Report by Azimkhan Dilnaz

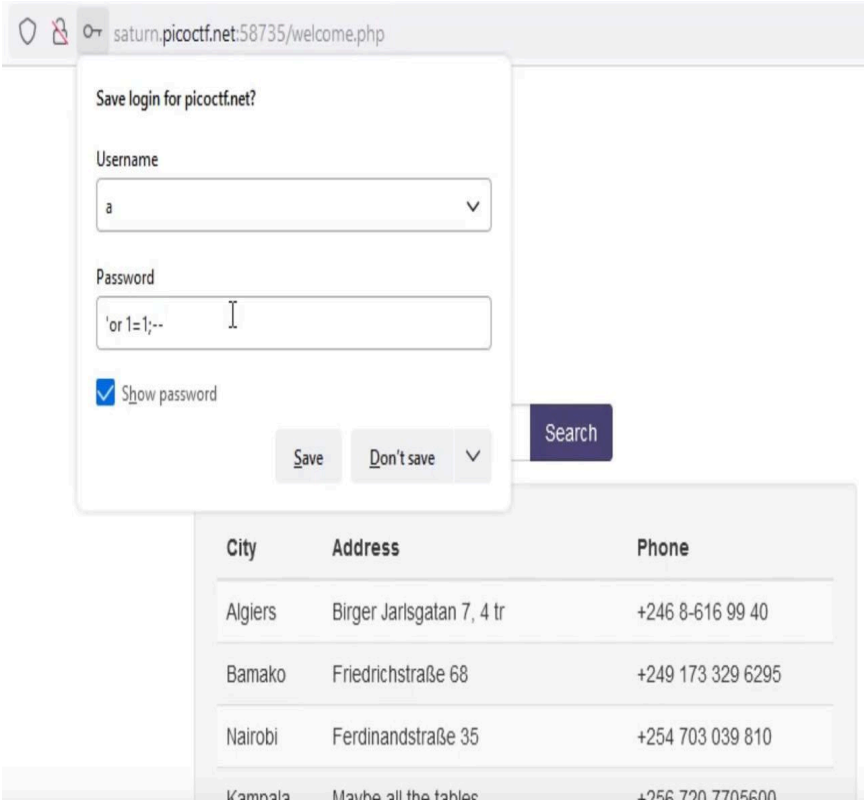
SQL Injection Exploitation: A Comprehensive Analysis

Introduction: SQL Injection (SQLi) remains one of the most potent threats to web applications, allowing attackers to manipulate databases via insecure user input fields. This report explores the diverse techniques used in SQL injection attacks, specifically focusing on advanced methods demonstrated in TryHackMe's rooms. It highlights various SQLi vulnerabilities such as Union-based, Blind, Time-based, and Boolean-based injections and delves into how these attacks can be automated for increased efficiency. Finally, the report discusses the importance of securing web applications through robust defense mechanisms to thwart these malicious attacks.

SQL Injection Techniques:

1. **Basic SQL Injection (Login Bypass):**
    - **Technique**: Attackers can bypass login forms by injecting a simple `OR 1=1;--` into the password field. This results in a query that always evaluates to true, granting unauthorized access.
    - **Example**: `SELECT * FROM users WHERE name = 'admin' AND password = '' OR 1=1 --`
    - **Impact**: This technique bypasses authentication, potentially exposing sensitive user data.
2. **Union-based SQL Injection:**
    - **Technique**: The attacker uses the `UNION SELECT` clause to retrieve additional information from the database by merging results from multiple queries.
    - **Example**: `Algiers' UNION SELECT sql, 1, 1 FROM sqlite_master;--`

- **Impact**: This allows the attacker to gather metadata, such as table structures and column names, providing insights into the underlying database.



3. **Blind SQL Injection (Boolean-based):**

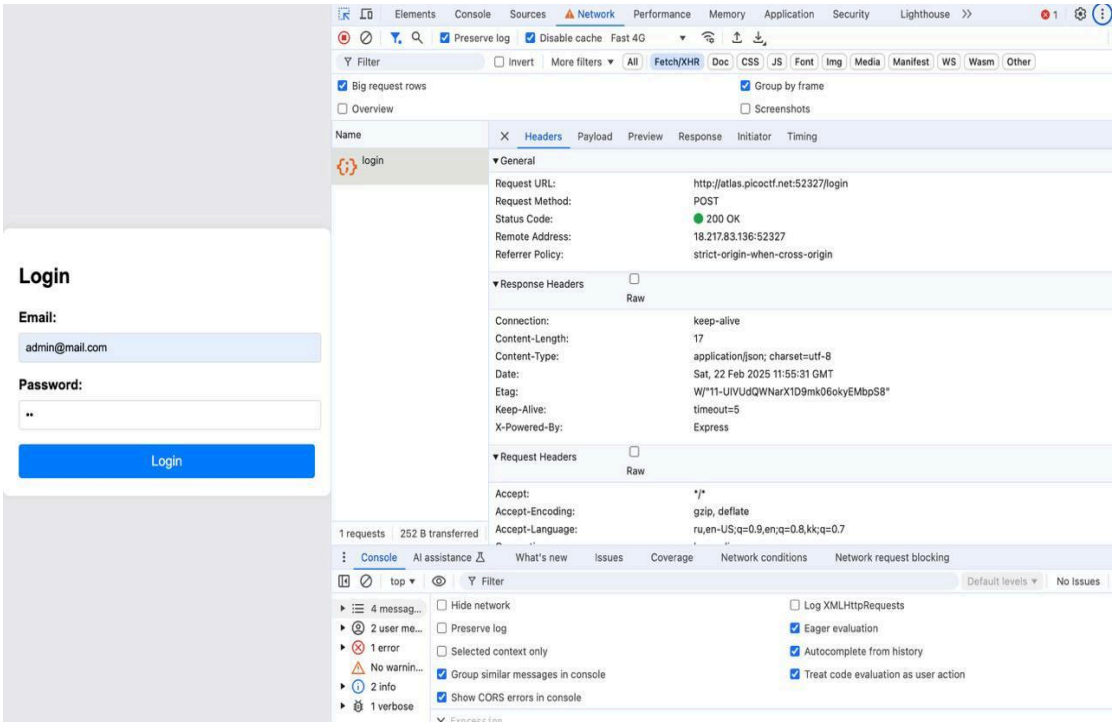
   - **Technique**: By manipulating SQL queries that return no visible output, attackers can infer database information based on the application's response behavior.
   - **Example**: `user' OR 1=1 --`
   - **Impact**: This technique does not directly reveal data but can be used to confirm the existence of records or infer database structures.
4. **Time-based Blind SQL Injection:**

   - **Technique**: This method relies on causing deliberate delays in the server's response to infer the success of certain queries.
   - **Example**: `user' OR IF(1=1, SLEEP(5), 0); --`

    - **Impact**: By measuring response times, the attacker can confirm the validity of a condition, allowing them to slowly gather information about the database.
5. **Inception-based SQL Injection:**

    - **Technique**: In this advanced exploitation method, an attacker uses nested `UNION SELECT` queries to extract sensitive data, such as flags or authentication credentials.
    - **Example**: `http://10.10.185.61/user?id=13 union all select 1,flag,4,5 from flag--`
    - **Impact**: Inception-based SQLi allows attackers to exfiltrate crucial data like flags, typically stored in database tables.
6. **Exfiltration via DNS (Out-of-Band SQL Injection):**

    - **Technique**: When direct data retrieval is not possible, attackers use DNS queries or HTTP requests to exfiltrate stolen data.
    - **Example**: Using DNS to send stolen data to an external server.
    - **Impact**: This method allows attackers to bypass traditional data extraction restrictions and transfer sensitive information to external systems.

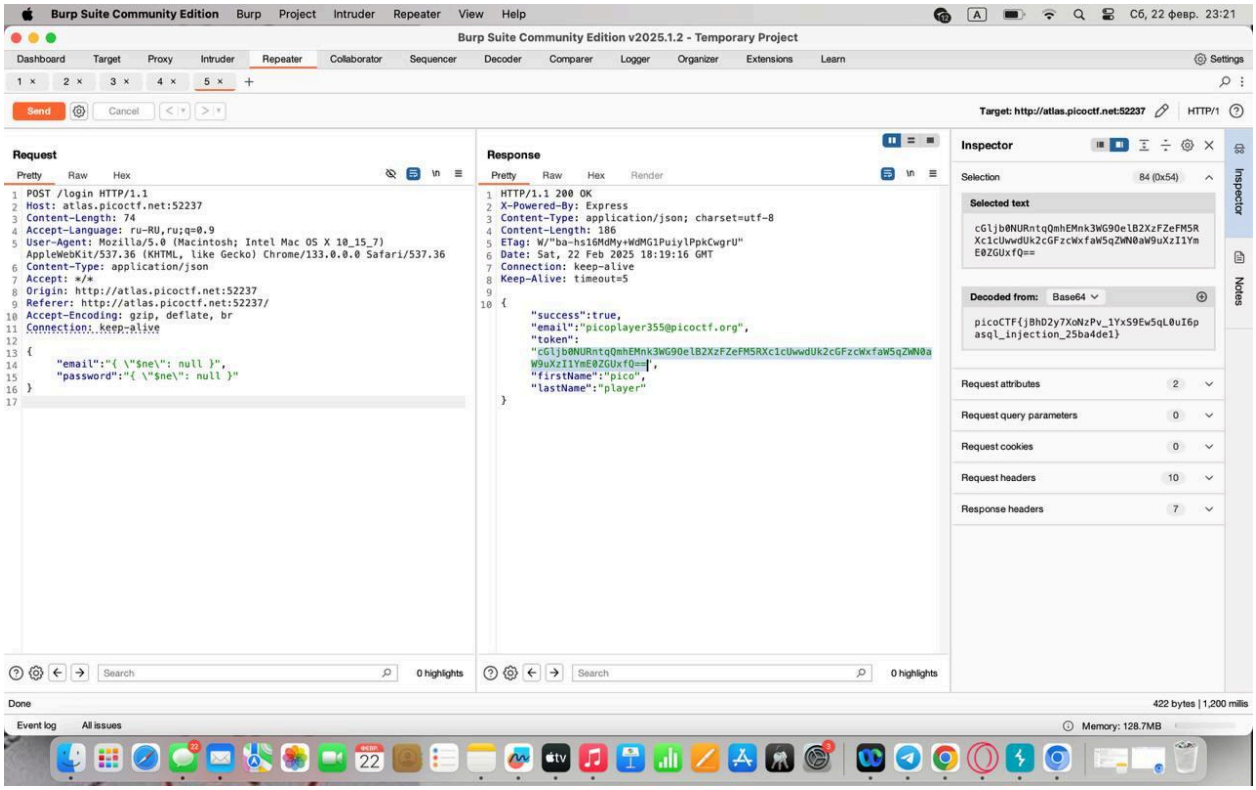


No Sql Injection

PICOCTF NoSQL Injection

To get the flag using **Burp Suite:**

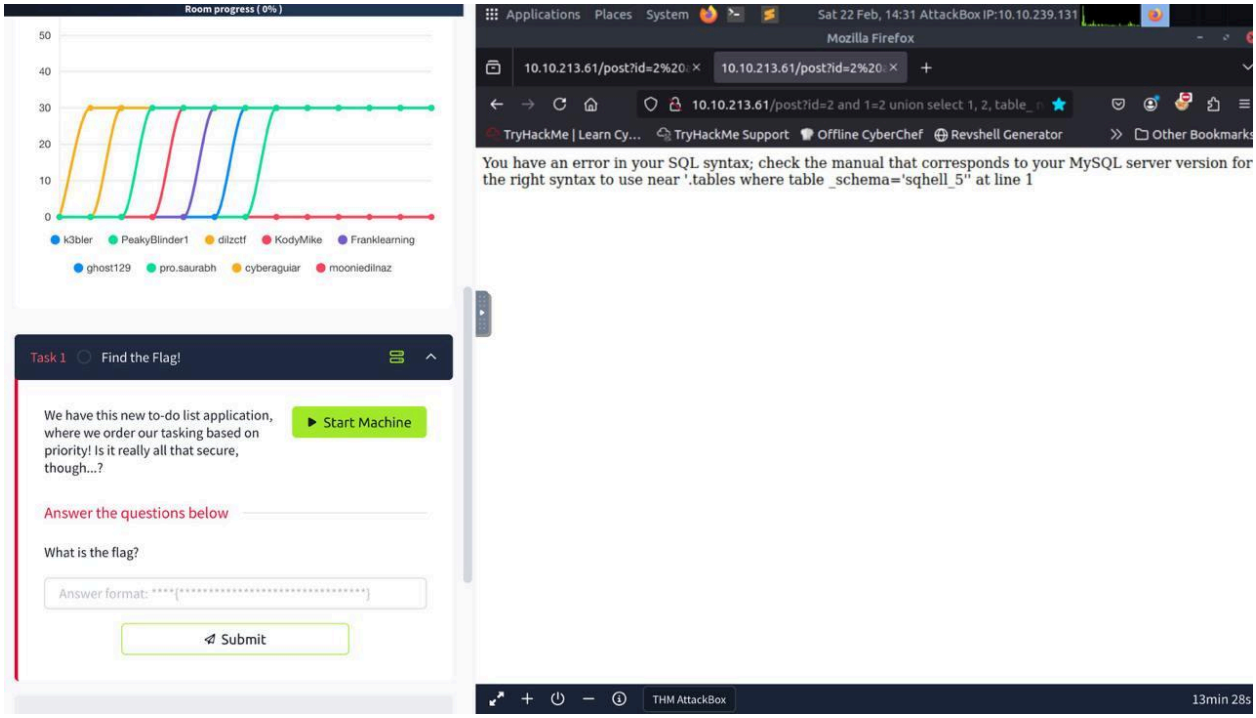
1. Launch Burp Suite and configure the proxy in your browser.
2. Find the web form and monitor the traffic through Burp Suite.
3. Modify the requests through Proxy, using a NoSQL injection payload (e.g., `{"$ne": null}`).
4. If the injection is successful, retrieve the flag from the server's response.
5. Enter the flag in the designated place in the challenge.



TryHackMe Room Walkthrough:

1. **SQHell:**

    - This room covers multiple SQL injection techniques across various pages, such as login and registration. Attacks such as Boolean-based SQLi and UNION SQLi were employed to extract flags like `THM{FLAG1:E786483E5A53075750F1FA792E823BD2}`.
    - The use of different SQLi techniques demonstrates the versatility of SQL injection in bypassing authentication and extracting sensitive data.



2. **Prioritize Room:**
    - This room focuses on exploiting a Boolean-based blind SQL injection vulnerability in a to-do list application. By manipulating the `order` GET parameter, attackers could inject SQL queries that allowed them to enumerate database columns and extract flags using automation techniques.
    - Automation, through Python scripts with threading, significantly sped up the process of extracting table names, column names, and the flag, showcasing the power of automated exploitation in real-world scenarios.

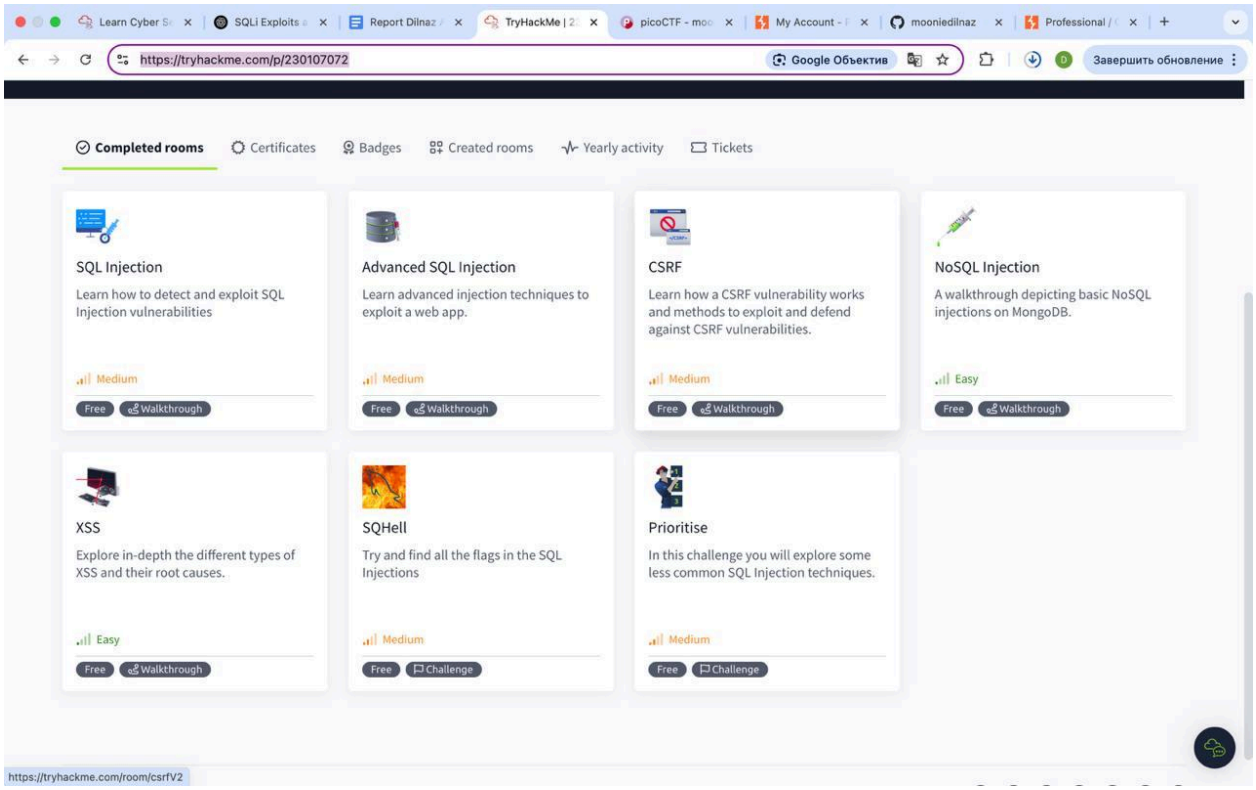
SQL Injection Automation:

- **Automation**: Leveraging Python scripts with threading for concurrent query execution, attackers can efficiently enumerate database details, extract flags, and bypass security measures. This automation accelerates the exploitation process, making it easier for attackers to discover and extract sensitive data.

- **Impact**: Automation increases the efficiency of SQL injection attacks, allowing attackers to exploit vulnerabilities on a larger scale and retrieve data with minimal effort.

Prevention and Mitigation of SQL Injection:

1. **Prepared Statements**: A fundamental defense mechanism that ensures SQL queries are parameterized, preventing malicious data from being executed as part of the query.
2. **Input Validation**: Ensuring that user input is validated and sanitized before being passed to the database is essential in preventing SQL injection.
3. **Escaping User Input**: Properly escaping user inputs helps prevent dangerous characters (like `'`, `--`) from being interpreted as part of SQL queries.
4. **Output Encoding**: Ensuring that any data output to the user is encoded to prevent XSS and other injection attacks.
5. **Web Application Firewalls (WAFs)**: WAFs act as an additional layer of defense, filtering malicious SQL queries before they can reach the database.



Conclusion: SQL Injection remains one of the most critical vulnerabilities in web applications. The TryHackMe rooms provided a comprehensive look at various SQLi techniques, from basic to advanced, and demonstrated how attackers can bypass authentication, exfiltrate data, and automate these processes for efficiency. While SQLi poses significant risks, proper defenses, such as prepared statements, input validation, and automated security measures, can help mitigate these threats. By adhering to secure coding practices and employing robust security measures, web developers can defend against SQL injection attacks and protect sensitive data.

Key Takeaways:

- SQL Injection attacks can take many forms, including Union-based, Blind, and Time-based injections.
- Automation plays a crucial role in modern SQLi exploitation, enhancing both the speed and scale of attacks.
- SQL injection vulnerabilities can be mitigated through careful coding practices, such as parameterized queries and input validation.

References:

- TryHackMe Rooms: SQHell, Prioritize Room, and SQL Injection Walkthrough.
- OWASP SQL Injection Prevention Cheat Sheet.
- PicoCTF

Note: The VR machine's limit expired, so I had to create a second account to complete the SQHell room. :))

Links:
PicoCTF : **https://play.picoctf.org/users/mooniedilnaz**
Try Hack Me: **https://tryhackme.com/p/230107072**
GitHub : **https://github.com/mooniedilnaz**