SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Sri Lanka Institute of Information Technology

Bug Bounty Report - 09

Module: IE2062

Web Security

Year 2, Semester 2

Aazaf Ritha. J – IT23151710

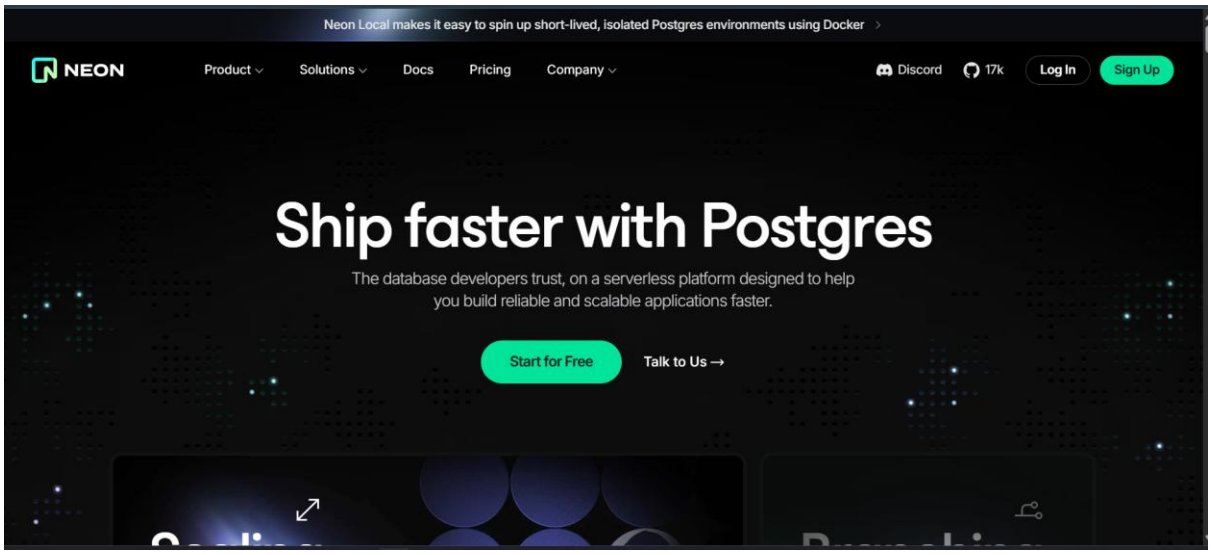
B.Sc. (Hons) in Information Technology

Specialized in Cyber Security

Table of Contents

Introduction

This report outlines a hash disclosure vulnerability involving the unintended exposure of a **BCrypt** hash on Neon's public website (https://neon.tech), identified as part of the HackerOne bug bounty program. The issue was confirmed through both manual inspection and automated security scanning. This document provides detailed technical context, clear reproduction steps, proof-of-concept evidence, and remediation guidance to help address and mitigate the risk associated with this exposure.



Vulnerability Title

Title – Hash Disclosure – BCrypt

Risk Level – High

Domain – https://neon.tech

Description

A **Hash Disclosure – BCrypt** vulnerability was identified on Neon's public website (https://neon.tech). During passive content inspection and automated security scanning with OWASP ZAP, a BCrypt-formatted hash was discovered embedded within an HTTP response. This type of hash typically represents a securely hashed password or token, and its exposure in public-facing content indicates a potential lapse in data sanitization or debug data leakage.

The presence of a BCrypt hash in a client-accessible context increases the risk of offline brute-force or dictionary attacks, particularly if the associated plaintext values are weak or reused. While BCrypt is intentionally slow to resist brute-force cracking, the visibility of such hashes provides valuable footholds to attackers and could be leveraged in further exploitation or enumeration attempts.

This vulnerability falls under **OWASP Top 10 – A06:2021 (Vulnerable and Outdated Components)** or **A01:2021 (Broken Access Control)** depending on the root cause. If the hash represents sensitive application data (e.g., user credentials, tokens, or configuration secrets), its disclosure could lead to unauthorized access, impersonation, or security policy bypass. Eliminating such exposures is essential to maintaining data confidentiality and system integrity.

Affected Component

HTTP Response Body: A BCrypt-formatted hash was found embedded within the raw HTML content of a response from https://neon.tech, indicating that sensitive backend data is being exposed unintentionally to end users.

Web Application Frontend: The exposed hash is visible on a publicly accessible page (https://neon.tech/blog), meaning that any unauthenticated visitor could retrieve it by simply accessing the page.

Server-Side Rendering / Debug Output: The hash likely originates from server-side logic leaking sensitive variables into the rendered frontend, possibly due to misconfigured debug templates, logging, or template injection issues.

Impact Assessment

Offline Brute-Force Attacks: The exposed BCrypt hash can be used by attackers to attempt offline password cracking using dictionary or brute-force techniques. Although BCrypt is computationally expensive by design, weak or reused passwords may still be recoverable over time.

Credential or Token Exposure: If the disclosed hash represents an actual user credential or API token, its exposure could lead to unauthorized access, privilege escalation, or account takeover especially if no additional safeguards (e.g., rate-limiting, MFA) are in place.

Information Disclosure: Revealing internal hashes in client-side content provides attackers with insights into the application's structure, potential debugging artifacts, or backend logic that could be further exploited.

Reconnaissance Vector: Even if the hash itself is not immediately useful, it indicates insecure development practices or improper data sanitization, encouraging attackers to look deeper for related vulnerabilities such as insecure direct object references (IDOR), debug endpoints, or exposed APIs.

Compliance and Privacy Risks: Exposure of hashed credentials or sensitive tokens may violate data protection policies or legal regulations (e.g., GDPR, CCPA), especially if user-related data is involved or logging mechanisms inadvertently leak PII.

Steps to Reproduce

Visit the following page in any modern web browser: **https://neon.tech/blog**

Open Developer Tools

Go to the Network Tab

In the "Response" tab, scroll or search (Ctrl + F) for the pattern: **$2a$**

Observe the Hash Disclosure

You will find a **BCrypt hash** embedded in the response, such as:

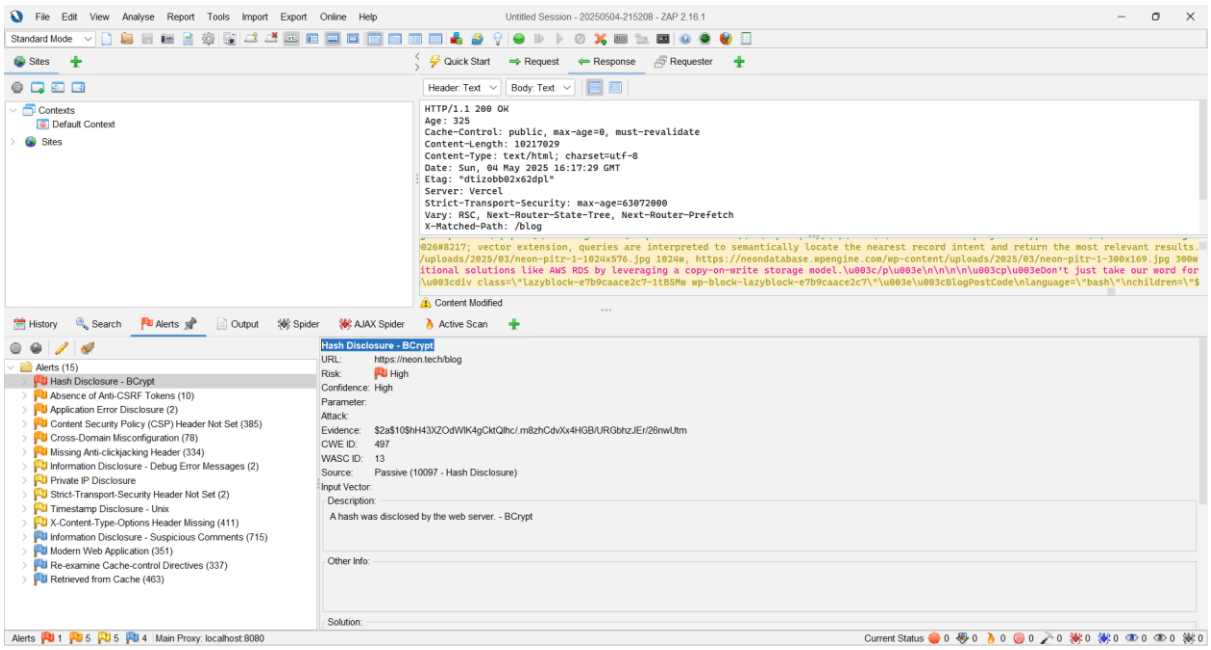
$2a$10$hH43XCOdWlK4gCktQlhc/.m8zhCdvXx4HGB/URGbhzJEr/26nwUtm

Confirm Context

Note that the hash appears inside an HTML element on the public blog page, confirming it is **visible to any unauthenticated visitor**.

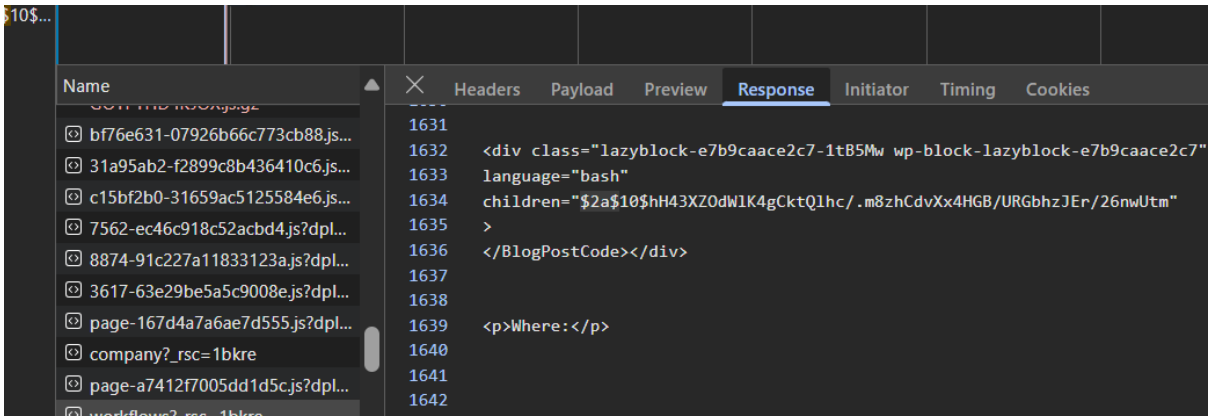
Proof of Concept (Screenshots)

OWASP ZAP Scan:



This screenshot from OWASP ZAP confirms a **Hash Disclosure – BCrypt** vulnerability on https://neon.tech/blog. A valid **BCrypt hash** ($2a$10$...) was found in the HTTP response body. The risk is marked **High**, with **High confidence**, as this kind of data should never be exposed publicly. If the hash represents a password or token, it could be used in **offline brute-force attacks**. This issue likely stems from a **template or debug data leak** and should be remediated immediately.

Manual Verification via Browser:



This screenshot shows a **BCrypt hash** ($2a$10$...) exposed directly in the HTML response of the https://neon.tech/blog page. It appears inside a element rendered on the frontend, confirming that the hash is publicly accessible without authentication. This type of disclosure can lead to **offline brute-force attacks** and indicates a **server-side data leak or insecure template rendering**.

Proposed Mitigation or Fix

Remove Sensitive Data from Client-Side Responses

Audit the server-side code responsible for rendering public pages (e.g., https://neon.tech/blog) to ensure that no hashed credentials, tokens, or debug data are exposed in HTML or JSON responses. Ensure that only non-sensitive content is returned to unauthenticated users.

Review Template Rendering Logic

Inspect view templates and rendering functions for any accidental exposure of backend variables, logs, or debugging artifacts. Avoid passing sensitive values such as password hashes to the frontend under any circumstances.

Implement Output Filtering and Sanitization

Apply strict output sanitization to ensure that even if backend data is unintentionally exposed, it is filtered before reaching the client. Remove or redact any server-side logs or placeholders that might contain hashes or internal data.

Conduct Periodic Security Scans

Regularly scan the application using security tools like OWASP ZAP, Burp Suite, or custom scripts to detect any accidental hash disclosures or other sensitive information leaks across public-facing pages.

Add Monitoring for Sensitive Patterns

Implement logging and alerting mechanisms that flag occurrences of sensitive data patterns (e.g., $2a$) in outgoing responses. This allows early detection of future leaks.

Apply Defense-in-Depth Controls

Even if a hash is exposed, ensure that other controls—such as multi-factor authentication (MFA), strong password policies, rate-limiting, and session expiration—are in place to reduce the impact of a successful brute-force attack.

Conclusion

The presence of a BCrypt hash in a publicly accessible HTTP response on Neon’s website introduces an unnecessary and avoidable security risk. While BCrypt is a strong hashing algorithm, its disclosure can aid attackers in reconnaissance or offline brute-force attempts. This exposure reflects a lapse in proper data handling and output sanitization within the application’s frontend or backend logic. Addressing this issue by removing sensitive data from responses and improving secure coding practices will significantly enhance the application's security posture and reduce the likelihood of future data leaks. Proactive remediation ensures better alignment with security best practices and industry compliance standards.