Software Security Assessment

Lab 1:
Set-UID Privileged Programs

Access Control in Linux/Unix

User 1	User 2	…	root

Access Control in Linux/Unix

User 1	User 2	…	root
UID = 5000	UID = 5001		UID = 0

Access Control in Linux/Unix

User 1	User 2	…	root
UID = 5000	UID = 5001		UID = 0

Access Control in Linux/Unix

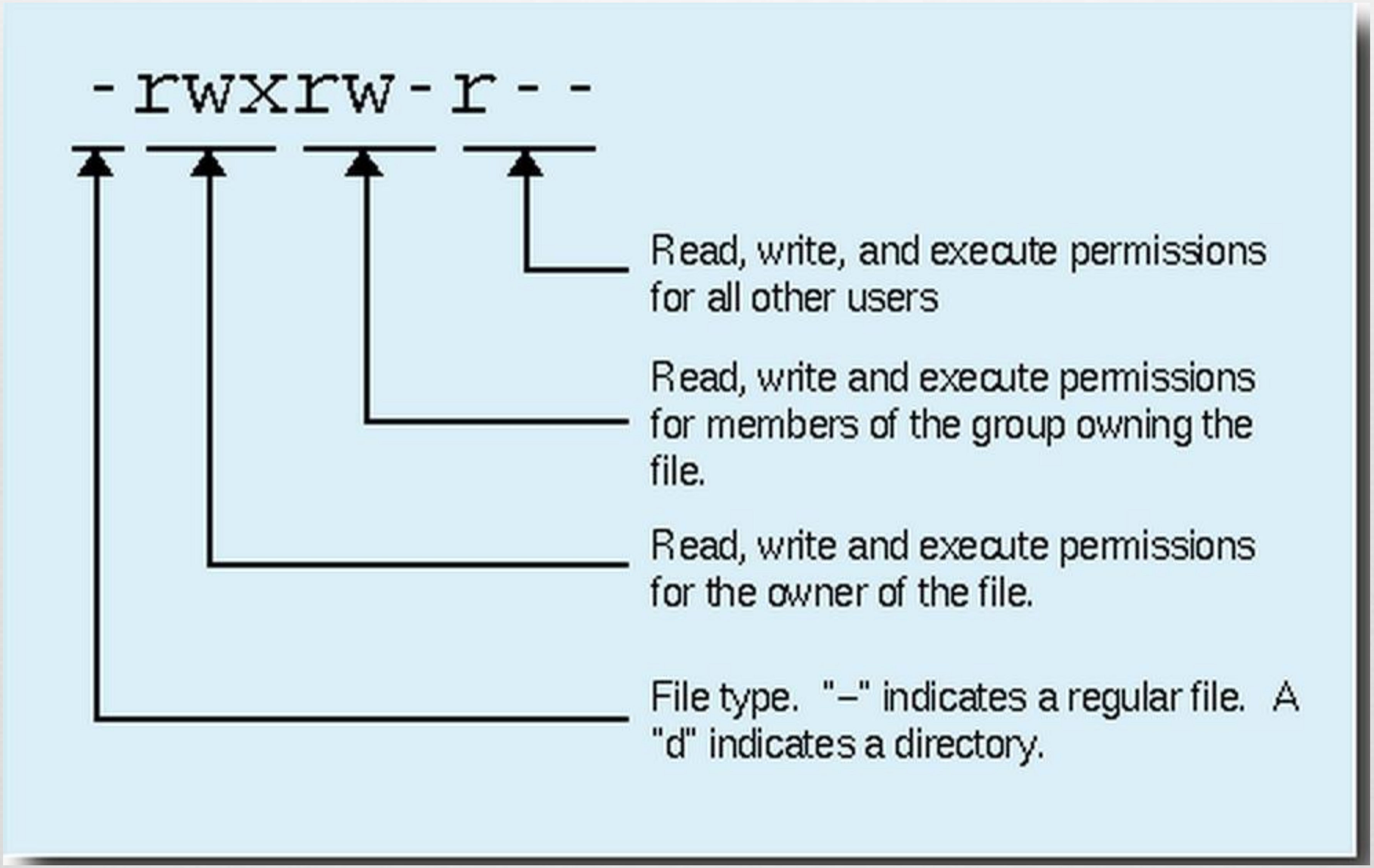
User 1	User 2	…	root
UID = 5000	UID = 5001		UID = 0

Access Control List (ACL)

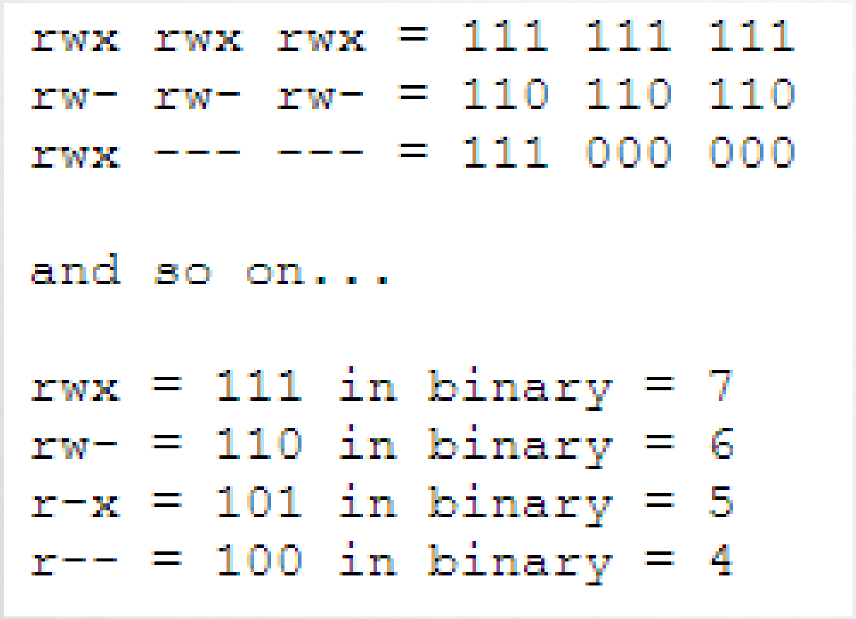
Permission: rwx r-x ---

Owner Group Other

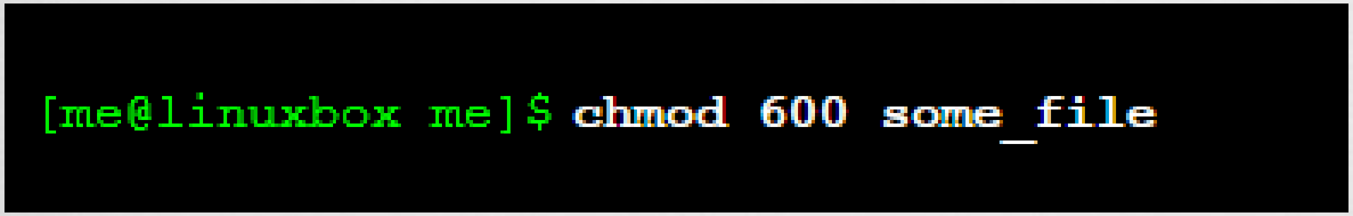
Access Control in Linux/Unix


-rwxrw-r--
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

Change Resource Permission



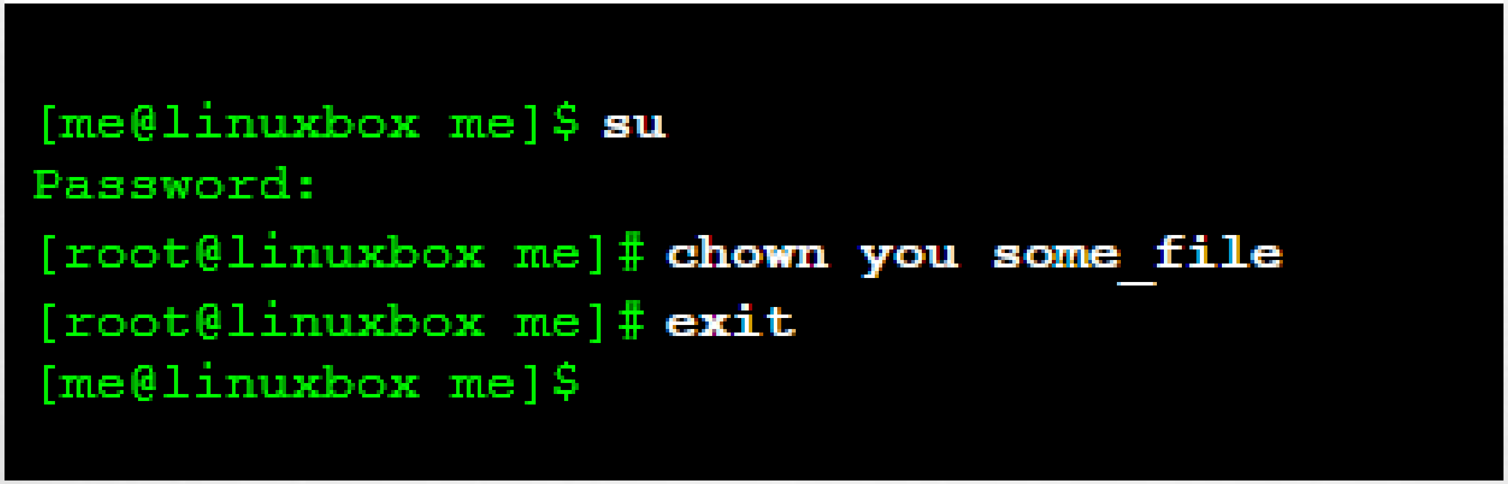
```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000

and so on...

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```



```
[me@linuxbox me]$ chmod 600 some_file
```

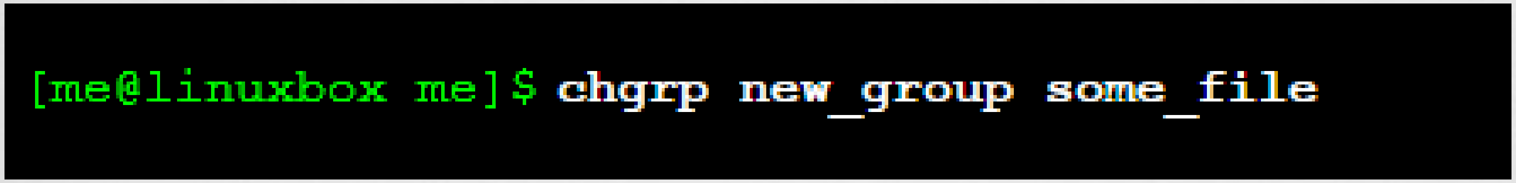
Change Resource Permission

Change Ownership of a file



```
[me@linuxbox me]$ su
Password:
[root@linuxbox me]# chown you some_file
[root@linuxbox me]# exit
[me@linuxbox me]$
```

Change Group Ownership



```
[me@linuxbox me]$ chgrp new_group some_file
```

Password Dilemma

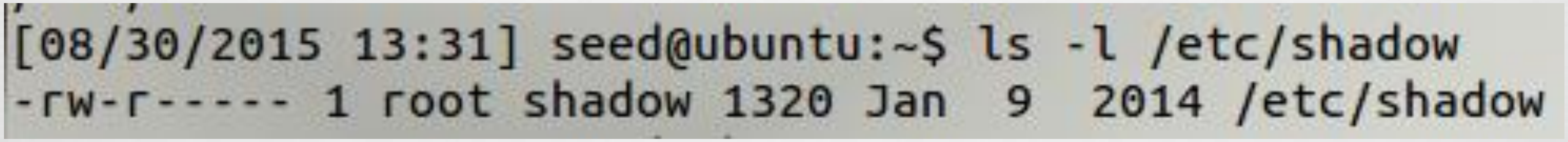
- Where your password is stored in Linux?

Password Dilemma

- Where your password is stored in Linux?
  - /etc/shadow

Password Dilemma

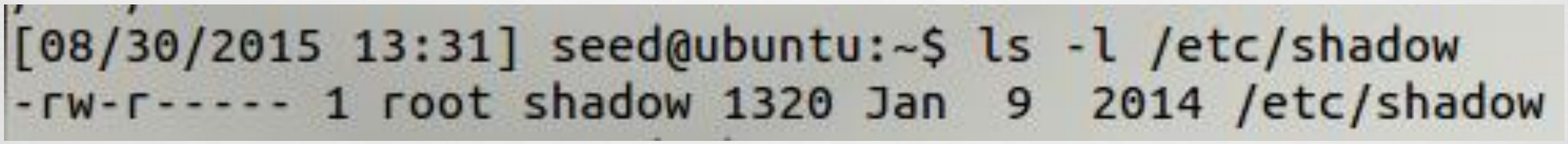
- Where your password is stored in Linux?
  - /etc/shadow



```
[08/30/2015 13:31] seed@ubuntu:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1320 Jan  9  2014 /etc/shadow
```

Password Dilemma

- Where your password is stored in Linux?
  - /etc/shadow

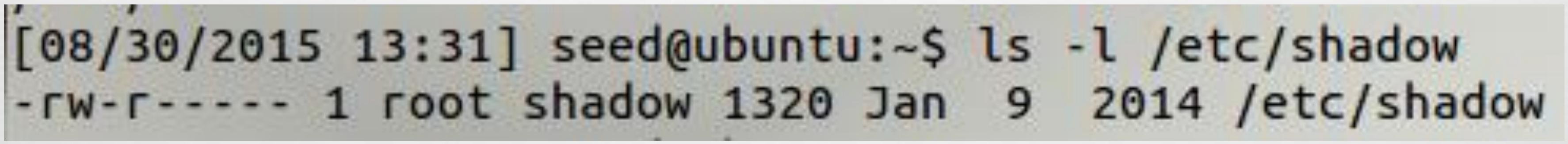


```
[08/30/2015 13:31] seed@ubuntu:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1320 Jan  9  2014 /etc/shadow
```

- Can we change our password in Linux?

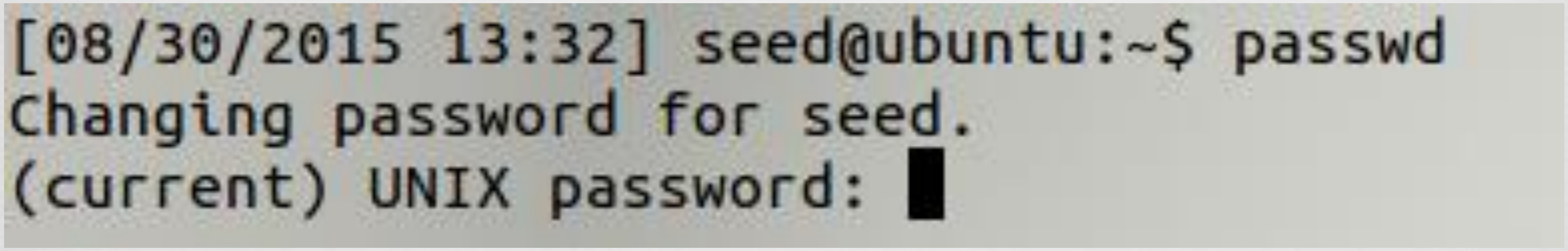
Password Dilemma

- Where your password is stored in Linux?
  - /etc/shadow



```
[08/30/2015 13:31] seed@ubuntu:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1320 Jan  9  2014 /etc/shadow
```

- Can we change our password in Linux?



```
[08/30/2015 13:32] seed@ubuntu:~$ passwd
Changing password for seed.
(current) UNIX password: █
```

Password Dilemma

- Possible Solutions:
  - Give the permission
  - Add a master user, or run a powerful program in background
  - Give partial permission, only change password of their own passwords.

Superman Story

Superman Story

Superman Story

Superman Story

Super People

Superman Story

Super People

Suggestions?

Superman Story

Super Robot

Suggestions?

Privileged Program

- A **privileged program** is one that can give users **extra privileges** beyond that are already assigned to them.

Privileged Program

- A **privileged program** is one that can give users **extra privileges** beyond that are already assigned to them.

- **Set-UID Programs**
  - Example: passwd

- Nomal Programs
  - Example: ls

Privileged Program

- How does the system know which program is a privileged program?
  - mark the program somehow, let the system know this program is special

Privileged Program

- How does the system know which program is a privileged program?
  - mark the program somehow, let the system know this program is special

- In Linux
  - **Permission: --- rwx r-x ---**

    rwS r-x ---

Privileged Program

- How does the system know which program is a privileged program?
  - mark the program somehow, let the system know this program is special

- In Linux
  - **Permission: --- rwx r-x ---**

    rwS r-x ---

Effective User Id

Real User Id

Privileged Program

- What is the effective user id and real user id when you run ls?
  - Suppose your current user id = 5000

Privileged Program

- What is the effective user id and real user id when you run ls?
  - Suppose your current user id = 5000

- Effective User Id = 5000
- Real User Id = 5000

Exercise

- Give you 60 seconds, what you can do?
- I am using a library computer and login to the remote server with **root** account. I leave the computer there without locking it for 60 seconds.
- If you also have a normal user account on the server, but no root permission. What you will do in 60 seconds, to take over my root account?
  - When you go back home, you can use your normal account to perform root tasks

Exercise

- Give you 60 seconds, what you can do?
  - Create a Set-UID program with root privilege.
    - What kind of program you want to create?

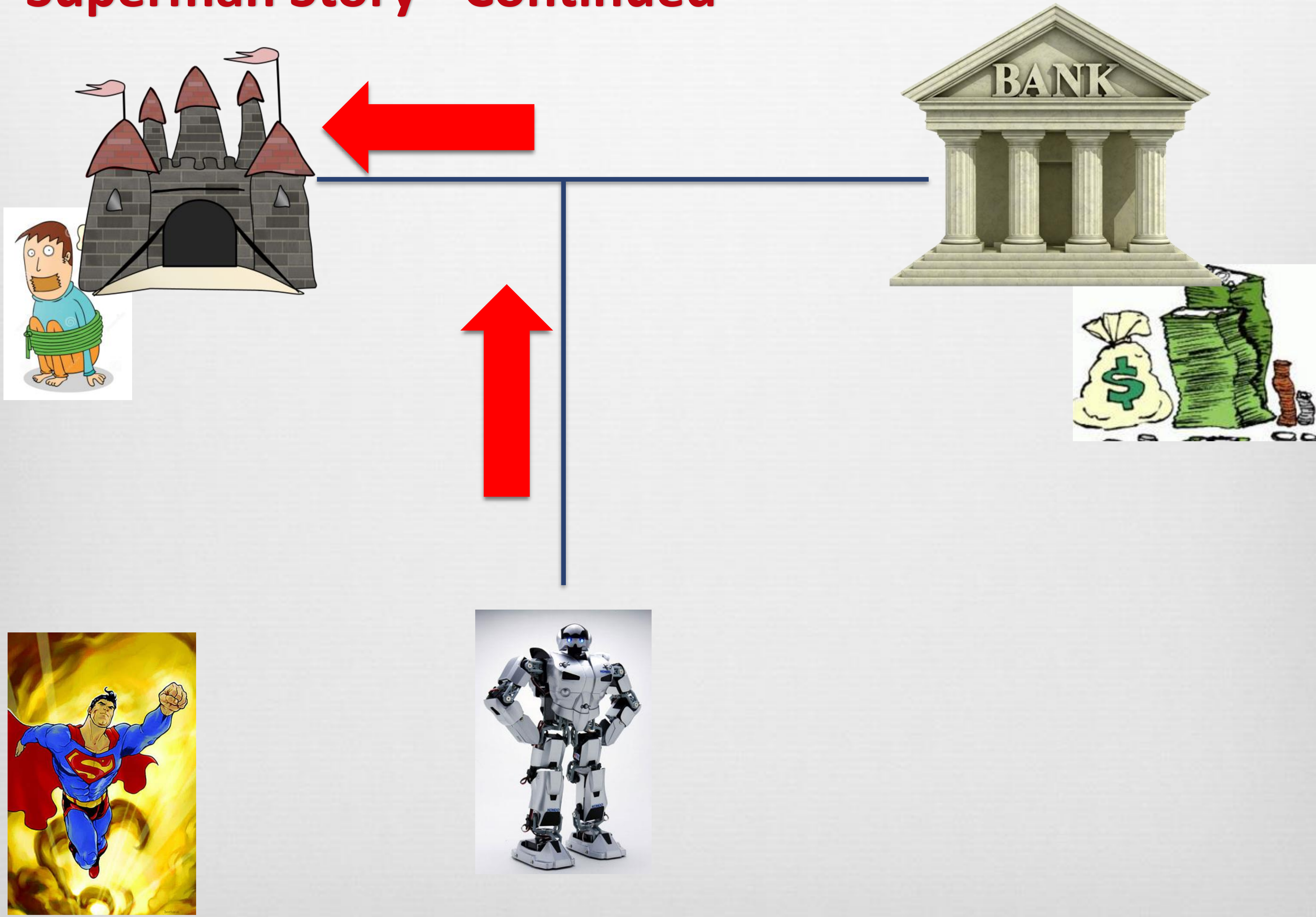
Exercise

- Give you 60 seconds, what you can do?
  - Create a Set-UID program with root privilege.
    - What kind of program you want to create?
- **Shell**

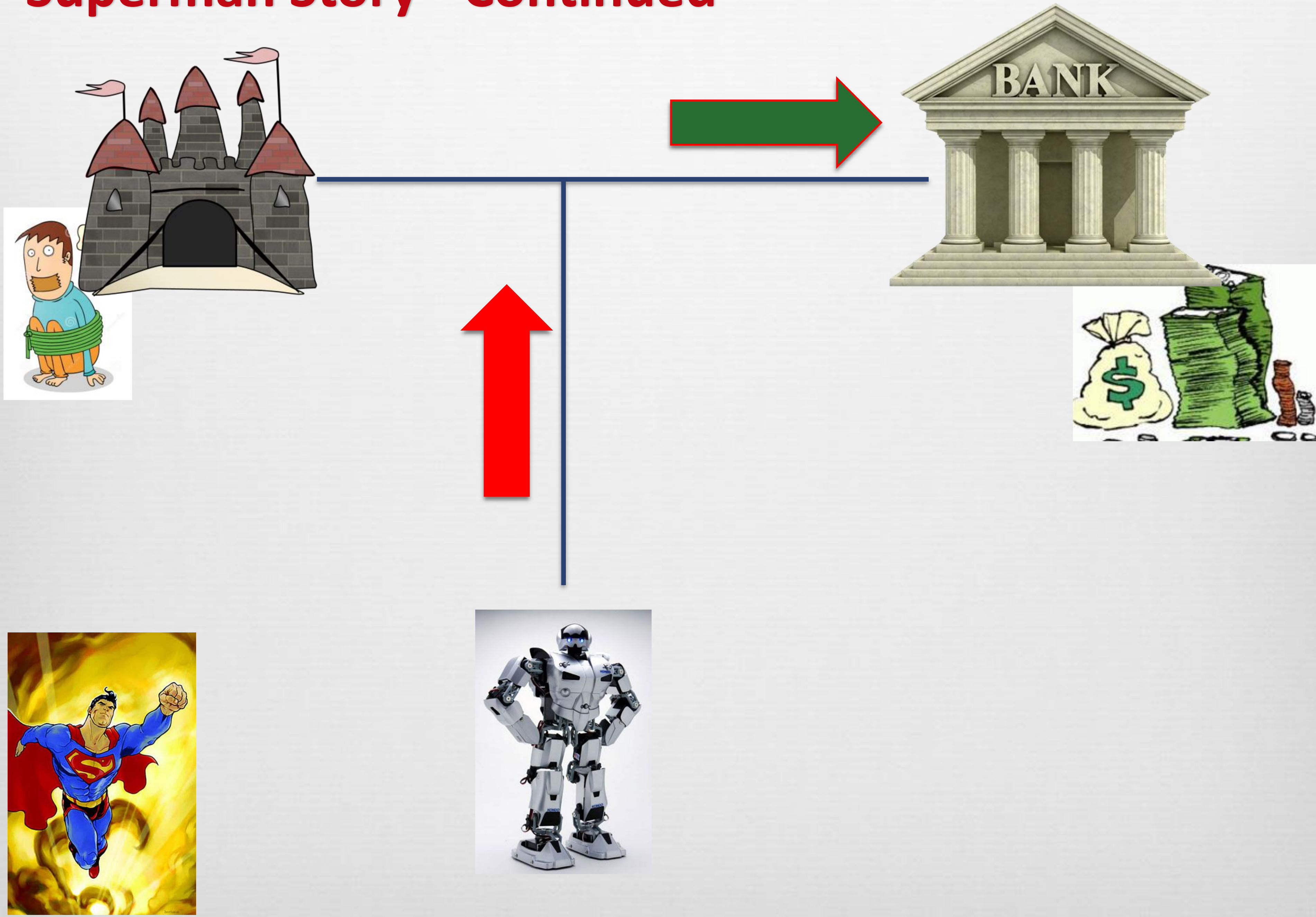
Exercise

- Give you 30 seconds, what you can do?
  - Create a Set-UID program with root privilege.
    - What kind of program you want to create?
- **Shell**
  - **cp /bin/sh /tmp/sh**
  - **chmod 4777**

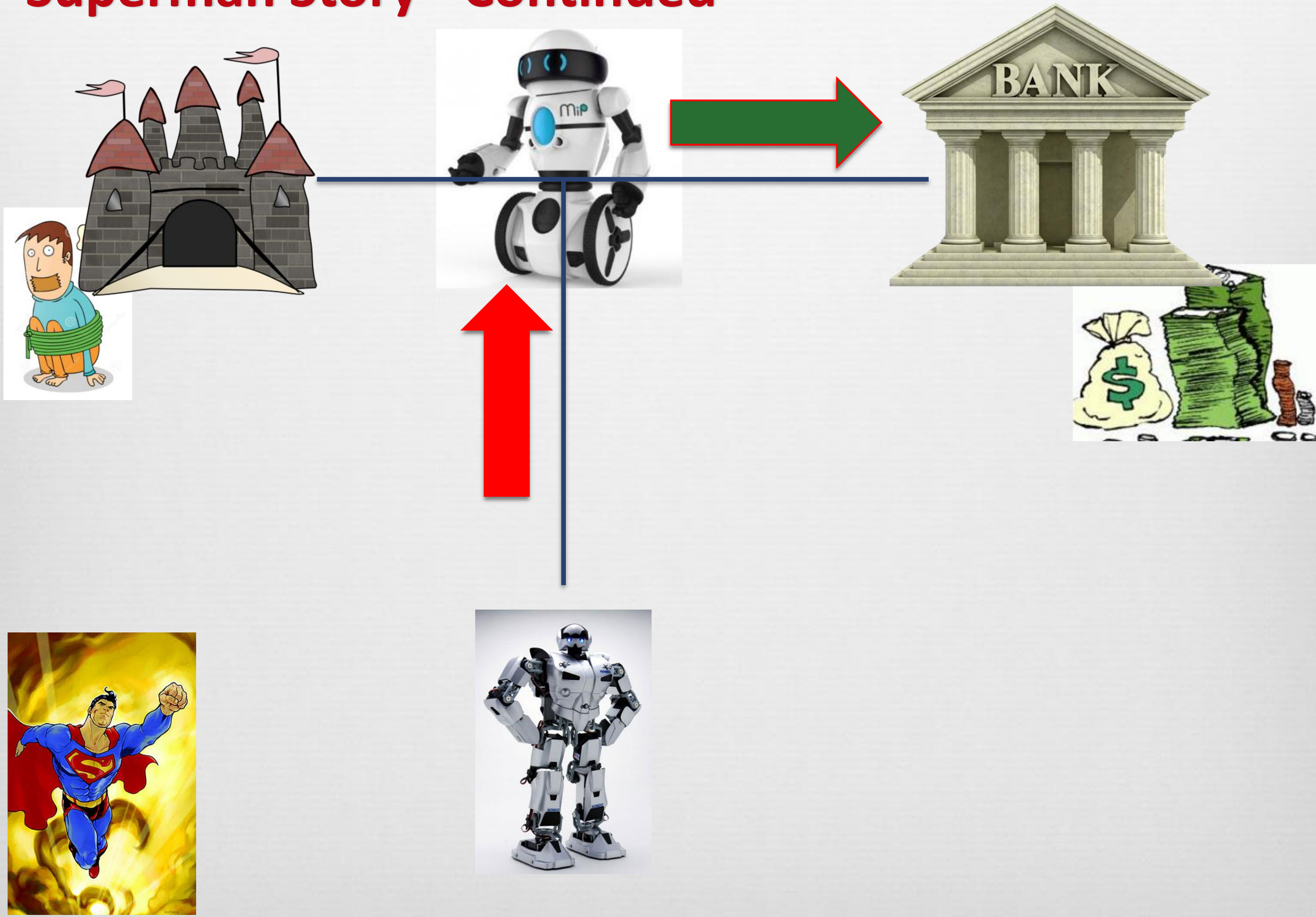
Superman Story - Continued



Superman Story - Continued

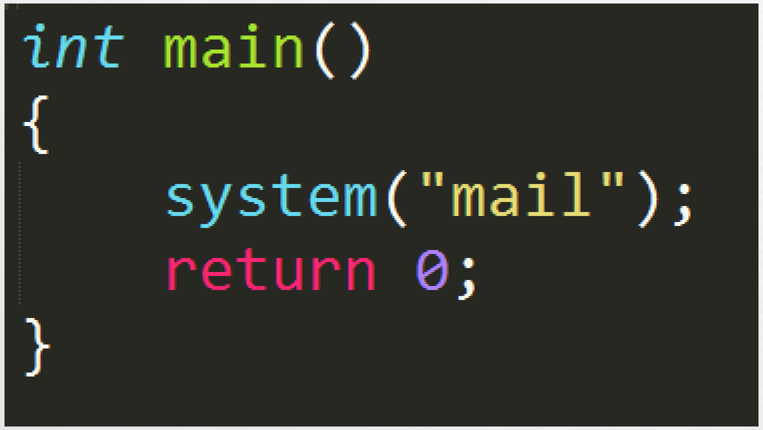


Superman Story - Continued



PATH Attack

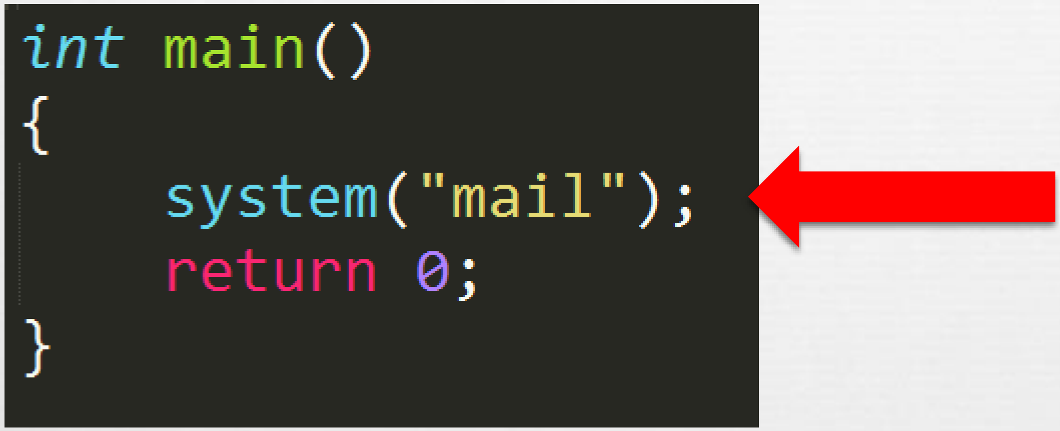
- A Set-UID program



```
int main()
{
    system("mail");
    return 0;
}
```

PATH Attack

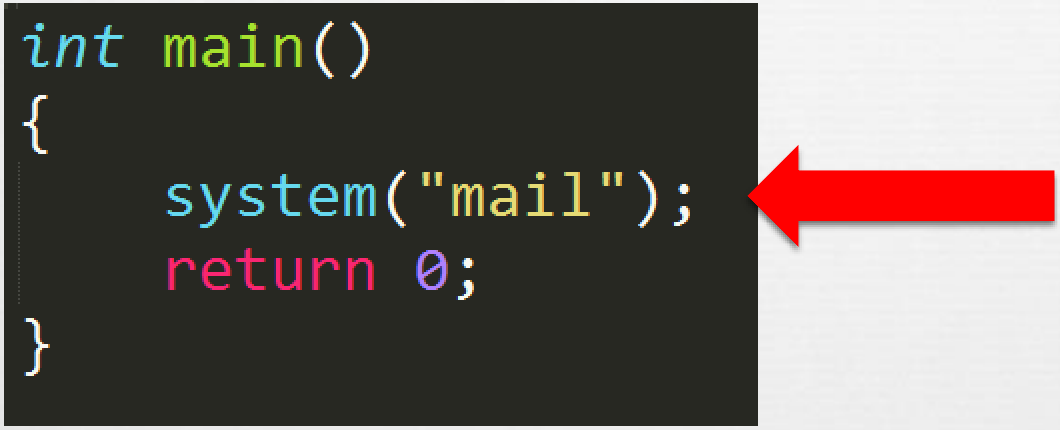
- A Set-UID program



```
int main()
{
     system("mail");
     return 0;
}
```

PATH Attack

- A Set-UID program

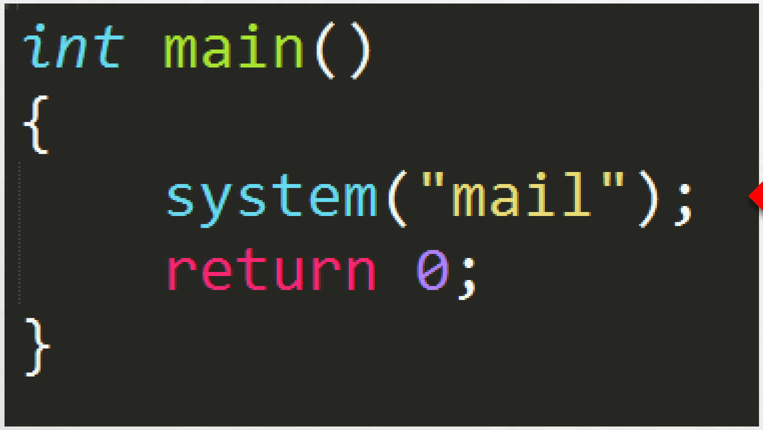


```
int main()
{
    system("mail");
    return 0;
}
```

"mail" program is located at /bin/mail, but how the system know this?

PATH Attack

- A Set-UID program



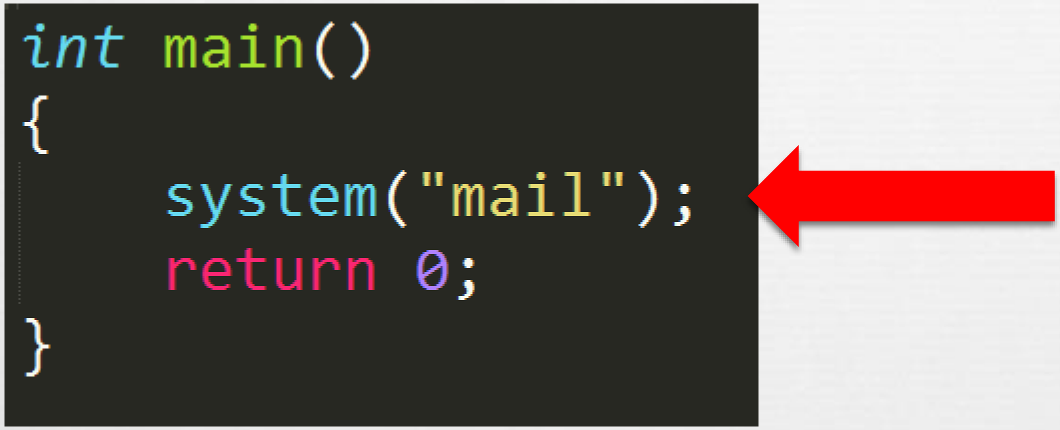
```
int main()
{
    system("mail");
    return 0;
}
```

“mail” program is located at /bin/mail, but how the system know this?

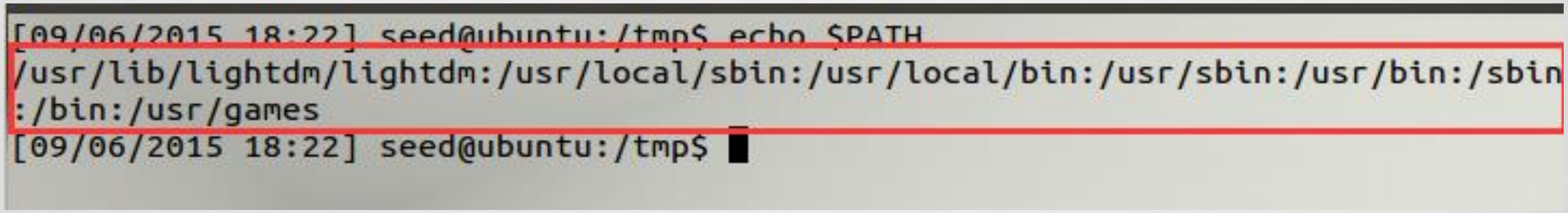
PATH: When running a command in a shell, the shell searches for the command using the PATH environment variable, which consists of a list of directories

PATH Attack

- A Set-UID program



```
int main()
{
    system("mail");
    return 0;
}
```

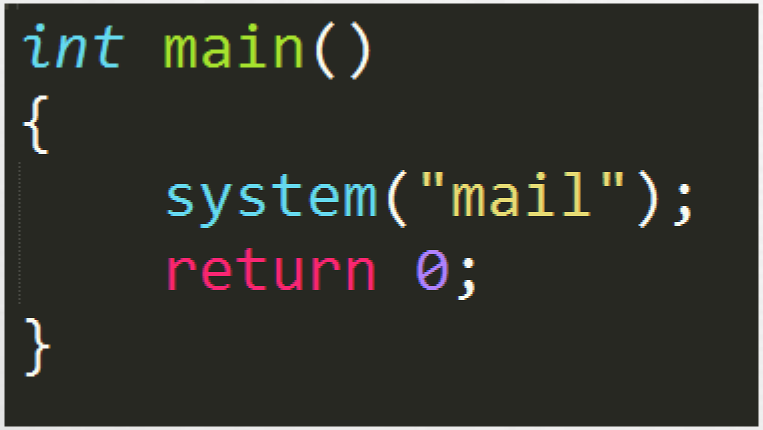


```
[09/06/2015 18:22] seed@ubuntu:/tmp$ echo $PATH
/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games
[09/06/2015 18:22] seed@ubuntu:/tmp$
```

PATH: When running a command in a shell, the shell searches for the command using the PATH environment variable, which consists of a list of directories

PATH Attack

- A Set-UID program

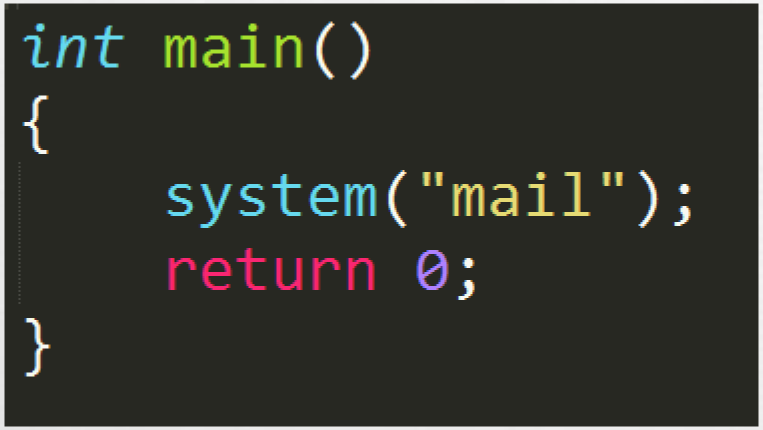


```
int main()
{
    system("mail");
    return 0;
}
```

1. Modify the PATH, add your current directory at the beginning of it.

PATH Attack

- A Set-UID program



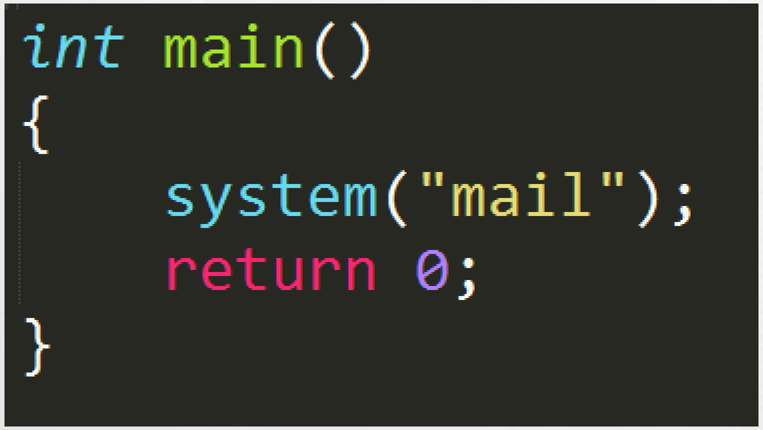
```
int main()
{
    system("mail");
    return 0;
}
```

1. Modify the PATH, add your current directory at the beginning of it.

2. Create your program, and name it as “mail”. Which program?

PATH Attack

- A Set-UID program



```
int main()
{
    system("mail");
    return 0;
}
```

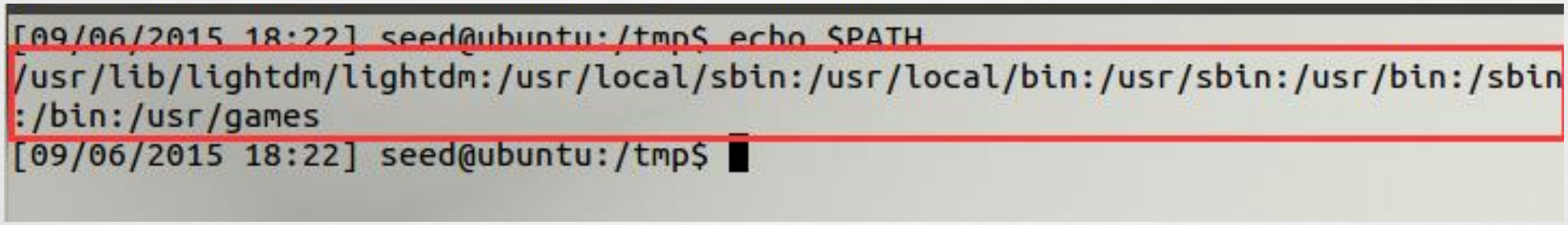
1. Modify the PATH, add your current directory at the beginning of it.

2. Create your program, and name it as “mail”. Which program?

3. **Shell**, will be executed with the root permission.

PATH Attack

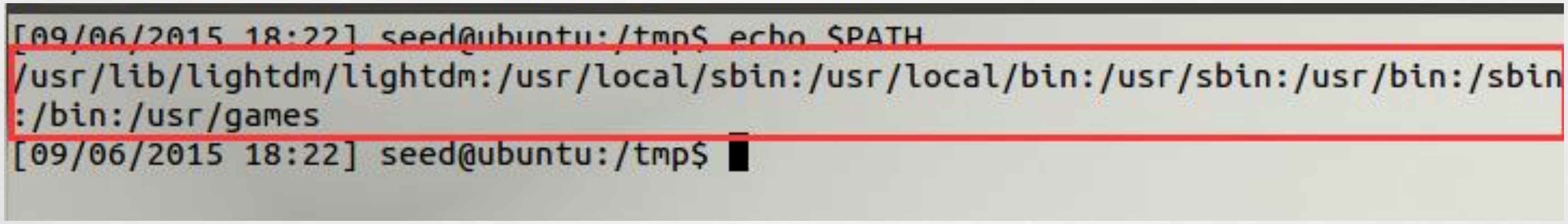
- Example:



```
[09/06/2015 18:22] seed@ubuntu:/tmp$ echo $PATH
/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games
[09/06/2015 18:22] seed@ubuntu:/tmp$
```

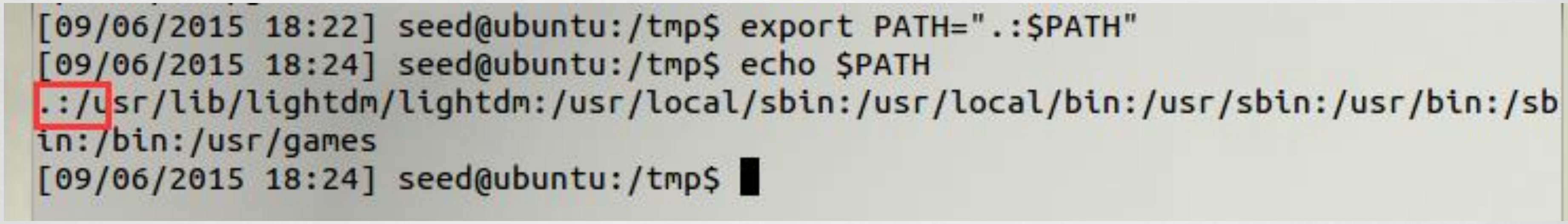
PATH Attack

- Example:



```
[09/06/2015 18:22] seed@ubuntu:/tmp$ echo $PATH
/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games
[09/06/2015 18:22] seed@ubuntu:/tmp$ 
```

- Add current directory to PATH



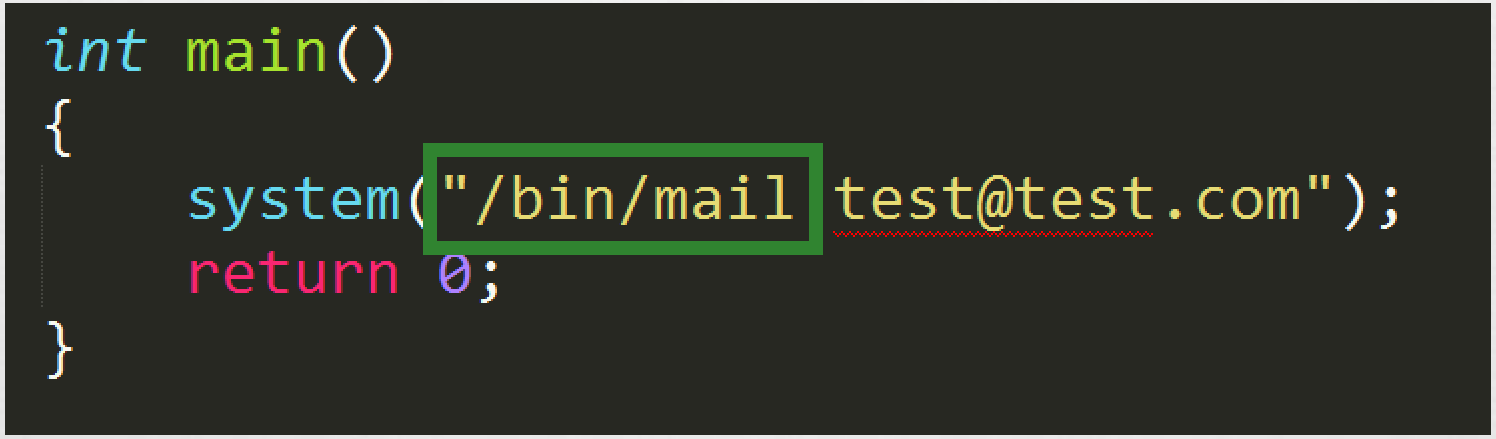
```
[09/06/2015 18:22] seed@ubuntu:/tmp$ export PATH=".:$PATH"
[09/06/2015 18:24] seed@ubuntu:/tmp$ echo $PATH
.:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sb
in:/bin:/usr/games
[09/06/2015 18:24] seed@ubuntu:/tmp$ 
```

- When you call a program now, the system will look up at your current directory first.

Are We Secure?

- The PATH attack happened since the programmer use the **relative** path for the program.

- If we use **absolute** path, **/bin/mail** instead of **mail**



```
int main()
{
	system("/bin/mail test@test.com");
	return 0;
}
```

- Is our Set-UID secure now?

Execution Attack

```
sprintf(command, "/bin/mail %s",user-input);
system(command)
```

- Now, you define the input.
- Can you consider other possible attacks?

Execution Attack

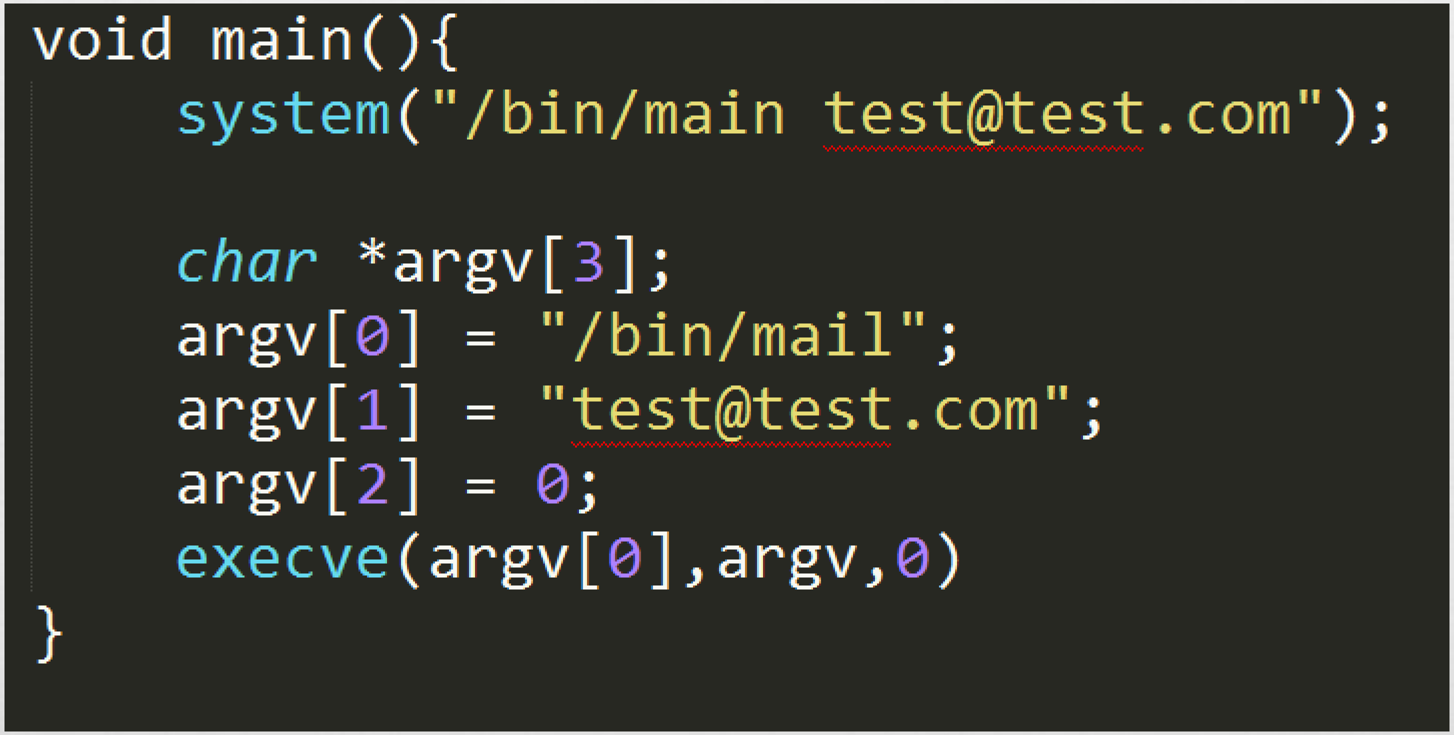
```
sprintf(command, "/bin/mail %s",user-input);
system(command)
```

- Now, you define the input.
- Can you consider other possible attacks?
- You input: test@test.com; your command
- The system will run the mail program first, then also runs your command

Execve

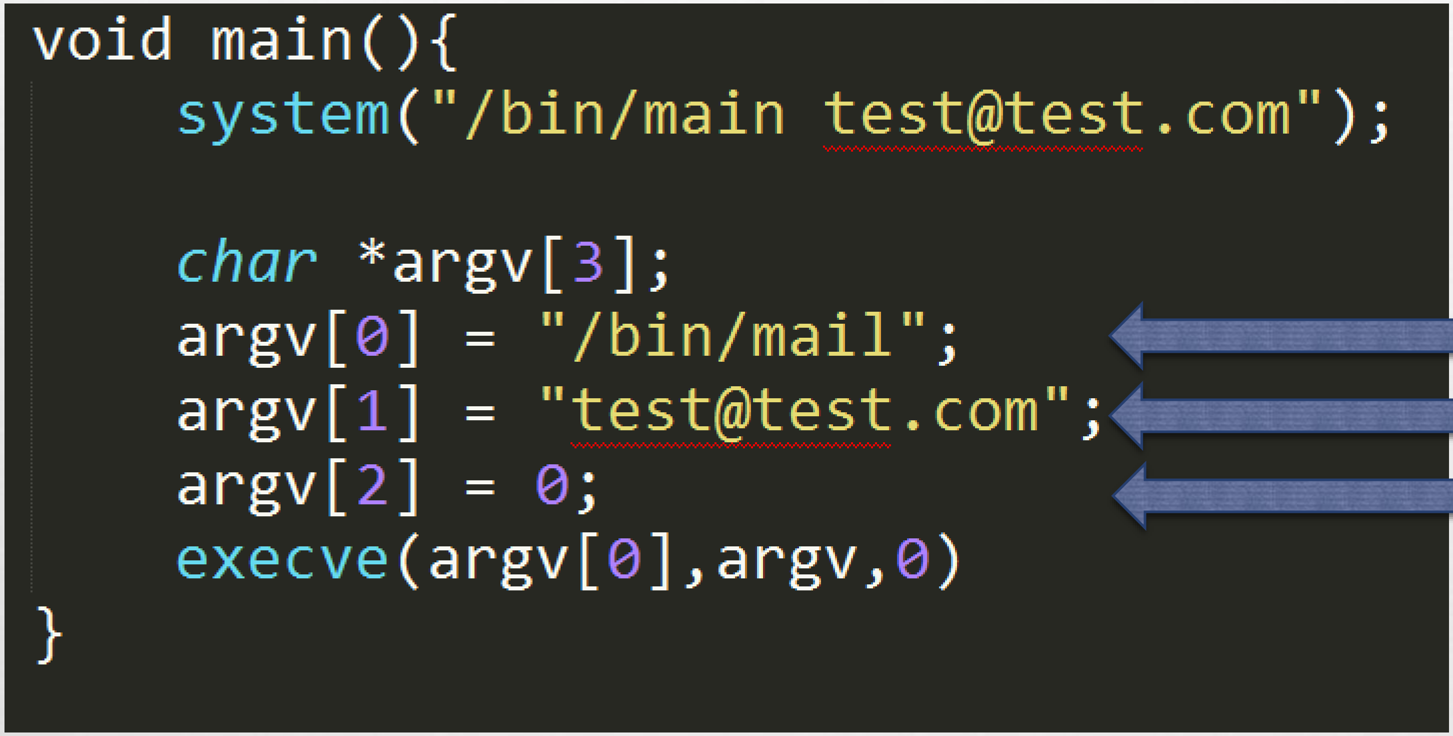
- **system()** invokes the **shell**, which can cause too many potential risks

- Use **execve()** instead of **system()** when you need to invoke a sub program



```
void main(){
    system("/bin/main test@test.com");

    char *argv[3];
    argv[0] = "/bin/mail";
    argv[1] = "test@test.com";
    argv[2] = 0;
    execve(argv[0],argv,0)
}
```

Execve



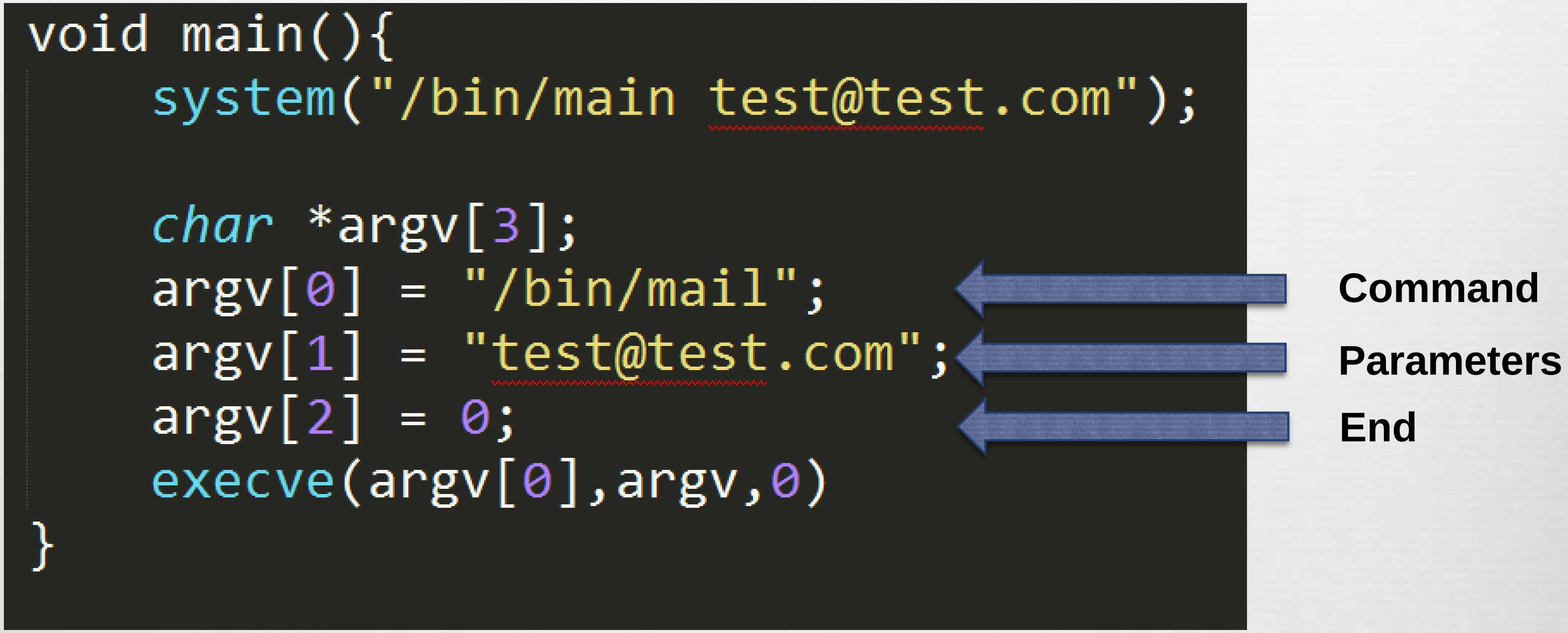
```
void main(){
     system("/bin/main test@test.com");

     char *argv[3];
     argv[0] = "/bin/mail";
     argv[1] = "test@test.com";
     argv[2] = 0;
     execve(argv[0],argv,0)
}
```

Command

Parameters

End

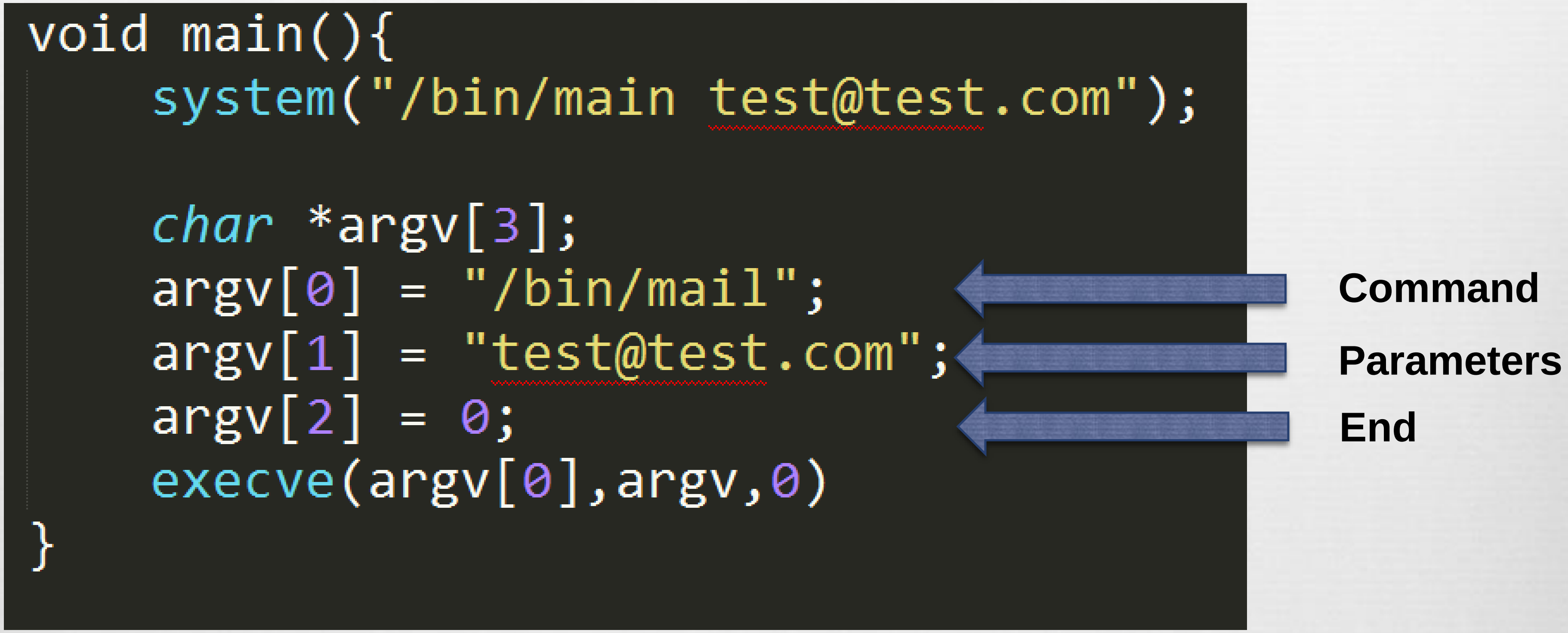
Execve



```c
void main(){
    system("/bin/main test@test.com");

    char *argv[3];
    argv[0] = "/bin/mail";
    argv[1] = "test@test.com";
    argv[2] = 0;
    execve(argv[0],argv,0)
}
```

test@test.com; your command, work or not?

Execve

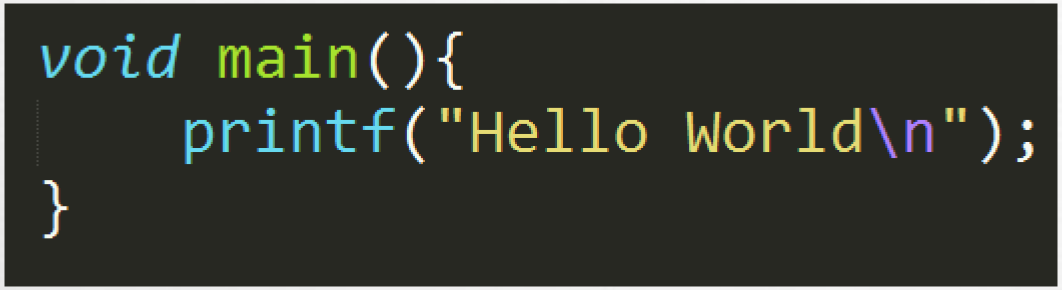


```
void main(){
    system("/bin/main test@test.com");

    char *argv[3];
    argv[0] = "/bin/mail";
    argv[1] = "test@test.com";
    argv[2] = 0;
    execve(argv[0],argv,0)
}
```

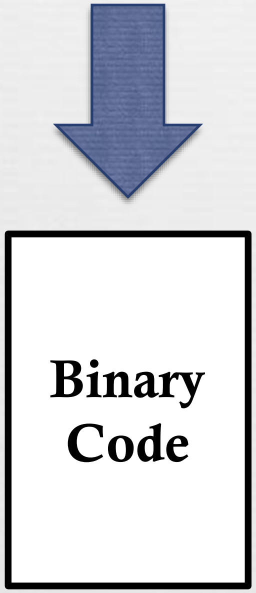
test@test.com; your command, work or not?

No, treat it as a whole, because we are not using the shell

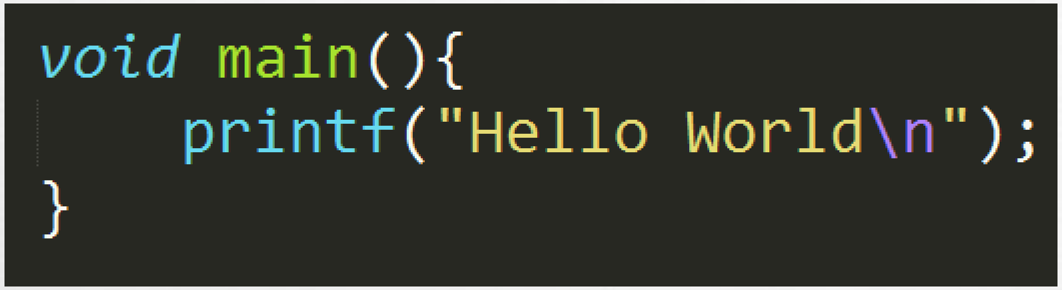
Library Interposition



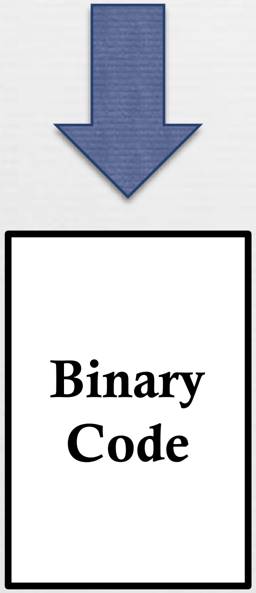
```
void main(){
    printf("Hello World\n");
}
```



Library Interposition

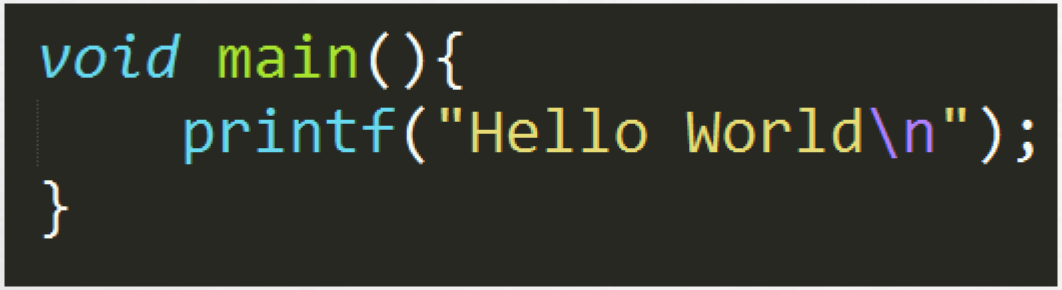


```
void main(){
    printf("Hello World\n");
}
```

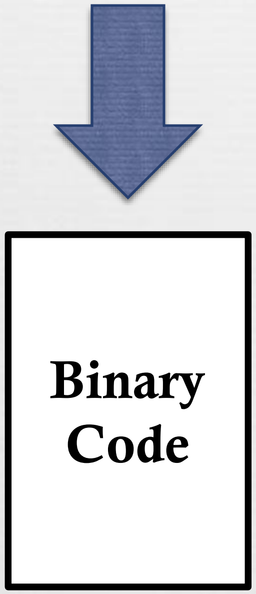


Where is printf() comes from?

Library Interposition



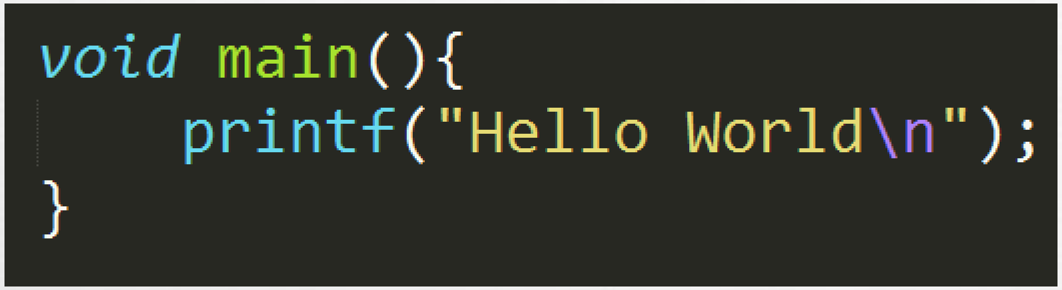
```
void main(){
    printf("Hello World\n");
}
```



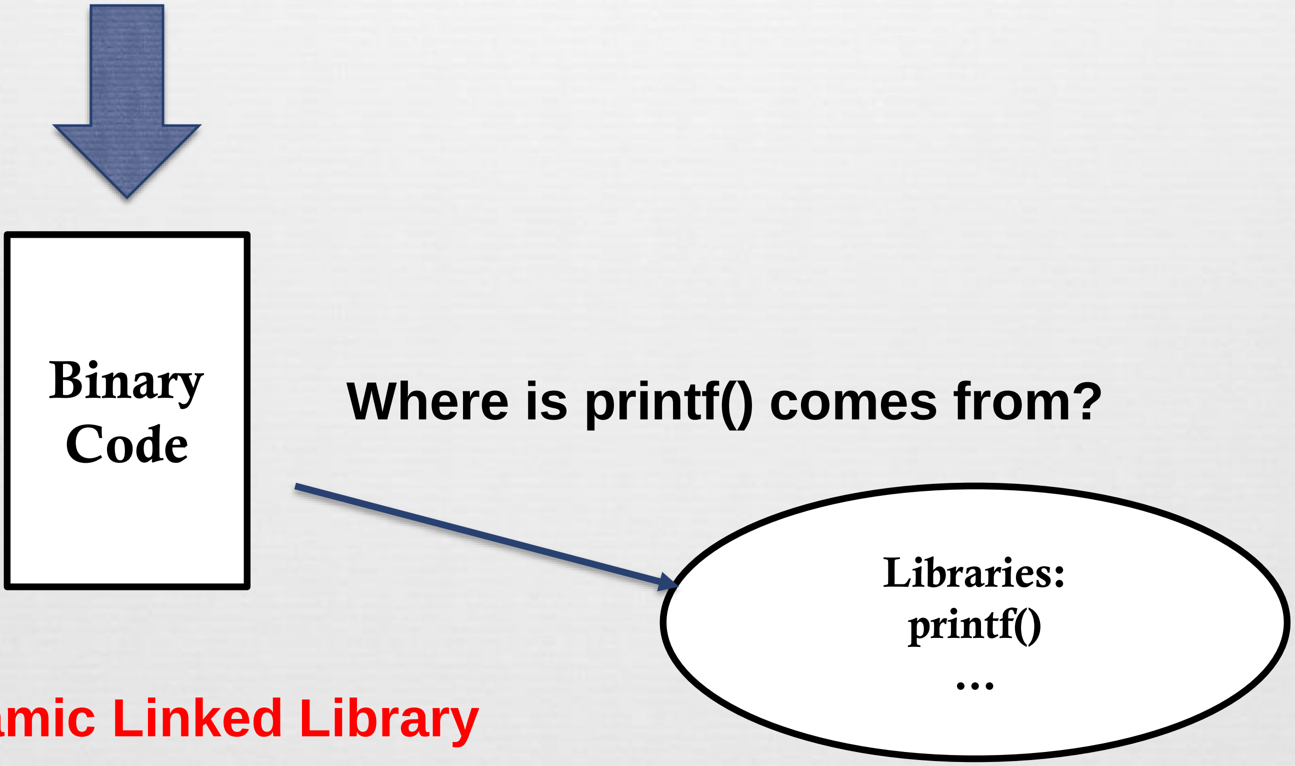
Where is printf() comes from?

Dynamic Linked Library

Library Interposition



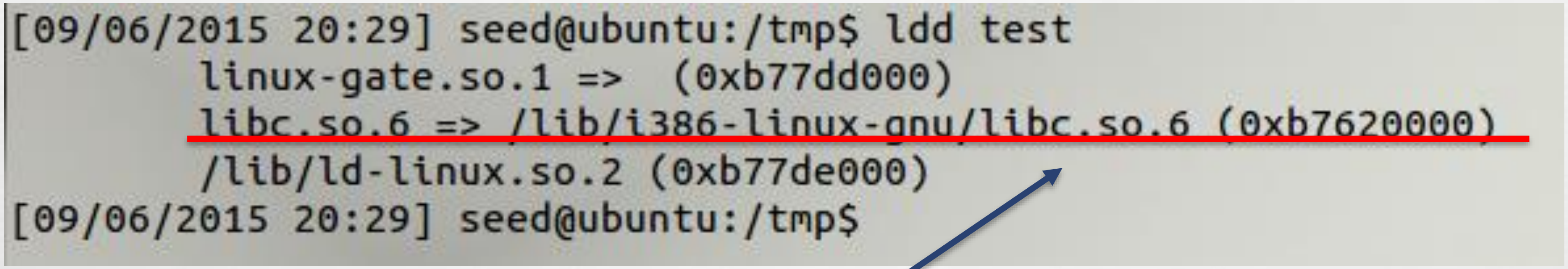
```c
void main(){
    printf("Hello World\n");
}
```



Where is printf() comes from?

Dynamic Linked Library

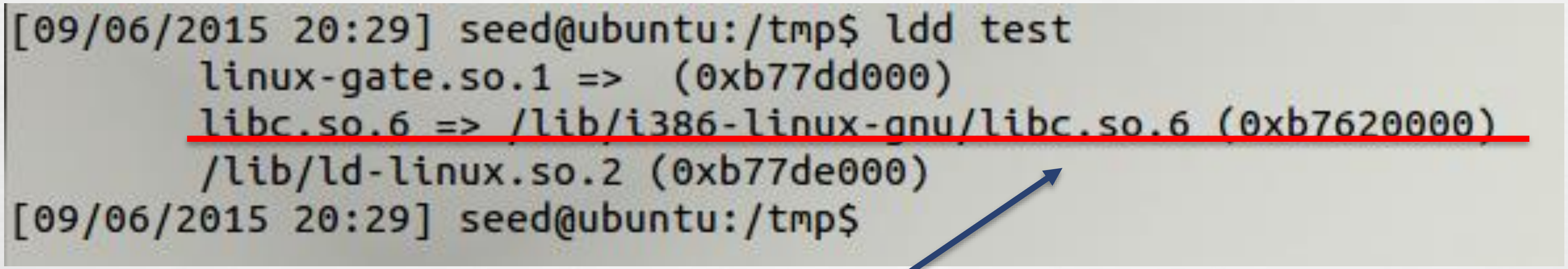
Library Interposition



```
[09/06/2015 20:29] seed@ubuntu:/tmp$ ldd test
        linux-gate.so.1 =>  (0xb77dd000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7620000)
        /lib/ld-linux.so.2 (0xb77de000)
[09/06/2015 20:29] seed@ubuntu:/tmp$
```

This is printf()

Library Interposition

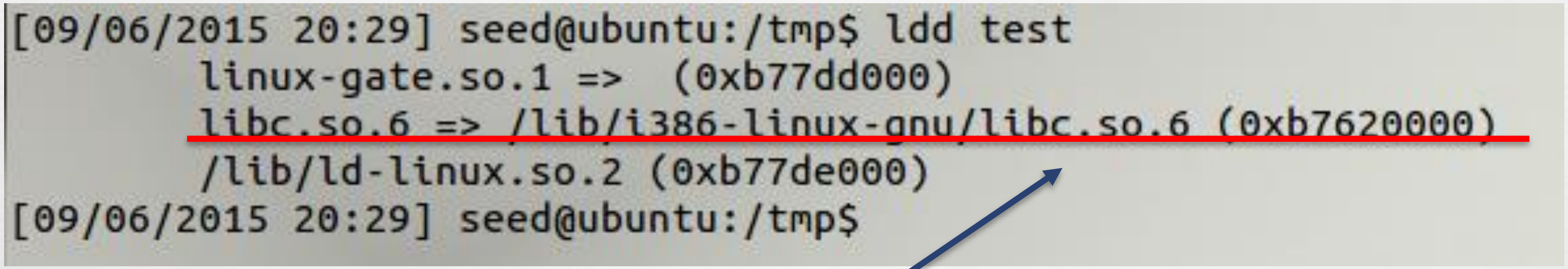


```
[09/06/2015 20:29] seed@ubuntu:/tmp$ ldd test
        linux-gate.so.1 =>  (0xb77dd000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7620000)
        /lib/ld-linux.so.2 (0xb77de000)
[09/06/2015 20:29] seed@ubuntu:/tmp$
```

This is printf()

How the system know where are these libraries?

Library Interposition



```
[09/06/2015 20:29] seed@ubuntu:/tmp$ ldd test
        linux-gate.so.1 =>  (0xb77dd000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7620000)
        /lib/ld-linux.so.2 (0xb77de000)
[09/06/2015 20:29] seed@ubuntu:/tmp$
```

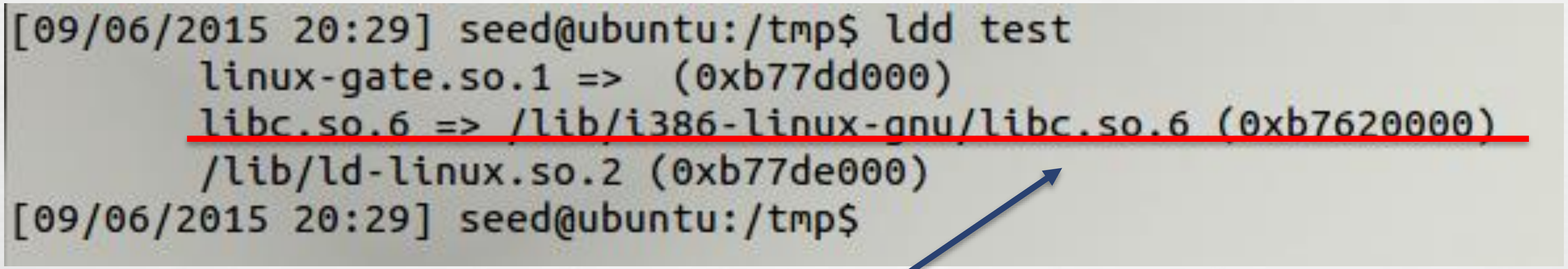
This is printf()

How the system know where are these libraries?

Environment Variable: LD_LIBRARY_PATH="xxx:xxxx:xxxx:xxx….."

A list of directories, similar to **PATH**

Library Interposition



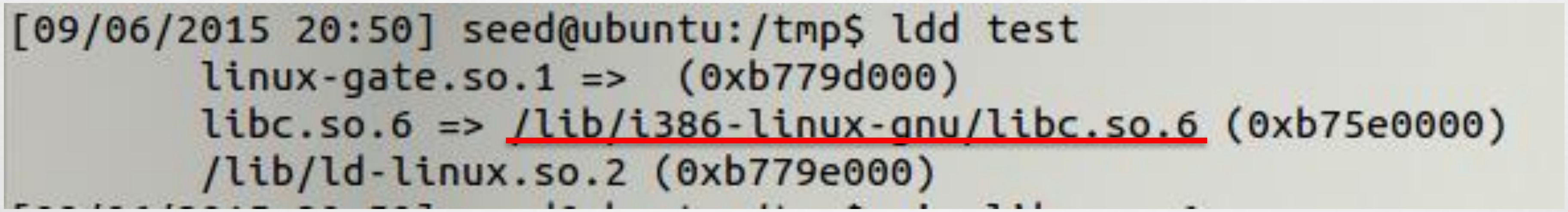
```
[09/06/2015 20:29] seed@ubuntu:/tmp$ ldd test
        linux-gate.so.1 =>  (0xb77dd000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7620000)
        /lib/ld-linux.so.2 (0xb77de000)
[09/06/2015 20:29] seed@ubuntu:/tmp$
```

This is printf()

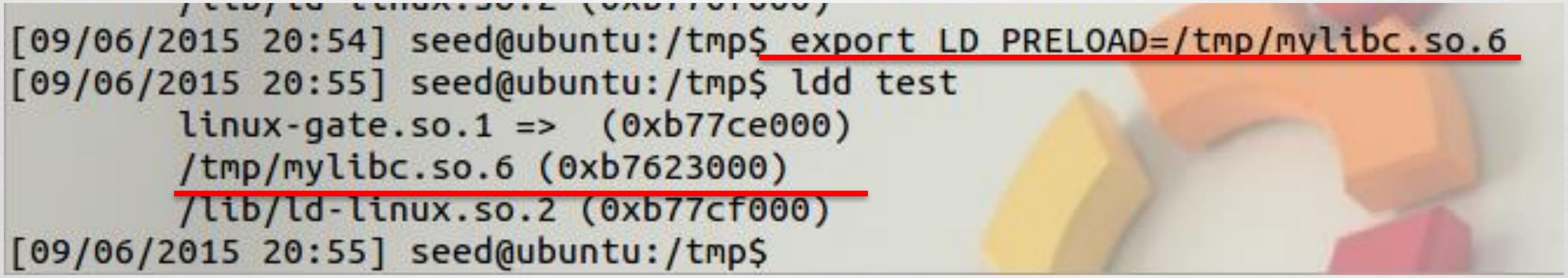
Environment Variable: LD_LIBRARY_PATH=“xxx:xxxx:xxxx:xxx…..”

LD_PROLOAD: Many Unix systems allow you to ”pre-load” shared libraries by setting an environment variable LD_PRELOAD. These user specified libraries will be **loaded before all others**. This can be used to selectively override functions in other libraries

Library Interposition



```
[09/06/2015 20:50] seed@ubuntu:/tmp$ ldd test
        linux-gate.so.1 =>  (0xb779d000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb75e0000)
        /lib/ld-linux.so.2 (0xb779e000)
```



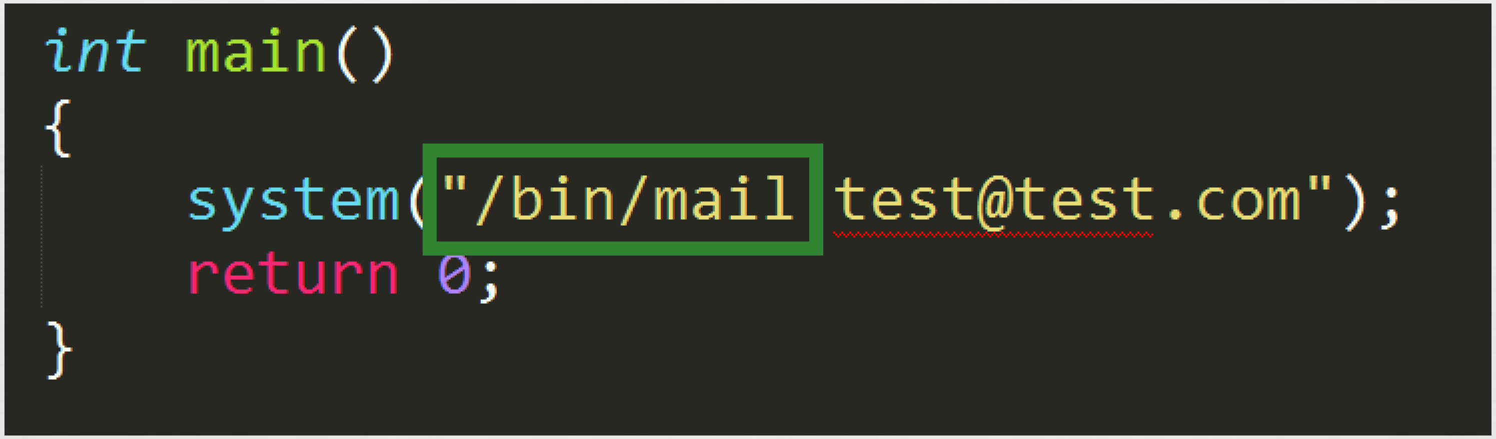
```
[09/06/2015 20:54] seed@ubuntu:/tmp$ export LD_PRELOAD=/tmp/mylibc.so.6
[09/06/2015 20:55] seed@ubuntu:/tmp$ ldd test
       linux-gate.so.1 =>  (0xb77ce000)
       /tmp/mylibc.so.6 (0xb7623000)
       /lib/ld-linux.so.2 (0xb77cf000)
[09/06/2015 20:55] seed@ubuntu:/tmp$
```

How the system protect this kind of vulnerability?

- Think about this based on your **lab task 5**.

Are We Secure?

- The PATH attack happened since the programmer use the **relative** path for the program.
- If we use **absolute** path, **/bin/mail** instead of **mail**



```
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

- Is our Set-UID secure now?
- **No!**

Internal field separator (IFS) Attack

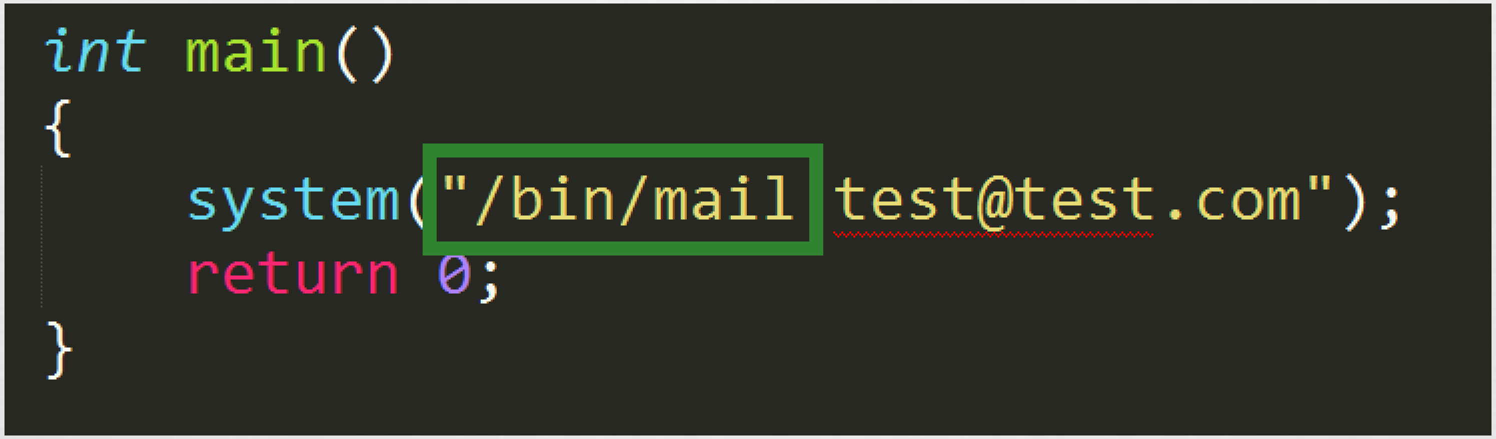
- **IFS:** variable that determines the characters which are to be interpreted as white spaces

- Example:
  - if we **IFS**=“abcd“, command “attat” -> “ tt t”

Internal field separator (IFS) Attack

- **IFS:** variable that determines the characters which are to be interpreted as white spaces

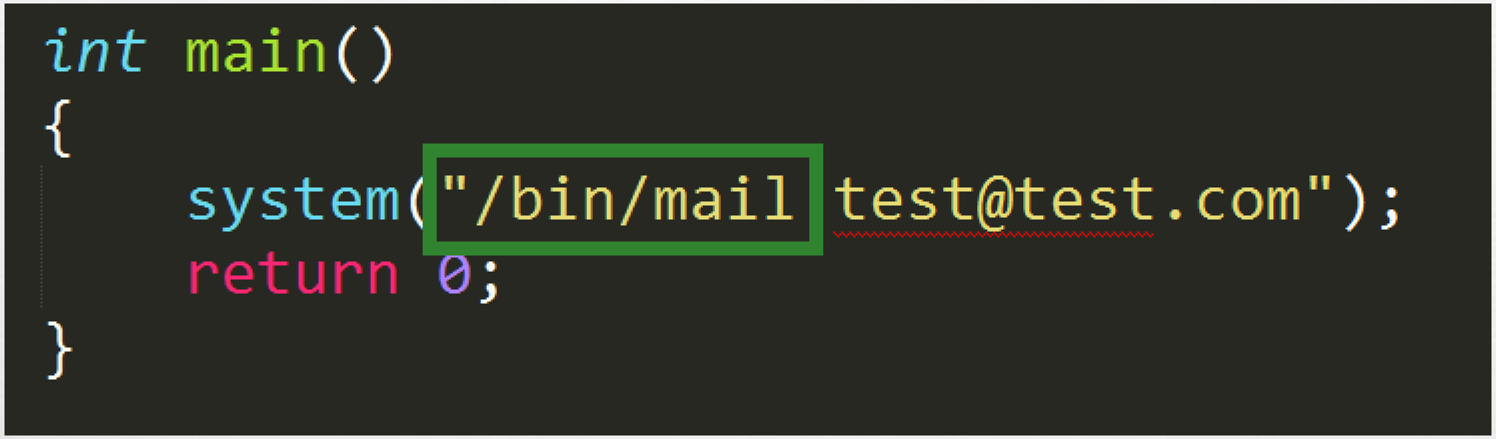
- Example:
    - if we **IFS**="abcd", command "attat" -> " tt t"



```
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

- Can you use IFS to perform attack on this Set-UID program?

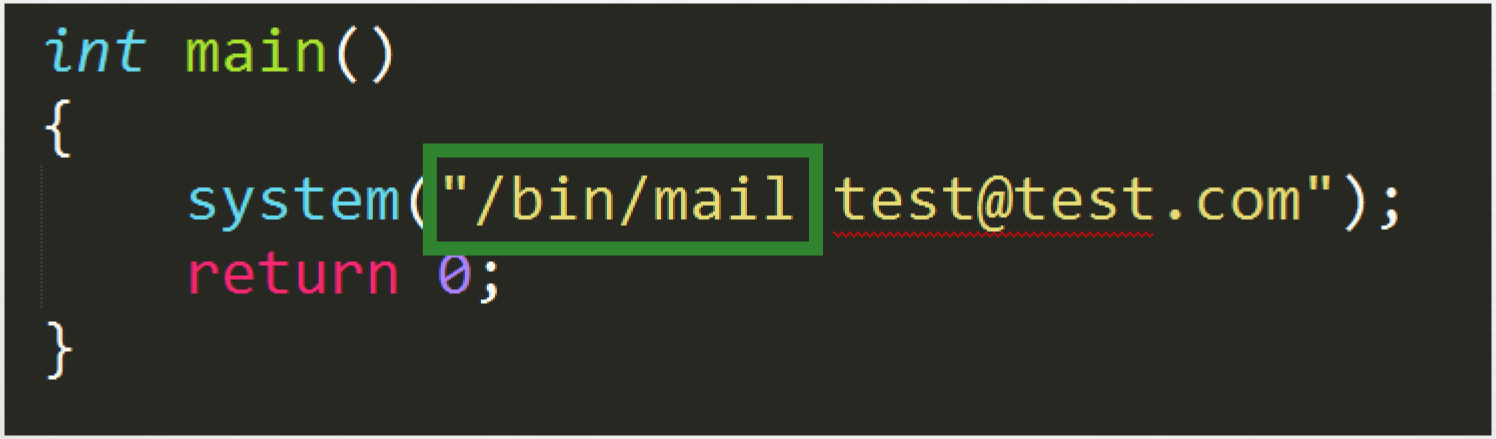
Internal field separator (IFS) Attack



```c
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

1. Add “/” into IFS, IFS=“/ ”; export IFS

Internal field separator (IFS) Attack

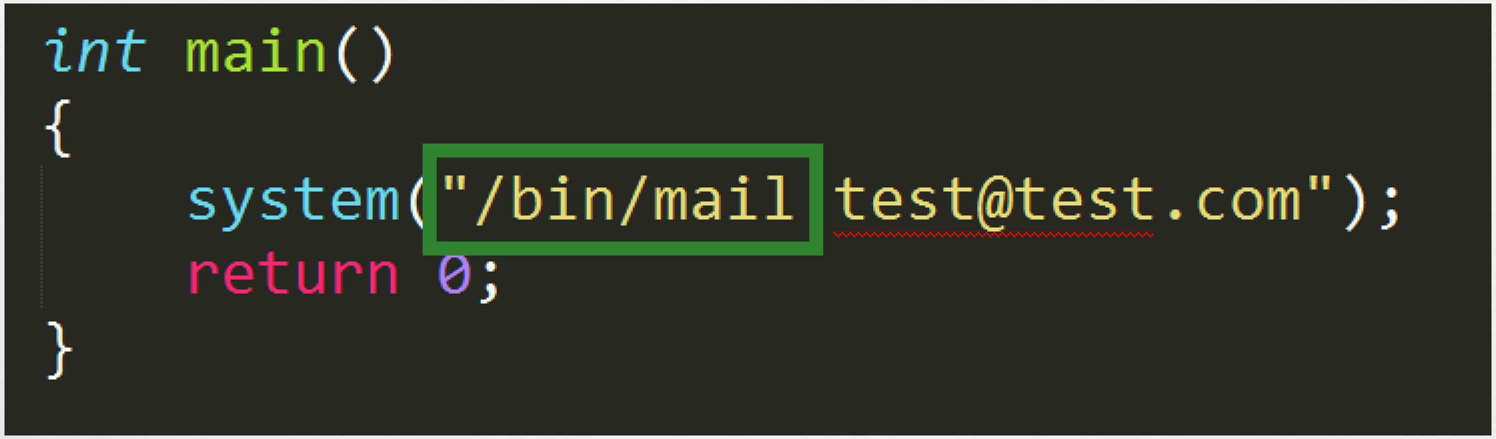


```
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

1. Add “/” into IFS, IFS=“/ ”; export IFS

2. Now, “/bin/mail test@test.com” -> “ bin mail test@test.com”

Internal field separator (IFS) Attack



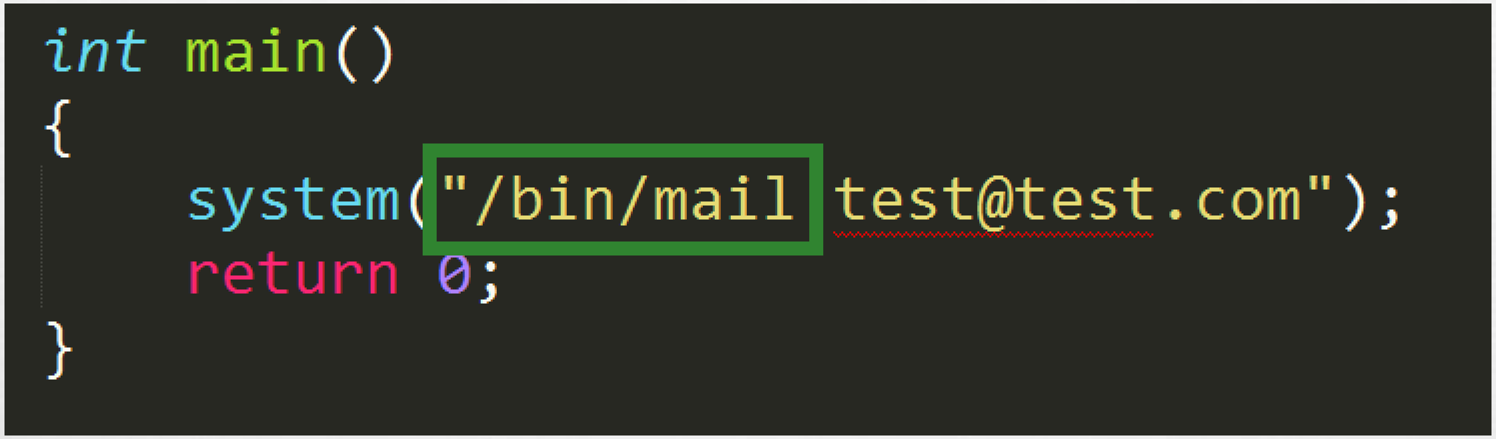
```
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

1. Add "/" into IFS, IFS="/ "; export IFS

2. Now, "/bin/mail test@test.com" -> " bin mail test@test.com"

3. The system thinks you are call program **bin** with **parameters mail** and **test@test.com**

Internal field separator (IFS) Attack



```
int main()
{
    system("/bin/mail test@test.com");
    return 0;
}
```

1. Add “/” into IFS, IFS=“/ ”; export IFS

2. Now, “/bin/mail test@test.com” -> “ bin mail test@test.com”

3. The system thinks you are call program **bin** with **parameters mail** and **test@test.com**

4. Now, you can perform the same attack as the PATH attack