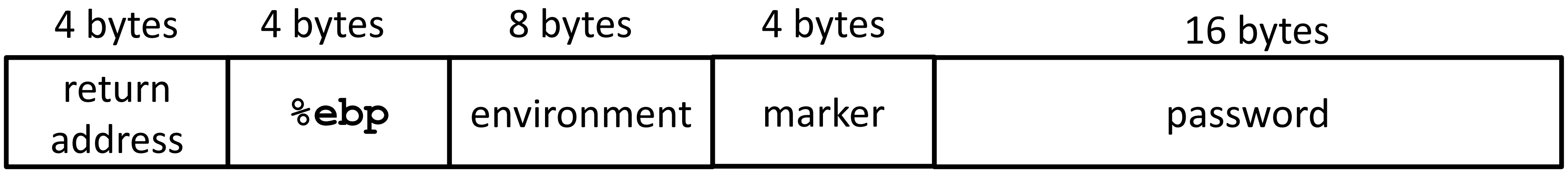
Function pointers

```
int vulnerable_func(...) {
    void (*fptr)(int);
    char buffer[N);

    /* initialize function pointer.  */
    fptr = myfunc();

    strcpy(buffer, malicious_input);

    fptr();
}
```

VTable pointers

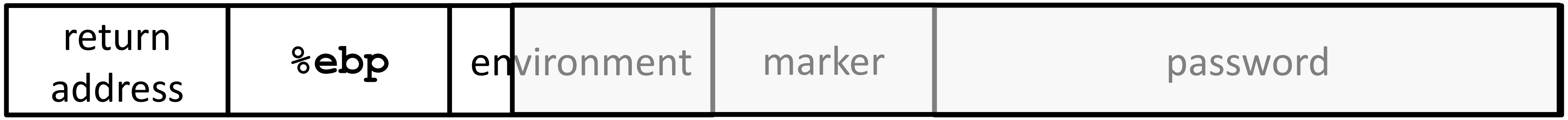
Only in C++. We will discuss later in the course.

Buffer Overflow



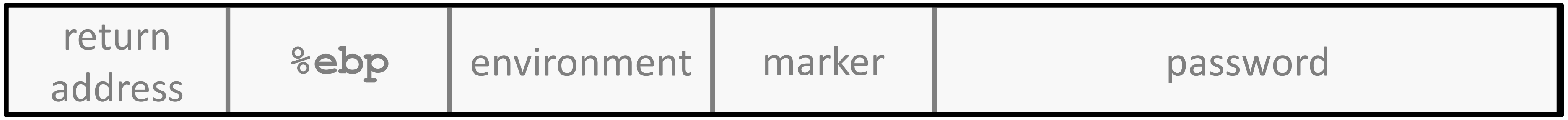
4 bytes	4 bytes	8 bytes	4 bytes	16 bytes
return address	**`%ebp`**	environment	marker	password

Bug (careless **`strcpy`**) can write over the stack (*write primitive*)



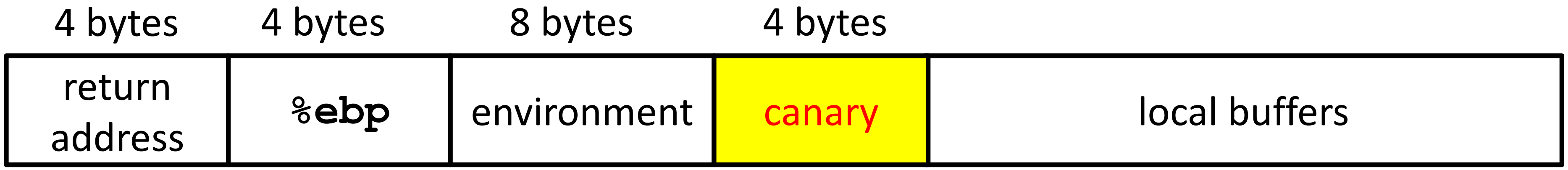
return address	**`%ebp`**	environment	marker	password

Use of write primitive can overwrite control-data (the return address)



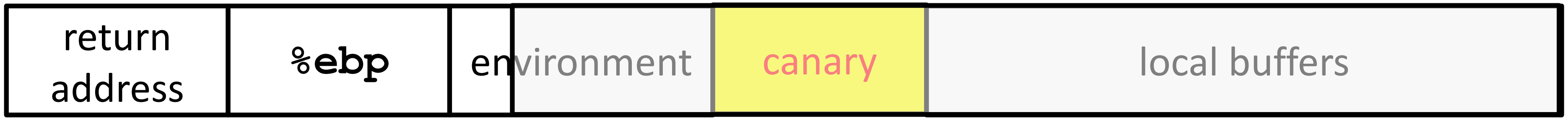
return address	**`%ebp`**	environment	marker	password

Defending return addresses



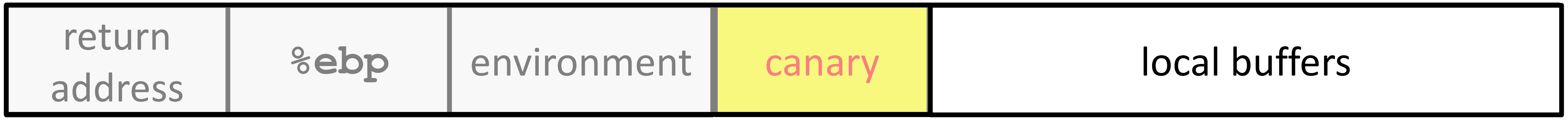
4 bytes	4 bytes	8 bytes	4 bytes	
return address	**%ebp**	environment	canary	local buffers

Bug (careless **`strcpy`**) can write over the stack (*write primitive*)



return address	**%ebp**	environment	canary	local buffers

Use of write primitive can overwrite the return address **and the canary value**



return address	**%ebp**	environment	canary	local buffers

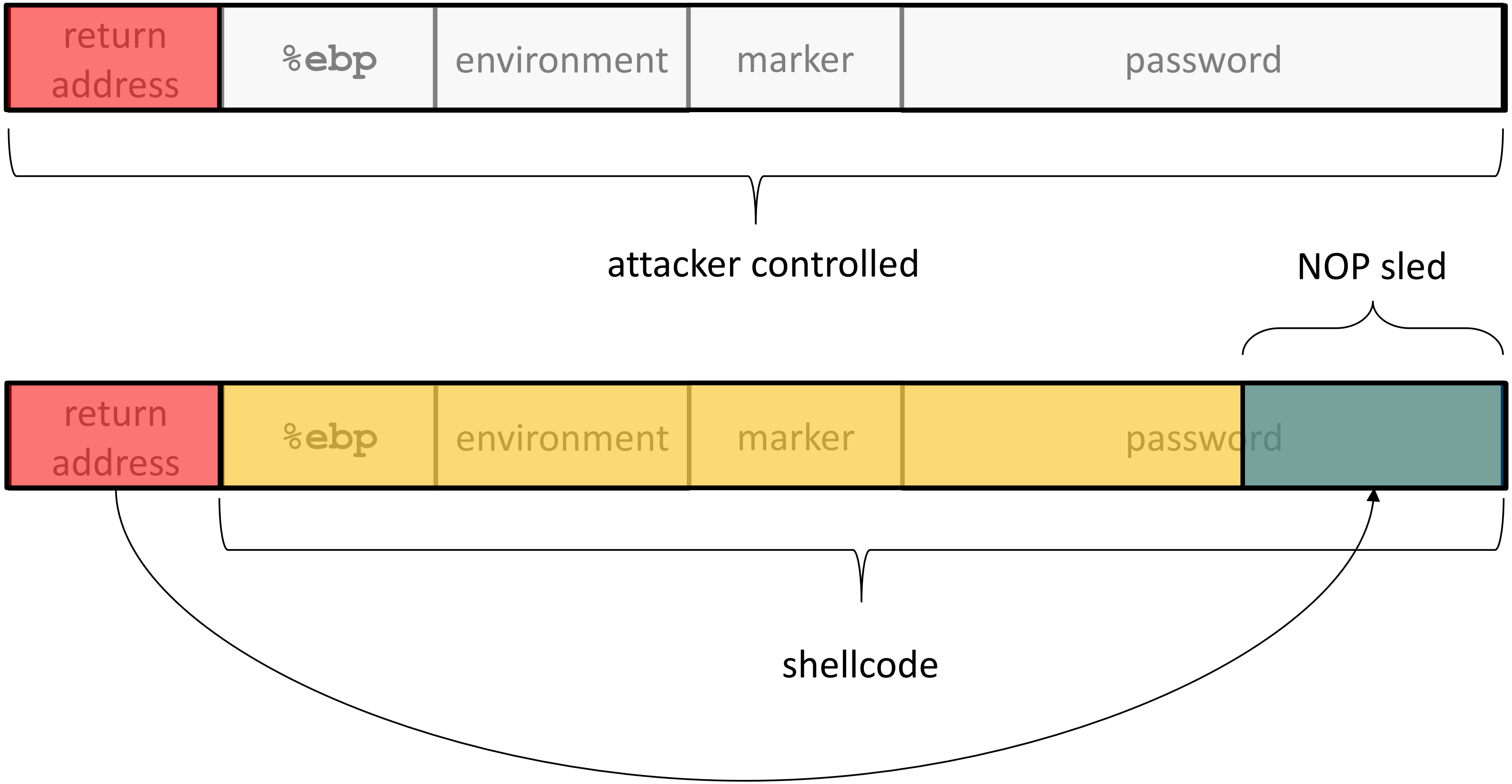
Defending return addresses

- Canary values are generated randomly upon process creation
  - Stored in a global variable
  - Usually `%gs` is involved
- All function epilogues are modified
  - Check if the canary value has been modified before returning back
- Bypassing canaries
  - Information leaks and read primitives (later in the course)

Code Injection

- Inject code to data
  - Code is contained in a malicious input
- Overwrite control data (e.g., return address)
  - New control data jump inside the malicious input
  - New code is executed
- Defense
  - Data is no longer executable (NX-bit)
  - Solution: ROP, change of permissions (later in the course)

Code Injection



return address
%ebp
environment
marker
password
attacker controlled
NOP sled
return address
%ebp
environment
marker
password
shellcode