Sri Lanka Institute of Information Technology

IE2062

Web Security

Bug Bounty Report VIII

Submitted by:

Student Registration Number	Student Name
IT21197550	Nihila Premakanthan

Date of Submission: 28.05.2023

Acknowledgement

I would like to express my special thanks to our mentor Ms. Chethana Liyanapathirana and Dr. Lakmal Rupansighe for their time and efforts she provided throughout the course, and for the Web Security lecture panel for guiding us through this semester and for helping us by giving examples, guidelines, and advice about the project. Your useful advice and suggestions were really helpful to me during the project's completion. In this aspect, I am eternally grateful to you.

Executive Summary

This report aims to provide an overview of the vulnerability identified in a particular domain. The bug bounty platform called Hackerone was used for this purpose. This report analyses the domain of Derbit (http://test.deribit.com/).

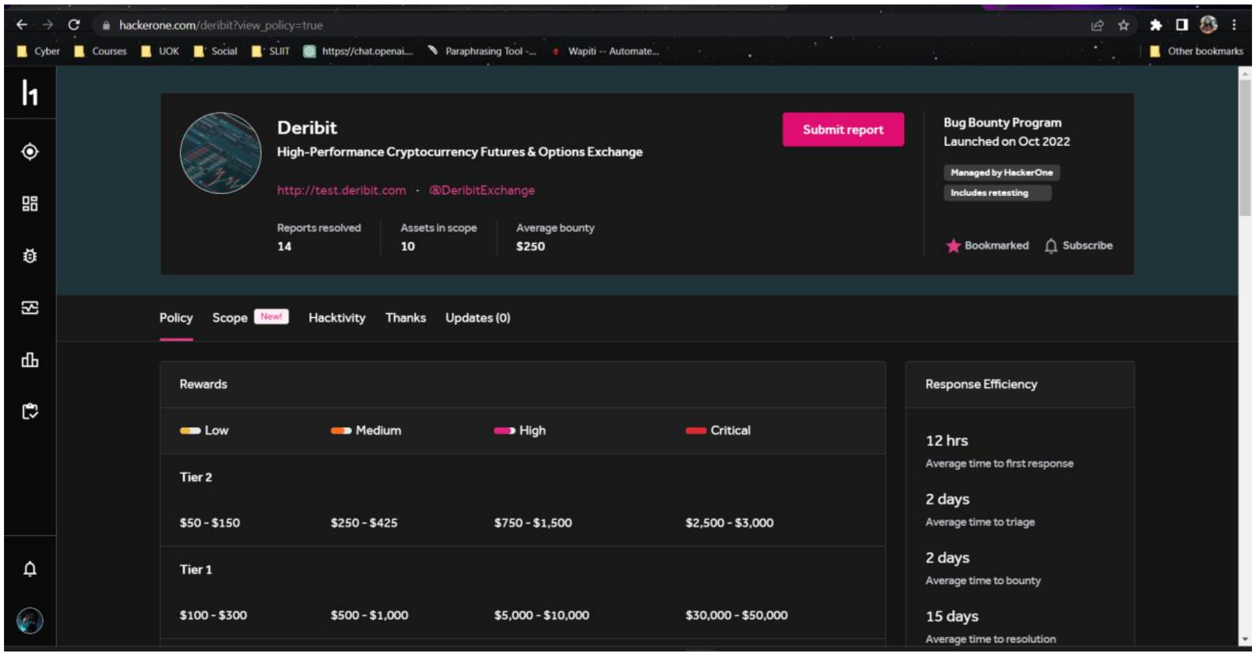
This report uses different tools to gather information detect vulnerabilities and perform penetration testing. The tool name Netsparker and Owasp Zap was mainly used to identify the vulnerability. Further this report provides the vulnerability title, vulnerability description, Affected Components, Impact Assessment, Steps to reproduce the vulnerability, proof of concept and the proposed mitigation.

By including these comprehensive details for each vulnerability, the report provides a comprehensive overview of the security weaknesses present within the system and offers actionable insights for remediation and improvement.

Contents

Introduction

Affected URL: http://test.deribit.com/



hackerone.com/deribit?view_policy=true

Deribit

High-Performance Cryptocurrency Futures & Options Exchange

http://test.deribit.com · @DeribitExchange

Reports resolved	Assets in scope	Average bounty
14	10	$250

Bug Bounty Program
Launched on Oct 2022

Managed by HackerOne
Includes retesting

Rewards

	Low	Medium	High	Critical
Tier 2	$50 - $150	$250 - $425	$750 - $1,500	$2,500 - $3,000
Tier 1	$100 - $300	$500 - $1,000	$5,000 - $10,000	$30,000 - $50,000

Response Efficiency

12 hrs	Average time to first response
2 days	Average time to triage
2 days	Average time to bounty
15 days	Average time to resolution

Vulnerability Title

Session Cookie Not marked as Secure

Severity:

Risk Level:
HIGH

Vulnerability Description

A web application security hole known as "Session Cookie Not Marked as Secure" occurs when the session cookie, which is used to authenticate and retain a user's session, is not set with the "Secure" mark. Cookies can have the "Secure" flag set to them to ensure that only secure, encrypted connections (HTTPS) are used to send them, lowering the possibility of interception and unauthorized access.

Session cookies can be sent over both secure (HTTPS) and non-secure (HTTP) connections when a web application fails to set the "Secure" mark for them. Due to this behavior, the session cookie is susceptible to being intercepted and used by attackers who may carry out a variety of harmful tasks, such as session hijacking, identity theft, and illegal account access.

Affected Components

The web application's session management and cookie handling procedures are the main aspects of the "Session Cookie Not Marked as Secure" vulnerability that are affected. The following elements may be impacted:

- Session Cookies: The main element impacted by this vulnerability is session cookies. In order to preserve the user's session state and keep data specific to that session, session cookies are utilized. Session cookies that are not designated as secure can be transmitted via insecure connections, increasing the likelihood of their being intercepted and being accessed by unwanted parties.
- Web Application Server: The "Secure" flag and other suitable cookie properties are set by the web application server, which also manages session cookies and generates them. The vulnerability increases if the server fails to classify session cookies as secure.
- Web application framework: The session management and cookie handling features of the web application framework, such as Django, Ruby on Rails, or Laravel, are important. These frameworks frequently provide with built-in settings and options for managing session cookies safely. Session cookies may not be marked as secure if these features are not used or are configured incorrectly.
- Web browsers: These are involved in the transmission and storage of session cookies. Even in the absence of the "Secure" setting, a web browser that does not enforce secure cookie transmission may permit the sending of session cookies over insecure connections, further exposing the vulnerability.
- Network Infrastructure: The web application's operating network infrastructure can also have an effect. The probability of session cookies being captured during transmission increases if the network is not properly secured using firewalls and encryption techniques.

Impact assessment

The "Session Cookie Not Marked as Secure" vulnerability can have a substantial negative effect, putting users and the web application at risk in several ways. An evaluation of the probable effects is provided below:

- Session Hijacking: An attacker can take over a user's session by intercepting session cookies sent across insecure connections. As a result, they can access the user's account without authorization, act on their behalf, and perhaps compromise important data.

- Account Compromise: If a session cookie is successfully intercepted by an attacker and used to hijack a user's session, the user's account may be compromised. This could result in unwanted access, changes to settings or preferences on the account, or even full account takeover.
- Unauthorized Data Access: An attacker who seizes control of the session can access and steal this data if the session cookie grants access to confidential information or privileged parts of the web application. Personal information, financial information, and any other confidential data saved during the user's session may fall under this category.
- Data Manipulation or Forgery: An attacker with access to a user's session may alter or fabricate data within the session. This may result in unauthorized data modifications, such as the submission of harmful transactions, tampering with user preferences, or the creation of false records.

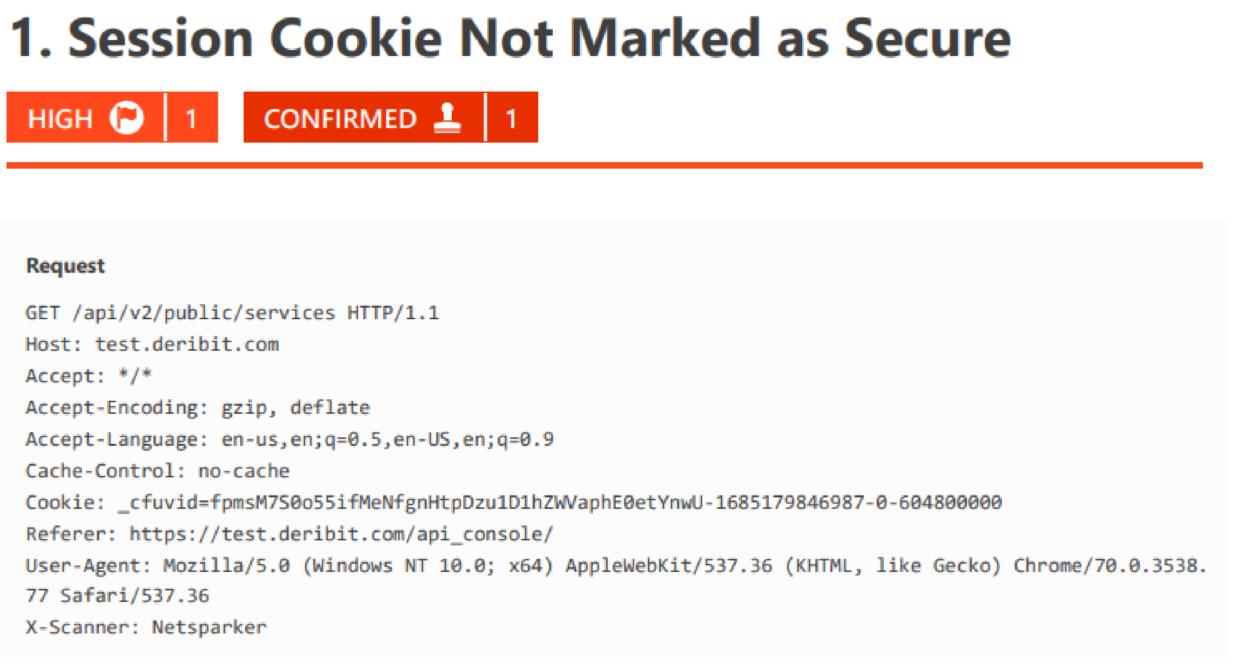
Steps to reproduce

The "Session Cookie Not Marked as Secure" vulnerability often requires access to the web application's source code or configuration settings. Depending on the technology stack being utilized, the particular processes may vary, however the following is a broad outline:

- Finding the target web application: Select the web application you want to examine for vulnerabilities. Make sure you are legally permitted to conduct security audits on the application.
- Create a testing environment: Create a testing environment that replicates the infrastructure of the online application, including the web server, database, and any other pertinent components. Local development tools or virtualized environments can be used for this.
- Analyze the session management mechanism: Understanding how the web application manages user sessions and session cookies can help you analyze the session management method. This include looking over the source code, configuration files, or any pertinent session management documentation.
- Find the handling code for session cookies: Find the code portion in charge of creating and managing session cookies. Look for the code that configures the "Secure" flag and other session cookie properties.
- Look to see if the "Secure" flag is present: Check to see if the session cookies have the secure designation. Examining the code or configuration options relating to cookie generation will help with this. Look for situations where the "Secure" flag is implicitly set, or is set to "false" or another appropriate value.

- Check the program: Tests should be run using the identified code portion to verify the vulnerability. This may entail exploiting an insecure connection (HTTP) to access the online application and employing tools like packet sniffers or browser developer tools to observe network traffic.
- Capture and analyze the session cookie: Intercept network traffic to acquire and analyze session cookies that are being transmitted through insecure connections. Check the cookie that was intercepted to make sure the "Secure" flag isn't there.
- Validate the vulnerability: Test the vulnerability's validity Utilize the session cookie that was acquired to pretend to be the user and check to see if the online application permits harmful or illegal activity.

Proof of concept

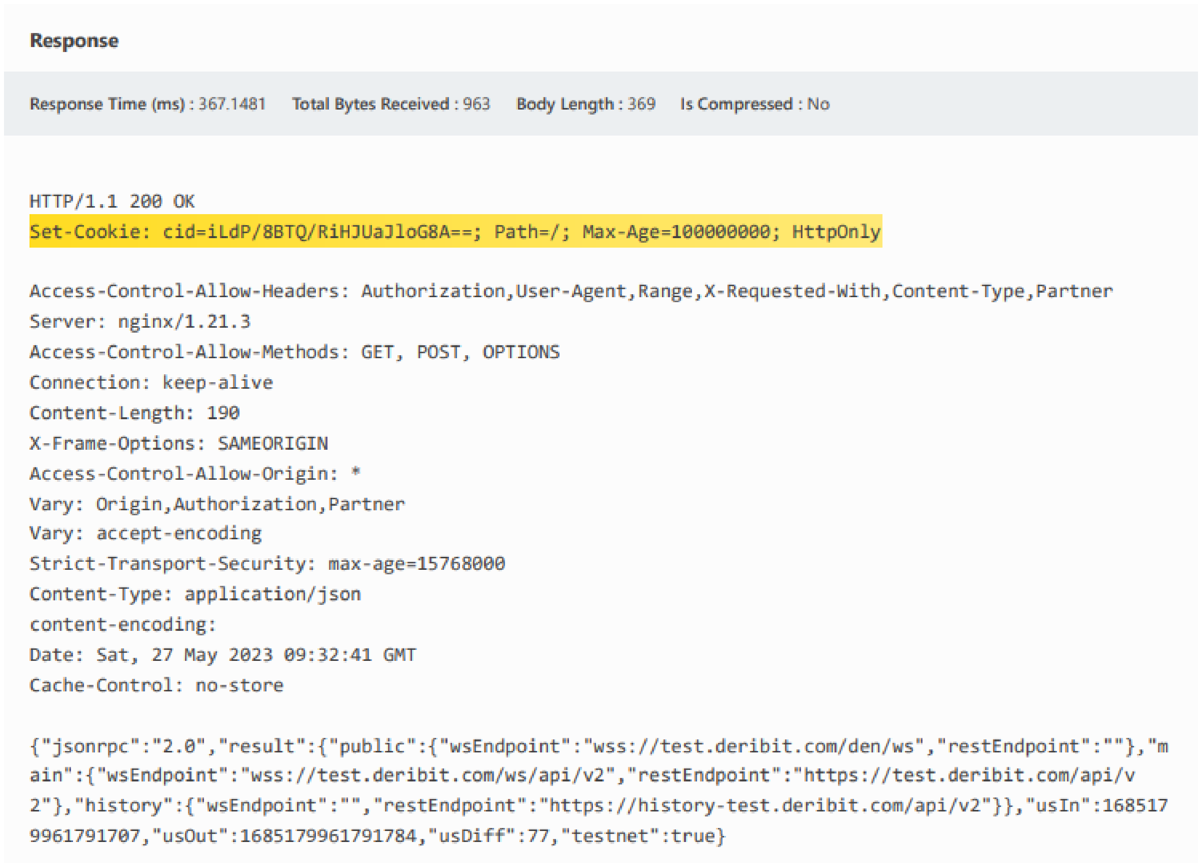


1. Session Cookie Not Marked as Secure

HIGH 1 CONFIRMED 1

Request

```
GET /api/v2/public/services HTTP/1.1
Host: test.deribit.com
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5,en-US,en;q=0.9
Cache-Control: no-cache
Cookie: _cfuvid=fpmsM7S0o55ifMeNfgnHtpDzu1D1hZWVaphE0etYnwU-1685179846987-0-604800000
Referer: https://test.deribit.com/api_console/
User-Agent: Mozilla/5.0 (Windows NT 10.0; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.
77 Safari/537.36
X-Scanner: Netsparker
```



Response

Response Time (ms) : 367.1481   Total Bytes Received : 963   Body Length : 369   Is Compressed : No

```
HTTP/1.1 200 OK
Set-Cookie: cid=iLdP/8BTQ/RiHJUaJloG8A==; Path=/; Max-Age=100000000; HttpOnly

Access-Control-Allow-Headers: Authorization,User-Agent,Range,X-Requested-With,Content-Type,Partner
Server: nginx/1.21.3
Access-Control-Allow-Methods: GET, POST, OPTIONS
Connection: keep-alive
Content-Length: 190
X-Frame-Options: SAMEORIGIN
Access-Control-Allow-Origin: *
Vary: Origin,Authorization,Partner
Vary: accept-encoding
Strict-Transport-Security: max-age=15768000
Content-Type: application/json
content-encoding:
Date: Sat, 27 May 2023 09:32:41 GMT
Cache-Control: no-store

{"jsonrpc":"2.0","result":{"public":{"wsEndpoint":"wss://test.deribit.com/den/ws","restEndpoint":""},"m
ain":{"wsEndpoint":"wss://test.deribit.com/ws/api/v2","restEndpoint":"https://test.deribit.com/api/v
2"},"history":{"wsEndpoint":"","restEndpoint":"https://history-test.deribit.com/api/v2"}},"usIn":168517
9961791707,"usOut":1685179961791784,"usDiff":77,"testnet":true}
```

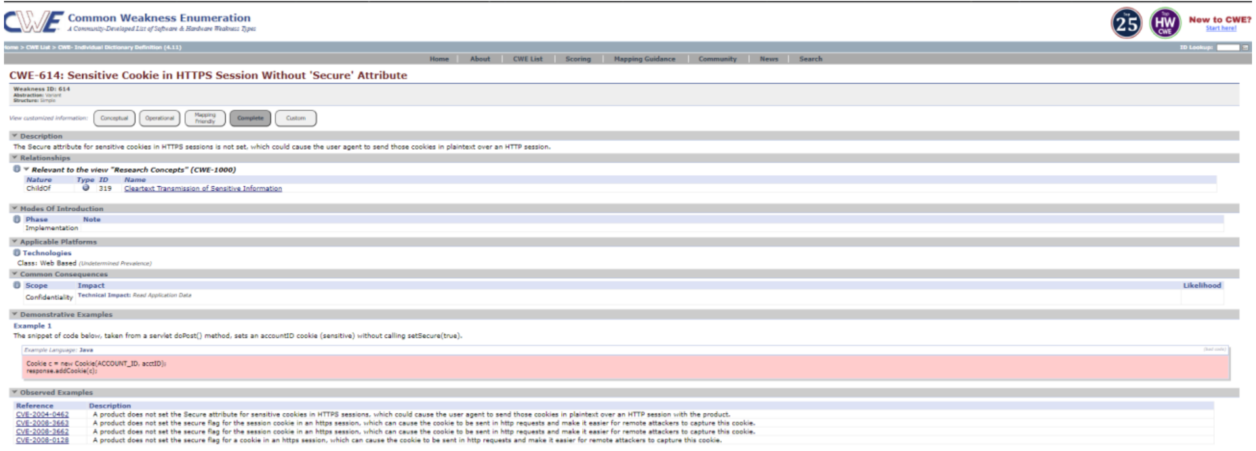
Proposed mitigation

The following best practices should be used by web application developers to solve the "Session Cookie Not Marked as Secure" vulnerability:

- Set the "Secure" flag: To ensure that session cookies are only communicated over secure, encrypted connections (HTTPS), make sure to mark them with the "Secure" flag. This can be done by setting the web server or application framework to generate session cookies that enforce the flag.
- Use HTTPS: Enforce the use of HTTPS fully throughout the online application, not simply during the login or authentication procedure, to implement secure communication. By doing this, it is guaranteed that all sensitive data, including session cookies, is transmitted through encrypted connections at all times.
- Implement HTTP Strict Transport Security (HSTS): To tell browsers to always use HTTPS while interacting with the web application. This makes it more difficult for users to unintentionally access the program via HTTP connections that aren't encrypted.

- Conduct frequent security assessments: To find and fix any session management and cookie security issues, conduct periodic security assessments and vulnerability scans. By doing this, vulnerabilities like the "Session Cookie Not Marked as Secure" problem can be found and mitigated.

<u>External Reference</u>


Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types
New to CWE? Start here!
Home > CWE List > CWE- Individual Dictionary Definition (4.11)
ID Lookup:
Home | About | CWE List | Scoring | Mapping Guidance | Community | News | Search
View customized information: Conceptual | Operational | Mapping Friendly | Complete | Custom

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Weakness ID: 614
Abstraction: Variant
Structure: Simple

Description

The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	●	319	Cleartext Transmission of Sensitive Information

Modes Of Introduction

Phase	Note
Implementation	

Applicable Platforms

Technologies

Class: Web Based *(Undetermined Prevalence)*

Common Consequences

Scope	Impact	Likelihood
Confidentiality	Technical Impact: *Read Application Data*	

Demonstrative Examples

Example 1

The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Example Language: Java (bad code)

```
Cookie c = new Cookie(ACCOUNT_ID, acctID);
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2004-0462	A product does not set the Secure attribute for sensitive cookies in HTTPS sessions, which could cause the user agent to send those cookies in plaintext over an HTTP session with the product.
CVE-2008-3663	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.
CVE-2008-3662	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.
CVE-2008-0128	A product does not set the secure flag for a cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.