

A (really) simple introduction to buffer overflows

Herbert Bos
Vrije Universiteit Amsterdam



Herbert Bos
VU University Amsterdam

syssec 
course repository

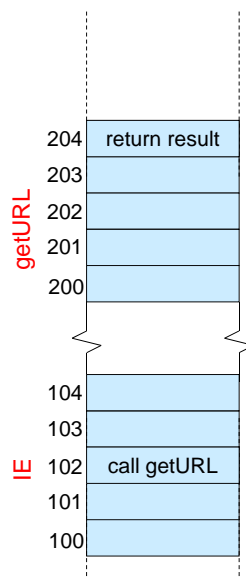
Exploits

- program has a security hole
- exploit = input that abuses the vulnerability

- In this module we will discuss an example:
the Buffer overflow

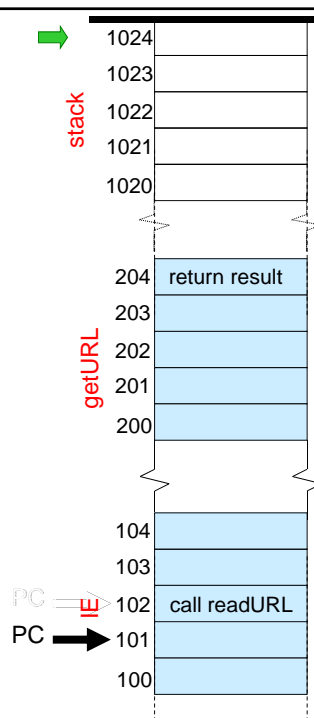
software

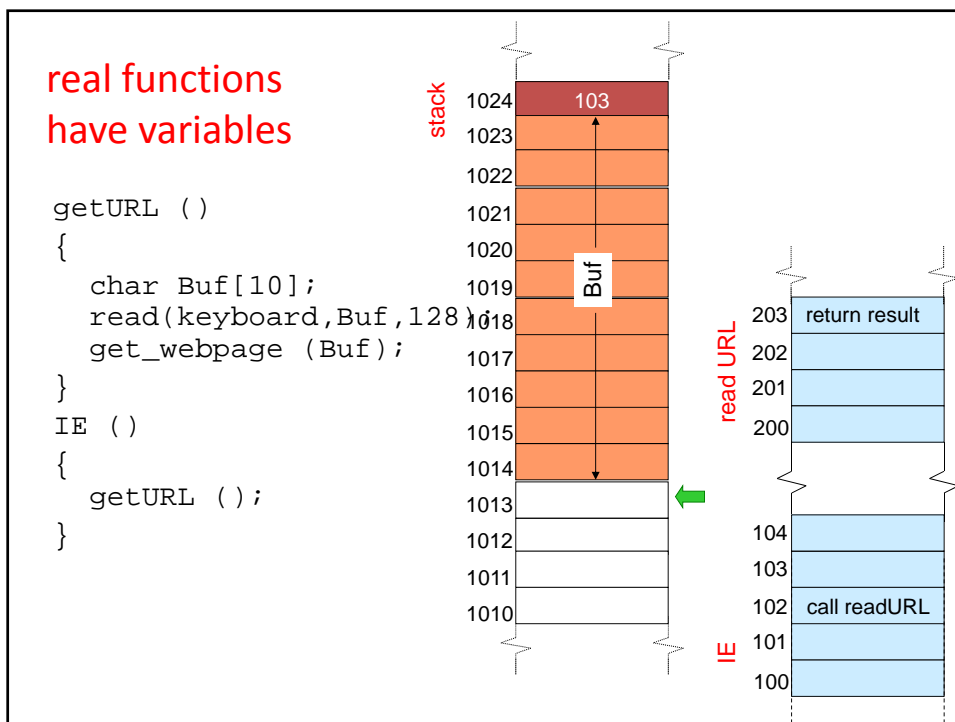
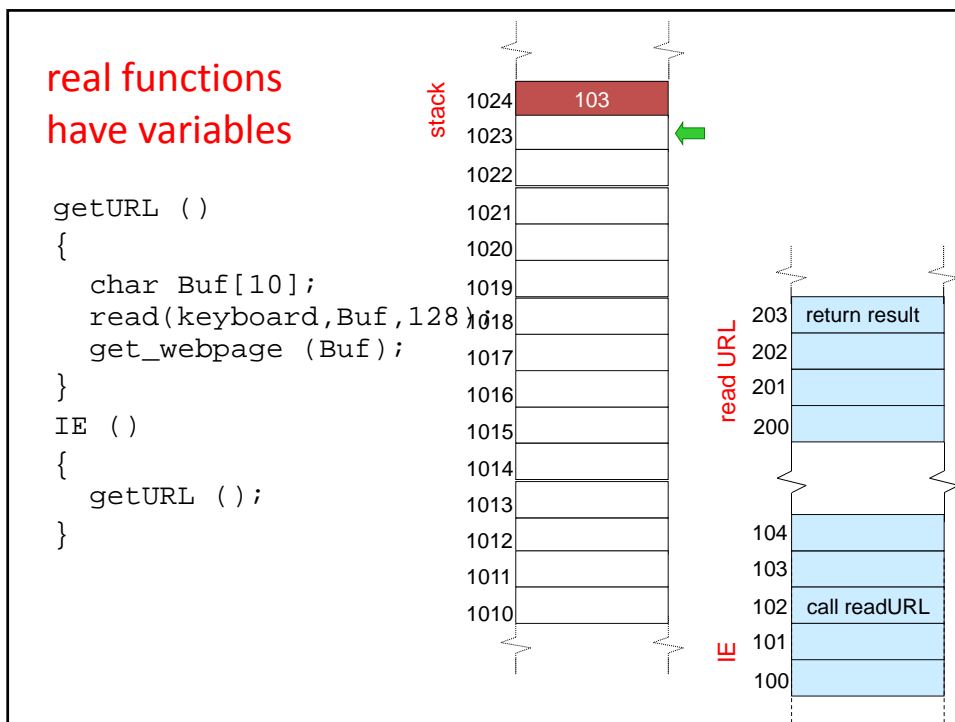
- sequence of instructions in memory
- logically divided in functions that call each other
 - function 'IE' calls function 'getURL' to read the corresponding page
- in CPU, the program counter contains the address in memory of the next instruction to execute
 - normally this is the next address (instruction 100 is followed by instruction 101, etc)
 - not so with function call

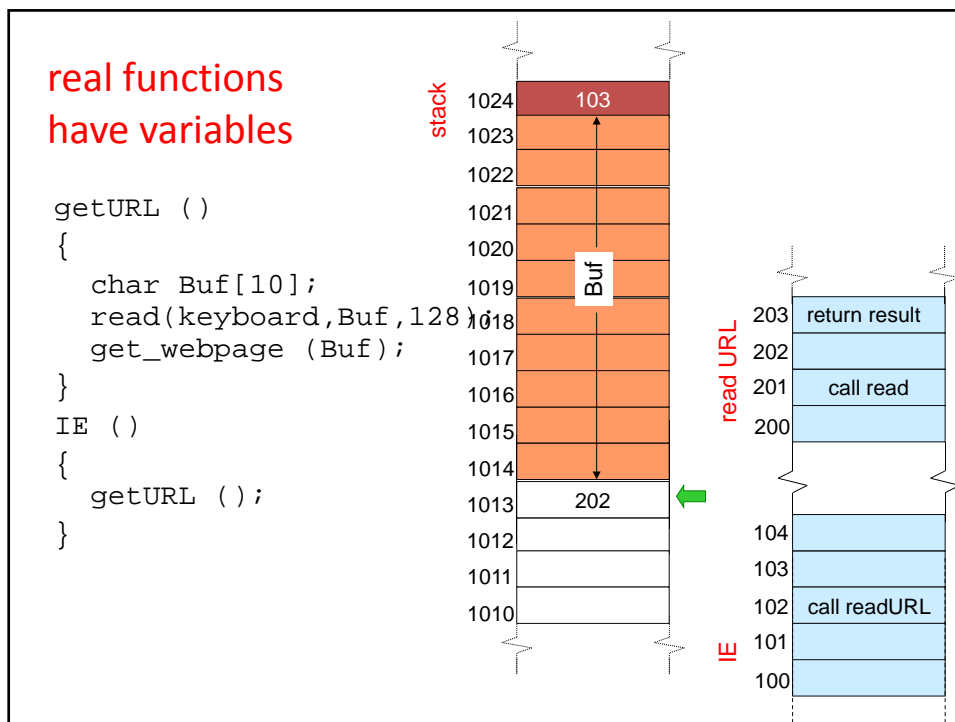


software

- so how does our CPU know where to return?
 - it keeps administration
 - on a 'stack'







what is next?

- we have learned a lot
- but where are the vulnerabilities?
- and how do we exploit them?

Exploit

```
getURL ()  
{  
    char Buf[10];  
    read(keyboard, Buf, 128);  
    get_webpage (Buf);  
}  
IE ()  
{  
    getURL ();  
}
```

The diagram illustrates a memory stack with addresses 1010 to 1024. A buffer labeled 'Buf' is shown between addresses 1014 and 1019. A red arrow indicates that data is being written from address 1024 down to 1014, overflowing the buffer. The address 1014 is highlighted in yellow, and a green oval encircles it and the address 1024.

That is it, really

- all we need to do is stick our program in the buffer