10K STUDENTS TO IMPROVE CYBER SECURITY
Part I
The Basic Idea
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

- 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
• so how does our CPU know where to return?
  – it keeps administration
  – on a ‘stack’
real functions

→ variables

definition:

```cpp
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}
```
real functions

getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

return result

get_URL

call read

call getURL

100
101
102
103
104
200
201
202
203
getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
real functions

variables

getURL ()
{
  char buf[10];
  read(keyboard, buf, 64);
  get_webpage(buf);
}

IE ()
{
  getURL();
}
real functions \(\rightarrow\) variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

getURL();
real functions ➔ variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

IE
real functions

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}
real functions \rightarrow variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

frame pointer (FP)
real functions ➞ variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage(buf);
}

IE ()
{
    getURL();
}

on “return from read”
real functions → variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

getURL ()

real functions ➔ variables

getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage(buf);
}

IE ()
{
    getURL();
}

getURL();

IE();
real functions
→ variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

getURL ()
{
    return result
}

IE ()
{
    getURL ();
    return result
}
real functions \rightarrow variables

getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}

getURL ()
{
    return result
}

IE ()
{
    getURL ();
    return result
}
Where is the vulnerability?
getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}

IE ()
{
    getURL ();
}
You may also overwrite other things

• For instance:
  – Other variables that are also on the stack
  – Other addresses
  – Etc.
Memory Corruption

Final words Part I

• We have sketched only the most common memory corruption attack
  – many variations, e.g.:
    • heap ⇔ stack
    • more complex overflows
    • off-by-one

• But there are others also
  – integer overflows
  – format string attacks
  – double free
  – etc.

• Not now, perhaps later...