**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**

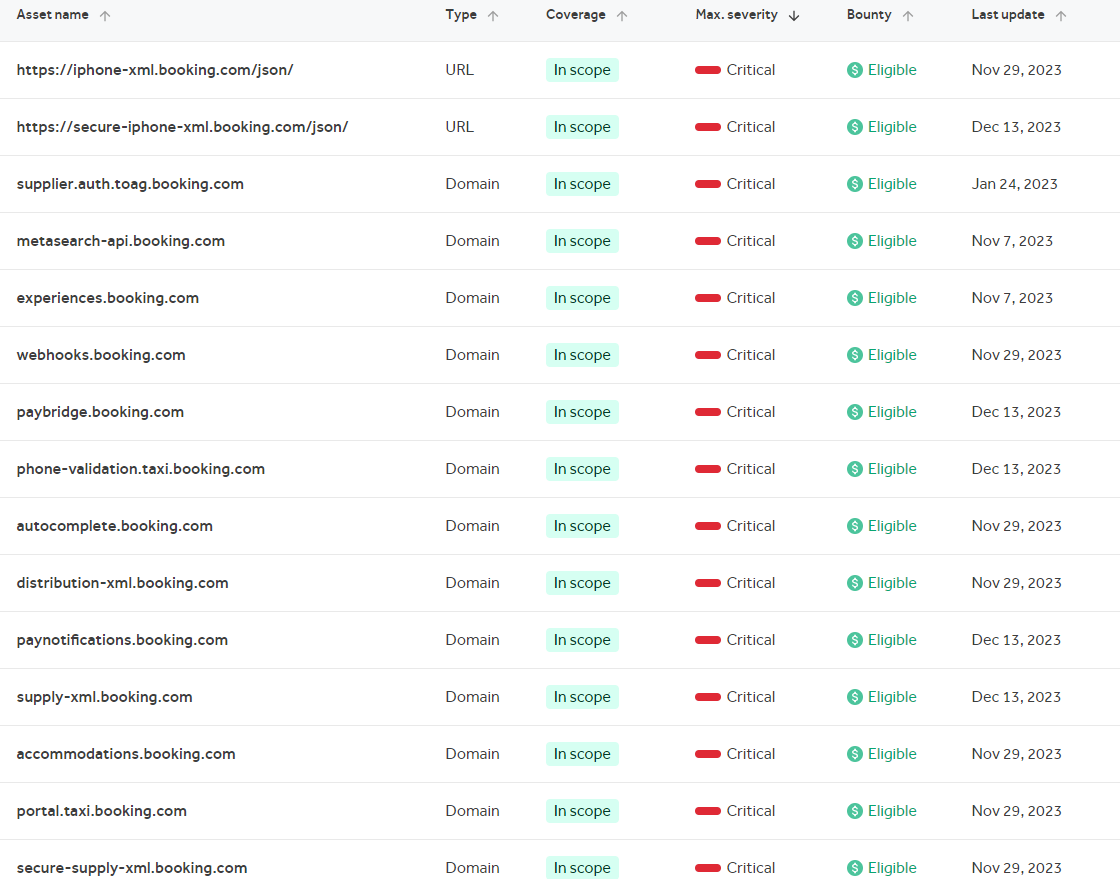
**Accepted, in-scope vulnerabilities include, but are not limited to:**

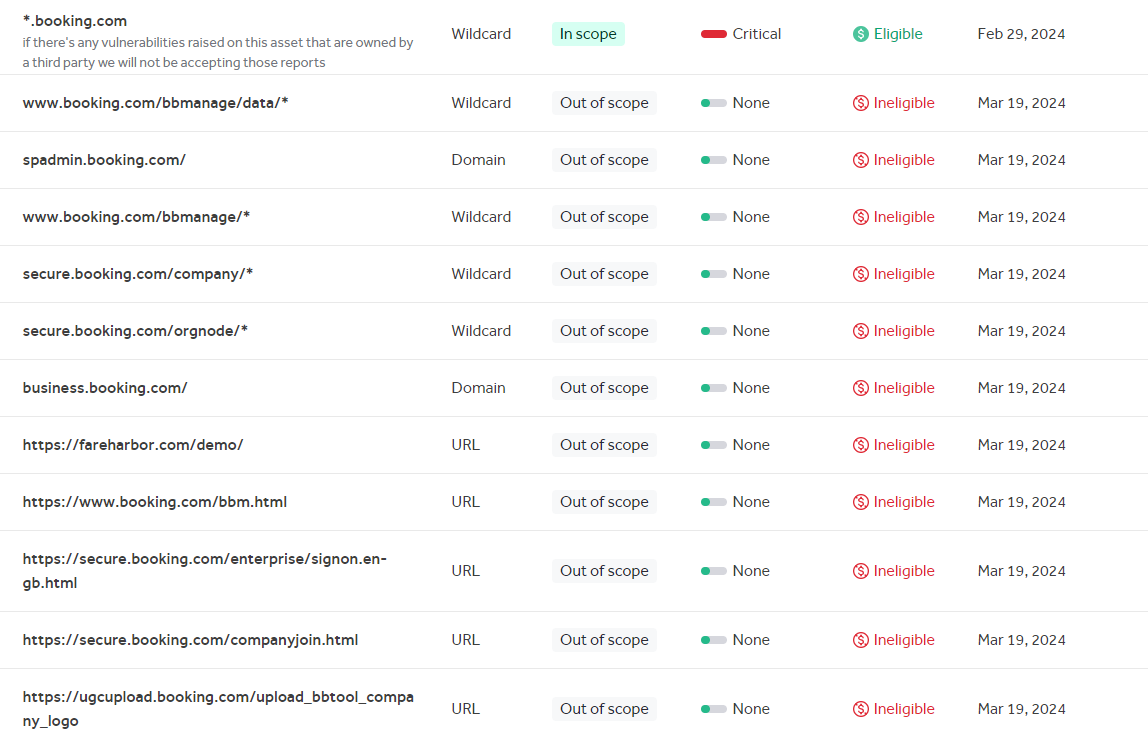
* Disclosure of sensitive or personally identifiable information
* Cross-Site Scripting (XSS) - Please note, for XSS if the same issue is reported for the different subdomains but with the same root cause, it will be considered duplicate
* Cross-Site Request Forgery (CSRF) for sensitive functions in 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:**

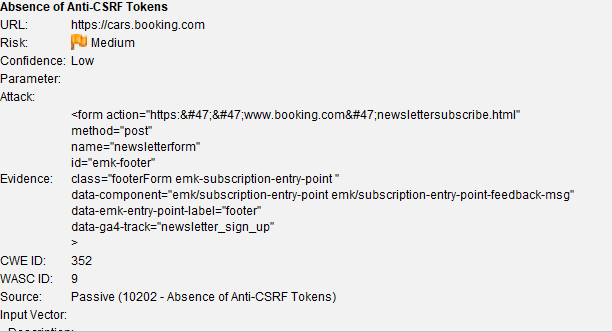
* Any vulnerability which requires access to a compromised email account or Booking.com account for successful exploitation
* Vulnerabilities on Third Party Products
* Attacks requiring physical access to a user's device or network.
* Forms missing CSRF tokens (we require evidence of actual CSRF vulnerability)
* 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.)





<https://cars.booking.com>





No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

CSRF attacks are effective in a number of situations, including: \* The victim has an active session on the target site. \* The victim is authenticated via HTTP auth on the target site. \* The victim is on the same local network as the target site.

CSRF has primarily been used to perform an action against a target site using the victim’s privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

Solution

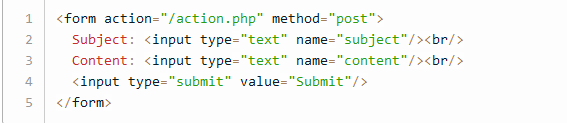
Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Normally Anti-CSRF tokens are used to protect against cross-site request forgery attacks. This post explains the basics of CSRF tokens and shows how to use them to protect the users of your websites and applications against CSRF.

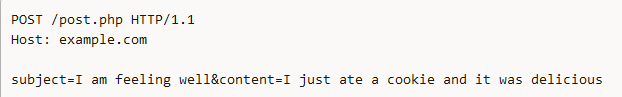
**Anti-CSRF token basics**

The idea behind anti-CSRF tokens (aka synchronizer token patterns or simply CSRF tokens) is give the user’s browser a piece of information (a token) that the browser then has to send back. The token must be unique and impossible to guess by a third party, and the application must only process HTTP requests once the token has been verified. This ensures that only the original user can send requests within an authenticated session.

For a basic example without CSRF protection, say you run a web application on *www.example.com*. To publish a message on their profile in the app, a user completes an HTML form and clicks the *Submit* button:



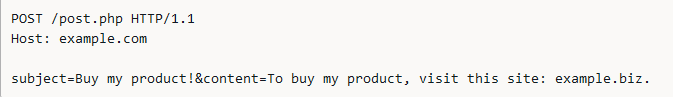
The submit action causes the web browser to send a POST request to the server, with whatever data the user entered being sent as parameters.



If the user is logged in and the attacker knows the request syntax, it may be possible to use a CSRF attack to publish an ad on that user’s profile:



As a result, the web browser sends the following POST request:

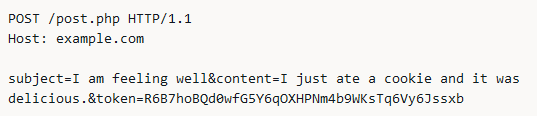


On an unprotected page, this could achieve CSRF if the server treats the forged request as coming from an authenticated user.

But now let’s say your site uses simple token-based CSRF mitigation, and your web server sets the token in a session cookie sent to the browser right after login. All the form submissions then include a hidden field containing the token. Assuming proper token validation, this completely eliminates the CSRF vulnerability:

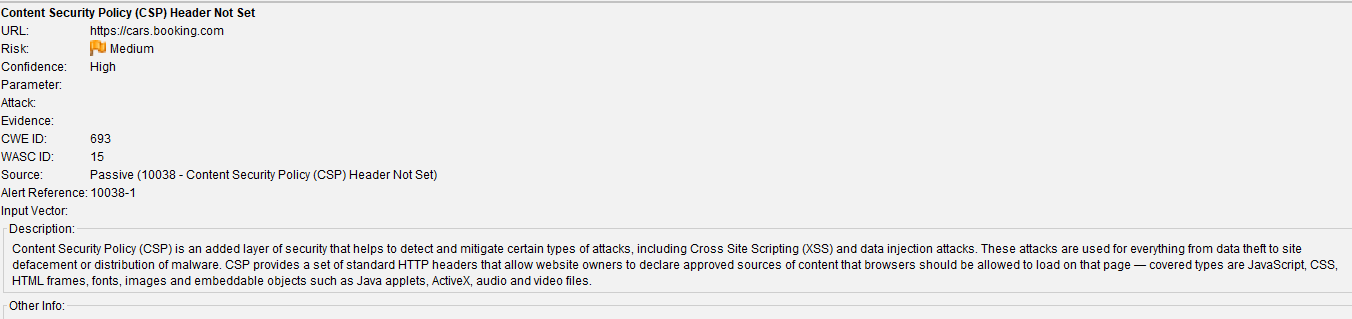


The server should then only accept POST requests from the same user that include this exact token value, for example:



With this protection in place, an attacker who tries to perform CSRF using a malicious site cannot fake HTTP requests without knowing the current token set in the valid user’s cookie. Because your server rejects all requests without this token, any attack attempts will fail.

<https://www.invicti.com/blog/web-security/protecting-website-using-anti-csrf-token/>

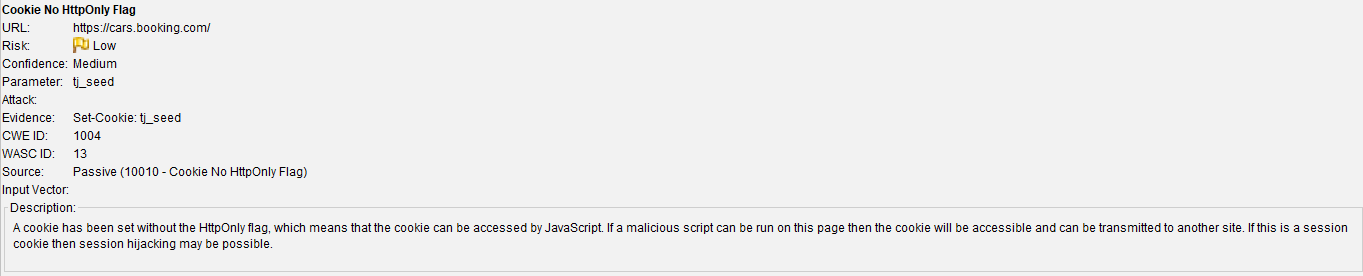
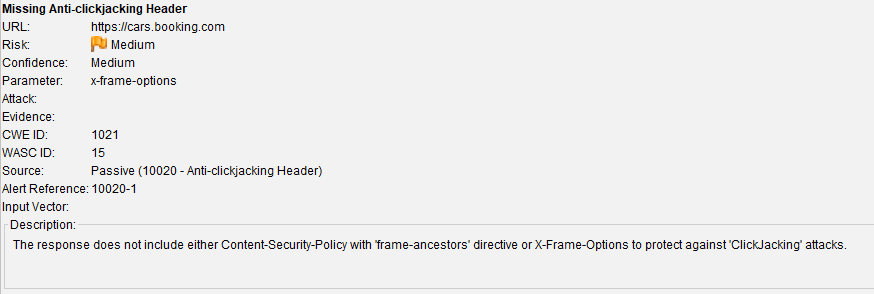


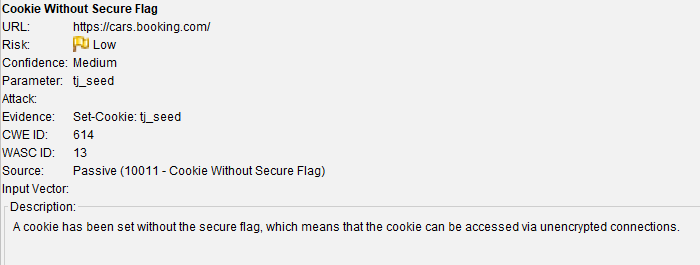
Content Security Policy (CSP) is a security feature that helps prevent code injection attacks by defining and enforcing a whitelist of approved content sources. It does this by defining a policy. If the CSP header is not set correctly, attackers can inject malicious scripts into your web application, leading to potential data theft, or unauthorized access.

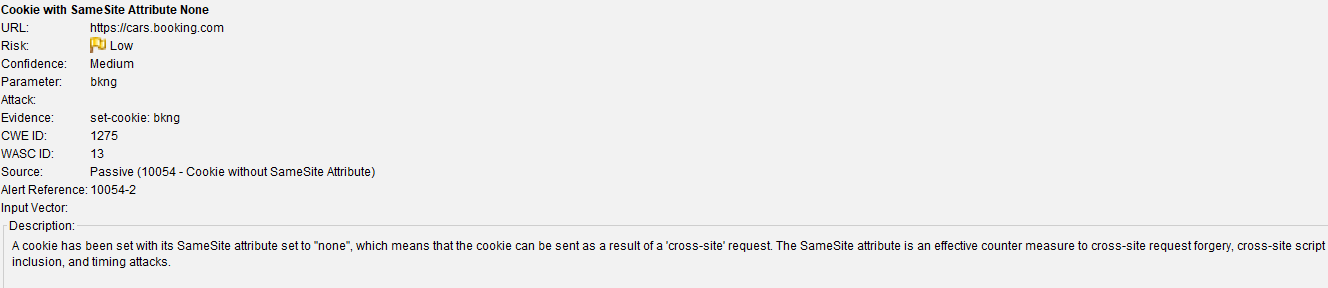
https://cheatsheetseries.owasp.org/cheatsheets/XSS\_Filter\_Evasion\_Cheat\_Sheet.html

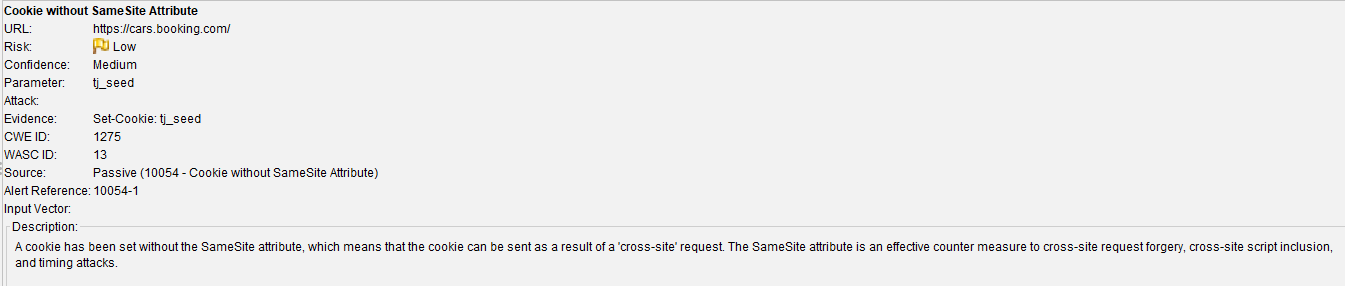
<https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html> this is a cheat sheet for content security policy.

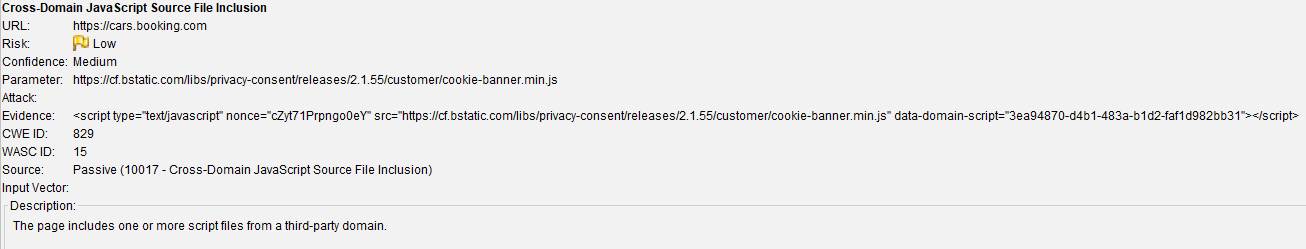
We can prevent this by defining the CSP correctly.

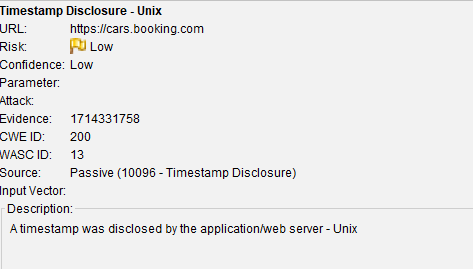




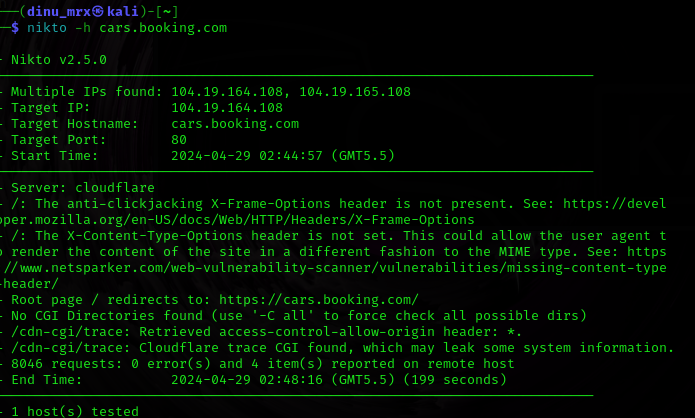


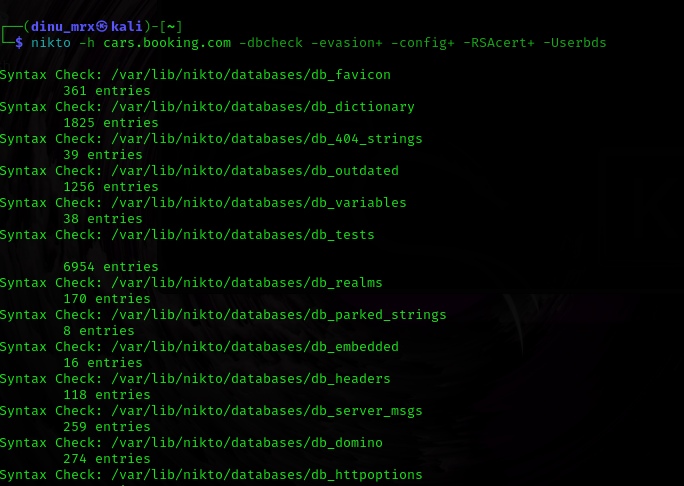






Results that were obtained when scanned through nikto





Didn’t give any configuration files, client certificate files.

After manually testing the using OWSAP ZAP I found 12 hidden fields, however these hidden fields are sanitized, and they also validate the input

A screenshot of a computer

Description automatically generated

In here it’s inputs are sanitized

Results that I obtained when tested with the URL.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

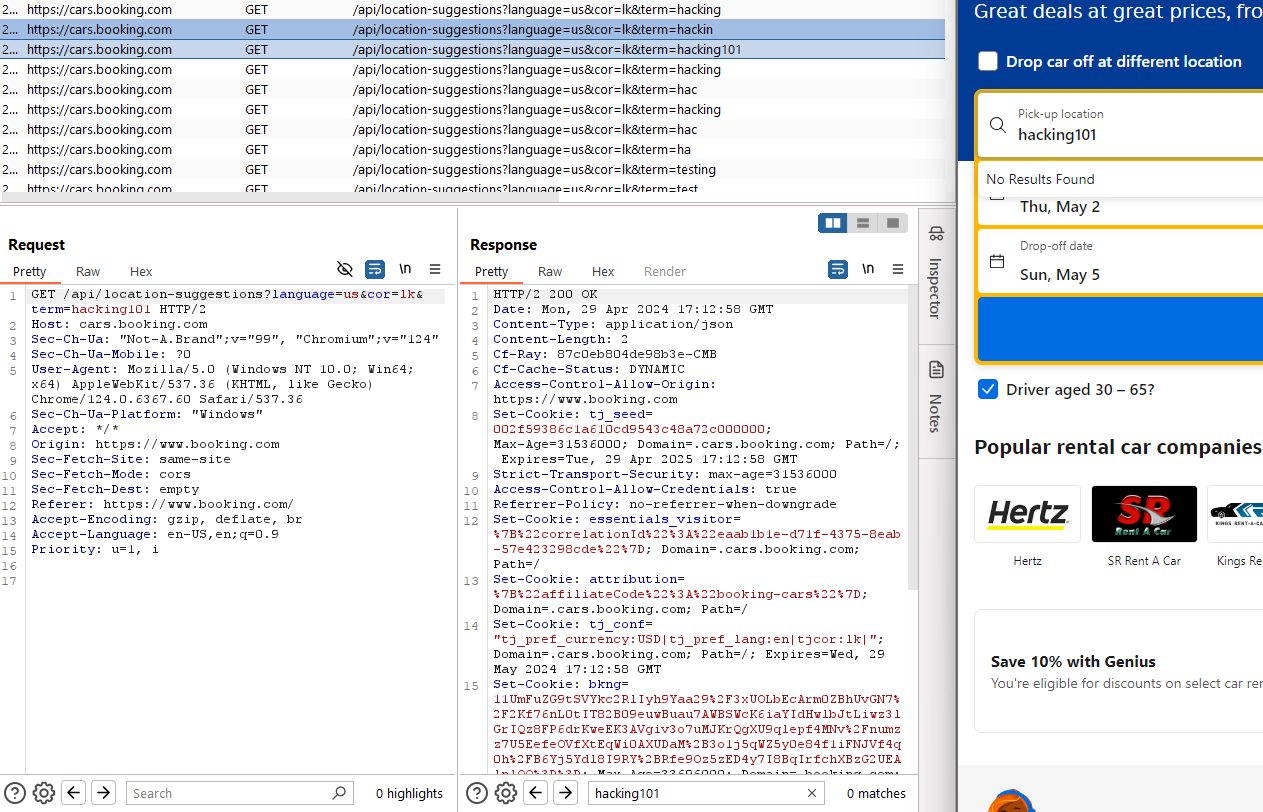
It is blocking the request which means even the URL is properly encoded, with special characters and Js commands.

And when we see the cookie value it is also encode

A screenshot of a computer

Description automatically generated

When it come to cross site scripting when I enter the location it not reflected in the DOM so it is impossible to inject XSS



No highlights in the response so this site is invulnerable to XSS.