**BUB BOUNTY**

A blue and orange logo

Description automatically generated

**IT NUMBER: IT22345332**

**NAME: G.P DINUJAYA THAMARA**

**WEEKEND BATCH**

**MALABE CAMPUS**

**Bug Bounty Platform – Hacker One**

**Bug Bounty Program - Booking.com**

**Scope**

**In Scope Assets**

For in Scope Assets please refer to the Scope tab

**Out-Of-Scope Applications** Any application whether owned by Booking.com or third-party vendor **not included as an in-scope asset** will be mentioned on the scope tab as out of scope.

For Out Of Scope Assets please refer to the Scope tab

**In-scope Vulnerabilities**

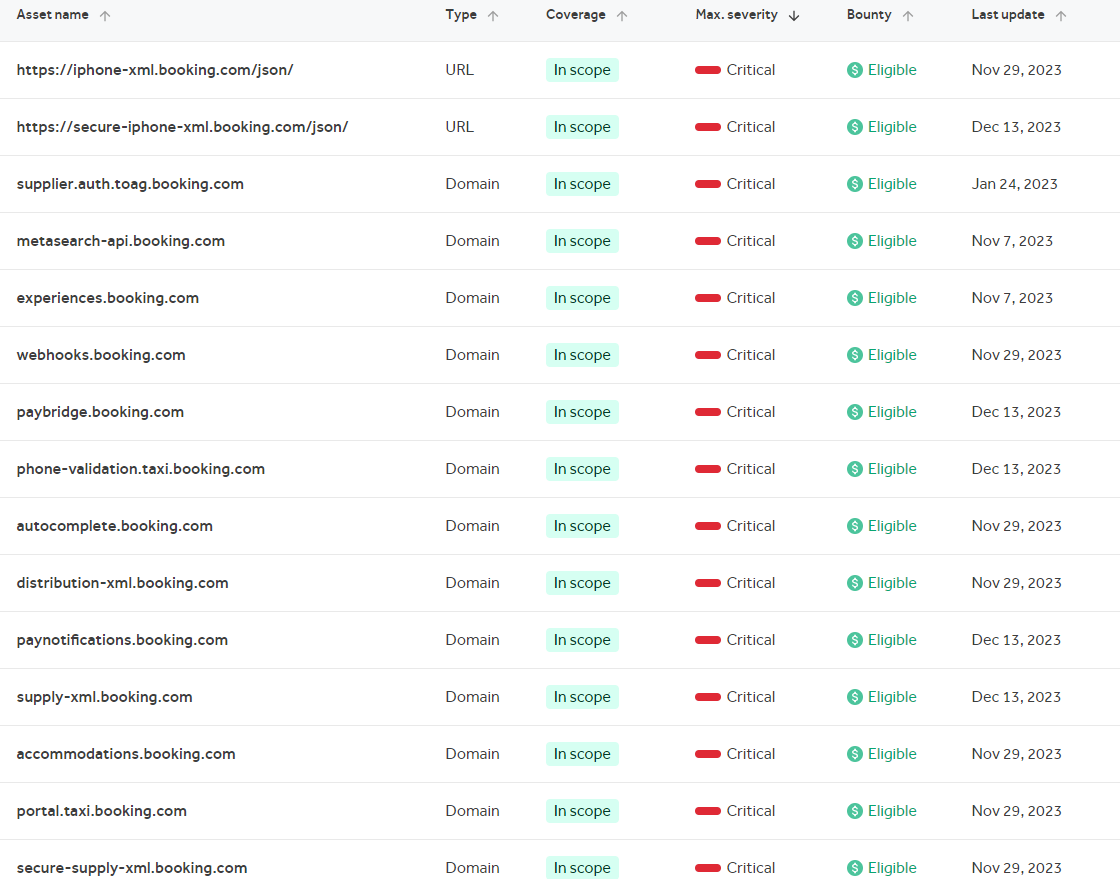
**Accepted, in-scope vulnerabilities include, but are not limited to:**

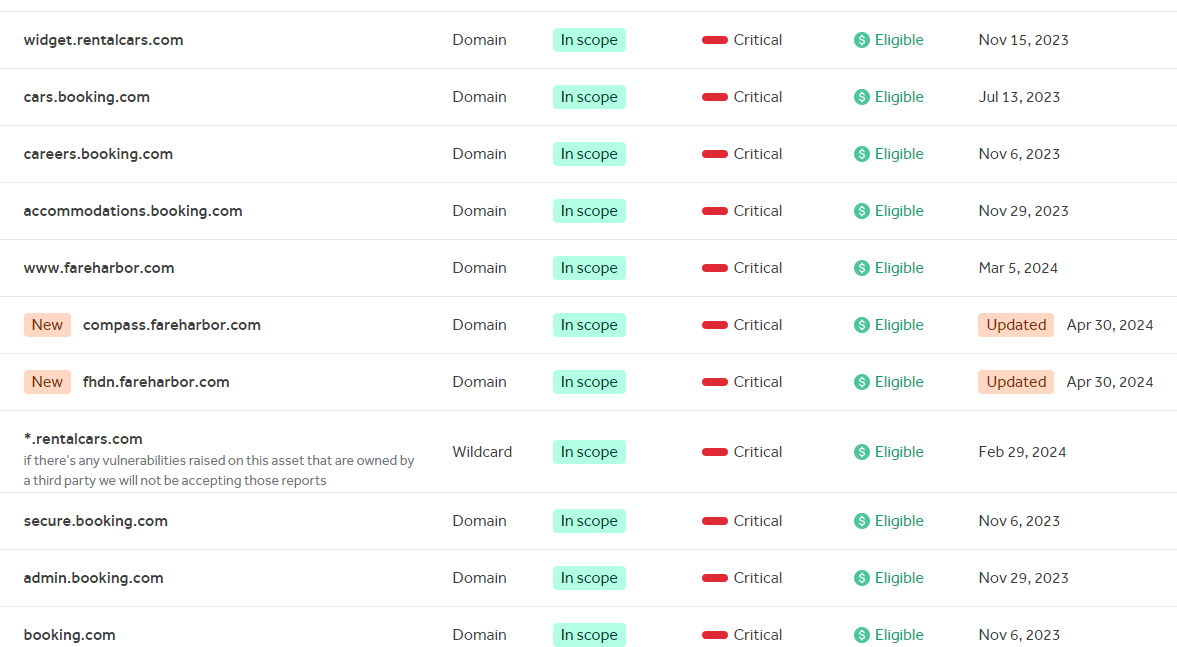
* Disclosure of sensitive or personally identifiable information
* Cross-Site Scripting (XSS) - Please note, for XSS if the same issue is reported for the different subdomains but with the same root cause, it will be considered duplicate
* Cross-Site Request Forgery (CSRF) for sensitive functions in a privileged context
* Remote code execution (RCE)
* Authentication or authorization flaws, including insecure direct object references and authentication bypass
* Injection vulnerabilities, including SQL and XML injection
* Directory traversal
* Significant security misconfiguration with a verifiable vulnerability
* Account takeover by exploiting a vulnerability
* SSRF
* XXE
* Subdomain takeover in \*.booking.com domains

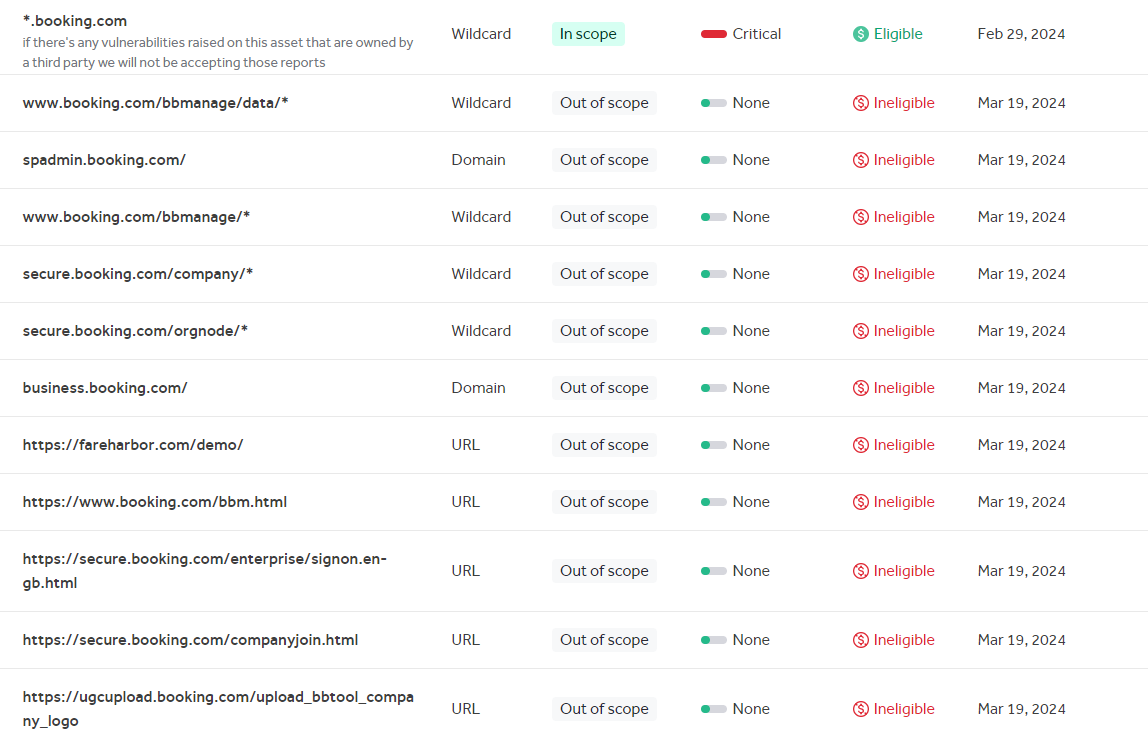
**Out-Of-Scope Vulnerabilities** Depending on their impact, not all reported issues may qualify for a monetary reward. However, all reports are reviewed on a case-by-case basis and any report that results in a change being made will at a minimum receive recognition. Please note that our **program terms and rules of engagement** still apply.

**The following issues are outside the scope of our vulnerability rewards program:**

* Any vulnerability which requires access to a compromised email account or Booking.com account for successful exploitation
* Vulnerabilities on Third Party Products
* Attacks requiring physical access to a user's device or network.
* Forms missing CSRF tokens (we require evidence of actual CSRF vulnerability)
* Login/Logout CSRF
* Missing security headers which do not lead directly to a vulnerability
* Use of a known-vulnerable library (without evidence of exploitability)
* Reports from automated tools or scans
* Social engineering of Booking staff or contractors
* Denial of Service attacks and/or reports on rate limiting issues
* Not enforcing certificate pinning
* Any issues that require a rooted or jailbroken device or a compromised device
* Clickjacking
* Improper session invalidation
* User enumeration
* Host header injections without a specific, demonstrable impact
* Self-XSS, which includes any payload entered by the victim
* Any vulnerabilities requiring significant and unlikely interaction by the victim, such as disabling browser controls
* Content spoofing without embedded HTML or JavaScript
* Hypothetical issues that do not have any practical impact
* Infrastructure vulnerabilities, including:
* Issues related to SSL certificates
* DNS configuration issues
* Server configuration issues (e.g. open ports, TLS versions, etc.)

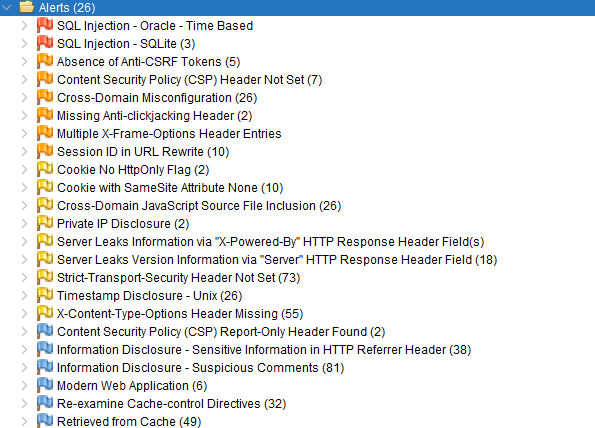


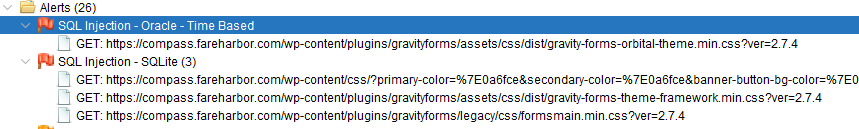




compass.fareharbor.com

From scan that has been done through the OWSAP ZAP these are the vulnerabilities it found







**Time-based Blind SQLi**

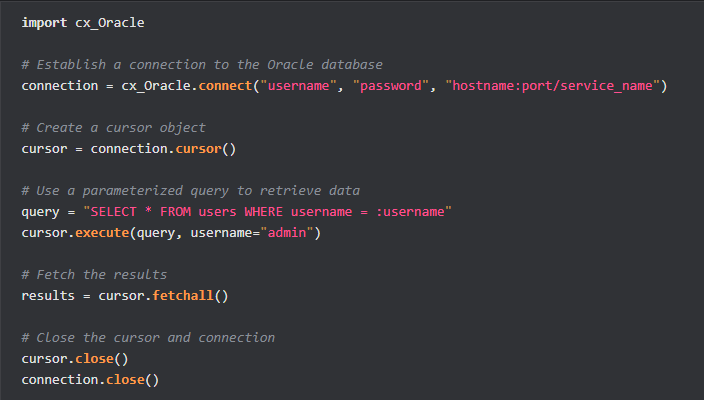
Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.

Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database character by character.

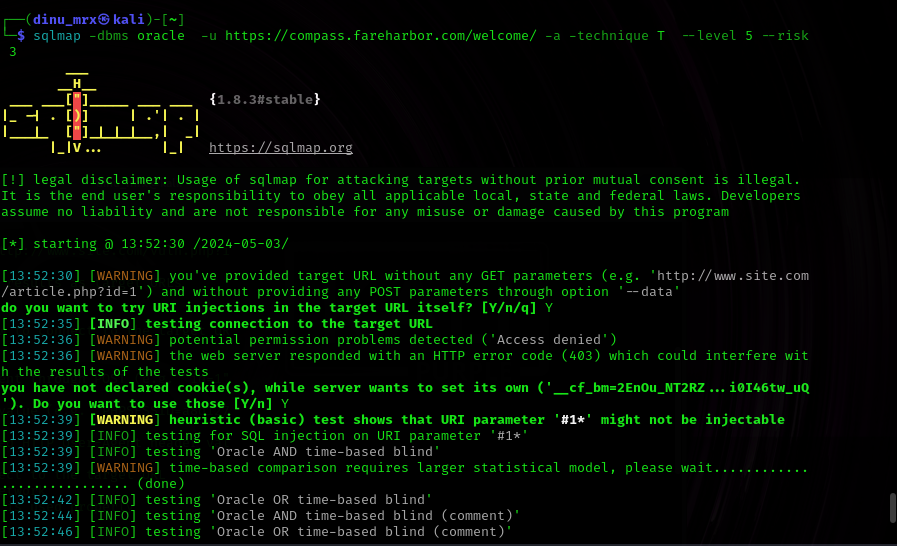
<https://www.acunetix.com/websitesecurity/sql-injection2/#:~:text=Time%2Dbased%20SQL%20Injection%20is,query%20is%20TRUE%20or%20FALSE>.

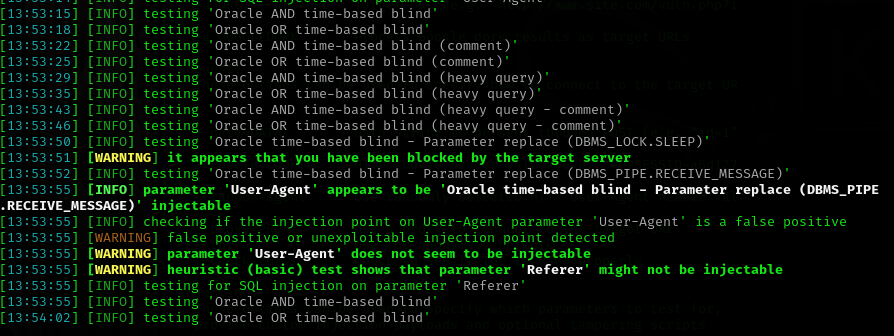
**Remediation**

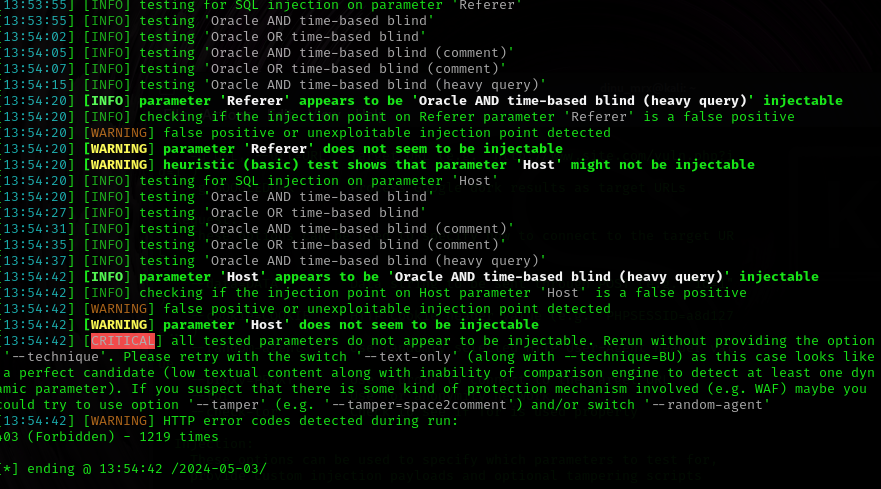
1. **Use built-in Object Data Models:** Instead of directly passing user input to the back-end SQL server, use built-in Object Data Models to gather and handle data. This helps to ensure that the input is properly sanitized and validated before being used in SQL queries.
2. **Parameterized queries:** Use parameterized queries in the language framework to construct SQL statements. This helps to prevent SQL injection by separating the SQL code from the user input. Here’s an example in Python using the cx\_Oracle library:
3. Avoid string concatenation: Avoid directly concatenating user input with SQL statements in the code base. This can make the application vulnerable to SQL injection attacks. Instead, use parameterized queries or stored procedures to handle dynamic SQL statements.



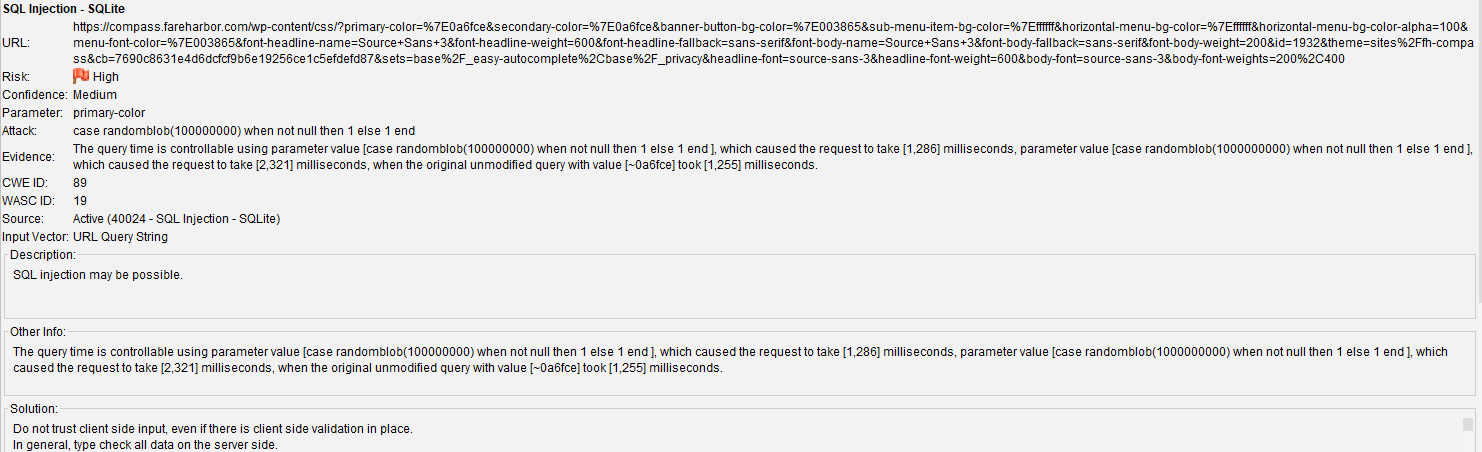
Attacking through SQLmap







According to the results there may be a oracle time based vulnerability in this domain.





SQLite Injection Attacks

In SQLite injection means injecting some malicious code to gain access to other databases while accepting the input from web application. Suppose we have a registration page where the user needs to enter username but instead of that if he enters SQLite statement then it will run on our database and return the data based on his query statement. The basic idea for SQLite injection attacks is to get secure information from your database and to perform some vulnerable actions like updating existing records information or delete/drop tables in the database, etc.Generally, these SQLite injection attacks can happen whenever your application relies on user input to construct the SQLite query statements. So while taking the input from users we need to validate that data before we send it to the database by defining pattern validations or accepting the input parameters in standard way.

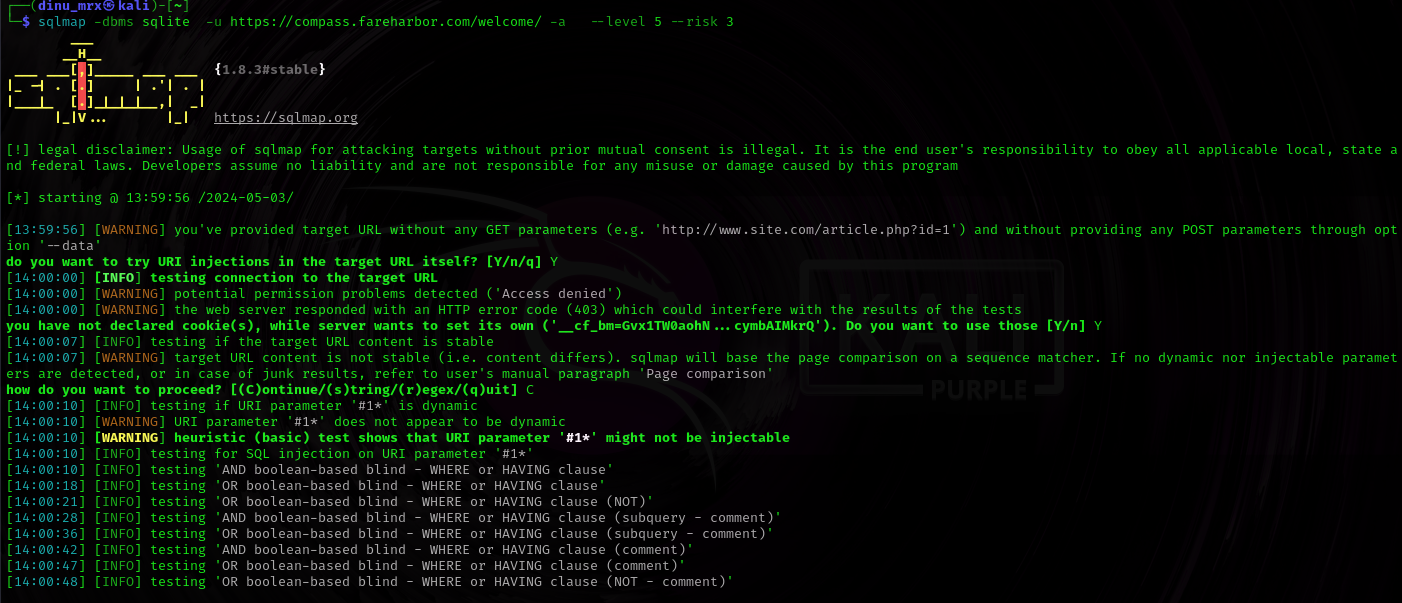
SQLite Injection Attacks Example

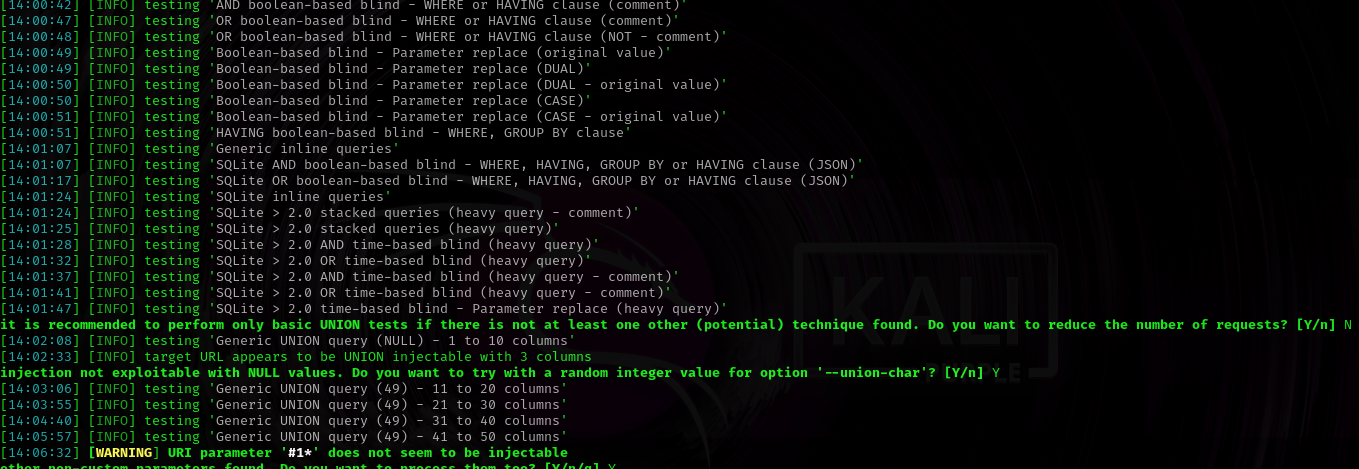
Now we will see how SQLite injection attacks can happen and how we can prevent it with examples for that create table **emp\_master** in your database using the following queries.



For more information <https://www.tutlane.com/tutorial/sqlite/sqlite-injection-attacks>

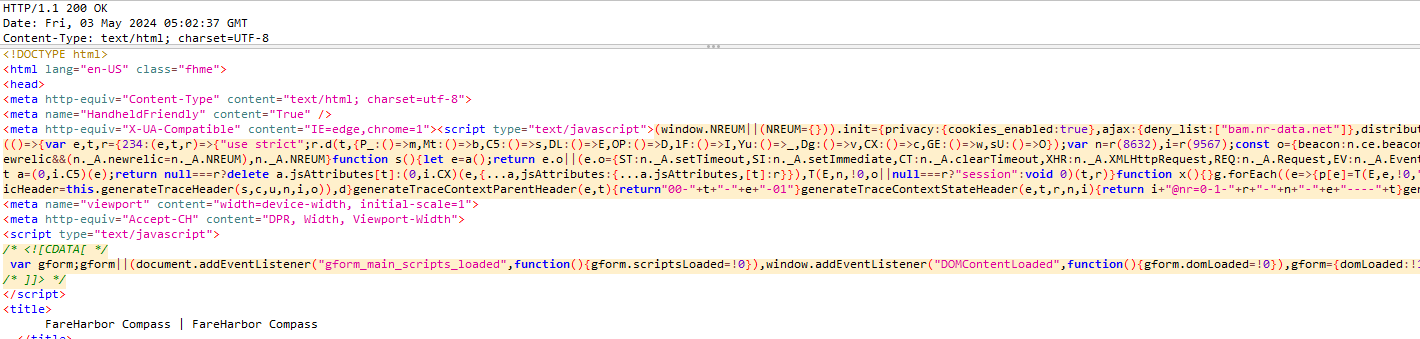
Performing the attack through SQLmap.

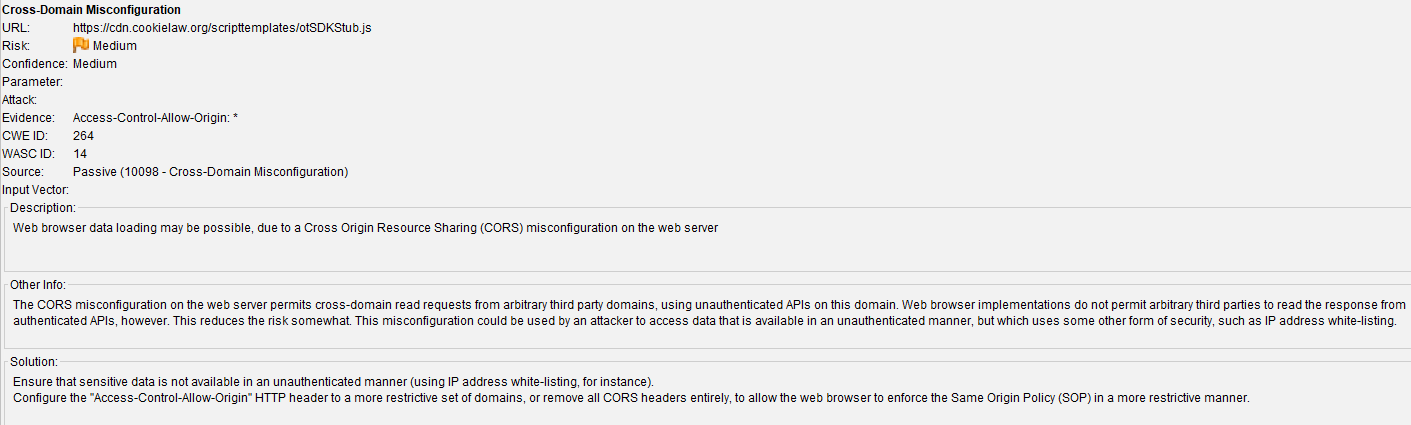


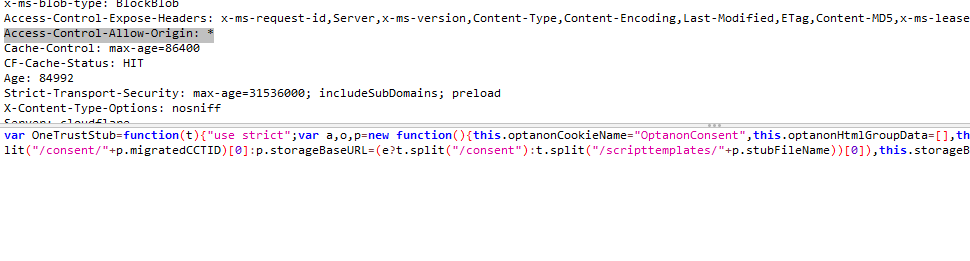


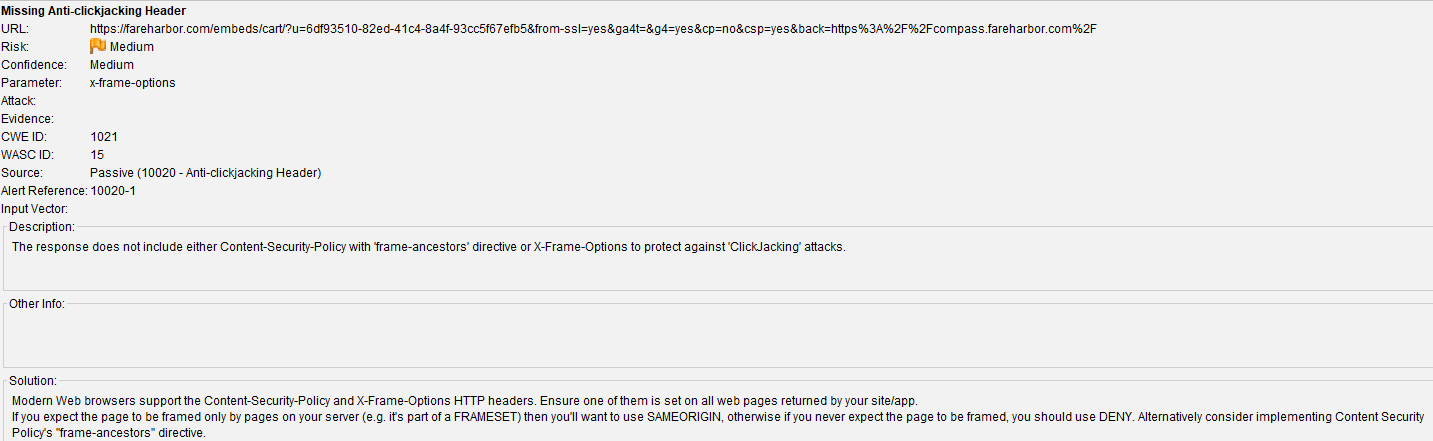


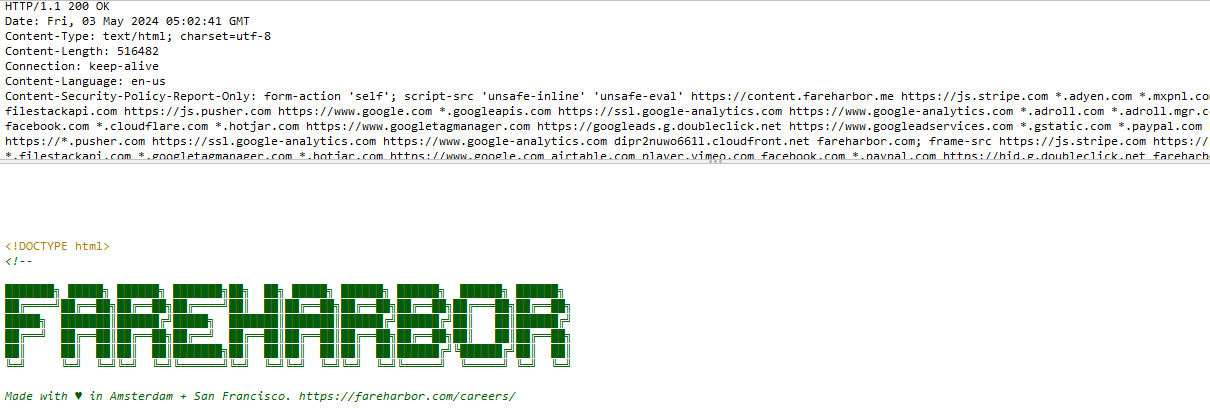


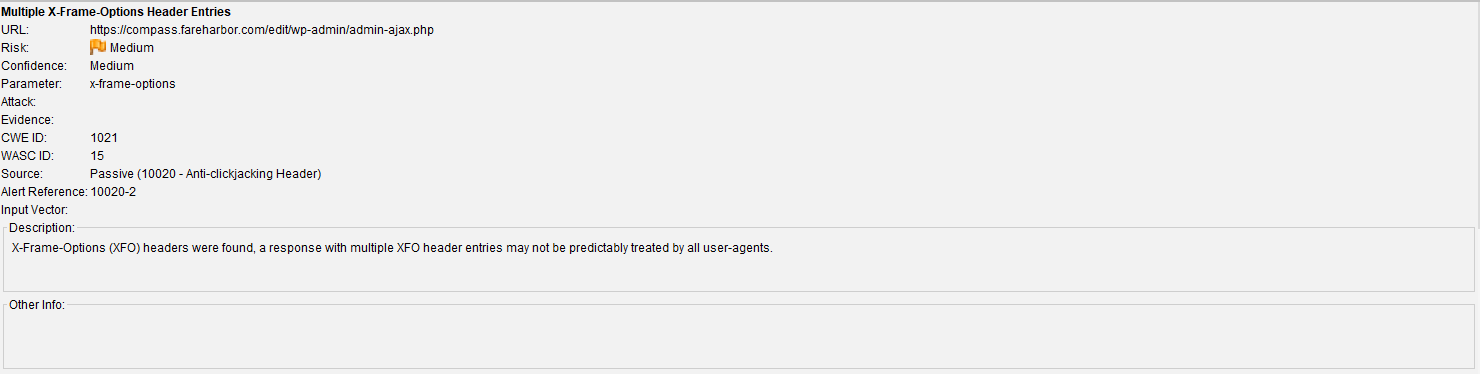




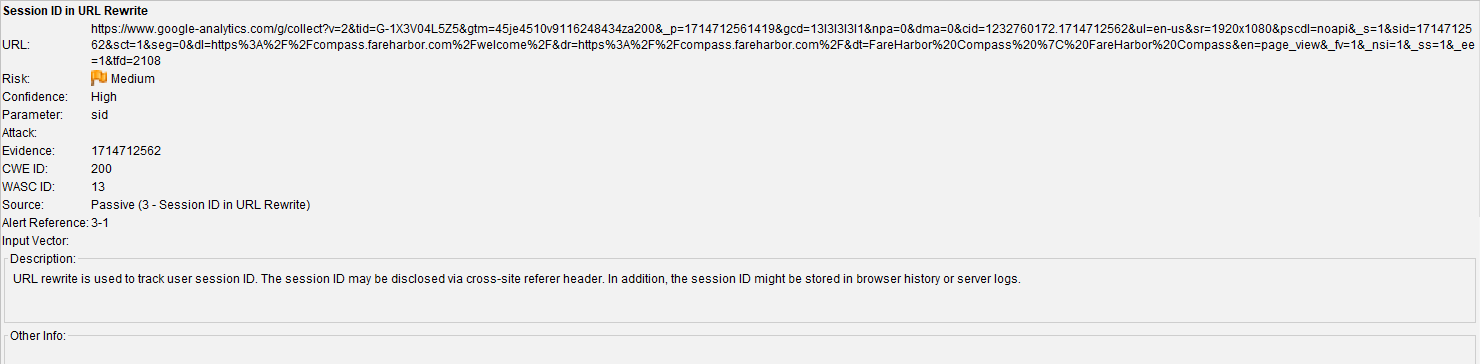


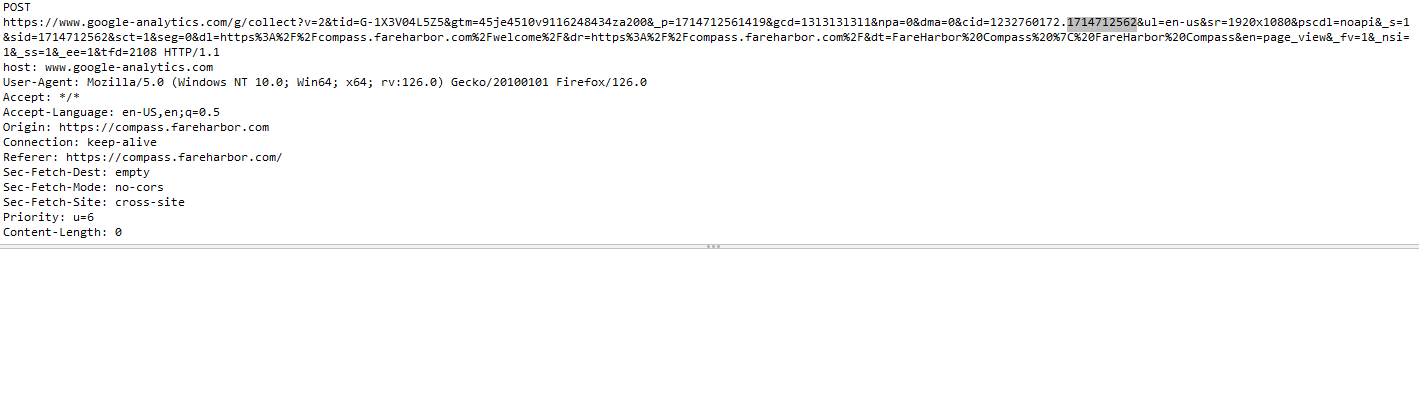




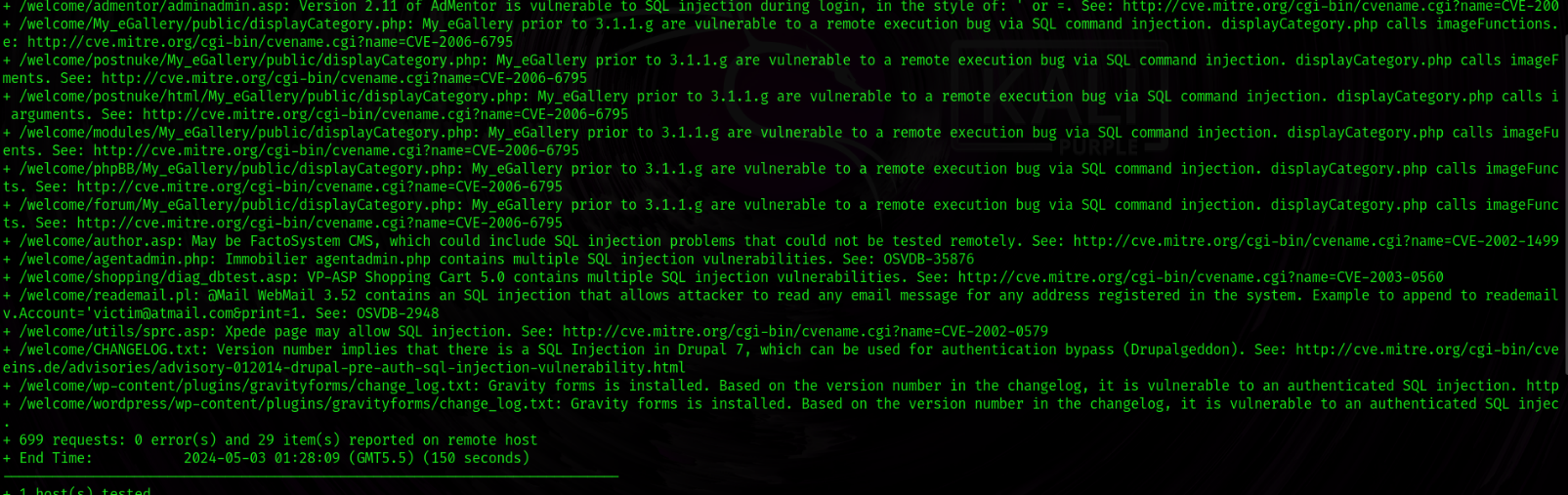








According to the nikto scan there are some SQL injection vulnerabilities.

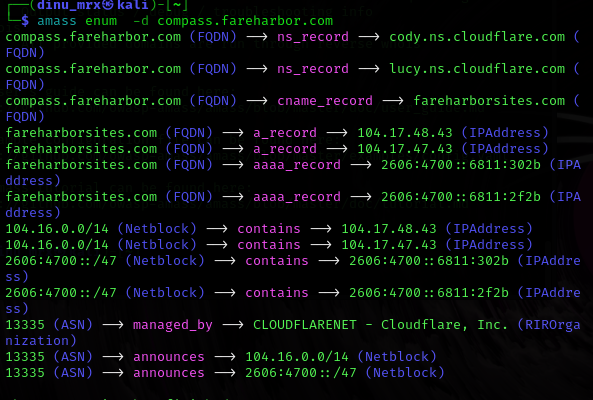


<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6795>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1499>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3704>

Amass enumeration scan.



compass.fareharbor.com (FQDN) --> ns\_record --> cody.ns.cloudflare.com (FQDN)

This indicates that the domain compass.fareharbor.com has a nameserver (NS) record pointing to cody.ns.cloudflare.com. Nameservers are servers responsible for handling DNS queries for a domain.

compass.fareharbor.com (FQDN) --> ns\_record --> lucy.ns.cloudflare.com (FQDN)

Similarly, this shows another nameserver (lucy.ns.cloudflare.com) responsible for handling DNS queries for compass.fareharbor.com.

compass.fareharbor.com (FQDN) --> cname\_record --> fareharborsites.com (FQDN)

This indicates a Canonical Name (CNAME) record where compass.fareharbor.com is pointing to fareharborsites.com. CNAME records are used to alias one domain name to another.

fareharborsites.com (FQDN) --> a\_record --> 104.17.48.43 (IPAddress)

fareharborsites.com has an A record pointing to the IPv4 address 104.17.48.43. A records map domain names to their corresponding IPv4 addresses.

fareharborsites.com (FQDN) --> a\_record --> 104.17.47.43 (IPAddress)

Similarly, fareharborsites.com also has an A record pointing to another IPv4 address 104.17.47.43.

fareharborsites.com (FQDN) --> aaaa\_record --> 2606:4700::6811:302b (IPAddress)

fareharborsites.com has an AAAA record pointing to the IPv6 address 2606:4700::6811:302b. AAAA records map domain names to their corresponding IPv6 addresses.

fareharborsites.com (FQDN) --> aaaa\_record --> 2606:4700::6811:2f2b (IPAddress)

Another AAAA record for fareharborsites.com, pointing to the IPv6 address 2606:4700::6811:2f2b.

104.16.0.0/14 (Netblock) --> contains --> 104.17.48.43 (IPAddress)

This indicates that the IP address 104.17.48.43 is part of the IP netblock 104.16.0.0/14, which is owned by Cloudflare (ASN 13335).

104.16.0.0/14 (Netblock) --> contains --> 104.17.47.43 (IPAddress)

Similarly, the IP address 104.17.47.43 is also part of the same IP netblock 104.16.0.0/14.

2606:4700::/47 (Netblock) --> contains --> 2606:4700::6811:302b (IPAddress)

The IPv6 address 2606:4700::6811:302b is within the IP netblock 2606:4700::/47 owned by Cloudflare.

2606:4700::/47 (Netblock) --> contains --> 2606:4700::6811:2f2b (IPAddress)

Similarly, 2606:4700::6811:2f2b is also within the same IP netblock 2606:4700::/47.

13335 (ASN) --> managed\_by --> CLOUDFLARENET - Cloudflare, Inc. (RIROrganization)

ASN 13335 (Autonomous System Number) is managed by Cloudflare (CLOUDFLARENET).

13335 (ASN) --> announces --> 104.16.0.0/14 (Netblock)

Cloudflare announces ownership of the IP netblock 104.16.0.0/14.

13335 (ASN) --> announces --> 2606:4700::/47 (Netblock)

Cloudflare also announces ownership of the IPv6 netblock 2606:4700::/47