

Programmazione Sicura



Iniezione locale
(prima parte)



Barbara Masucci

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Nelle lezioni precedenti abbiamo visto come vengono gestiti **elevazione e ripristino dei privilegi** nei vari sistemi UNIX-like



- **Scopo della lezione di oggi:**
 - Analizzare la tecnica di manipolazione delle variabili d'ambiente per **l'iniezione locale di codice**
 - Risolvere le **prime due sfide** Capture The Flag su una particolare macchina virtuale: **NEBULA**



Cosa faremo

- A partire da questa lezione studieremo in profondità alcune **tipologie di vulnerabilità**
 - Sotto quali ipotesi si verificano?
 - Quali conseguenze hanno?
 - Come si possono mitigare?
- L'indagine avrà una **forte connotazione pratica**
 - Avremo a disposizione una **macchina virtuale** su cui fare prove, in piena autonomia e libertà



Come agire?

- Il **modo non corretto di agire** prevede l'esecuzione delle azioni seguenti:
 - Provare comandi a casaccio, senza avere idea di cosa si stia facendo
 - Copiare soluzioni messe a punto da altri, senza avere idea di cosa si stia facendo
 - Utilizzare strumenti automatici di attacco, senza avere idea del loro funzionamento



Come agire?

- Il **modo corretto di agire** invece prevede la piena consapevolezza delle proprie azioni
 - Conoscere tutti i dettagli dell'ambiente che si sta studiando
 - Identificare tutti i modi possibili (plausibili ed improbabili) di condurre un attacco
 - Provare l'attacco sui sistemi su cui si ha il permesso di operare
 - Capire nel dettaglio modalità e conseguenze dell'attacco
 - Capire come mitigare l'attacco



Albero di attacco

- E' uno strumento utile per la conduzione ragionata di attività di attacco
- Rappresenta una **vista gerarchica** dei possibili attacchi ad un sistema
 - Ogni nodo dell'albero è un'azione
 - Il nodo radice è l'azione finale dell'attacco
 - Ciascun nodo foglia è una azione iniziale dell'attacco
 - Ciascun nodo intermedio rappresenta un'azione preliminare per poter svolgere l'azione rappresentata dal nodo padre



Albero di attacco

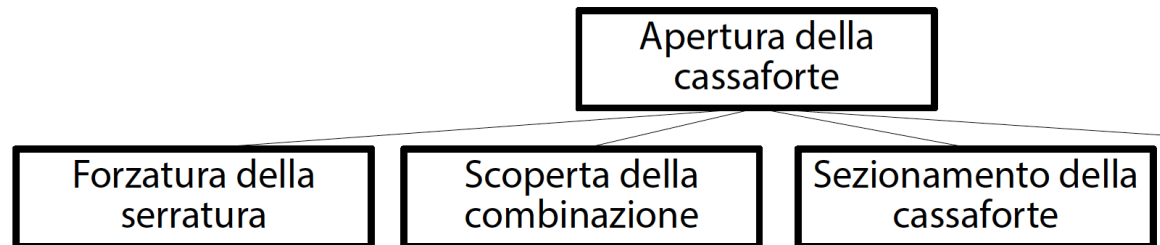
- Supponiamo di voler aprire una cassaforte
- Il **nodo radice** dell'albero rappresenta proprio l'obiettivo dell'attacco

Apertura della
cassaforte



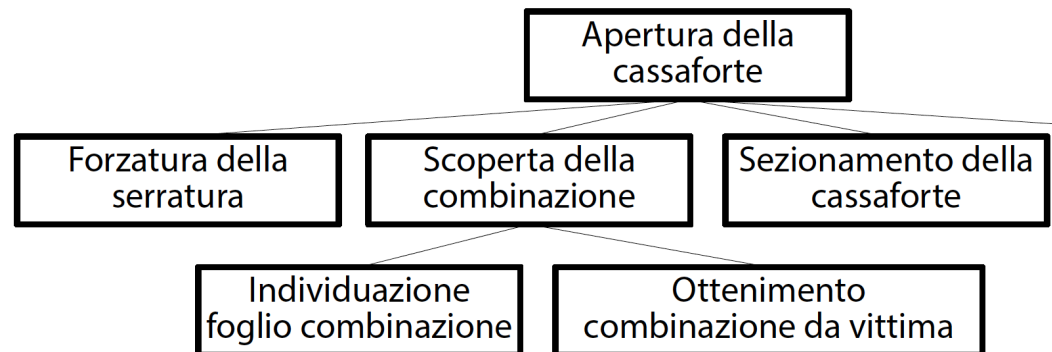
Albero di attacco

- La cassaforte può essere aperta se almeno una delle azioni rappresentate nei **nodi foglia** ha successo



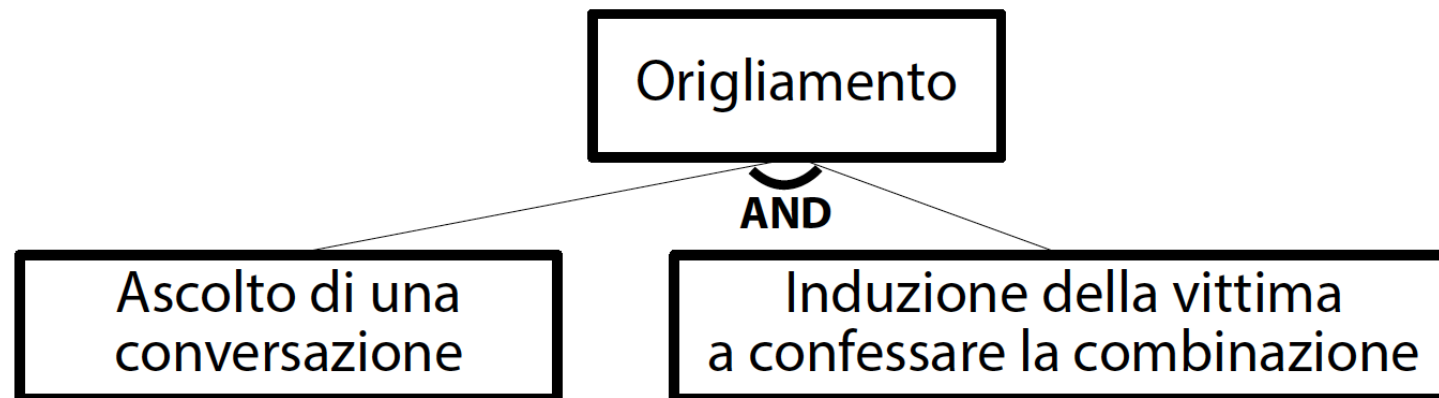
Albero di attacco

- Le **azioni intermedie** hanno bisogno, a loro volta, del successo di almeno un'altra azione preliminare



Albero di attacco

- Alcune azioni necessitano l'esecuzione di più azioni preliminari
 - Si modellano con un AND e un arco



Albero di attacco

- Un **possibile attacco** è un OR di percorsi (incrocianti su un nodo AND) da nodi foglia al nodo radice



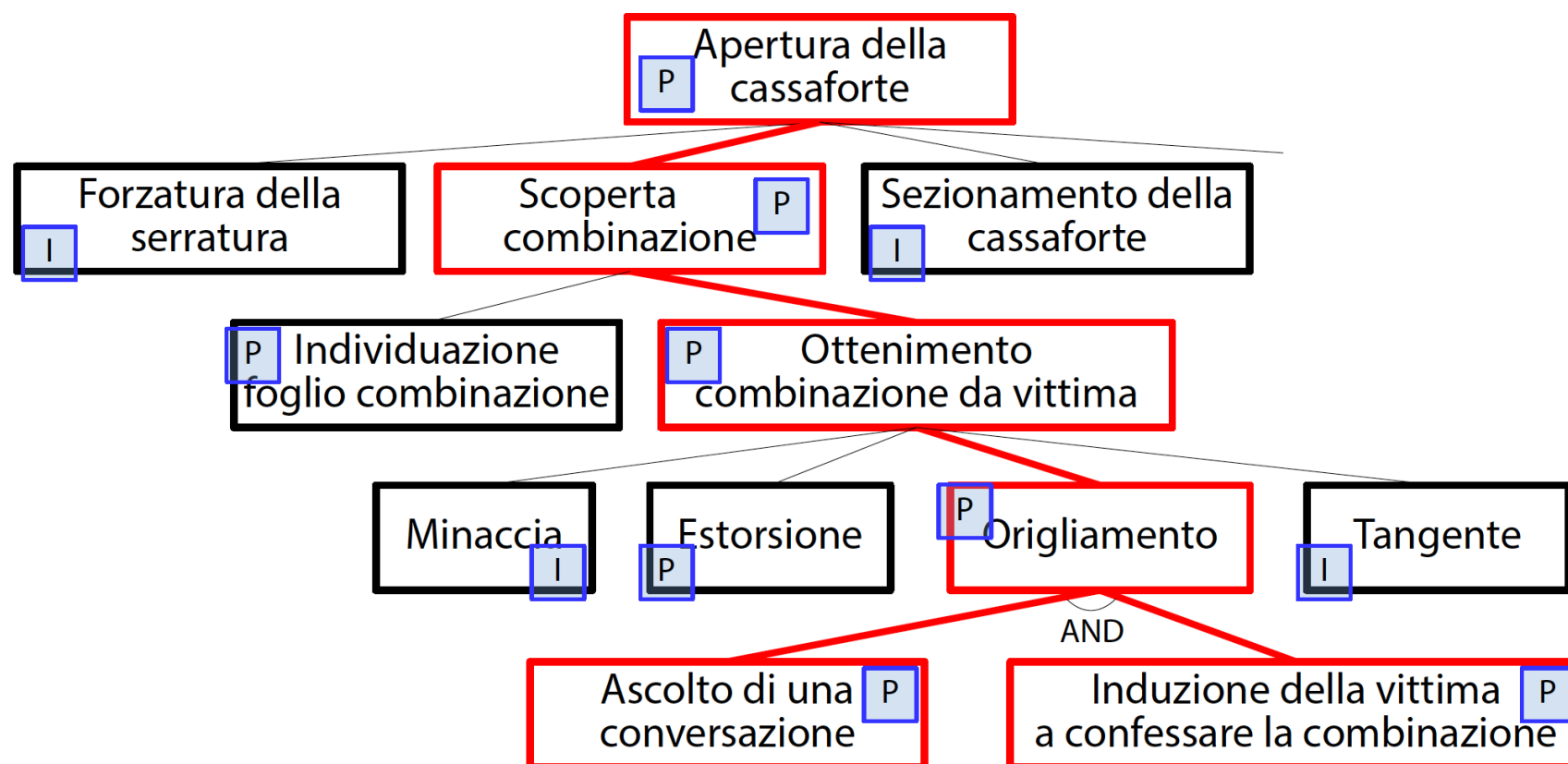
Etichettatura dei nodi

- Una volta definito, l'albero di attacco può essere arricchito con opportune **etichette** sui nodi
 - Fattibilità dell'azione (Possibile, Impossibile)
 - Costo dell'azione
 - Probabilità di successo
- Aggregando le etichette nel percorso da una foglia alla radice, è possibile **stimare l'attacco**



Etichettatura: Fattibilità

➤ Etichette: P (Possibile), I (Impossibile)

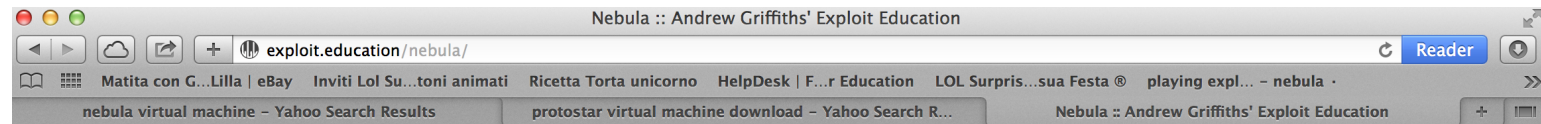


La macchina virtuale Nebula

- La macchina virtuale **Nebula** contiene esercizi di sicurezza, organizzati come sfide (**challenge**)
- Ciascun esercizio corrisponde a un livello, per un totale di 20 livelli
 - Level00, Level01, ..., Level19
 - Ciascun livello **dichiara un obiettivo non banale** che l'utente deve cercare di ottenere con ogni mezzo possibile
 - I livelli dovrebbero essere eseguiti in sequenza
 - Vedremo solo alcuni di questi livelli
 - Gli altri sono disponibili per i progetti



La macchina virtuale Nebula



Exploit Education

1. Phoenix

2. Nebula

[Level 00](#)

[Level 01](#)

[Level 02](#)

[Level 03](#)

[Level 04](#)

[Level 05](#)

[Level 06](#)

[Level 07](#)

[Level 08](#)

[Level 09](#)

[Level 10](#)

[Level 11](#)

[Level 12](#)

[Level 13](#)

[Level 14](#)

[Level 15](#)

[Level 16](#)

[Level 17](#)

Exploit Education > Nebula

Nebula takes the participant through a variety of common (and less than common) weaknesses and vulnerabilities in Linux. It takes a look at

- SUID files
- Permissions
- Race conditions
- Shell meta-variables
- \$PATH weaknesses
- Scripting language weaknesses
- Binary compilation failures

At the end of Nebula, the user will have a reasonably thorough understanding of local attacks against Linux systems, and a cursory look at some of the remote attacks that are possible.

Download

Downloads are available from the [downloads](#) page.

Getting started



La macchina virtuale Nebula

- Disponibile al link
<http://exploit.education/nebula/>
- Installazione:
 - Scarichiamo l'immagine ISO
exploit-exercises-nebula-5.iso
da <http://exploit.education/downloads/>
 - Successivamente, importiamola in VirtualBox o VMware, creando una nuova macchina virtuale



La macchina virtuale Nebula

- Gli account a disposizione sono di due tipi:
 - **Giocatori**
Un utente che intende partecipare alla sfida (simulando il ruolo dell'**attaccante**) si autentica con le credenziali seguenti
 - Username: levelN (N=00, 01,...,19)
 - Password: levelN (N=00, 01,...,19)
 - **Vittime**
Gli account flag00,...,flag19 simulano una **vittima** e contengono vulnerabilità di vario tipo



La macchina virtuale Nebula

- C'è anche un account che simula un **amministratore di sistema**
 - Username: nebula
 - Password: nebula
- L'elevazione dei privilegi a root può essere effettuata manualmente tramite il comando `sudo`



La macchina virtuale Nebula

- Cosa fa l'utente levelN dopo l'autenticazione?
- Usa le informazioni contenute nella directory /home/flagN per conseguire uno specifico obiettivo
 - Esecuzione di un programma con privilegi elevati
 - Ottenimento di informazioni sensibili



Level 00

"This level requires you to find a Set User ID program that will run as the flag00 account.

You could also find this by carefully looking in top level directories in / for suspicious looking directories."



Level 00

- Per individuare tutti i file con il bit **SETUID** acceso possiamo usare utilizzare il comando **find** con l'opzione **-perm** (filtro per permessi)

```
find / -perm /u+s
```

- Per evitare di visualizzare i messaggi di errore (permission denied)

```
find / -perm /u+s 2>/dev/null
```



Level 00

- Per agevolare la consultazione, salviamo i risultati della ricerca in un file nella nostra home

```
find / -perm /u+s > /home/level00/permessi
```

- Tra i vari risultati della ricerca, notiamo il file

```
/bin/.../flag00
```



Level 00

- Con `ls -la` visualizziamo i metadati del file individuato
- E' di proprietà di `flag00` e ha il bit **SETUID** acceso

```
level00@nebula:/bin/...$ ls -la
total 8
drwxr-xr-x 2 root    root      29 2011-11-20 21:22 .
drwxr-xr-x 3 root    root     2728 2012-08-18 02:50 ..
-rwsr-x--- 1 flag00  level00  7358 2011-11-20 21:22 flag00
```

- Ora mandiamo in esecuzione il file
`./flag00`

- Viene stampato il messaggio
`Congrats, now run getflag to get your flag!`



Sfida vinta!



Level 01

- "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 `level01.c` e il suo eseguibile ha il seguente percorso:
`/home/flag01/flag01`



Level 01

level01.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)
{
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

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

    system("/usr/bin/env echo and now what?");
}
```



Capture the Flag!

- L'**obiettivo della sfida** è l'esecuzione del programma `/bin/getflag` con i privilegi dell'utente `flag01`
- L'obiettivo è visto come una "**bandierina**" da acciuffare per primi in un gioco a squadre
- Queste sfide vengono spesso chiamate con il termine **Capture the Flag (CTF)**



Costruzione di un albero di attacco

- Costruiamo l'albero di attacco del sistema considerato
 1. Iniziamo impostando il nodo radice
 2. Poi studiamo il sistema in profondità
 3. In seguito, aggiorniamo l'albero di attacco
 4. Se esiste un percorso fattibile da una foglia alla radice, STOP. Altrimenti vai al passo 2.

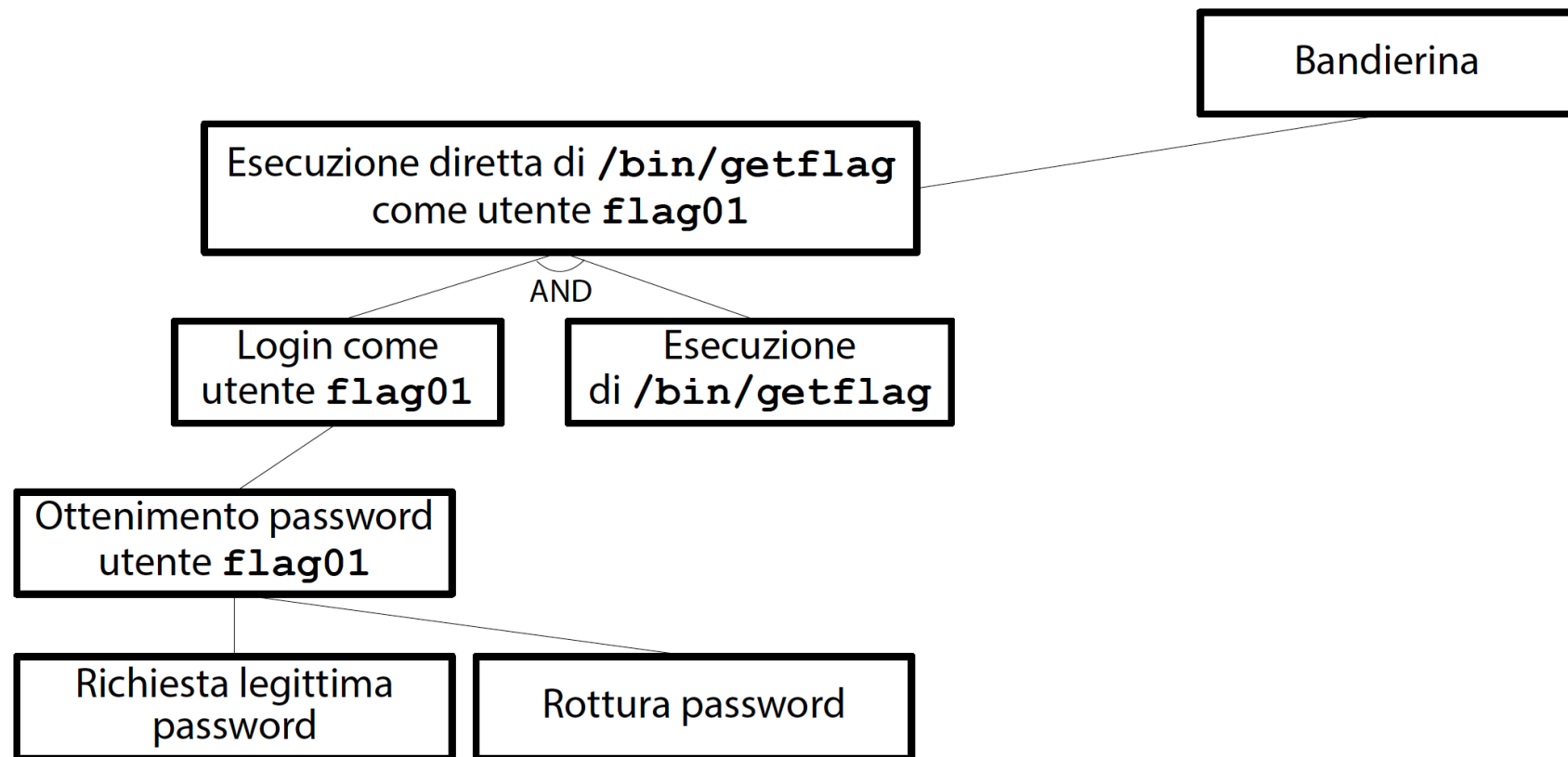


Costruzione di un albero di attacco

- L'obiettivo dell'attacco è l'esecuzione del programma `/bin/getflag` con i privilegi dell'utente `flag01`
- Come prima strategia (naïve) eseguiamo il login come utente `flag01` e proviamo ad eseguire `/bin/getflag`



Costruzione di un albero di attacco



Richiesta password

- A chi si potrebbe chiedere la password dell'account flag01?
 - 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

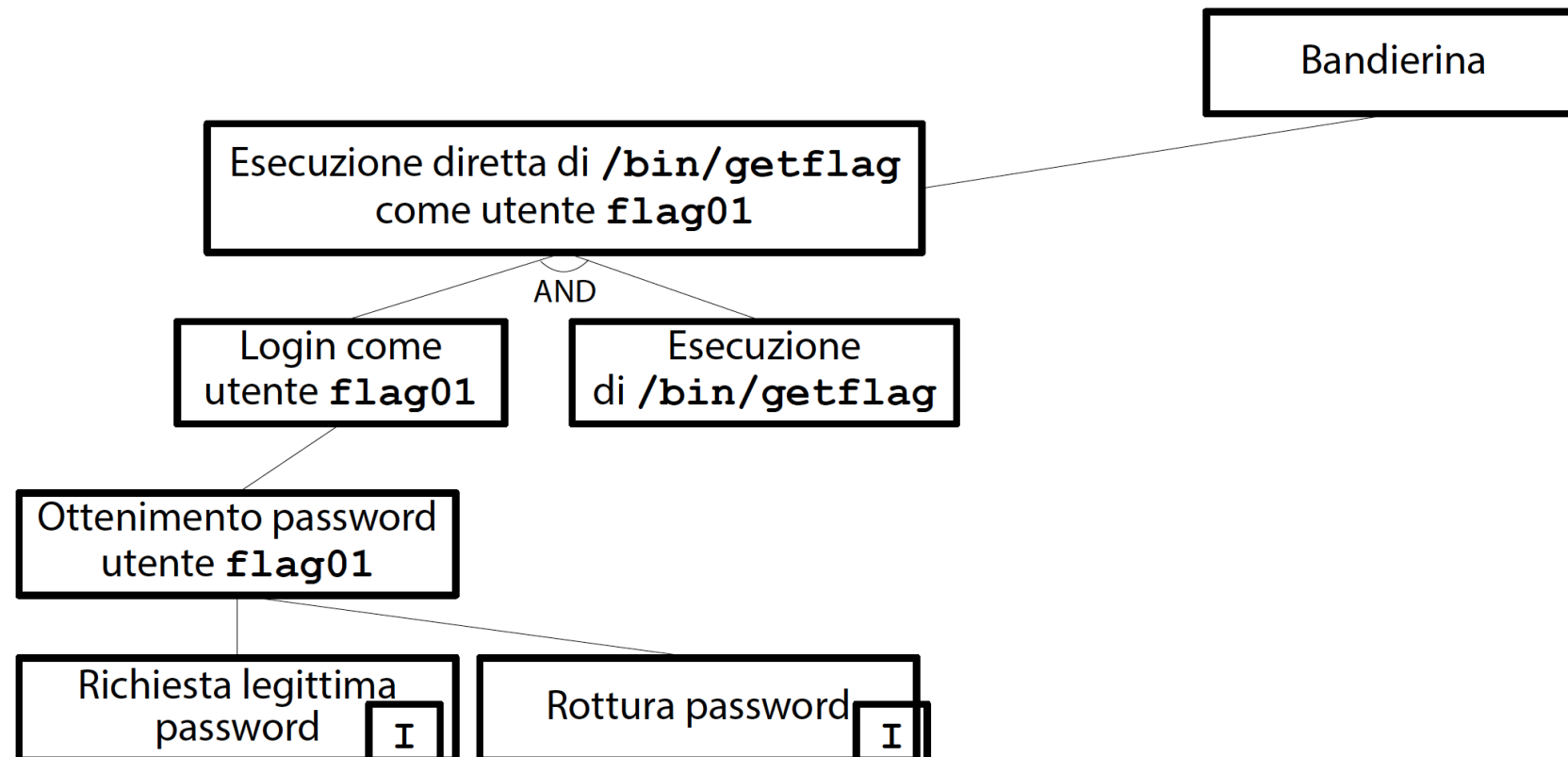


Rottura password

- E' possibile rompere la password dell'account flag01?
 - 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 level01

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

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

- L'utente level01 può accedere solamente alle directory

```
/home/level01
```

```
/home/flag01
```



Strategia alternativa

- La directory `/home/level01` non sembra contenere materiale interessante
- La directory `/home/flag01` contiene file di configurazione di BASH e un eseguibile:
`/home/flag01/flag01`
 - Digitando `ls -la /home/flag01/flag01` otteniamo

```
-rwsr-x--- 1 flag01 level01
```
 - Il file `flag01` è di proprietà dell'utente `flag01` ed è eseguibile dagli utenti del gruppo `level01`
 - Inoltre, è **SETUID**

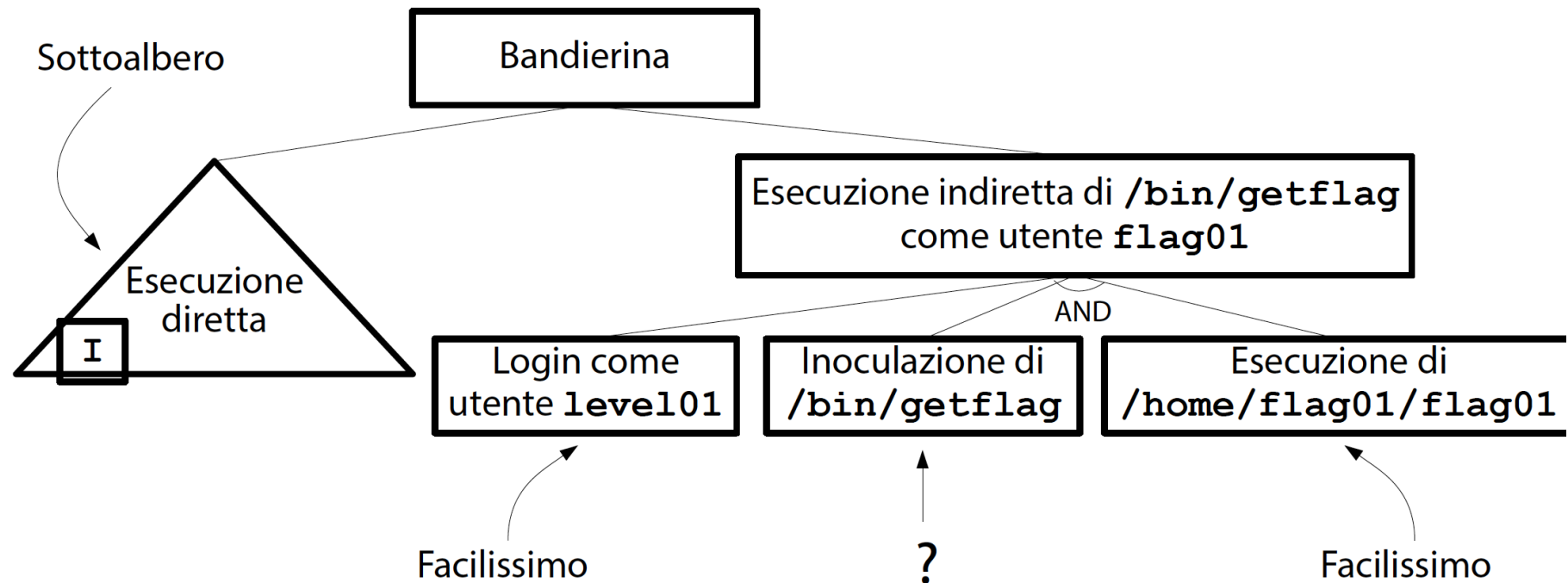


Strategia alternativa

- Nota: il file `/home/flag01/flag01` è eseguibile e permette di ottenere i privilegi dell'utente `flag01`
- Idea: provocare indirettamente (**inoculare**) l'esecuzione del binario `/bin/getflag` sfruttando il binario `/home/flag01/flag01`
- Conseguenza: `/bin/getflag` è eseguito come utente `flag01` (**si vince la sfida!**)



Aggiornamento dell'albero di attacco



Strategia alternativa

- Autenticarsi come level01 è facilissimo (abbiamo la password)
- Eseguire il comando `/home/flag01/flag01` è facilissimo (abbiamo i permessi)
- L'ostacolo da superare è quello di **trovare un modo di inoculare** `/bin/getflag` in `home/flag01/flag01`
 - Analizziamo il file sorgente dell'eseguibile `home/flag01/flag01`
 - Il file in questione è `level01.c`



Level 01

level01.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)
{
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

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

    system("/usr/bin/env echo and now what?");
}
```



Analisi del sorgente

- Le operazioni svolte da `level01.c` sono le seguenti
 - Imposta tutti gli user ID al valore effettivo (elevazione dell'utente al valore associato a `flag01`)
 - Imposta tutti i group ID al valore effettivo (elevazione del gruppo al valore associato a `level01`)
 - Esegue un comando, tramite la funzione di libreria `system()`
`system("/usr/bin/env echo and now what?");`



La funzione `system()`

- La funzione di libreria `system()` esegue un comando di shell, passato come argomento e restituisce -1 in caso di errore
`/bin/sh -c argomento`
- Vediamo i dettagli del comando con
`man 3 system`



La funzione `system()`

- Leggendo la sezione NOTES scopriamo una cosa interessante:

"Do not use `system()` from a program with `set-user-ID` or `set-group-ID` privileges, because strange values for some environment variables might be used to subvert system integrity."



La funzione `system()`

- Cosa ne deduciamo?
- Mai eseguire `system()` con il SETUID bit impostato
 - Giocando con le variabili di ambiente si può violare la sicurezza del programma
(anche se ancora non sappiamo come)
- Questo è esattamente il caso del binario `home/flag01/flag01`



La funzione `system()`

➤ Un'altra cosa interessante:

"`system()` will not, in fact, work properly from programs with `set-user-ID` or `set-group-ID` privileges on systems on which `/bin/sh` is `bash`...".



La funzione `system()`

- Cosa ne deduciamo?
- La funzione di libreria `system()` non funziona correttamente se `/bin/sh` corrisponde a `bash`
- E' il nostro caso?
 - Controlliamo se `/bin/sh` punta effettivamente a `bash`

```
ls -l /bin/sh
```

```
lrwxrwxlrxw 1 root root ... /bin/sh → /bin/bash
```
 - E' proprio così!



La funzione `system()`

- Cosa fa la funzione di libreria `system()`?
 - Usa proprio `sh` per eseguire un comando
 - Tale comando è eseguito mediante un processo figlio che **eredita tutti i privilegi del padre**
 - Entriamo nei dettagli del comando eseguito da `system()` nel file `level01.c`



La funzione `system()`

- Nel sorgente `leve101.c`, la funzione `system()` esegue il comando seguente
`/usr/bin/env echo and now what`
- Scopriamo qualche dettaglio in più sui comandi `env` e `echo`



Il comando env

- Innanzitutto, vediamo cosa è env:
`type -a env`
`env è usr/bin/env`
- Leggiamone la documentazione: `man env`
 - `env name=value name2=value2 command`
 - Si tratta di un comando di shell
 - Se invocato da solo, stampa la lista delle variabili di ambiente
 - Altrimenti, esegue il comando `command` nell'ambiente modificato ottenuto dopo aver settato le variabili ai valori specificati



Il comando echo

- Poi, vediamo cosa è echo:

`type -a echo`

`echo` è un comando interno di shell

`echo` è `usr/bin/echo`

- Leggiamone la documentazione: `man echo`

- Il comando `echo` stampa i suoi argomenti sullo standard output



Che comando esegue `system()` ?

- Il comando `/usr/bin/env echo` and now what
esegue il comando `/usr/bin/echo` che
stampa a video la stringa `and now what`
- Possiamo `inoculare` qualcosa di diverso da
`/usr/bin/echo`?
 - Ad esempio `/bin/getflag`?
 - Se si, abbiamo vinto la sfida

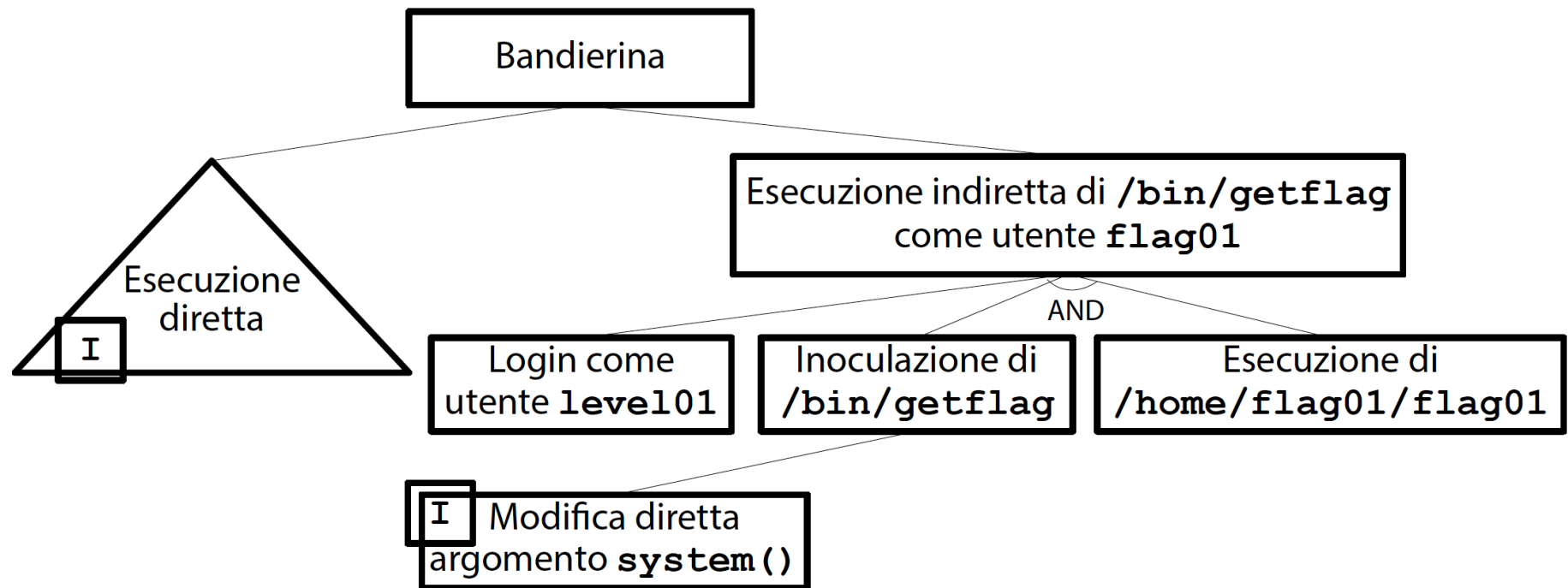


Modifica diretta del comando

- Purtroppo non possiamo modificare il comando eseguito da `system()`, poichè si tratta di una stringa costante
- L'aggiornamento dell'albero di attacco mostra che ancora non siamo sulla strada giusta



Aggiornamento dell'albero di attacco



Modifica dell'ambiente di shell

- Possiamo provare a modificare l'ambiente di shell ereditato da `/home/flag01/flag01`
- Vediamo quali **variabili di ambiente** influenzano l'esecuzione di un comando
- Scorriamo il manuale alla ricerca di pagine sulle variabili di ambiente
apropos environment



Lettura della documentazione

- Scorrendo i risultati, scopriamo la voce seguente nella Sezione 7:
`environ (7) –user environment`
- Leggiamo la pagina di manuale alla ricerca di qualche variabile di ambiente interessante
`man 7 environ`
- Scopriamo l'esistenza della `variabile PATH`



La variabile di ambiente PATH

- La variabile di ambiente PATH imposta la sequenza ordinata di cartelle scandite dai programmi di sistema alla ricerca di file specificati con un percorso incompleto



Idea



- Possiamo **modificare indirettamente** la stringa eseguita da `system()`
 - Copiamo `/bin/getflag` in una cartella temporanea e diamogli il nome `echo`
`cp /bin/getflag /tmp/echo`
 - Alteriamo il percorso di ricerca in modo da anticipare `/tmp` a `/usr/bin`
`PATH=/tmp:$PATH`

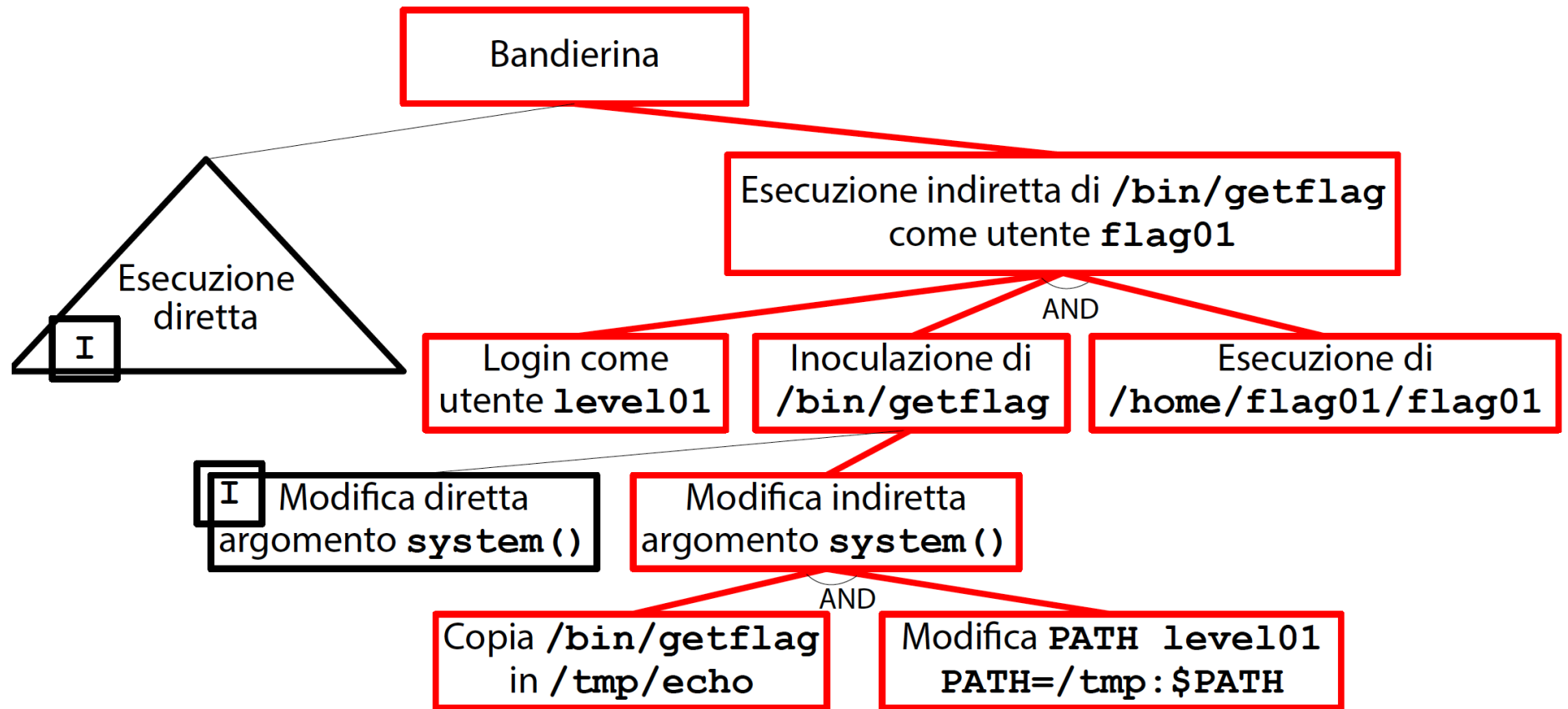


La conseguenza

- Cosa succede lanciando il programma `/home/flag01/flag01`?
 - Il comando `env` prova a caricare il file eseguibile `echo`
 - Poichè `echo` non ha un percorso, `sh` usa i percorsi di ricerca per individuare il file da eseguire
 - `sh` individua `/tmp/echo` come primo candidato all'esecuzione
 - `sh` esegue `/tmp/echo` con i privilegi dell'utente `flag01`



Aggiornamento dell'albero di attacco

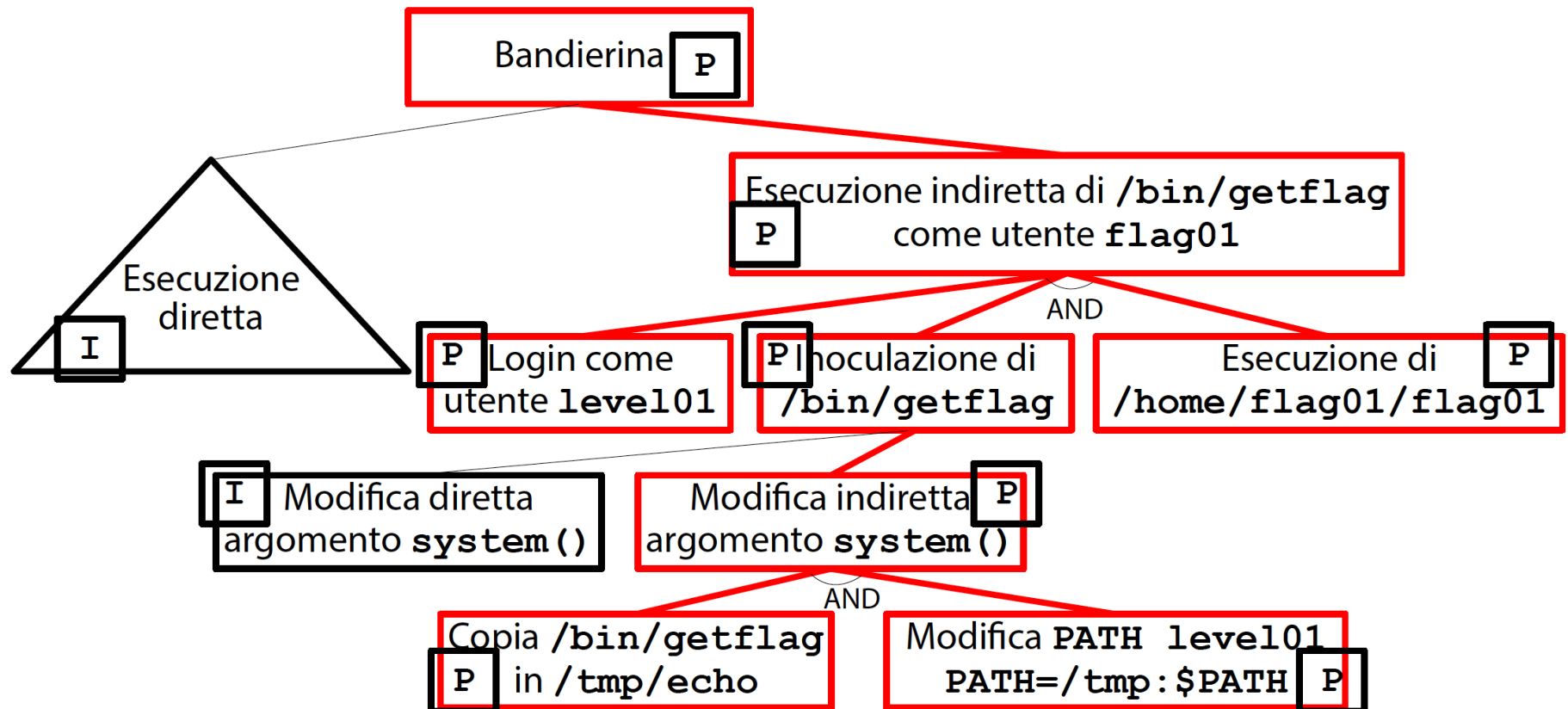


Aggiornamento dell'albero di attacco

- 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 level01?
 - Copia di /bin/getflag in /tmp/echo: **SI**
 - Modifica PATH=/tmp:\$PATH: **SI**
 - Login come utente level01: **SI**
 - Esecuzione di /home/flag01/flag01: **SI**



Aggiornamento dell'albero di attacco



Procedura di verifica dell'attacco

- 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 level01

Login come
utente **level01**



Passo 2

Copia di `/bin/getflag` in `/tmp/echo`

Login come
utente `level01`

Copia `/bin/getflag`
in `/tmp/echo`



Passo 3

Modifica PATH=/tmp:\$PATH

Login come
utente **level01**

Inoculazione di
/bin/getflag

Modifica indiretta
argomento **system()**

AND

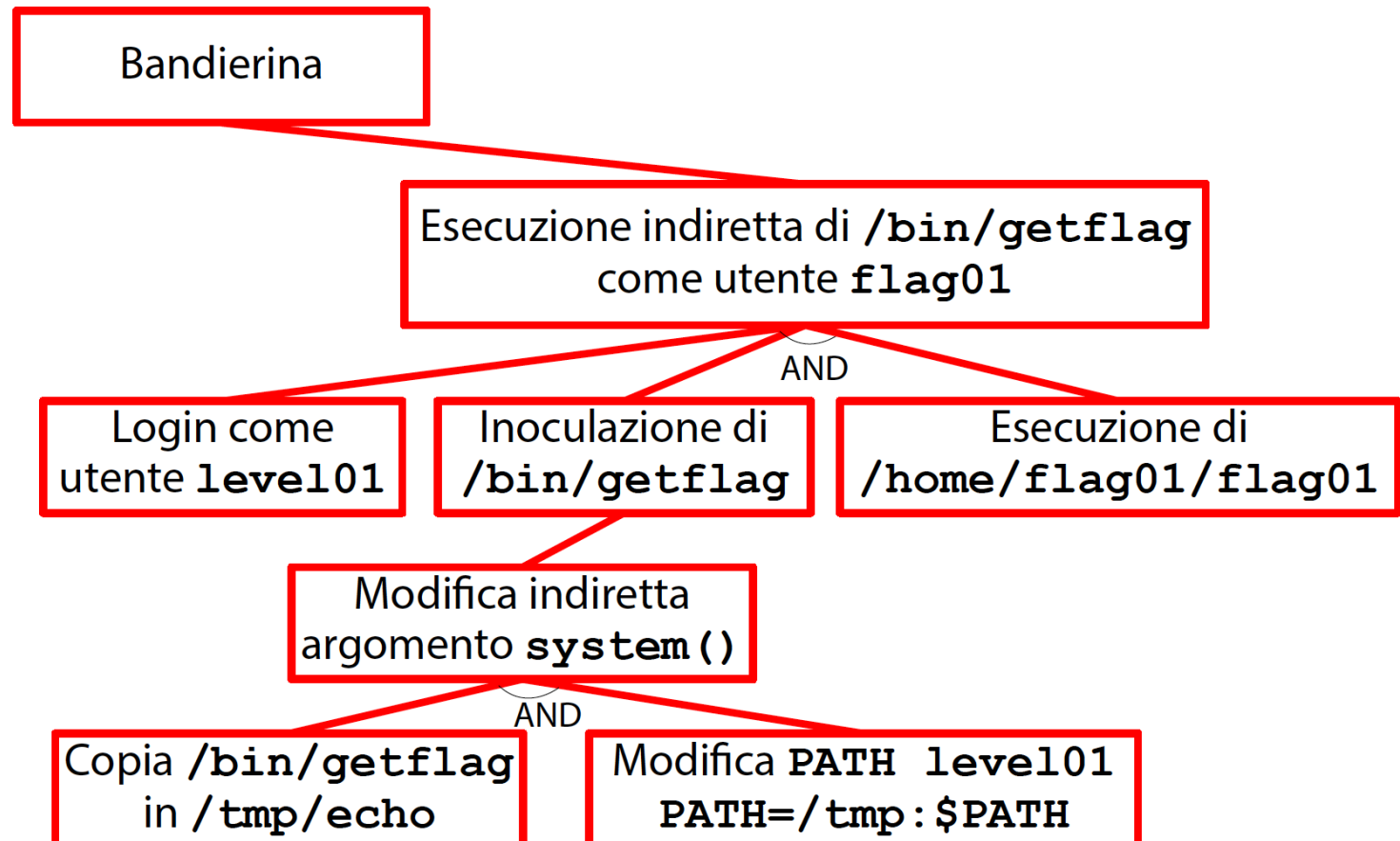
Copia **/bin/getflag**
in **/tmp/echo**

Modifica PATH **level01**
PATH=/tmp:\$PATH

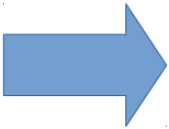


Passo 4

Esecuzione di `/home/flag01/flag01`



Sfruttamento della vulnerabilità



```
Ubuntu 11.10 ubuntu tty1
ubuntu login: level01
Password:
Last login: Thu Apr  6 17:17:50 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.

level01@ubuntu:~$ cp /bin/getflag /tmp/echo
level01@ubuntu:~$ PATH=/tmp:$PATH
level01@ubuntu:~$ /home/flag01/flag01
You have successfully executed getflag on a target account
level01@ubuntu:~$ _
```



Sfida vinta!



La vulnerabilità in Level01

- La **vulnerabilità** presente in level01.c si verifica solo se diverse **debolezze** sono presenti e sfruttate contemporaneamente
 - Quali sono queste debolezze?
 - Che CWE ID hanno?



Debolezza #1

- Il binario `/home/flag01/flag01` ha privilegi di esecuzione ingiustamente elevati
- CWE di riferimento: **CWE-276**
Incorrect Default Permissions
<https://cwe.mitre.org/data/definitions/276.html>



Debolezza #2

- La versione di bash utilizzata in Nebula non abbassa i propri privilegi di esecuzione
- CWE di riferimento: **CWE-272**
Least Privilege Violation
<https://cwe.mitre.org/data/definitions/272.html>



Debolezza #2

- Di default, molte shell moderne inibiscono l'elevazione dei privilegi tramite SETUID
 - In tal modo si evitano attacchi in grado di provocare **esecuzione di codice arbitrario**
 - Ad esempio, quando BASH esegue uno script con privilegi elevati, ne **abbassa i privilegi** a quelli dell'utente che ha invocato la shell
- Invece, la versione di bash utilizzata in Nebula non abbassa i propri privilegi di esecuzione



Debolezza #3

- Manipolando una variabile di ambiente (PATH), si sostituisce echo con un comando che esegue lo stesso codice di /bin/getflag
- CWE di riferimento: **CWE-426**
Untrusted Search Path
<https://cwe.mitre.org/data/definitions/426.html>



Mitigare le debolezze

- La vulnerabilità è un AND di tre debolezze
- Per annullarla è sufficiente inibire una delle tre debolezze
- Ovviamente, sarebbe preferibile inibirle tutte e tre!
 - Le prime due le può inibire l'amministratore di sistema
 - La terza, la può inibire il programmatore



Mitigare le debolezze

- Di seguito descriveremo come mitigare la **prima** e la **terza** debolezza:
 - Rimozione dei privilegi non minimi per `/home/flag01/flag01`
 - Impostazione sicura di PATH
- La mitigazione della **seconda** debolezza è più complessa
 - Prevede l'installazione di una diversa versione di BASH che eviti il problema del mancato abbassamento dei privilegi



Mitigazione #1


- Autentichiamoci come utente nebula e poi otteniamo una shell di root tramite `sudo -i`
- Spegliamo il bit **SETUID** sul file eseguibile `/home/flag01/flag01:`

```
chmod u-s /home/flag01/flag01
```



Mitigazione #1

/bin/getflag non riceve più i privilegi di flag01



```
Ubuntu 11.10 ubuntu tty1
ubuntu login: nebula
Password:
Last login: Thu Apr  6 19:47:16 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.

nebula@ubuntu:~$ sudo -i
root@ubuntu:~# chmod u-s /home/flag01/flag01
root@ubuntu:~# su - level01
level01@ubuntu:~$ PATH=/tmp:$PATH
level01@ubuntu:~$ /home/flag01/flag01
getflag is executing on a non-flag account, this doesn't count
level01@ubuntu:~$ _
```



Mitigazione #3

- Modifichiamo il sorgente `level01.c` in modo da impostare in maniera sicura la variabile di ambiente `PATH` prima di eseguire `system()`
 - **Idea:** rimuovere `/tmp` da `PATH`
- Possiamo usare la funzione di libreria `putenv()`
 - Modifica una variabile di ambiente già impostata
 - Ad esempio, per modificare `PATH`:

```
putenv( "PATH=/bin:/sbin:/usr/bin:usr/sbin" );
```



Una modifica mirata a level01.c

level01-env.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)
{
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);
    putenv("PATH=/bin:/sbin:/usr/bin:usr/sbin");
    system("/usr/bin/env echo and now what?");
}
```



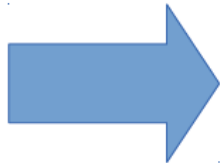
Mitigazione #3

- Compiliamo level01-env.c:
`gcc -o flag01-env level01-env.c`
- Impostiamo i privilegi su flag01-env:
`chown flag01:level01 /home/flag01/flag01-env`
`chmod u+s /home/flag01/flag01-env`
- Impostiamo PATH ed eseguiamo flag01-env:
`PATH=/tmp:$PATH`
`/home/flag01/flag01-env`



Risultato

/bin/getflag non è più eseguito
al posto dell'echo originale



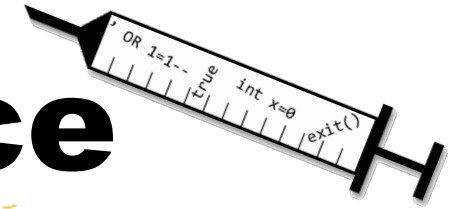
```
Ubuntu 11.10 ubuntu tty2
ubuntu login: level01
Password:
Last login: Fri Apr  7 01:25:07 PDT 2017 on tty2
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

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

level01@ubuntu:~$ PATH=/tmp:$PATH
level01@ubuntu:~$ /home/flag01/flag01-env
and now what?
level01@ubuntu:~$ _
```



Iniezione di codice



- Nell'esercizio visto, abbiamo utilizzato la **manipolazione di una variabile di ambiente** (PATH) per provocare l'esecuzione di codice arbitrario (/bin/getflag)
- Questa tecnica si chiama **iniezione di codice** e consiste appunto nell'**iniettare** il codice da eseguire dall'applicazione vulnerabile al posto del codice presente
 - Vedremo anche altri metodi per iniettare codice in un eseguibile

