QuillAudits

Audit Report
July, 2023

For

Dregn

Table of Content

Executive Summary

Project Name	Dregn
Overview	Dregn's mission is to significantly improve the art & science of IT networking through ambitious innovations. Dregn vision is to boost network tools and enhance the value & future of work for network personnel with an extensive network tools SaaS suite, an AI roadmap, a blockchain utility token ecosystem and a revolutionary 3d game-like real time interface for better visibility, analysis and control. Dregn's platform is empowered by leveraging the DREGN token and the plans for functions of the platform in the coming years are just industry-changing.
Timeline	20 July 2023 - 27 July 2023
Scope of Audit	The scope of this pentest was to analyze the Web App and API calls for quality, security, and correctness. **In Scope:** *https://qa.dregn.org* *https://qa-api.dregn.org*

7
Issues Found

🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	**2**	**3**	**2**	0

Techniques and Methods

Throughout the pentest of BIT-Wallet, care was taken to ensure:

- Information gathering – Using OSINT tools information concerning the web architecture, information leakage, web service integration, and gathering other associated information related to web server & web services.
- Using Automated tools approach for Pentest like Nessus, Wireshark, etc.
- Platform testing and configuration
- Error handling and data validation testing
- Encryption-related protection testing
- Client-side and business logic testing

Tools and Platforms used for Pentest

- Burp Suite
- DNSenum
- Dirbuster
- SQLMap
- Acunetix
- Neucli
- Nabbu
- Turbo Intruder
- Nmap
- Metasploit
- Horusec
- Postman
- Netcat
- Nessus and many more.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Checked Vulnerabilities

- ✓ Improper Authentication
- ✓ Improper Resource Usage
- ✓ Improper Authorization
- ✓ Insecure File Uploads
- ✓ Insecure Direct Object References
- ✓ Client-Side Validation Issues
- ✓ Rate Limit
- ✓ Input Validation
- ✓ Injection Attacks
- ✓ Cross-Site Scripting (XSS)
- ✓ Cross-Site Request Forgery
- ✓ Security Misconfiguration
- ✓ Broken Access Controls
- ✓ Insecure Cryptographic Storage
- ✓ Insufficient Cryptography
- ✓ Insufficient Session Expiration
- ✓ Insufficient Transport Layer Protection
- ✓ Unvalidated Redirects and Forwards
- ✓ Information Leakage
- ✓ Broken Authentication and Session Management
- ✓ Denial of Service (DoS) Attacks
- ✓ Malware
- ✓ Third-Party Components

Manual Testing

High Severity Issues

1. Weak Authorization

Description

A weak authorization issue can occur when an authorization token is not properly protected. In this case, the authorization token is a wallet address. A wallet address is a unique identifier that is used to store cryptocurrency. Anyone who knows the wallet address can access the cryptocurrency that is stored in it.
In this case, the authorization token is being used to authenticate users to a web application. This means that anyone who knows the wallet address can authenticate themselves as any user. This could allow an attacker to gain unauthorized access to the web application and its resources.

Vulnerable Endpoint: Authorization Header in qa-api.dregn.org or Dregen-backend

Steps to Reproduce

1. Visit the website and try to access transaction orders of user
2. You will see a request sent to api with the authorization header in the qa-api.dregn.org hostname.
3. Replace it to any valid wallet address and it will still show the output.

Recommendation

To mitigate this risk, the web application should use a more secure authorization token, such as a JWT token. JWT tokens are signed and encrypted, which makes them more difficult to forge or tamper with.

Here are some additional recommendations for mitigating this risk:
Use a strong encryption algorithm to sign and encrypt the JWT token.
Use a unique secret key for each JWT token.
Store the secret keys in a secure location.
Expire JWT tokens after a certain period of time.
Implement a token rotation mechanism.

Status

Resolved

2. PII Information Leaking (Email Address)

Description

Admin API endpoints are usually meant for only to be accessed by admin privileged users, as the information it contains can be critical to users and needs to be kept private for internal use only. Such as who all user's are KYC'd or which user has what amount of ico token and all such details.
Qa-api Subdomain allows us to fetch details even from admin endpoints to an unauthenticated user without checking for proper headers.

Vulnerable Endpoint:
https://qa-api.dregn.org/admin/ico/*
https://qa-api.dregn.org/admin/ico/getRecords
https://qa-api.dregn.org/admin/ico/getUserManagementList
https://qa-api.dregn.org/admin/ico/getReferralCommission
And more such endpoints.

Recommendation

1. To mitigate this risk, the api subdomain should only allow authenticated high privileged users to access such information of the whole system.
2. The Auth header needs to be checked
   IP Whitelisting can be done for some critical endpoints to only be accessed by certain
3. individuals.

Status

Resolved

Medium Severity Issues

3. No Rate Limit on Contact Us and Subscribe

Description

The Contact Us form and subscribe on the website does not have a rate limit. This means that anyone can send unlimited requests to the form in a short amount of time. This can be used to jam the inbox of the company and hide the important requests from legit customers. Rate limits are important in order to keep servers from jamming and letting spammers be away and keep the data clean.

Vulnerable Endpoint

1. https://qa.dregn.org/contact-us
2. https://qa-api.dregn.org/admin/mail/sendmail

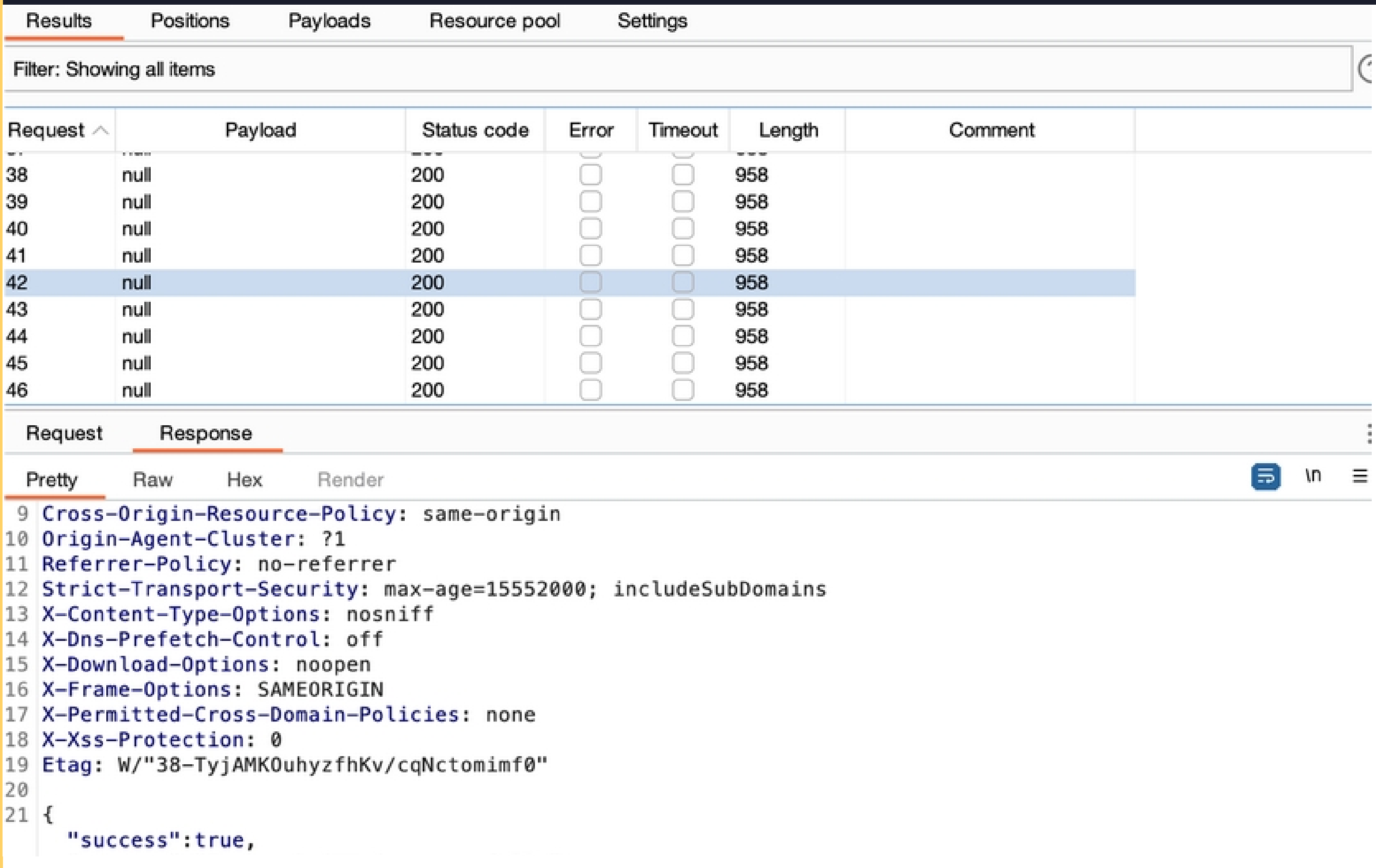
Steps to Reproduce

1. Visit the https://qa.dregn.org/contact-us url
2. Fill the form and Intercept the request in Burp
3. Send the request to Intruder and starrt an attack for 1000 empty payload sniper attack
4. All the requests sent will have a 200 OK response.

Recommendation

The company can mitigate this vulnerability by adding a rate limit to the contact us form. This will limit the number of requests that can be sent to the form in a short amount of time. The company can also implement other security measures, such as CAPTCHAs and email verification, to prevent spam and abuse.

POC


Results | Positions | Payloads | Resource pool | Settings

Filter: Showing all items

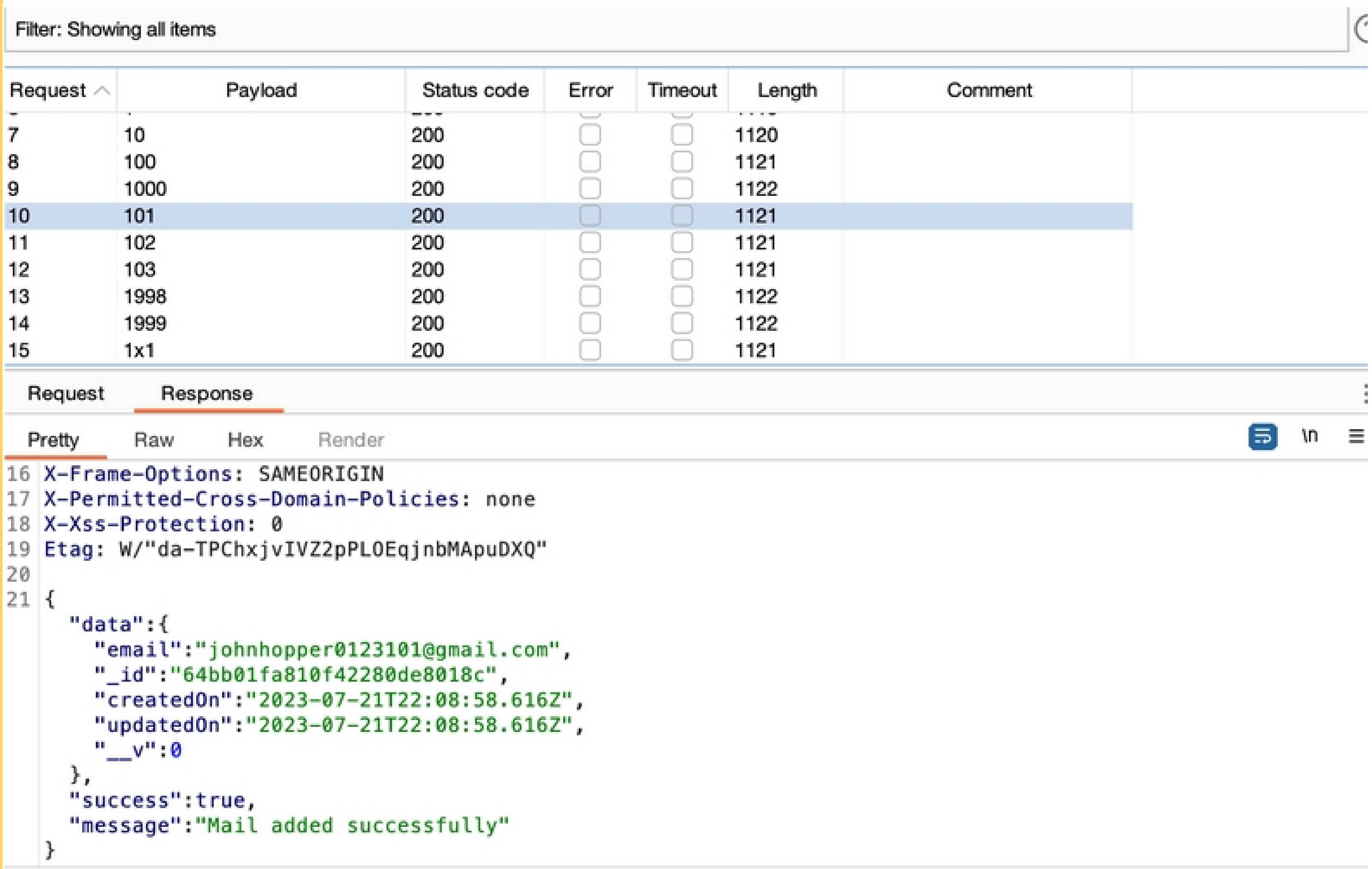
Request	Payload	Status code	Error	Timeout	Length	Comment
38	null	200	☐	☐	958	
39	null	200	☐	☐	958	
40	null	200	☐	☐	958	
41	null	200	☐	☐	958	
42	null	200	☐	☐	958	
43	null	200	☐	☐	958	
44	null	200	☐	☐	958	
45	null	200	☐	☐	958	
46	null	200	☐	☐	958	

Request | Response

Pretty | Raw | Hex | Render

```
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"38-TyjAMKOuhyzfhKv/cqNctomimf0"

{
  "success":true,
```


Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
7	10	200	☐	☐	1120	
8	100	200	☐	☐	1121	
9	1000	200	☐	☐	1122	
10	101	200	☐	☐	1121	
11	102	200	☐	☐	1121	
12	103	200	☐	☐	1121	
13	1998	200	☐	☐	1122	
14	1999	200	☐	☐	1122	
15	1x1	200	☐	☐	1121	

Request | Response

Pretty | Raw | Hex | Render

```
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"da-TPChxjvIVZ2pPLOEqjnbMApuDXQ"

{
  "data":{
    "email":"johnhopper0123101@gmail.com",
    "_id":"64bb01fa810f42280de8018c",
    "createdOn":"2023-07-21T22:08:58.616Z",
    "updatedOn":"2023-07-21T22:08:58.616Z",
    "__v":0
  },
  "success":true,
  "message":"Mail added successfully"
}
```

Status

Resolved

4. Indirect Object Reference in Transaction Order and Referral

Description

The Transaction Details and Referral function allows you to view the transaction history of any user. However, the userAddress parameter is not properly sanitized, which allows attackers to specify any user's wallet address. This can be used to view the transaction history of any user. Referral shows you how much commission you earned and such details.
Here when tampered the userAddress parameter to any valid user's address you can see their details and this can cause issues regarding Confidentiality.

Vulnerable Endpoint

1. https://qa-api.dregn.org/admin/ico/getTxnDetails
2. https://qa-api.dregn.org/user/ico/getRefersTo
   Parameter- userAddress,referralAddress

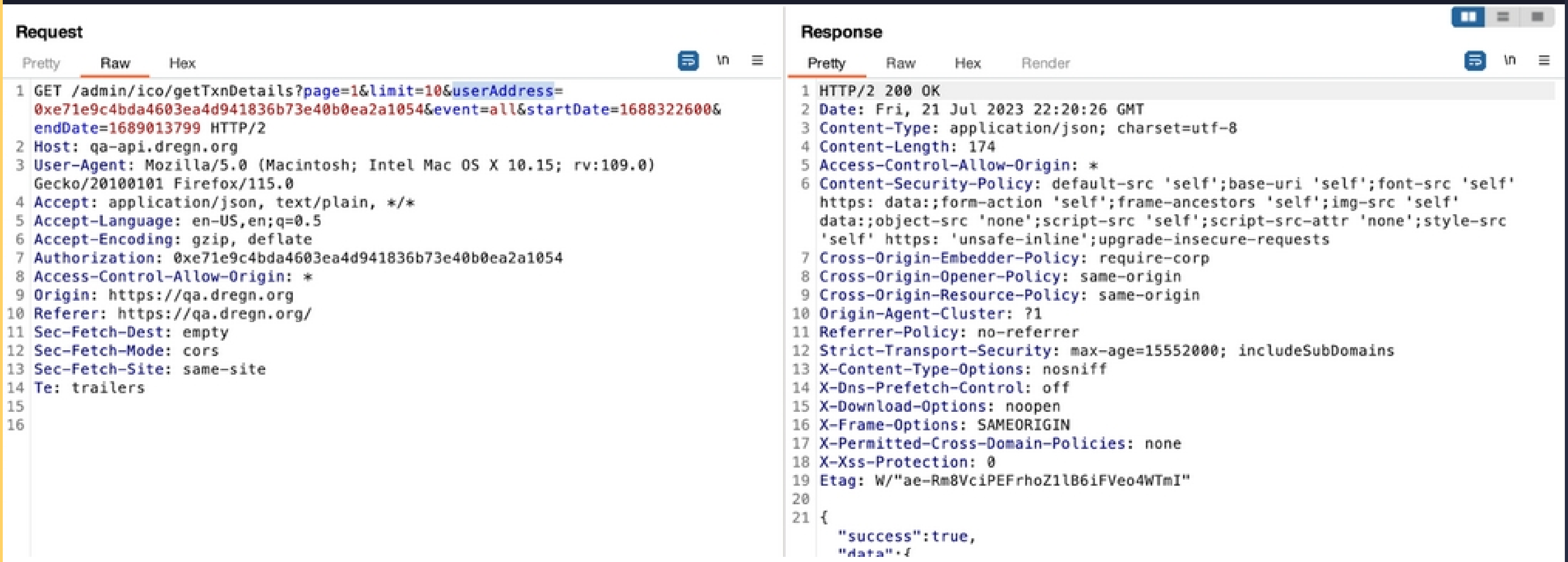
Steps to Reproduce

1. Visit the transaction Details tab in *https://qa.dregn.org/transaction-detail*
2. Fill out details and Capture the request in BurpSuite
3. Send the request to Repeater
4. Test out other user's wallet address in userAddress Parameter.
5. Change the same address in Authorization header too.

Recommendation

1. Using proper authentication mechanism so that only valid user can access only their details ad not of other users
2. Encrypt the parameters with a salt so no one can actually tamper with the parameters

POC



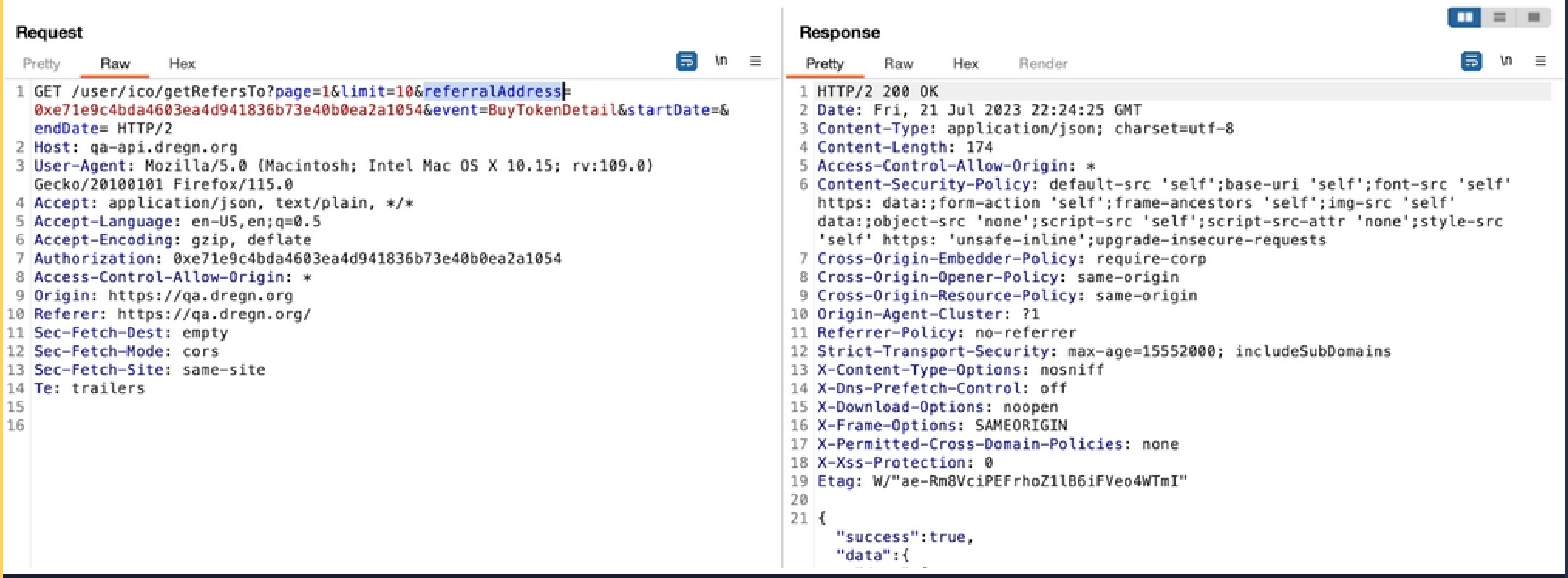
Request

```
GET /admin/ico/getTxnDetails?page=1&limit=10&userAddress=
0xe71e9c4bda4603ea4d941836b73e40b0ea2a1054&event=all&startDate=1688322600&
endDate=1689013799 HTTP/2
Host: qa-api.dregn.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: 0xe71e9c4bda4603ea4d941836b73e40b0ea2a1054
Access-Control-Allow-Origin: *
Origin: https://qa.dregn.org
Referer: https://qa.dregn.org/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
```

Response

```
HTTP/2 200 OK
Date: Fri, 21 Jul 2023 22:20:26 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 174
Access-Control-Allow-Origin: *
Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self'
https: data:;form-action 'self';frame-ancestors 'self';img-src 'self'
data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src
'self' https: 'unsafe-inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"ae-Rm8VciPEFrhoZ1lB6iFVeo4WTmI"

{
  "success":true,
  "data":{
```



Request

```
GET /user/ico/getRefersTo?page=1&limit=10&referralAddress=
0xe71e9c4bda4603ea4d941836b73e40b0ea2a1054&event=BuyTokenDetail&startDate=&
endDate= HTTP/2
Host: qa-api.dregn.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: 0xe71e9c4bda4603ea4d941836b73e40b0ea2a1054
Access-Control-Allow-Origin: *
Origin: https://qa.dregn.org
Referer: https://qa.dregn.org/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
```

Response

```
HTTP/2 200 OK
Date: Fri, 21 Jul 2023 22:24:25 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 174
Access-Control-Allow-Origin: *
Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self'
https: data:;form-action 'self';frame-ancestors 'self';img-src 'self'
data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src
'self' https: 'unsafe-inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Etag: W/"ae-Rm8VciPEFrhoZ1lB6iFVeo4WTmI"

{
  "success":true,
  "data":{
```

Status

Resolved

5. API keys Leaked in JS file

Description

The JS files were found to contain sensitive information, including API tokens, access keys, and other credentials that should not be exposed publicly. The presence of these keys in client-side scripts poses a significant risk, as malicious actors can exploit them to gain unauthorized access to sensitive data, impersonate legitimate users, or perform other malicious activities.

Vulnerable Endpoint

1. https://qa.dregn.org/static/js/main.db9956c4.js

Risk Impact

- **Unauthorized Access:** Malicious actors can utilize the leaked keys to bypass security controls and gain unauthorized access to sensitive APIs or backend systems.
- **Data Breach:** The leaked keys could grant unauthorized access to databases or cloud storage, leading to potential data breaches and exposure of sensitive information.
- **Financial Loss:** If the keys are associated with billing systems or payment gateways, unauthorized access may result in financial losses or fraudulent transactions.

Recommendation

1. Secure Storage for Sensitive Information
2. Implement Server-Side Authentication and Authorization
3. Use Environment Variables
4. Monitor and Rotate Keys

Status

Resolved

Low Severity Issues

6. Multiple Deprecated Libraries in package-lock.json

Description

package-lock Stores files that can be useful for the dependency of the application. This is used for locking the dependency with the installed version. It will install the exact latest version of that package in your application and save it in package. This arises a problem if the dependency used has an exploit in the version mentioned. It can create a backdoor for an attacker.

Vulnerable Dependencies:
mongoose
tough-cookie
jsonwebtoken
word-wrap
semver
request
fast-xml-parser

Recommended Fix

1. Update all the above mentioned Dependencies
2. Remove any Library Not needed.

Impact

Multiple of these libraries have public exploits and CVE-registered issues that have been patched and can help your application stay more secure from any dependency-vulnerable issues.

Status

Resolved

7. Clickjacking

Description

A Clickjacking vulnerability was identified on the website https://qa.dregn.org/. Clickjacking, also known as a UI redress attack, is a technique that tricks users into clicking on malicious content or performing unintended actions without their knowledge or consent. In this case, the vulnerability allows an attacker to overlay or embed malicious content on top of the legitimate Dregn website, potentially leading to various forms of abuse or exploitation.

Vulnerable Endpoint:
https://qa.dregn.org/

Steps to Reproduce

1. Create a malicious web page or use an existing website under your control.
2. Modify the malicious web page's HTML to include an iframe that loads <html>

```
  <body>
    <h1>Malicious Website</h1>
    <iframe src="https://qa.dregn.org/"></iframe>
  </body>
</html>
```

3. Host the malicious web page on a web server.
4. Open the link where the malicious web page is hosted in your browser. You will find your website embedded in an iframe.

Recommendation

1. Set the X-Frame-Options HTTP response header to deny or sameorigin. This will prevent the website from being loaded inside an iframe on malicious websites.
2. Implement a strong Content Security Policy that includes the frame-ancestors directive with 'self' or specific trusted domains to restrict which websites can embed Google's content.

Status

Resolved

Closing Summary

In this report, we have considered the security of the Dregn ICO Web App. We performed our audit according to the procedure described above.

Some issues of High, medium, low, and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2023

For

Dregn

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com