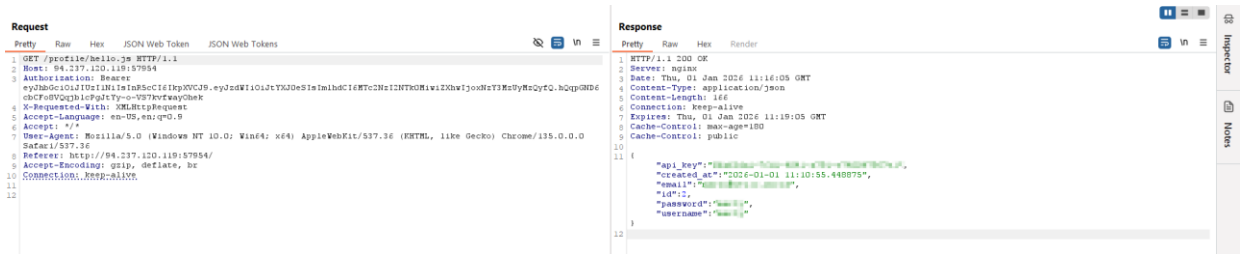
Created By: [Mritunjay Kumar]

Web Cache Deception Leading to Information Exposure

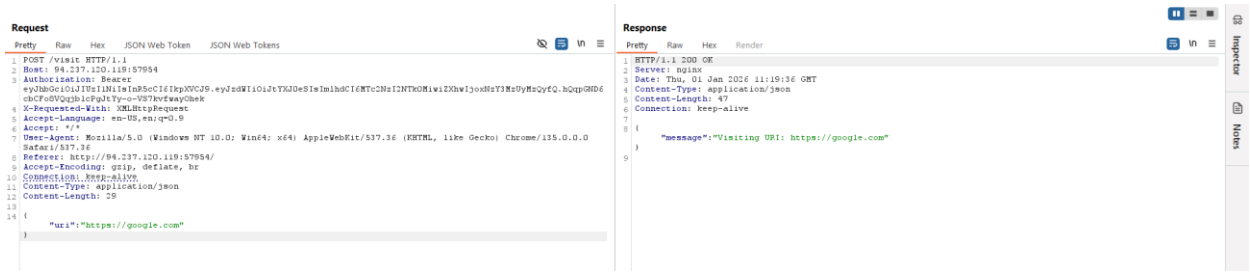
Description:
Web Cache Deception is a vulnerability that occurs when a web application incorrectly allows sensitive, user-specific content to be cached by intermediate caching mechanisms such as CDNs, reverse proxies, or browser caches. Attackers exploit discrepancies between how the origin server and cache interpret URLs.
By appending cacheable file extensions (e.g., .css, .jpg, .js) or crafted paths to authenticated endpoints, an attacker can trick the cache into storing private responses. Once cached, this sensitive content can be retrieved by unauthenticated users

Steps to reproduce:

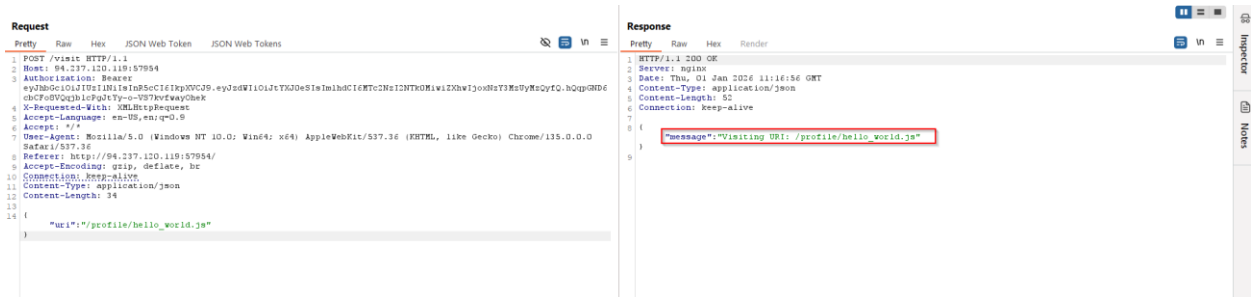
1. The application has a /profile endpoint to show us our user's details such as api key, email address, id, username, and password. The static files at this endpoint get cached at the server side. Surprisingly the response of /profile is the same as /profile/hello.js. So, the contents of hello.js has the user details getting cached.



2. The application also provides us an option to fetch a URI of our choice. From the context of the application, the URI is visited by a bot authorized by the administrator.

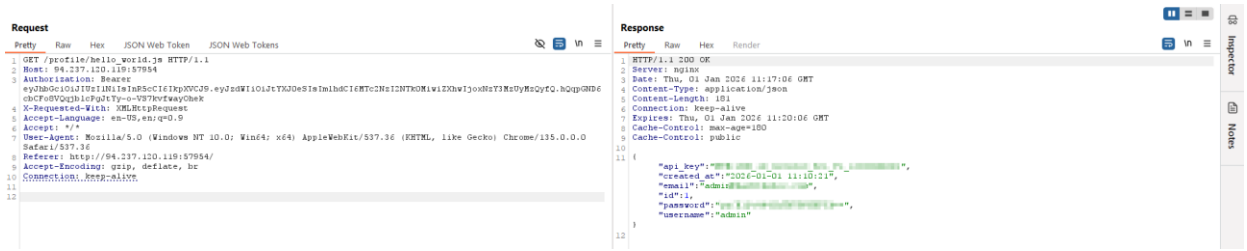


3. The attacker can now direct the bot to visit the /profile endpoint to cache the administrator's user details.



The administrator's details get cached and his API key and password gets exposed.

Created By: [Mritunjay Kumar]



Impact:
Successful exploitation of Web Cache Deception can lead to severe information disclosure. Attackers may retrieve sensitive data such as session tokens, CSRF tokens, personal user information, account details, or API responses belonging to other users.
The above application exposes the user details such as the API key, username, password of the administrator which can further be used to take over his account.

Solution:
To mitigate Web Cache Deception vulnerabilities, applications should strictly control cache behavior for authenticated or sensitive endpoints.
Ensure that responses containing user-specific data include appropriate cache-control headers such as 'Cache-Control: no-store, no-cache, must-revalidate' and 'Pragma: no-cache'. CDN and reverse proxy configurations should be reviewed to prevent caching based solely on file extensions.
Additionally, implement strict URL normalization and routing logic so that dynamic endpoints cannot be accessed via misleading or unintended paths.