A (really) simple introduction to buffer overflows

Herbert Bos
Vrije Universiteit Amsterdam

Herbert Bos
VU University Amsterdam

syssec
course repository

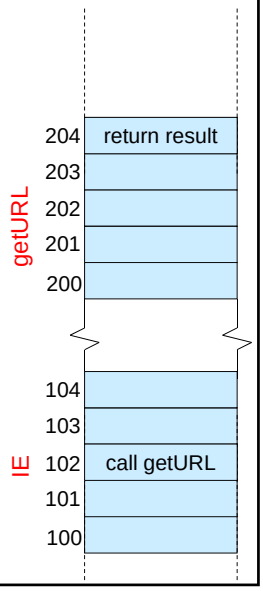
Exploits

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

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

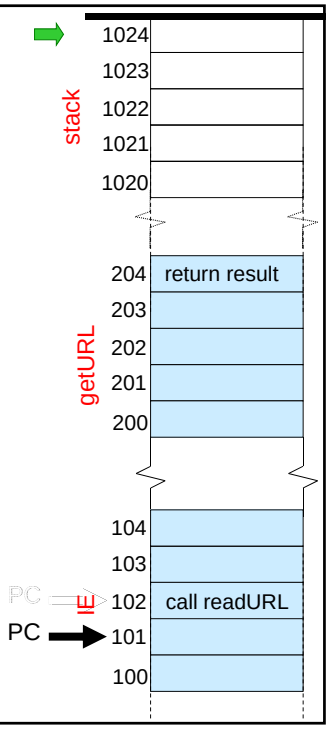
software

- 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



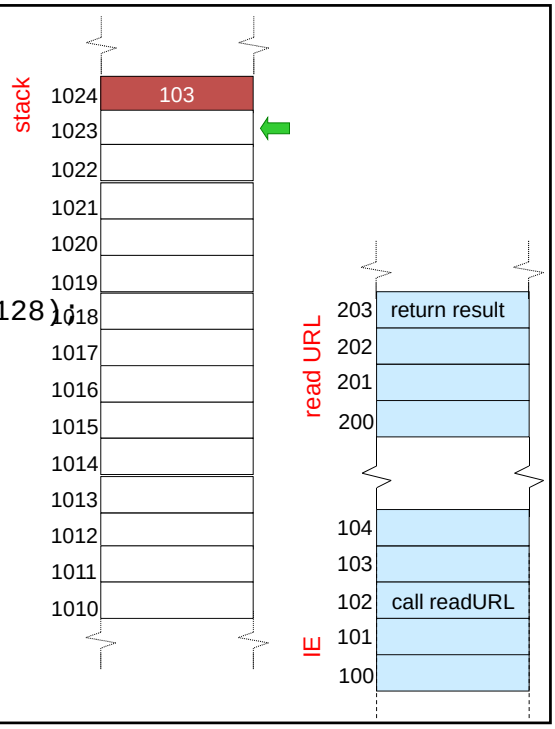
software

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



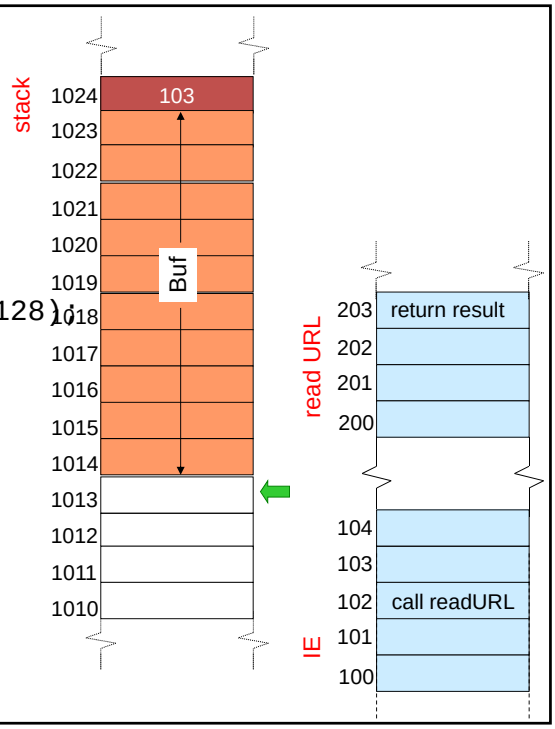
real functions have variables

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



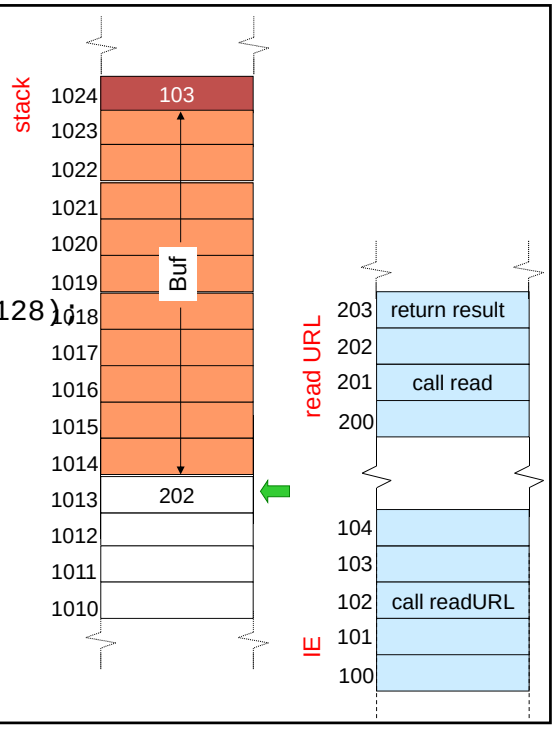
real functions have variables

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



real functions have variables

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

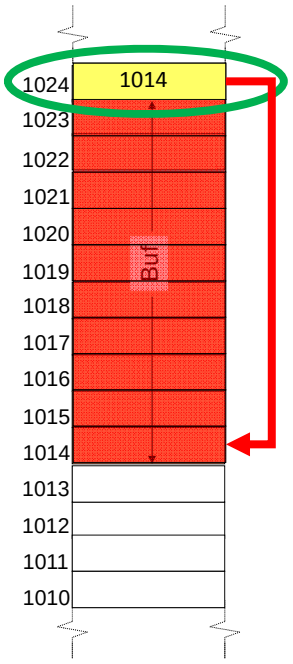


what is next?

- 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 ();
}
```



That is it, really

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