// HALBORN

Haqq - Backend

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **August 21st, 2023 - September 8th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/08/2023
0.2	Draft Review	09/11/2023
0.3	Draft Review	09/11/2023
1.0	Remediation Plan	10/23/2023
1.1	Remediation Plan Review	10/23/2023
1.2	Remediation Plan Review	10/24/2023
1.3	Remediation Plan Update	11/01/2023
1.4	Remediation Plan Update Review	11/02/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Carlos Polop	Halborn	Carlos.Polop@halborn.com
Afaq Abid	Halborn	Afaq.Abid@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Haqq engaged Halborn to conduct a security assessment on their backend code and its respective API and services, beginning on August 21st, 2023 and ending on September 8th, 2023. The security assessment was scoped to the backend and underlying API service. Halborn was provided access to the application, and its respective source code to conduct security testing using tools to scan, detect, validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided a timeline for the engagement and assigned a full-time security engineer to verify the security of the assets in scope. The security engineer is a penetration testing expert with advanced knowledge in web, mobile, recon, discovery & infrastructure penetration testing.

In summary, Halborn identified several vulnerabilities. A potential for a Denial of Service attack due to long JWT tokens handling at backend that could adversely affect the application's availability. Similarly, we discovered a Docker Compose Environment Variable Misconfiguration and Cacheable HTTPS Response vulnerabilities, which could expose sensitive information or configurations. These vulnerabilities are particularly concerning given the potential for attackers to exploit the environment or intercept and cache sensitive data.

Furthermore, the application's API endpoints exhibit a Lack of Rate Limiting, increasing the risk of brute-force attacks or abuse by automated bots. Another significant concern revolves around the Outdated Versions of TLS Supported, which can leave encrypted communications susceptible to modern attack techniques.

On the web application front, Missing Security Headers were observed, which can enhance the risk of client-side attacks if not appropriately addressed. The presence of Reflected Unsanitized Input suggests poten-

tial vectors for cross-site scripting or similar client-side exploits, although the current impact is limited.

It was also identified Vulnerable Third-Party Dependencies within the application stack, potentially introducing known vulnerabilities that malicious actors could exploit. Lastly, the occurrence of an Internal Server Error on Input Handling was noted, which, while not immediately indicative of a vulnerability, but recommended fixing due to the potential implications and risks associated with such errors.

Throughout the remediation process, the Haqq team successfully resolved the majority of the identified issues. However, there were specific instances where the Haqq team acknowledged and accepted the risk associated with certain unresolved issue, choosing to proceed with awareness of the potential implications.

1.3 SCOPE

IN-SCOPE:
The following API and its respective repository were in scope:

URLs:

- https://generator-shares.social.production.haqq.network/
- https://social-share-1.social.production.haqq.network/
- https://metadata.social.production.haqq.network/

Repos:
- https://github.com/haqq-network/social/tree/main/generator-shares
- https://github.com/haqq-network/social/tree/main/metadata
- https://github.com/haqq-network/social/tree/main/social-share
- https://github.com/haqq-network/social/tree/main/shared/src

- Commit/Branch: 57d951a31350c0c22607778564d362c13fb7ce45

OUT-OF-SCOPE:
- External Libraries.

1.4 TEST APPROACH & METHODOLOGY

Halborn followed Whitebox and Blackbox methodology as per the scope and performed a combination of manual and automated security testing with both to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the pentest. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques assist enhance coverage of the infrastructure and can quickly identify flaws in it.

Throughout the assessment, it was followed the following phases, but not limited to:

- Mapping Content and Functionality
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Rate Limitations Tests
- Brute Force Attempts
- Input Handling
- Response Manipulation
- Source Code Review
- Fuzzing of all input parameters
- Multiple Type of Injection (SQL/JSON/HTML/Command)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	5	4	0

LIKELIHOOD

IMPACT

	(HAL-03) (HAL-05)	(HAL-01)		
	(HAL-06)	(HAL-02) (HAL-04)		
	(HAL-07) (HAL-08) (HAL-09)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) POTENTIAL DENIAL OF SERVICE	Medium	SOLVED - 10/23/2023
(HAL-02) LACK OF RATE LIMITING IN API ENDPOINTS	Medium	SOLVED - 10/23/2023
(HAL-03) DOCKER COMPOSE ENVIRONMENT VARIABLE MISCONFIGURATION	Medium	SOLVED - 10/31/2023
(HAL-04) OUTDATED VERSIONS OF TLS SUPPORTED	Medium	SOLVED - 09/30/2023
(HAL-05) CACHEABLE HTTPS RESPONSE	Medium	SOLVED - 10/18/2023
(HAL-06) MISSING SECURITY HEADERS	Low	SOLVED - 10/18/2023
(HAL-07) REFLECTED UNSANITIZED INPUT	Low	SOLVED - 11/01/2023
(HAL-08) VULNERABLE THIRD PARTY DEPENDENCIES	Low	RISK ACCEPTED
(HAL-09) INTERNAL SERVER ERROR ON INPUT HANDLING	Low	SOLVED - 10/18/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) POTENTIAL DENIAL OF SERVICE - MEDIUM

Description:

During the assessment, it was discovered that the long JWT tokens generated from API endpoint /api/token were accepted and processed by the backend.

This oversight can lead to potential resource exhaustion attacks by allowing malicious actors to create and process large JWT tokens, which can strain the server's memory and computational resources. The server might not be the only system component affected by large JWT tokens. Client applications, middleware, logging systems, or databases that handle or store these tokens could also face performance issues or errors.
Combined with the lack of rate limiting as reported, this vulnerability can potentially be exploited to create a Denial-of-Service (DoS) attack.

Proof of Concept:

A POST request with an extremely long email address causes significant resource utilization on the server.

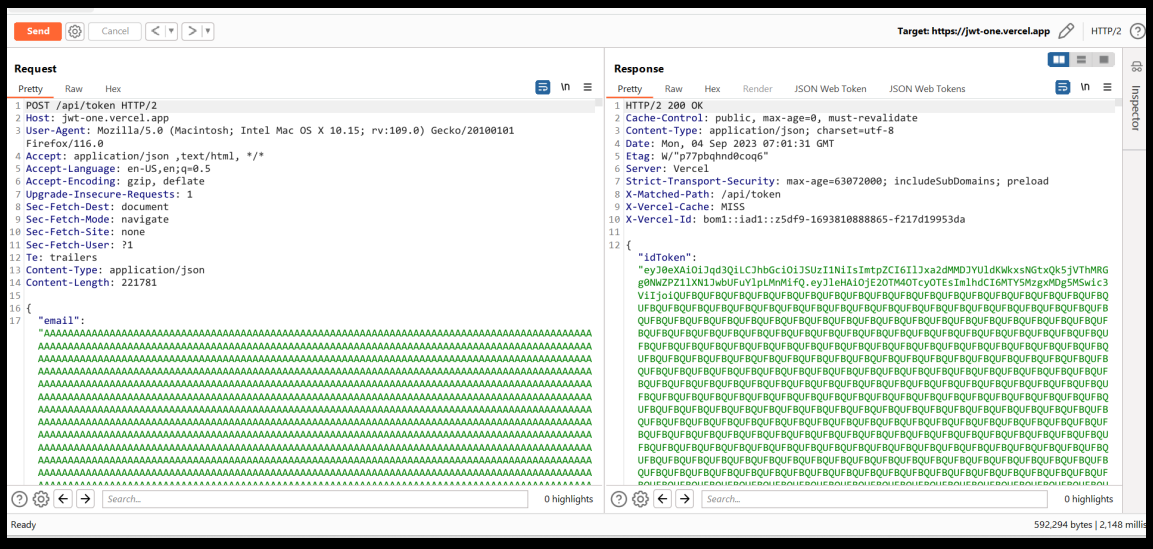

Figure 1: oversized JWT generation

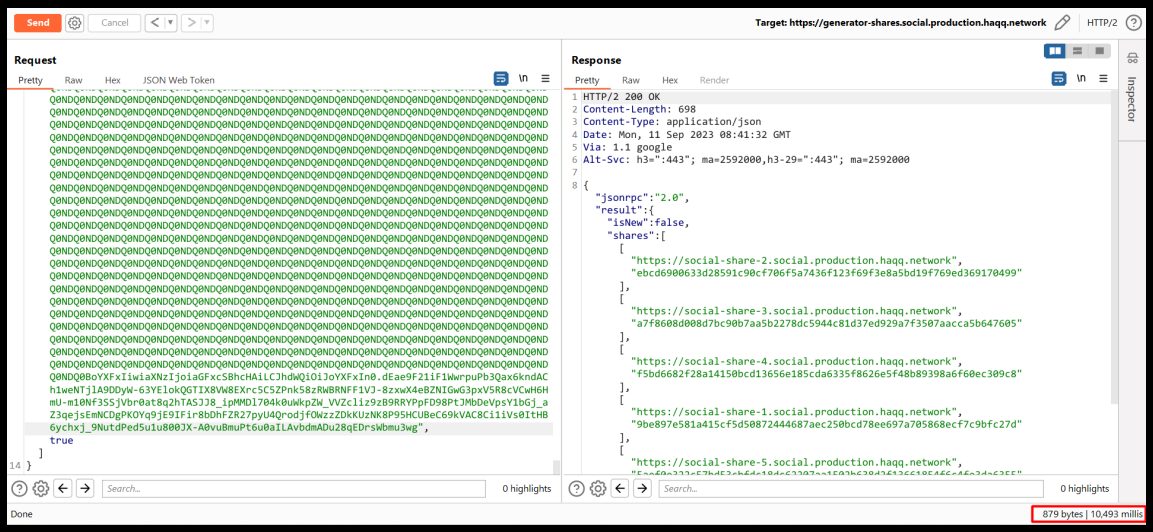

Figure 2: Backend processed the oversized JWT

NOTE: It was observed that every new JWT token processing takes some time (8-10 ms) to process and there is a window in which multiple long JWT processing could potentially overload the server.

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to implement validation for long JWT tokens at the backend, ensuring a reasonable maximum length. It is also recommended to implement rate limiting for the endpoint to prevent rapid repeated requests.

Remediation Plan:

SOLVED: The Haqq team informed that the service to create the test JWT removed and will not be part of the production, other than that the client also implemented the rate limitation protection against the identified services.

3.2 (HAL-02) LACK OF RATE LIMITING IN API ENDPOINTS - MEDIUM

Description:

API requests consume resources such as network, CPU, memory, and storage. This vulnerability occurs when too many requests arrive simultaneously, and the API does not have enough compute resources to handle those requests.

During the assessment, no rate limitation was found on the API services. An attacker could exploit this vulnerability to overload the API by sending more requests than it can handle. As a result, the API becomes unavailable or unresponsive to new requests.

Proof of Concept:

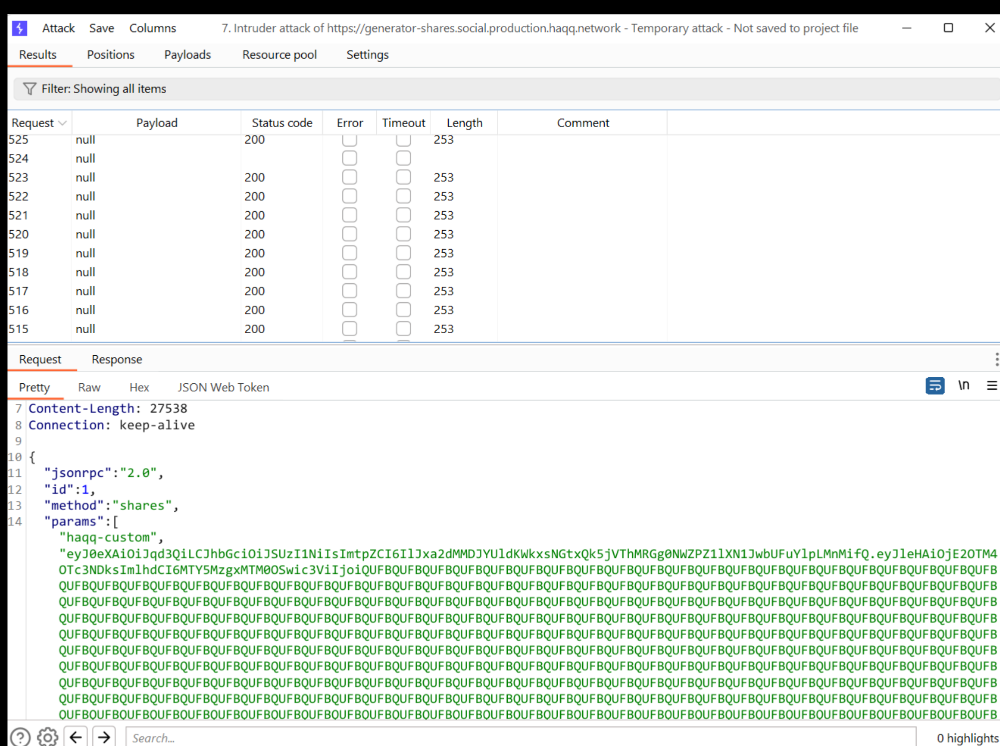

Figure 3: No rate limitation found on https://generator-shares.social.production.haqq.network/

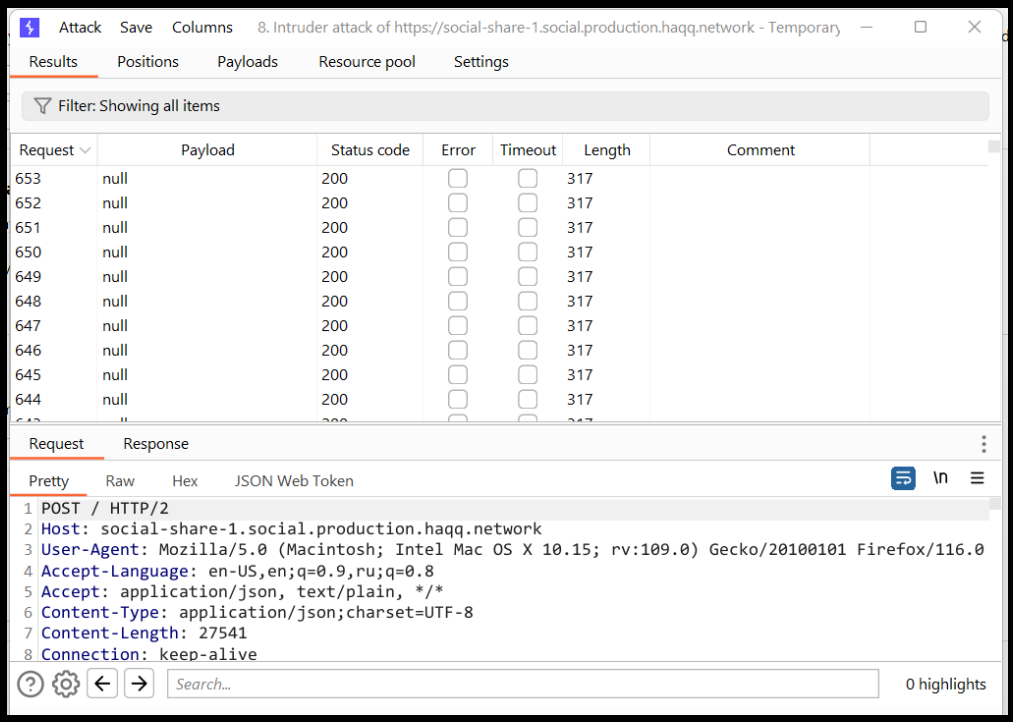

Figure 4: No rate limitation found on https://social-share-1.social.production.haqq.network/

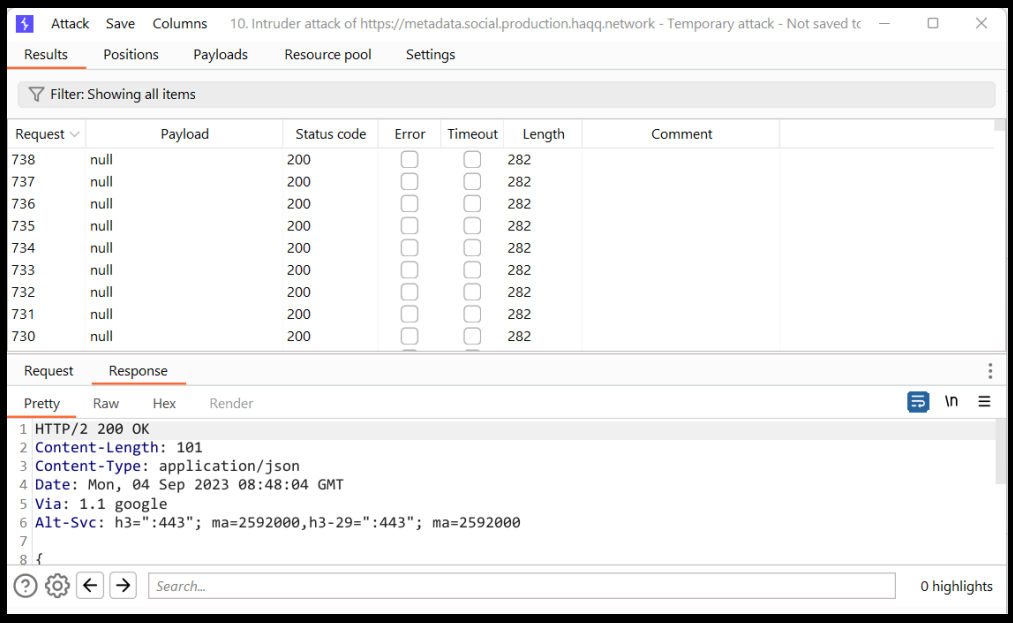

Figure 5: No rate limitation found on https://metadata.social.production.haqq.network/

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

This vulnerability is due to the application accepting requests from users at a given time without performing request throttling checks. It is recommended to follow the following best practices:

- Implement a limit on how often a client can call the API within a defined timeframe.
- Notify the client when the limit is exceeded by providing the limit number and the time the limit will be reset.

Reference:

CWE-770: Allocation of Resources Without Limits or Throttling

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing the appropriate rate limitation against the identified services.

3.3 (HAL-03) DOCKER COMPOSE ENVIRONMENT VARIABLE MISCONFIGURATION - MEDIUM

Description:

The application has a security misconfiguration vulnerability that exposes sensitive environment variables due to a misconfigured Docker Compose file. This vulnerability arises from adding an environment variable to the .env file and defining it in the **docker-compose.yml** file, which results in the variable being accessible to all running containers. This misconfiguration can lead to the exposure of sensitive information and introduce potential vulnerabilities. It also increases the chances of single point failure where, If all credentials and keys are centralized in one .env file, and file be exposed or accidentally get committed to a public version control repository. If all credentials are in a single file, the potential damage from such an oversight is significantly amplified., all associated services and data are at risk also mistakes can happen.

Code Location:

Listing 1: /docker-compose.dev.yml (Line 27)

```
24  generator-shares:
25    build: ./generator-shares
26    restart: always
27    env_file: .env
28    environment:
29      GENERATOR_SHARES_STORAGE_PREFIX: "generator_shares"
30    links:
31      - postgres
32      - redis
33    depends_on:
34      - postgres
35      - redis
```

Listing 2: /docker-compose.dev.yml (Line 48)

```
45   metadata:
46     build: ./metadata
47     restart: always
48     env_file: .env
49     environment:
50       STORAGE_CONTRACT_PRIVATE_KEY: ${METADATA_STORAGE_PRIVATE_KEY
↳ }
51     links:
52       - postgres
53       - redis
54     depends_on:
55       - postgres
56       - redis
```

Listing 3: /docker-compose.dev.yml (Line 70)

```
67   social_share_1:
68     build: ./social-share
69     restart: always
70     env_file: .env
71     environment:
72       STORAGE_CONTRACT_PRIVATE_KEY: ${
↳ STORAGE_CONTRACT_PRIVATE_KEY_1}
73       SOCIAL_SHARE_SECRET_TOKEN: ${SOCIAL_SHARE_SECRET_TOKEN_1}
74     ports:
75       - 8070:8080
76     links:
77       - postgres
78       - redis
79     depends_on:
80       - postgres
81       - redis
```

Listing 4: /docker-compose.dev.yml (Line 92)

```
89   social_share_2:
90     build: ./social-share
91     restart: always
92     env_file: .env
93     environment:
```

```
94        STORAGE_CONTRACT_PRIVATE_KEY: ${
↳ STORAGE_CONTRACT_PRIVATE_KEY_2}
95        SOCIAL_SHARE_SECRET_TOKEN: ${SOCIAL_SHARE_SECRET_TOKEN_2}
96      ports:
97        - 8071:8080
```

Listing 5: /docker-compose.dev.yml (Line 114)

```
111   social_share_3:
112     build: ./social-share
113     restart: always
114     env_file: .env
115     environment:
116       STORAGE_CONTRACT_PRIVATE_KEY: ${
↳ STORAGE_CONTRACT_PRIVATE_KEY_3}
117       SOCIAL_SHARE_SECRET_TOKEN: ${SOCIAL_SHARE_SECRET_TOKEN_3}
118     ports:
119       - 8072:8080
```

Listing 6: /docker-compose.dev.yml (Line 136)

```
133   social_share_4:
134     build: ./social-share
135     restart: always
136     env_file: .env
137     environment:
138       STORAGE_CONTRACT_PRIVATE_KEY: ${
↳ STORAGE_CONTRACT_PRIVATE_KEY_4}
139       SOCIAL_SHARE_SECRET_TOKEN: ${SOCIAL_SHARE_SECRET_TOKEN_4}
140     ports:
141       - 8073:8080
142     links:
```

Listing 7: /docker-compose.dev.yml (Line 158)

```
155   social_share_5:
156     build: ./social-share
157     restart: always
158     env_file: .env
159     environment:
160       STORAGE_CONTRACT_PRIVATE_KEY: ${
↳ STORAGE_CONTRACT_PRIVATE_KEY_5}
```

```
161         SOCIAL_SHARE_SECRET_TOKEN: ${SOCIAL_SHARE_SECRET_TOKEN_5}
162       ports:
163         - 8074:8080
```

CVSS Vector:

- CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:L/I:L/A:N

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Docker-compose allows us to define environment variables to pass to running containers with the environment option. It is recommended to use this config instead of the env_file option.

Reference:

Why you should split your env file

Remediation Plan:

SOLVED: The Haqq team solved the issue as per the recommendations.

NOTE: Retest was conducted on the following commit:
- d1d6e759dfc73e291a77cfe90724e9dcf077f31b

3.4 (HAL-04) OUTDATED VERSIONS OF TLS SUPPORTED - MEDIUM

Description:

Several misconfigurations were identified within the application's TLS configuration that could compromise the security of communications.
Scoped URLs in the application have been identified to support the deprecated and vulnerable versions of TLS: v1.0 and TLS v1.1. These versions are no longer deemed secure due to numerous cryptographic design flaws. As technology advances, so does the capability to exploit these vulnerabilities. While current attacks targeting weak cipher suites or vulnerable algorithms might seem sophisticated and complex, the relentless growth in computational power means these attacks will only become more feasible and straightforward to execute as time goes on.

These outdated TLS versions have been superseded by newer versions (e.g., TLS v1.3) which provide enhanced security features and address many of the vulnerabilities found in their predecessors. By supporting these older versions, the application not only endangers the integrity and confidentiality of data in transit but also risks non-compliance with modern security standards and regulations.

Specifically, the risks of using these deprecated TLS versions include but are not limited to:

Cipher Suite Attacks: Adversaries can exploit weak cipher suites to decrypt sensitive data.
Session Hijacking: Vulnerabilities in older versions can allow attackers to hijack a user's session.
Man-in-the-Middle Attacks: An attacker can intercept and possibly alter the communication between two parties without either party noticing.

Proof of Concept:

```
Start 2023-09-04 10:46:14        -->> 34.160.229.79:443 (generator-shares.social.production.haqq.network) <<--

rDNS (34.160.229.79):   79.229.160.34.bc.googleusercontent.com.
Service detected:       HTTP


Testing protocols via sockets except NPN+ALPN

SSLv2      /usr/bin/testssl: connect: Connection refused
/usr/bin/testssl: line 10454: /dev/tcp/34.160.229.79/443: Connection refused
 Oops: TCP connect problem

Unable to open a socket to 34.160.229.79:443. Fixme: couldn't open socket
SSLv3      not offered (OK)
TLS 1      offered (deprecated)
TLS 1.1    offered (deprecated)
TLS 1.2    offered (OK)
TLS 1.3    offered (OK): final
NPN/SPDY   not offered
ALPN/HTTP2 h2, grpc-exp (offered)

Testing cipher categories

NULL ciphers (no encryption)                not offered (OK)
Anonymous NULL Ciphers (no authentication)  not offered (OK)
Export ciphers (w/o ADH+NULL)               not offered (OK)
LOW: 64 Bit + DES, RC[2,4] (w/o export)     not offered (OK)
Triple DES Ciphers / IDEA                   offered
Obsolete CBC ciphers (AES, ARIA etc.)       offered
Strong encryption (AEAD ciphers)            offered (OK)


Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA ECDHE-RSA-CHACHA20-POLY1305 TLS_AES_128_GCM_SHA256
                             ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA
Elliptic curves offered:     prime256v1 X25519
```

Figure 6: Multiple TLS issues affecting generator-shares.social.production.haqq.network

```
Start 2023-09-04 10:50:09        -->> 34.160.229.79:443 (social-share-1.social.production.haqq.network) <<--

rDNS (34.160.229.79):   79.229.160.34.bc.googleusercontent.com.
Service detected:       HTTP


Testing protocols via sockets except NPN+ALPN

SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      offered (deprecated)
TLS 1.1    offered (deprecated)
TLS 1.2    offered (OK)
TLS 1.3    offered (OK): final
NPN/SPDY   not offered
ALPN/HTTP2 h2, http/1.1, grpc-exp (offered)

Testing cipher categories

NULL ciphers (no encryption)                not offered (OK)
Anonymous NULL Ciphers (no authentication)  not offered (OK)
Export ciphers (w/o ADH+NULL)               not offered (OK)
LOW: 64 Bit + DES, RC[2,4] (w/o export)     not offered (OK)
Triple DES Ciphers / IDEA                   offered
Obsolete CBC ciphers (AES, ARIA etc.)       offered
Strong encryption (AEAD ciphers)            offered (OK)


Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA ECDHE-RSA-CHACHA20-POLY1305 TLS_AES_128_GCM_SHA256
                             ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA
Elliptic curves offered:     prime256v1 X25519


Testing server preferences
```

Figure 7: Multiple TLS issues affecting social-share-1.social.production.haqq.network

```
Start 2023-09-04 10:51:05        -->> 34.160.229.79:443 (metadata.social.production.haqq.network) <<--

rDNS (34.160.229.79):   79.229.160.34.bc.googleusercontent.com.
Service detected:       HTTP


 Testing protocols via sockets except NPN+ALPN

 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      offered (deprecated)
 TLS 1.1    offered (deprecated)
 TLS 1.2    offered (OK)
 TLS 1.3    offered (OK): final
 NPN/SPDY   not offered
 ALPN/HTTP2 h2, http/1.1, grpc-exp (offered)

 Testing cipher categories

 NULL ciphers (no encryption)                /usr/bin/testssl: connect: Connection refused
/usr/bin/testssl: line 10454: /dev/tcp/34.160.229.79/443: Connection refused
 Oops: TCP connect problem

Unable to open a socket to 34.160.229.79:443. not offered (OK)
 Anonymous NULL Ciphers (no authentication)    not offered (OK)
 Export ciphers (w/o ADH+NULL)                 not offered (OK)
 LOW: 64 Bit + DES, RC[2,4] (w/o export)       not offered (OK)
 Triple DES Ciphers / IDEA                     offered
 Obsolete CBC ciphers (AES, ARIA etc.)         offered
 Strong encryption (AEAD ciphers)              offered (OK)


 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

 PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA ECDHE-RSA-CHACHA20-POLY1305 TLS_AES_128_GCM_SHA256
                              ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA
 Elliptic curves offered:     prime256v1 X25519
```

Figure 8: Multiple TLS issues affecting metadata.social.production.haqq.network

CVSS Vector:

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:L/A:N

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

As the application functionality was hosted across multiple servers, the TLS configuration of each should be reviewed and updated to comply with security best practice and to minimize the risk of existing and future TLS vulnerabilities.

- Support for TLS 1.0 and TLS 1.1 should be disabled.
- Support for the weak cipher suites identified above should be removed.
- Key lengths of 256 bits and above should be used for symmetric encryption algorithms and a length of 4096 bits should be used for

RSA algorithms. For asymmetric keys generated with ECC algorithms, the minimum recommended key size is 512 bits.

References Lucky13 Vulnerability

Remediation Plan:

SOLVED: Haqq team solved the issue by disabling the support for outdated TLS version.

3.5 (HAL-05) CACHEABLE HTTPS RESPONSE - MEDIUM

Description:

During the assessment, it has been identified that the application did not define any cache directive and due to this security misconfiguration vulnerability HTTPS responses can be stored in intermediary caches, such as Content Delivery Network (CDN) caches, browser caches, or proxy caches. If sensitive information or responses are cached, unauthorized users may access them, leading to potential data leaks.

Proof of Concept:

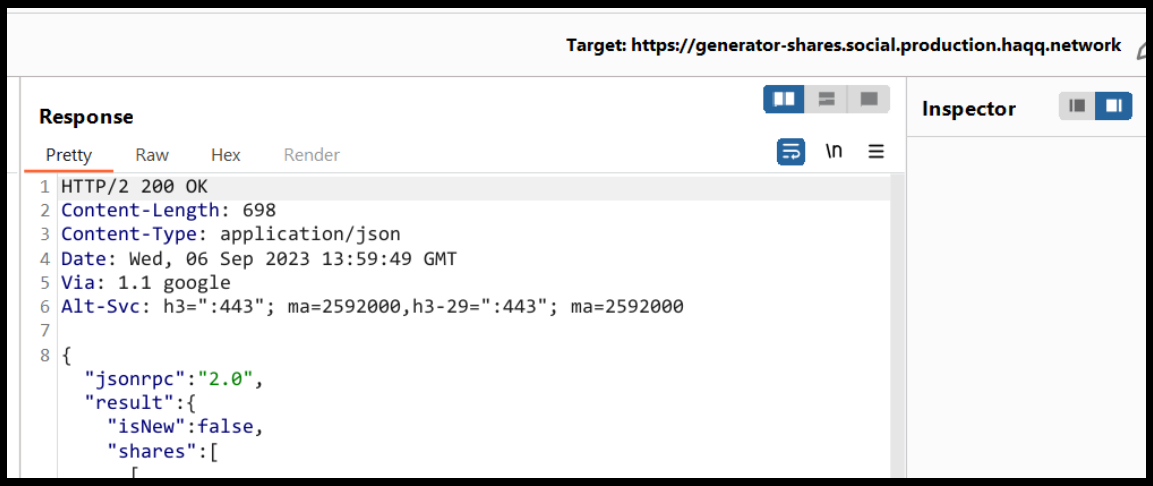

Figure 9: No caching directives found at https://generator-shares.social.production.haqq.network/

Figure 10: No caching directives found at https://social-share-1.social.production.haqq.network/

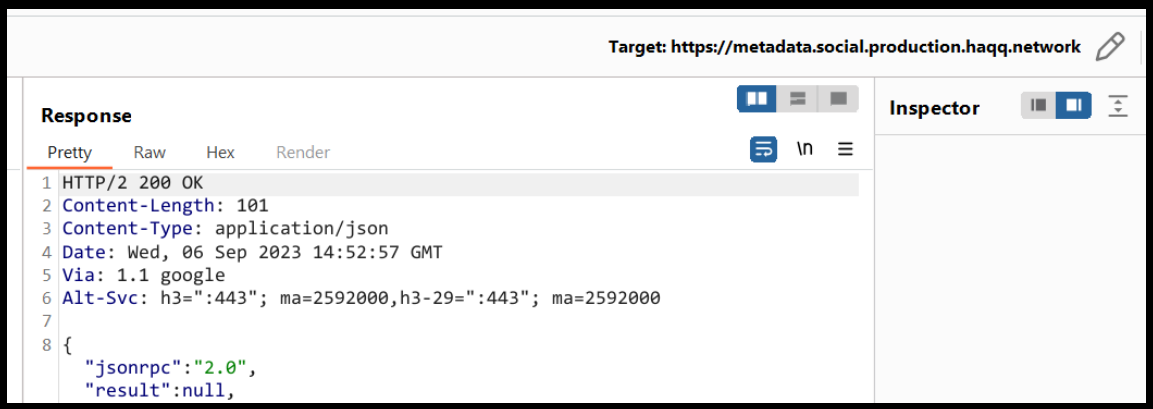

Figure 11: No caching directives found at https://metadata.social.production.haqq.network/

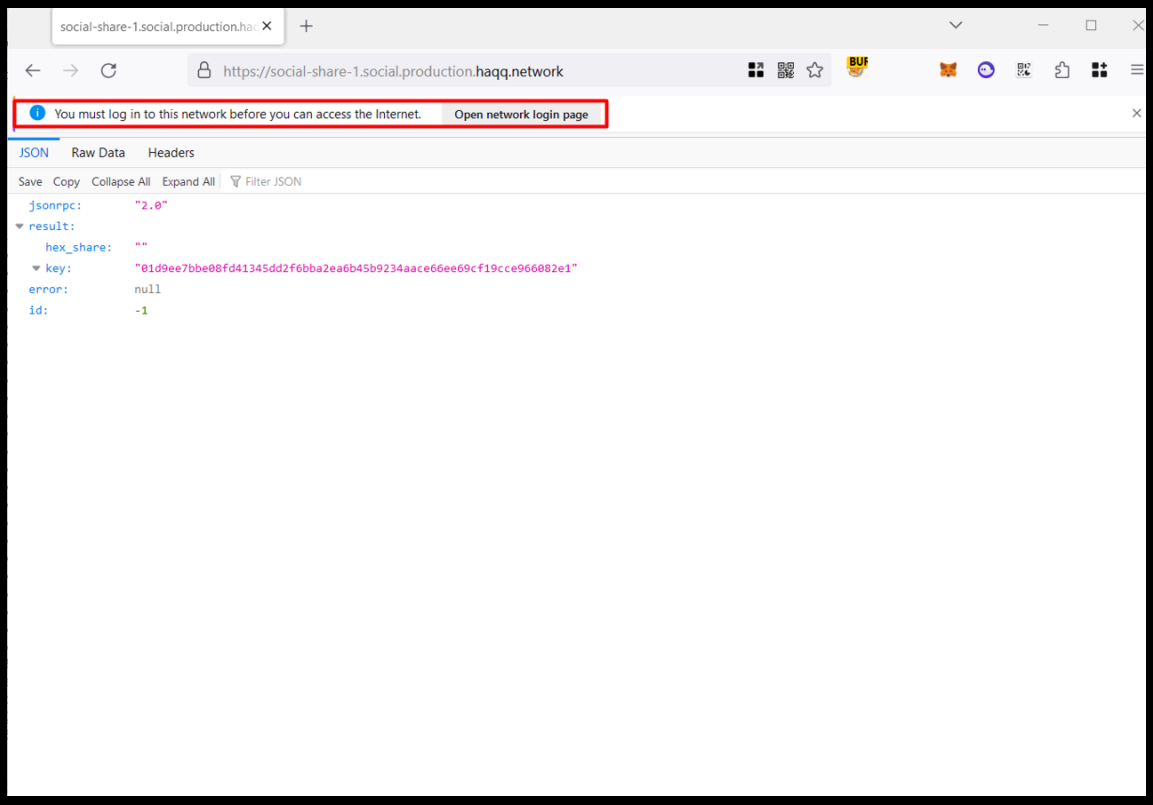

Figure 12: Browser stores the response, contains key share values in its cache

Risk Level:

Likelihood - 2
Impact - 4

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Recommendation:

Applications should return caching directives instructing browsers not to store local copies of any sensitive data. Often, this can be achieved by configuring the web server to prevent caching for relevant paths within the web root. Alternatively, most web development platforms allow you to control the server's caching directives from within individual scripts.

Ideally, the web server should return the following HTTP headers in all responses containing sensitive content:

Listing 8

```
1 Cache-control: no-store
2 Pragma: no-cache
```

Reference:

MDN Web Docs: Cache-Control

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing the appropriate cache headers against the services.

NOTE: Retest was conducted on the following commit:
- 15e71e0eb5467b4f9881a95404eb46ad2fa98582

3.6 (HAL-06) MISSING SECURITY HEADERS - LOW

Description:

Important security HTTP Headers were missing from the API services. These headers used by the client browser improve the security of end users against common attacks.

Missing important security headers;
Strict-Transport-Security and **Content-Security-Policy** response headers.

- Content-Security-Policy is an effective measure to protect your site from XSS attacks. By whitelisting sources of approved content, it is possible to prevent the browser from loading malicious assets.
- Strict-Transport-Security HTTP Strict Transport Security is an excellent feature to support on your site and strengthens your implementation of TLS by getting the User Agent to enforce the use of HTTPS. Recommended value “Strict-Transport-Security: max-age=31536000; includeSubDomains”.

Proof of Concept:

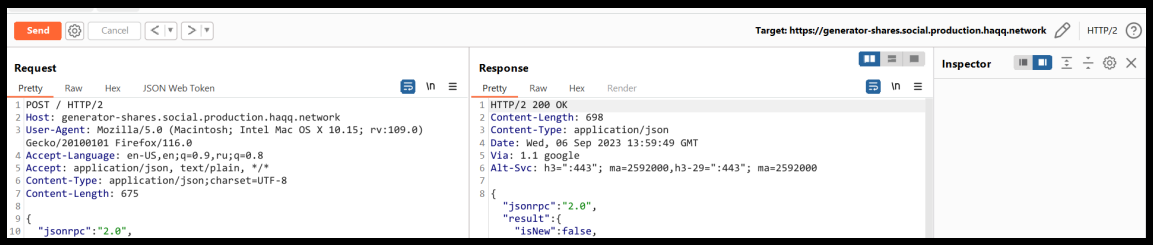

Figure 13: Missing security headers on https://generator-shares.social.production.haqq.network/

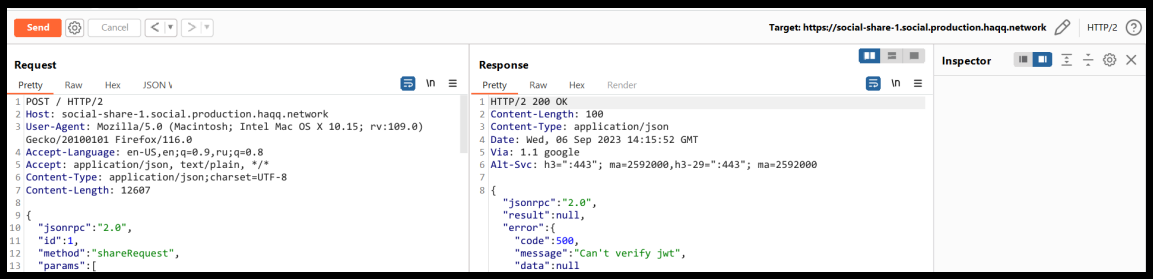

Figure 14: Missing security headers on https://social-share-1.social.production.haqq.network/

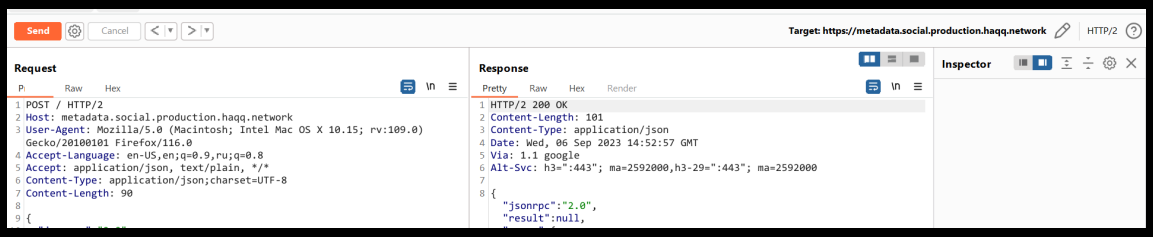

Figure 15: Missing security headers on https://metadata.social.production.haqq.network/

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to define these headers in API services; Content-Security-Policy and Strict-Transport-Security response headers with appropriate policies.

Reference:

Content Security Policy
Strict-Transport-Security

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing the security headers with appropriate policies.

NOTE: Retest was conducted on the following commit:
- 15e71e0eb5467b4f9881a95404eb46ad2fa98582

3.7 (HAL-07) REFLECTED UNSANITIZED INPUT - LOW

Description:

It was identified user's inputs reflected in the response of API request in id parameter. It is worth noting that these values did not cause any XSS type injection attack currently on the UI. This, however, does not mean that the response is not displayed, or will not be displayed in the future, on other web pages, making it a potential **Cross Site Scripting** attack vector.

Screenshot:

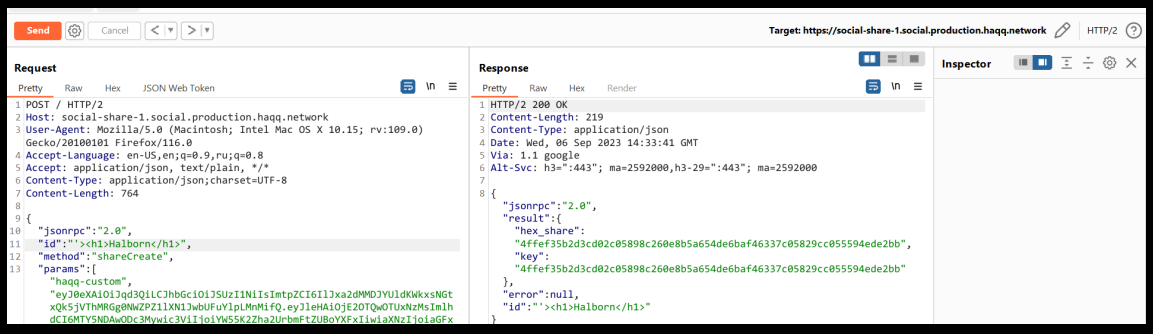

Figure 16: Reflected unsanitized input on API response in id parameter

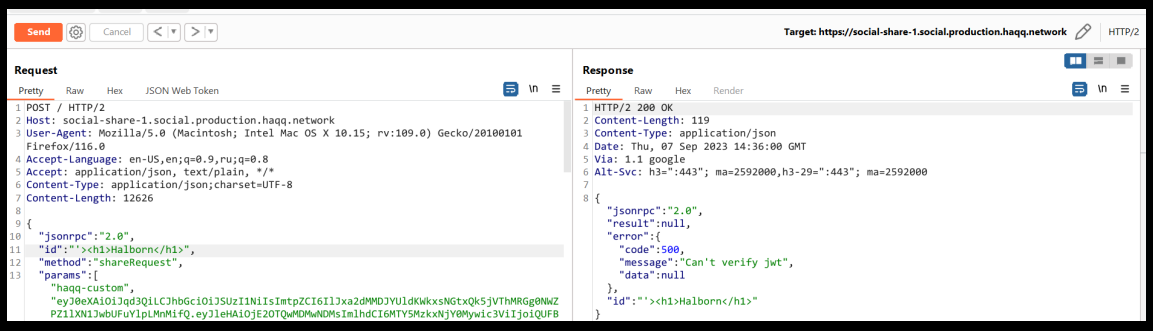

Figure 17: Reflected unsanitized input on API response in id parameter

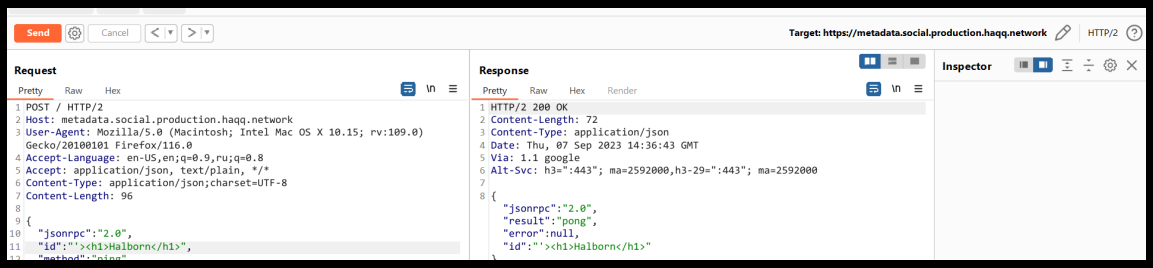

Figure 18: Reflected unsanitized input on API response in id parameter

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to sanitize and validate all user input on the API level as well. One commonly used technique is HTML entity encoding, which converts special characters into their corresponding HTML entities. This approach ensures that user input is safely handled by the server and can be safely displayed in the browser without being interpreted as HTML or script code. Additionally, implementing Content Security Policy (CSP) headers can further restrict potential malicious scripts from being executed.

Reference:

Cross Site Scripting Prevention Cheat Sheet

Content Security Policy

Remediation Plan:

SOLVED: The Haqq team solved the issue as per the recommendations.

NOTE: Retest was conducted on the following commit:
- 964dcf60eb894f352ca98fbfa5824b472372261a

3.8 (HAL-08) VULNERABLE THIRD PARTY DEPENDENCIES - LOW

Description:

The scoped repository uses multiple third-party dependencies. Using vulnerable third-party libraries can result in security vulnerabilities in the project that can be exploited by attackers. This can result in data breaches, theft of sensitive information, and other security issues. However, some of them were affected by public-known vulnerabilities that may pose a risk to the global application security level.

Proof of Concept:

```
Crate:     borsh
Version:   0.10.3
Warning:   unsound
Title:     Parsing borsh messages with ZST which are not-copy/clone is unsound
Date:      2023-04-12
ID:        RUSTSEC-2023-0033
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0033
Dependency tree:
borsh 0.10.3
└── rust_decimal 1.32.0
    ├── sqlx-core 0.6.3
    │   ├── sqlx-macros 0.6.3
    │   │   └── sqlx 0.6.3
    │   │       ├── sea-query-binder 0.3.1
    │   │       │   └── sea-orm 0.11.3
    │   │       │       ├── storage-postgres 0.1.0
    │   │       │       └── sea-orm-migration 0.11.3
    │   │       │           └── migration 0.1.0
    │   │       │               └── storage-postgres 0.1.0
    │   │       └── sea-orm 0.11.3
    │   └── sqlx 0.6.3
    ├── sea-query-binder 0.3.1
    ├── sea-query 0.28.5
    │   ├── sea-schema 0.11.0
    │   │   ├── sea-orm-migration 0.11.3
    │   │   └── sea-orm-cli 0.11.3
    │   │       └── sea-orm-migration 0.11.3
    │   ├── sea-query-binder 0.3.1
    │   └── sea-orm 0.11.3
    └── sea-orm 0.11.3

Crate:     ouroboros
Version:   0.15.6
Warning:   unsound
Title:     Ouroboros is Unsound
Date:      2023-06-11
ID:        RUSTSEC-2023-0042
```

Figure 19: Multiple vulnerable dependencies reported in repository (cargo)

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is strongly recommended to update all affected packages to its latest version. It is also recommended to perform an automated analysis of the dependencies from the birth of the project and if they contain any security issues. Developers should be aware of this and apply any necessary mitigation measures to protect the affected application.

References:

OWASP. Vulnerable and Outdated Components
OWASP. Vulnerable Dependency Management Cheat Sheet

Remediation Plan:

RISK ACCEPTED: The Haqq team accepted the risk of this finding.

3.9 (HAL-09) INTERNAL SERVER ERROR ON INPUT HANDLING - LOW

Description:

During the assessment, we identified that due to lack of input validation and error handling some input parameters lead to an **“Internal Server Error”**. This error suggests that the server encountered an unexpected condition which prevented it from fulfilling the request. This error gives an attacker insight of the backend or could be exploited to cause further issues. Such occurrences can signify potential vulnerabilities, misconfigurations, or other underlying issues that malicious actors might exploit. Beyond the immediate service disruption, this lack of error handling could lead to various security risks, from information disclosure to potential vectors for more sophisticated attacks.

Proof of Concept:

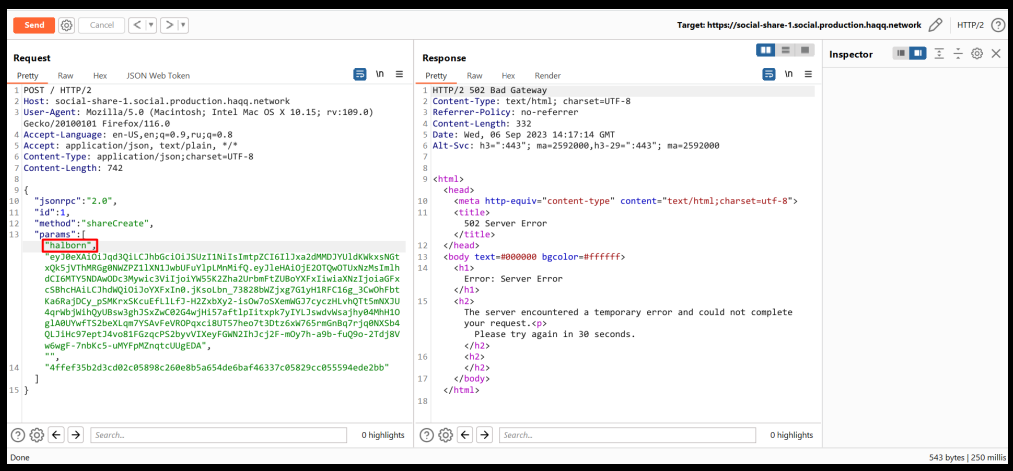

Figure 20: Internal server error in application response

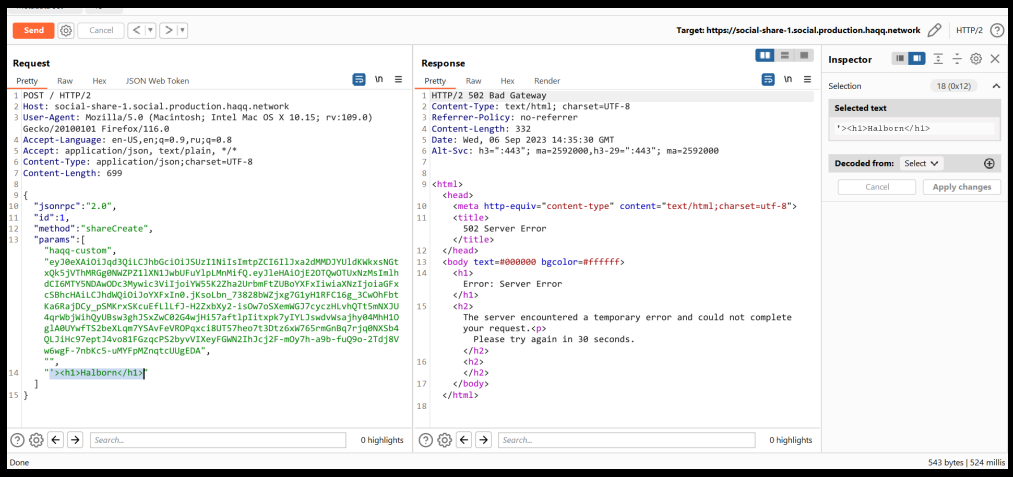

Figure 21: Internal server error in application response

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to implement robust error handling. Return user-friendly messages for errors and log technical details server-side, also ensuring no sensitive information leaks. It is also recommended to enhance input validation mechanisms to reject unexpected or malicious values. Ensure that specific parameter inputs are strictly validated.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing the appropriate checks.

NOTE: Retest was conducted on the following commit:
- 15e71e0eb5467b4f9881a95404eb46ad2fa98582

ANNEX

4.1 Web App Security Testing Methodology

Planning:

1. Gather Scoping Information

After starting the project, scope/target information is collected from the client. In the case of web apps penetration testing, this information includes IP addresses and URLs, a definition files or documentation such as swagger or postman, the source code (if possible), authentication credentials, and/or API tokens (2 sets of credentials for each feature being tested) and a list of any sensitive or restricted endpoints/functionalities that should not be scanned or exploited.

Halborn and the client jointly review and acknowledge the penetration testing rules of engagement, confirm project scope and testing timeline, identify specific testing objectives, document any testing limitations or restrictions, and answer any questions related to the project.

Execution:

1. Information gathering

The information-gathering phase consists of gathering information about the app, including its purpose, functionality, documentation, endpoints, and authentication mechanisms. Understand the app's intended use and potential attack vectors.

It includes application footprinting, metafile leakage review, listing services, operating system functions, and application fingerprinting. This step maps the in-scope application to prepare for identifying exploitable vulnerabilities collectively.

During the Information Gathering phase, Halborn does:

- Use discovery tools to uncover information about the application passively.

- Identify entry points into the application, such as administration portals or backdoors.
- Perform application fingerprinting to identify the underlying development language and components.
- Send fuzzing requests for analysis of error codes that may disclose sensitive data that could be used to launch a more targeted cyberattack.
- Actively scan for open services and develop a test plan for the latter phases of the vulnerability assessment.

If the source code was provided, all this discovery could be done in a more effective way by reading it.

2. Threat Modeling

The threat modeling phase serves to evaluate the types of threats that may affect the target apps that are in scope. The types of attacks and likelihood of these threats materializing serve to inform risk rankings/priorities that are assigned to vulnerabilities throughout the assessment.

The perspective of the testing (external/internal, authenticated/unauthenticated, black box/crystal box, etc.) is also identified to ensure the validity of vulnerabilities discovered. This phase of the assessment also includes manual review of the exposed endpoints, determining business functionality of the endpoints, and identifying unauthenticated/authenticated endpoint attack surface.

With the information collected from the previous step, the testing process transitions to identifying security vulnerabilities in the app.

During this phase, Halborn does:

- Use open-source, commercial, and internally developed tools to identify and confirm well-known vulnerabilities.
- Swagger/Postman documents in-scope to effectively build a map of each feature, component, and area of interest.

- Send fuzzing requests to analyze error codes that may disclose valuable information that can launch a more targeted attack.
- Build the application's threat model using the information gathered in this and the previous phase to be used as a plan of attack for later stages of the penetration test.

If the source code was provided, every unexpected behavior detected will be checked on it to find out the exact reason and discover if it is exploitable in any way.

3. Vulnerability Analysis

The vulnerability analysis phase will encompass the enumeration of all in-scope targets/applications at both the network layer and the application layer. The phase involves documenting and analyzing vulnerabilities discovered due to Information Gathering and Threat Modeling. This step includes the analysis of output from the various security tools and manual testing techniques.

In the Vulnerability Analysis phase, Halborn does:

- Compile the list of areas of interest and develop a plan for exploitation.
- Search and gather known exploits from various sources.
- Analyze the impact and likelihood of each potentially exploitable vulnerability.
- Select the best methods and tools for adequately exploiting each of the suspected security vulnerabilities.

In this phase, Halborn performs an in-depth security source code review with 2 stages:

- Automatic review: The source code is verified using specialized tools/software to scan and examine the code for common security vulnerabilities and insecure coding practices. These tools employ

various techniques, such as pattern matching, data flow analysis, and control flow analysis, to identify potential security risks. The tools used will vary depending on the language of the source code. Additionally, these tools may not detect all types of security vulnerabilities, and manual security assessments and penetration testing are still essential for a comprehensive security evaluation of an application.

- Manual review: The source code is reviewed by Halborn security experts to identify security vulnerabilities and weaknesses. Unlike automatic security source code reviews, which rely on automated tools, a manual review involves a careful and in-depth examination of the code by experienced security professionals. In this case, Halborn security professionals will perform a thorough examination of the code and identify complex vulnerabilities that may not be easily detected by automated tools.

4. Exploitation

This phase involves taking all potential vulnerabilities identified in the previous phases of the assessment and attempting to exploit them as an attacker would. This helps to evaluate the realistic risk level associated with the successful exploitation of the vulnerability, analyze the possibility of exploit/attack chains, and account for any mitigating controls that may be in place.

Exploitation involves establishing access to the application or connected components by bypassing security controls and exploiting vulnerabilities to determine their real-world risk through ethical hacking. Throughout this step, several manual tests will be performed simulating real-world exploits incapable of being achieved through automated means. During a Halborn penetration test, the exploitation phase involves heavy manual testing tactics and is often the most time-intensive.

This includes business logic flaws, authentication/authorization bypasses, direct object references, injection-style attacks (SQL, command, XPath, LDAP, XXE, XSS), error analysis, file uploads, parameter tampering, and session management etc.

The exploitation part includes:

- Authorization and Authentication Testing: Test the app's authentication mechanisms, such as API keys, tokens, OAuth, or other access control methods. Verify if proper authentication is enforced and that it cannot be easily bypassed or abused.
- Input Validation: Test for input validation vulnerabilities, such as injection attacks (SQL, command, or XML), cross-site scripting (XSS), Path, RCE, and cross-site request forgery (CSRF) etc. Send malicious input to the app endpoints to identify any security weaknesses.
- Error Handling: Check how the app handles various error conditions, such as invalid input, server-side errors, or exceptions. Determine if error messages reveal sensitive information that could aid attackers.
- Data Integrity and Confidentiality: Assess the app's ability to protect data integrity and confidentiality during transmission and storage. Evaluate the implementation of encryption, secure transport protocols (e.g., HTTPS), and secure storage practices.
- Session Management: Test how the app manages sessions, tokens, and state information. Identify any session-related vulnerabilities that could lead to unauthorized access or privilege escalation.
- Business Logic Testing: Analyze the app's business logic to identify vulnerabilities or flaws in the design and implementation. Test for authorization bypass, bypassing custom workflows, insecure direct object references, and other logical vulnerabilities.
- Rate Limiting and Throttling: Evaluate if the app has proper rate limiting and throttling mechanisms in place to prevent abuse and mitigate denial-of-service (DoS) attacks.
- Source code review: Potential vulnerabilities found in both automatic and manual source code review will be reviewed and exploited.

- Third-Party Integration: Assess the security of any third-party libraries, frameworks, or services used in the app implementation. Review their security controls and potential vulnerabilities.

In the Exploitation phase, Halborn does conduct:

- Attempt to manually exploit the vulnerabilities identified in the previous steps to determine the possible level of risk and level of exploitation.
- Capture and log evidence to provide proof of exploitation (images, screenshots, configs, etc.).
- Notify the client of findings.

5. Post Exploitation

After successful exploitation, analysis may continue, including infrastructure analysis, pivoting, sensitive data identification, data exfiltration, and identification of high-value targets/data. The information collected here is used in the prioritization and criticality ranking of identified vulnerabilities.

Furthermore, chaining different vulnerabilities can lead to proof-of-concept vulnerabilities with higher critically.

Post-Execution:

1. Reporting

After completing all the phases, Halborn formally documents the findings. The output provided generally includes an executive-level report and a technical findings report. The executive-level report is written for management consumption and includes a high-level overview of assessment activities, scope, most critical/thematic issues discovered, overall risk scoring, organizational security strengths, and applicable screenshots.

The technical findings report, on the other hand, includes all vulnerabilities listed individually, with details as to how to recreate

the issue, understand the risk, recommended remediation actions, and helpful reference links. Both include actionable recommendations for improving security.

2. Quality Assurance

All assessments go through a rigorous technical and editorial quality assurance phase. This may also include follow-ups with the client to confirm or deny details, as appropriate.

3. Presentation

The final activity in any assessment is a presentation of all documentation to the client. Halborn walks the client through the information provided, makes any updates needed, and addresses questions regarding the assessment output. Following this activity, new revisions of documentation and schedule any formal retesting are provided, if applicable.

THANK YOU FOR CHOOSING

// HALBORN