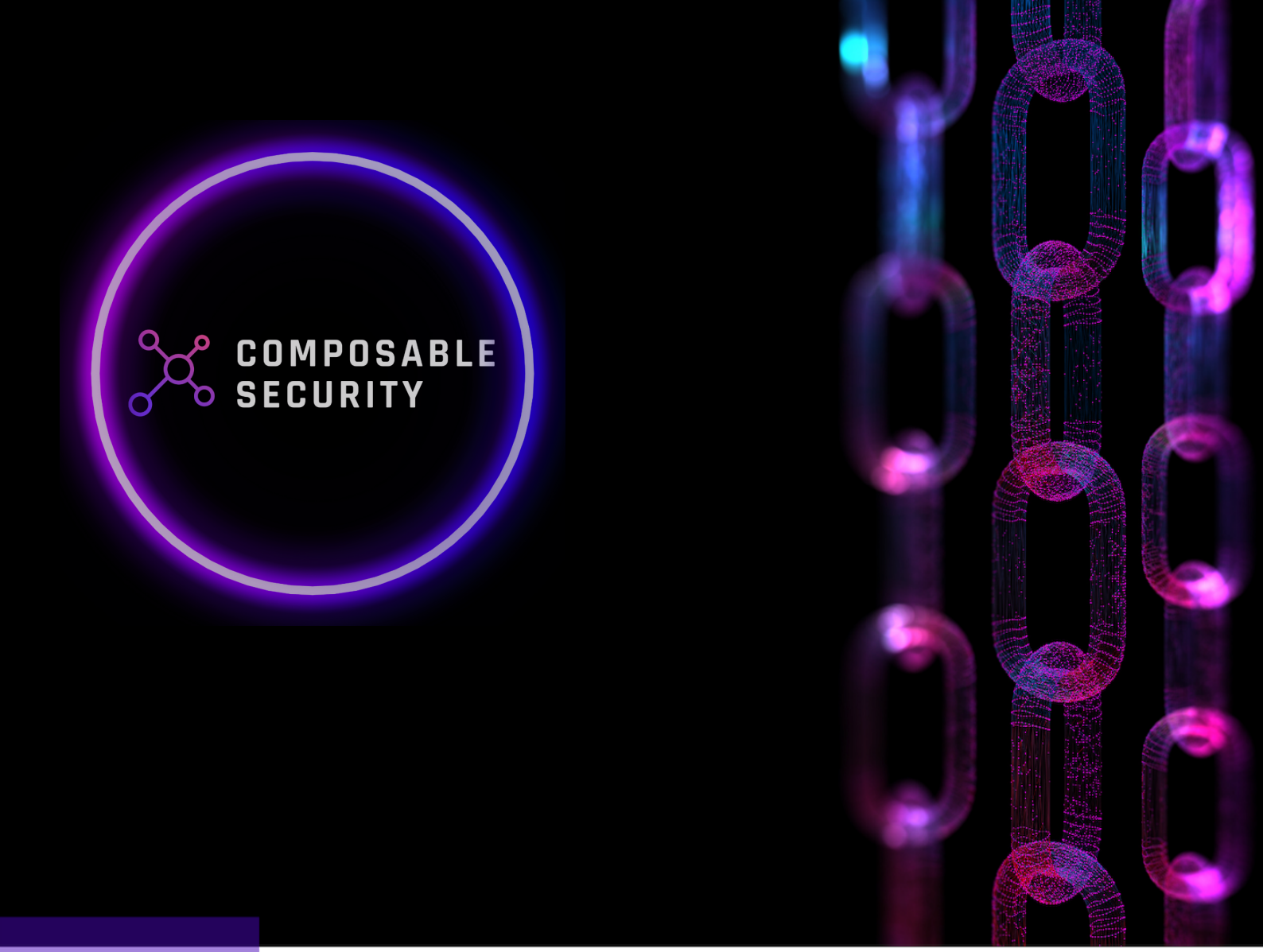


COMPOSABLE SECURITY

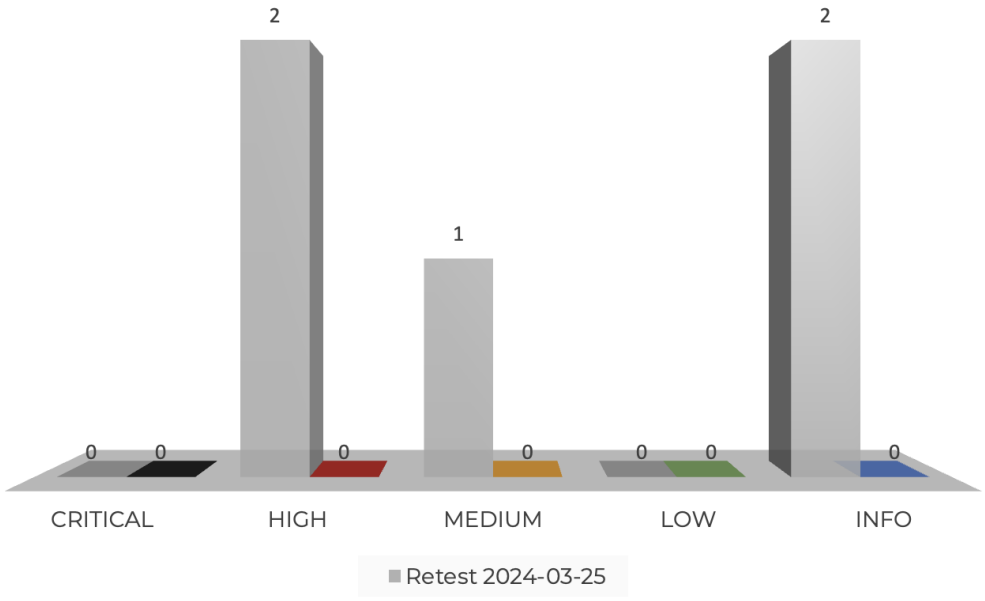
REPORT

Security consultation for ChickenDAO application

Prepared by: Composable Security
Report ID: CHD-8aa78524
Test time period: 2024-03-04 - 2024-03-06
Retest time period: 2024-03-19 - 2024-03-25
Report date: 2024-03-25
Version: 1.0
Visit: composable-security.com

Contents

1. Retest summary (2024-03-25)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in multiple iterations of verification whether the threats and vulnerabilities detected during the consultation (between 2024-03-04 and 2024-03-06) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- All **high** risk vulnerabilities have been fully fixed.
- One **medium** risk vulnerability has been fixed.
- **2** security **recommendations** were implemented.
- The detailed threat analysis (see Chapter 5) has been updated for points that has changed.
- During the retest, Composable Security team consulted the implementation of authentication using the custom messages signed by the TRON wallet.

1.2. Scope

The retest scope included the same repositories, on a different commit.

1.2.1 Frontend application

GitHub repository:
https://github.com/crytpto-dk-c/ccc-dao-web
CommitID: 0c920b63766cd19bb6e64b0e1ee57b4c4f056972

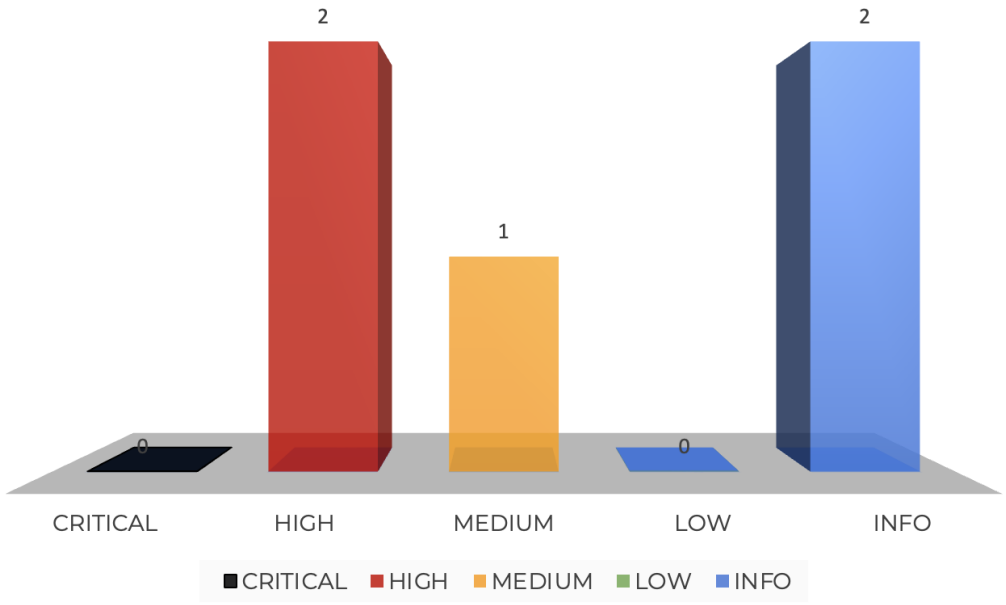
1.2.2 Telegram bot

GitHub repository:
https://github.com/crytpto-dk-c/ccc-dao-bot
CommitID: b419fc298c1742c9bcb4444948bff4793423afab

2. Current findings status

ID	Severity	Vulnerability	Status
CHD-8aa78524-H01	HIGH	Unprotected API endpoint	FIXED
CHD-8aa78524-H02	HIGH	Insufficient balance check in banning service	FIXED
CHD-8aa78524-M01	MEDIUM	NoSQL Injection	FIXED
ID	**Severity**	**Recommendation**	**Status**
CHD-8aa78524-R01	INFO	Turn off detailed error messages	IMPLEMENTED
CHD-8aa78524-R02	INFO	Remove unused code	IMPLEMENTED

3. Security consultation summary (2024-03-06)



3.1. Results

The **Chicken DAO** engaged Composable Security to consult security of **ChickenDAO**. Composable Security conducted this assessment over 3 person-day with 1 engineer.

The main goal of the consultation was to analyze security threats for users using the ChickenDAO front-end application and the security of the Telegram bot. The bot controls access to a private group dedicated to CCC token holders.

Detailed threat analysis results can be found in Chapter 5.

The summary of the consultation is as follows:

- The testing team has not identified any function that could be used to control users' funds. There are no functions that ask user to accept any transfers.
- There was an issue found that allowed anyone to join the private group dedicated for the core developers. The details can be found in CHD-8aa78524-H01.
- The wallet service issue responsible for detecting group members without the required amount of CCC tokens is not functioning correctly and does not remove members with 0 tokens. The details can be found in CHD-8aa78524-H02.

Composable Security recommends that **Chicken DAO** complete the following:

- Address all reported issues.
- Go through the configuration hardening process for selected applications (at least Telegram and X).
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.

3.2. Scope

The subjects of the security consultation were selected repositories of the **Chicken DAO**.

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this consultation on the following:

- Perform the threat analysis based on the predefined questions related to users' security.
- Identify security issues and potential threats both for **Chicken DAO** and their users.

4.2. Agreed scope of tests

The subjects of the test were selected repositories of the **Chicken DAO**.

4.2.1 Frontend application

GitHub repository:
https://github.com/crytpto-dk-c/ccc-dao-web
CommitID: e605ec968f376db9a814fb4498e97159ed244623
Available at: dao.coconutchicken.com

Files in scope:

```
src
├── src/App.test.tsx
├── src/App.tsx
├── src/index.tsx
├── src/pages
│   ├── src/pages/cccDao
│   │   └── src/pages/cccDao/index.tsx
│   └── src/pages/home
│       └── src/pages/home/index.tsx
├── src/react-app-env.d.ts
├── src/reportWebVitals.ts
├── src/setupTests.ts
└── src/theme.ts
```

4.2.2 Telegram bot

GitHub repository:
https://github.com/crytpto-dk-c/ccc-dao-bot
CommitID: c1b0d3f97231f9a95545798e648f4b533fe3eaea
Available at: bot.coconutchicken.com

Files in scope:

```
├── ./app.js
├── ./controllers
│   └── ./controllers/index.js
├── ./models
│   └── ./models/wallet.js
├── ./package-lock.json
├── ./package.json
├── ./routes
│   └── ./routes/index.js
├── ./services
│   └── ./services/walletService.js
└── ./utils
    ├── ./utils/telegramBot.js
    └── ./utils/tronwebUtil.js
```

4.2.3 Documentation

No documentation was provided.

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the consultation in cooperation with the client.

The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Potential attackers goals:

- Theft of users' tokens.
- Unauthorized access to the private group.

Potential scenarios to achieve the indicated attacker's goals:

- Backdoor in the front-end application that asks users to sign transfers.
- Cross-Site-Scripting in the front-end application allowing to modify the process of retrieving balances and asks users to sign transfers.
- Accessing the private groups via exposed invite link.
- Forging generation of an invite link by the application.
- Remaining in the private group despite losing the required number of funds.

4.4. Testing methodology

Security review and consultation was performed using the following methods:

- Q&A sessions with the **Chicken DAO** development team to thoroughly understand intentions and assumptions of the project.
- Threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Penetration tests focused on the confirmation of identified issues.
- **Manual review of the application code.**

4.5. Disclaimer

Security consultation **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing Web3 applications is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Threat analysis

Threat analysis aimed at verification of potential threats, focused to answer the questions below.

5.1. Front-end application threats

Does the ChickenDAO front-end application follow the best security practices in the integration with the TRON wallet?

The front-end application has been integrated with the official TronLink wallet utilizing the Tronwallet Adapter.

Currently, this is the sole wallet compatibility offered. This integration is facilitated through the employment of `WalletProvider` and `WalletModalProvider` components, sourced from the official TRON library (`@tronweb3/tronwallet-adapter-react-ui`), specifically tailored for React applications.

The application omitted any assignment of hooks to the TronLink adapter, nor is there invocation of its functions. The singular function employed to extract data from the wallet provider is the `useWallet` function. This function is instrumental in acquiring the wallet's address, its current state (whether connected or disconnected), and the wallet object itself.

The wallet object's sole purpose is to ascertain the name of the adapter, which is TronLink in this context. The state is conspicuously displayed on the website, and the wallet's address is transmitted to the back-end application, which is responsible for retrieving the balance of the specified address.

Error handling has been implemented.

Does the ChickenDAO front-end application contain any function that could steal users' CCC tokens?

Retest update: *The application now asks users to sign a JSON message with project name, timestamp and user's address. The message does not allow the application to steal user's tokens.*

The testing team conducted a thorough examination and confirmed the absence of any functions within the integration with TronLink wallet that would initiate requests for signing messages, including those for transfers.

The sole operation executed by the front-end application is establishing a connection with the wallet, with the primary objective of accessing the user's wallet address.

Does the ChickenDAO front-end application contain XSS vulnerability that could allow external users to change the logic of the integration with the wallet and ask users to confirm the malicious transfer?

The testing team did not find any input that could be used by external threat actor to inject XSS payload, neither as stored, nor as reflected type of XSS vulnerability.

Does the ChickenDAO front-end application save some sensitive data about users on servers?

The front-end application is designed to collect solely the wallet address from the user. It is important to highlight that the operation to verify the balance is conducted within the back-end application. In this process, the front-end application initiates a request to the back-end, which possesses the capability to ascertain the user's IP address.

Upon review, the testing team has not identified any segment within the bot's source code that actively collects the IP address. However, it should be acknowledged that there exists a potential for the IP address to be recorded within the server logs.

Can external applications retrieve any data from the back-end application (e.g. bypassing CORS)?

The front-end application does not use cookies and the API endpoints do not expose and protected data (e.g. where users are required to authenticate).

5.2. Telegram bot (back-end) application threats

Does the Telegram bot application expose any unprotected ports and access to internal components (e.g. database)?

Retest update: *The messages has been updated and server listens only on 80 and 433 ports, as expected.*

The server hosting the bot (bot.coconutchicken.com) is configured to expose only HTTP ports 80 and 443 to external traffic. Analysis of the source code indicates the presence of a scraping server operating on a port defined by the environment variable `process.env.PORT`. However, the absence of a `.env` file within the repository precludes the determination of this configuration parameter's actual value. Although log messages hint at the use of port 30000, it is important to note that only ports 80 and 443 are accessible from the internet, suggesting that port 30000 is not externally exposed.

```
23 app.listen(process.env.PORT, () => {
24   console.log("Scrapying server is running on port 30000");
25 });
```

The server hosting the website (dao.coconutchicken.com) similarly exposes only HTTP ports 80 and 443 to external traffic.

Can users bypass the balance check and join the closed Telegram group?

Retest update: *The vulnerability has been removed and now the balance check is performed*

on the back-end side as well.

The front-end application, after retrieving the wallet address, sends a request to the back-end application to check the balance. Additionally, the front-end application additionally sends the address of the token contract.

After the balance is received, the front-end application validates its minimum value if the requirement is met, it generates the invite code and stores it in the back-end application. This validation can be bypassed by any user and they can join the private group on Telegram. The details of this vulnerability have been presented in CHD-8aa78524-H01.

Can users bypass the periodic balance check and stay in the Telegram group without the required balance?

Retest update: *The vulnerability has been removed and now the periodic balance check is performed correctly. The only case where user is not banned having insufficient funds is when there is an error in communication with RPC provider.*

The bot application executes the `checkWalletBalances` function every 15 minutes. It's aim is to check the balances of all users in the group and if any user does not have required amount of tokens, they are kicked out from the group.

However, the check is insufficient and any user that has no tokens (0 value) will not be removed from the group. The details of this vulnerability have been presented in CHD-8aa78524-H02.

Does the CCC token contract contain any powerful functionality that could lead to the minting of new tokens, burning or stealing someone's tokens?

The CCC token has been found at *TRv9ipj4kKAZqQggQ7ceJpe5ERD1ZShpgs* address in TRON mainnet blockchain.

The token is typical TRC20 token (equivalent of ERC20 token) with the following state-changing functions:

- approve (095ea7b3)
- transferFrom (23b872dd)
- increaseAllowance (39509351)
- decreaseAllowance (a457c2d7)
- transfer (a9059cbb)

There are no functions that allow to mint or burn tokens.

Is there any entity that could use those powerful functions without requiring the confirmation by the majority of the community?

There are no powerful function in the CCC token contract. All functions are typical for tokens and there are no functions that are allowed to be executed only by any specific role.

Is the majority of the CCC tokens held by one entity?

The following information has been received on 4th of March 2024.

The token is distributed between 4,374 token holder accounts. The address that holds the biggest amount (6.60%) is the *Black hole* address (T9yD14Nj9j7xAB4dbGeiX9h8unkKLxmGkn). The second biggest holder is TGUX8s7JRLf4VZHDRYn8sRRZHKfPtSEYMd address with 3.005% of tokens.

There is no address that holds the significantly greater amount of tokens than the others.

6. Vulnerabilities

[CHD-8aa78524-H01] Unprotected API endpoint

HIGH FIXED

Retest (2024-03-19)

The vulnerability has been removed as recommended. The authentication mechanism has been added, using JWT.

Affected files

- /controllers/index.js#L25

Description

The bot application exposes the following endpoints:

- `/getTokenBalance`,
- `/save_data`,
- `/get_invitecode`,
- `/getTotalSupply`,
- `/getCirculateSupply`.

The endpoint `/get_invitecode` returns an information about the invite code for a given address.

A curl command to retrieve information about address TCuqovrsNSPGMwRGivGx2SsnTqkgJDdSY3:

```
curl 'https://bot.coconutchicken.com/get_invitecode' \
  -H 'content-type: application/json' \
  --data-raw '{"address":"TCuqovrsNSPGMwRGivGx2SsnTqkgJDdSY3"}'
```

The result either returns information that the code is new, expired or leaks the invite code for the adress that the code was not yet used and is not expired.

Furthermore, the endpoint `save_data` allows the front-end application to store generated invite code and assign it to user's address.

However, as the endpoint is not protected, anyone can call it and store the fake invite code to be later used.

Here is the example of a curl command that stores the code `12345678`:

```
curl 'https://bot.coconutchicken.com/save_data' \
```

```
-H 'content-type: application/json' \
--data-raw '{"inviteCode":"12345678","address":"
    TVEtg5KdNGMkAaLRJUUjQpfmvBjuzPrQxZ","expire_date":1759555387}'
```

Attack scenario

The attackers might take the following steps to join the private group using Telegram bot:

① Call the `save_data` endpoint to store code `12345678` and pass any wallet address in the address parameter.
② Send the message `/start 12345678` to the Telegram bot (e.g. by accessing https://t.me/test_eg_bot?start=12345678 link).
③ Use the invite link sent by the bot to access the private group.

Result of the attack: The attacker can generate a valid invite code and see codes for other users and join the private group.

Recommendation

- Implement authentication mechanism for API endpoints that return user-related data.
  - The back-end application must require a parameter that is a signed message. The message must be unique for the application and should include some nonce (e.g. current datetime).
  - The front-end application should ask user to sign the message to prove they control the address.
- Merge `/get_invitecode` and `/save_data` endpoints into one endpoint `/checkAddress` that will check signature to authenticate user, check the balance, generate the code in the back-end application (if all requirements are met) and return it.

[CHD-8aa78524-H02] Insufficient balance check in banning service

HIGH FIXED

Retest (2024-03-19)

The vulnerability has been removed as recommended.

Affected files

- /services/walletService.js#L39

Description

The wallet service plays a crucial role in monitoring the token balances of all group members, with a specific focus on identifying instances where a user's token balance falls below the requisite threshold.

Upon detection of such a scenario, the system is designed to automatically remove the user from the group. The code segment responsible for executing the user removal process is as follows:

```
38 const balance = await getBalanceOfCCCToken(address);
39 if (balance && balance < 10000000) {
40     console.log(`kickUserFromGroup: ${user_id}`);
41     await kickUserFromGroup(user_id);
42 }
```

The `balance` variable is utilized within an `if` condition, where it is converted to a boolean type. The issue encountered here is that the `balance` variable is cast to a false value in two distinct scenarios: when an error occurs and when the balance is zero. As a result of this casting to `false`, the `if` statement fails to execute in both situations.

Attack scenario

The attackers might take the following steps in turn:

① The attacker gets the required amount of tokens to join the private group.
② The attacker transfers all their tokens to other address.
③ The wallet service does not kick out the user with insufficient amount of tokens.

Result of the attack: The user without required amount of CCC tokens stays in the private group.

Recommendation

Make sure that the `getBalanceOfCCCToken` function allows to distinguish situation when there was an error and when the balance is zero, and handle it accordingly.

[CHD-8aa78524-M01] NoSQL Injection

MEDIUM FIXED

Retest (2024-03-19)

The vulnerability has been removed as recommended.

Affected files

- /controllers/index.js#L27
- /controllers/index.js#L58
- /models/wallet.js#L32

Description

The telegram bot application passes the request body parameters directly to the NoSQL ORM model, without any validation. That allows anyone to execute arbitrary NoSQL operators on database.

A curl command with injected `$regex` operator:

```
curl 'https://bot.coconutchicken.com/get_invitecode' \
  -H 'content-type: application/json' \
  --data-raw '{"address":{"$regex": "^TVEt.*"}}'
```

The result:

```
{"is_newcode":-1,"inviteCode":"12345678"}
```

Result of the attack: The attacker can execute arbitrary operators on database. An example could be finding multiple invite codes.

Recommendation

Validate input format and do not allow to pass NoSQL operators.
Consider using `express-mongo-sanitize` package.

References

1. Package: `express-mongo-sanitize`

7. Recommendations

[CHD-8aa78524-R01] Turn off detailed error messages

INFO IMPLEMENTED

Retest (2024-03-19)

The recommendation has been implemented. The middleware for handling errors has been added.

Description

The Telegram bot application presents a detailed error message when an error occurs in the application.

Example request with incorrect JSON structure (lack of closing quote):

```
curl 'https://bot.coconutchicken.com/get_invitecode' \
  -H 'content-type: application/json' \
  --data-raw '{"address":"TCuqovrsNSPGMwRGivGx2SsnTqkgJDdSY3}'
```

Result:

```
(...)
<pre>SyntaxError: Unterminated string in JSON at position 47 (line 1 column 48)<
    br>    at JSON.parse (&lt;anonymous&gt;)<br>    at parse
    (/app/node_modules/body-parser/lib/types/json.js:89:19)<br>    at /
    app/node_modules/body-parser/lib/read.js:128:18<br>    at
    AsyncResource.runInAsyncScope (node:async_hooks:206:9)<br>    at
    invokeCallback (/app/node_modules/raw-body/index.js:231:16)<br>    
    at done (/app/node_modules/raw-body/index.js:220:7)<br>    at
    IncomingMessage.onEnd (/app/node_modules/raw-body/index.js:280:7)<br>  
     at IncomingMessage.emit (node:events:519:28)<br>    at
    endReadableNT (node:internal/streams/readable:1696:12)<br>    at
    process.processTicksAndRejections (node:internal/process/task_queues:82:21)</
    pre>
(...)
```

Recommendation

Turn off the detailed error messages.

[CHD-8aa78524-R02] Remove unused code

INFO IMPLEMENTED

Retest (2024-03-19)

The recommendation has been partially implemented.

Description

The Telegram bot application contains commented out code, unused variables, duplicates code and fragments which are never reached. Here is the list:

- /utils/telegramBot.js#L21
- /utils/telegramBot.js#L28
- /utils/telegramBot.js#L34
- /utils/telegramBot.js#L63-L68
- /utils/telegramBot.js#L143-L156

Recommendation

Remove the code that is commented, unused or is never reached.

8. Impact on risk classification

Risk classification is based on the one developed by OWASP [1].

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com