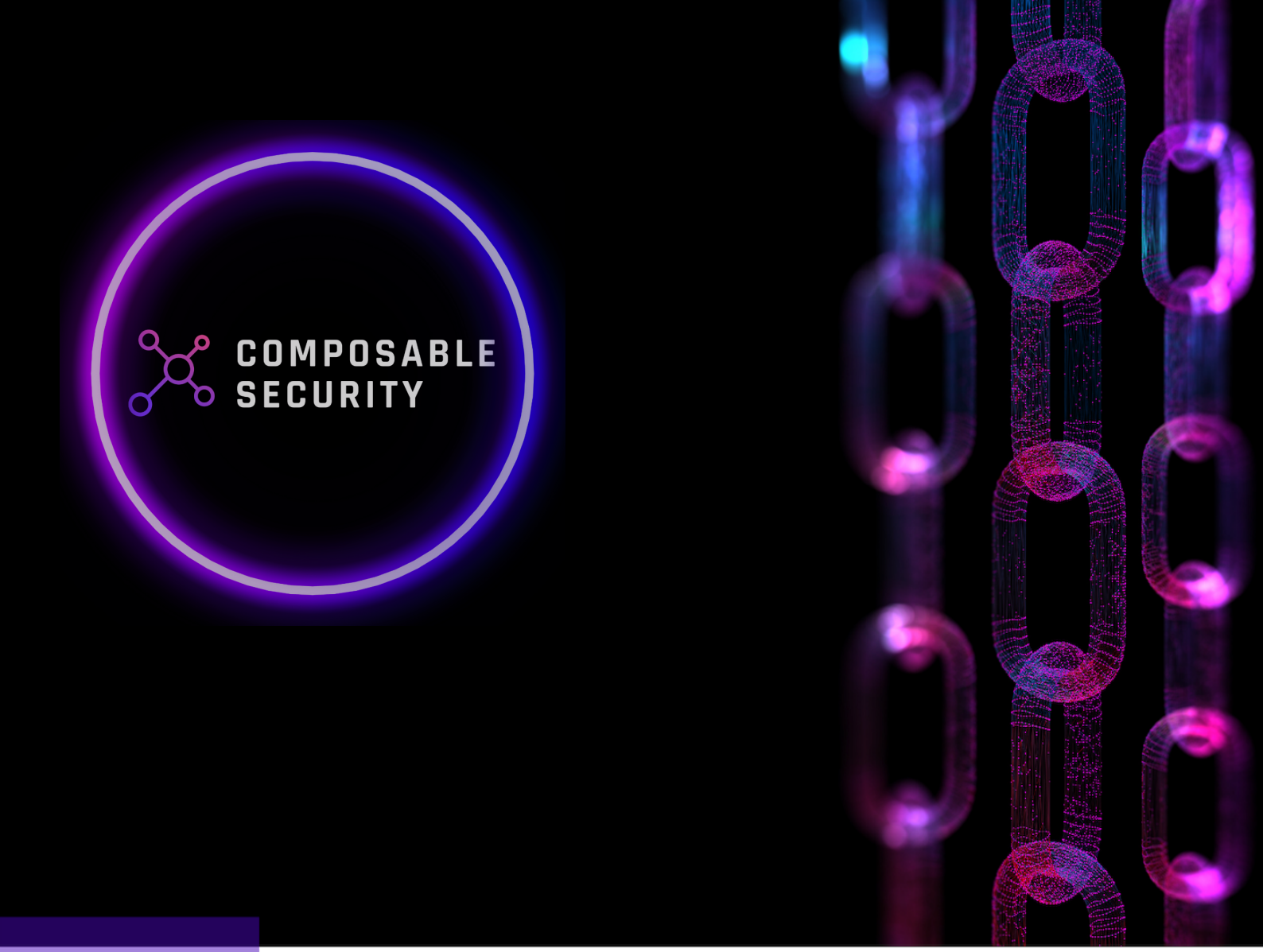


COMPOSABLE SECURITY

REPORT

Security consultation for Freelance Labs, Inc.

Prepared by: Composable Security
Report ID: BTRS-ee59003
Test time period: 2024-07-29 - 2024-08-30
Retest time period: 2024-09-30 - 2024-10-03
Report date: 2024-08-30
Version: 1.0
Visit: composable-security.com

Contents

1. Retest summary (2024-10-03) **3**
1.1 Results . . . 3
1.2 Scope . . . 4

2. Current findings status **5**

3. Security review summary (2024-08-30) **6**
3.1 Client project . . . 6
3.2 Results . . . 6
3.3 Scope . . . 7

4. Project details **8**
4.1 Projects goal . . . 8
4.2 Agreed scope of tests . . . 8
4.3 Threat analysis . . . 9
4.4 Testing methodology . . . 10
4.5 Disclaimer . . . 10

5. Vulnerabilities **12**
[BTRS-ee59003-H01] Race condition on BTRST payments and withdrawals . . . 12
[BTRS-ee59003-H02] Improper validation of Coinbase transaction . . . 13
[BTRS-ee59003-M01] Invalid Safe wallet selection . . . 14
[BTRS-ee59003-M02] Reverts on Safe wallet multi transfers . . . 15
[BTRS-ee59003-M03] Improper event logs parsing . . . 17
[BTRS-ee59003-M04] Improper validation of the signature . . . 18
[BTRS-ee59003-M05] Withdrawal limit bypass . . . 20
[BTRS-ee59003-L01] Exceeding reward cap per member . . . 22
[BTRS-ee59003-L02] Invalid error message . . . 23
[BTRS-ee59003-L03] Use one key per wallet . . . 24
[BTRS-ee59003-L04] Invalid BTRST balance when Token API is not working . . . 25
[BTRS-ee59003-L05] Session ID leakage . . . 26
[BTRS-ee59003-L06] Long session age . . . 27

6. Recommendations **29**
[BTRS-ee59003-R01] Use Onramp Session Token . . . 29
[BTRS-ee59003-R02] Consider improvements for secure key management . . . 29
[BTRS-ee59003-R03] Remove redundant filtering . . . 30

7. Smart contract security review **32**

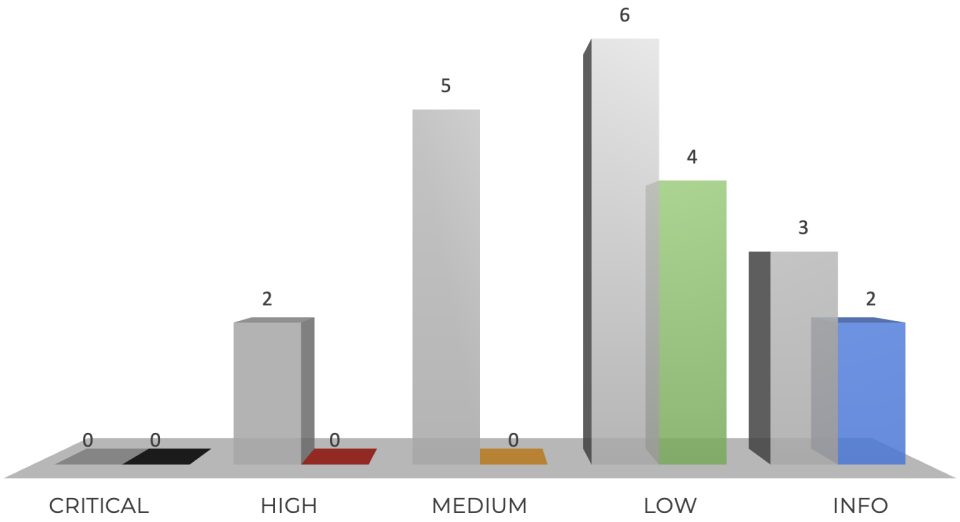
8. Answers to the consultation questions **33**

9. Impact on risk classification **35**

10Long-term best practices **36**

10.1 Use automated tools to scan your code regularly 36
10.2 Perform threat modeling 36
10.3 Use Smart Contract Security Verification Standard 36
10.4 Discuss audit reports and learn from them 36
10.5 Monitor your and similar contracts 36

1. Retest summary (2024-10-03)



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 a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2024-07-29 and 2024-08-30) were removed correctly and no longer appear in the code.

All issues have been addressed, and those resolved have been managed through separate merge requests. Collaboration with the team has been highly efficient and professional throughout the process.

The current status of detected issues is as follows:

- All (2) **high** vulnerabilities have been fixed.
- All (5) **medium** vulnerabilities have been fixed.
- 6 vulnerabilities with a **low** impact on risk were handled as follows:
  - 4 have been acknowledged,
  - 2 have been fixed.
- 3 security **recommendations** were handled as follows:
  - 1 have been implemented,
  - 2 have been acknowledged.

The issue BTRS-ee59003-L03 has been addressed and is scheduled for resolution in accordance with the recommended actions. However, as it is still under analysis, it has been marked as acknowledged and accompanied by an appropriate comment for tracking purposes.

1.2. Scope

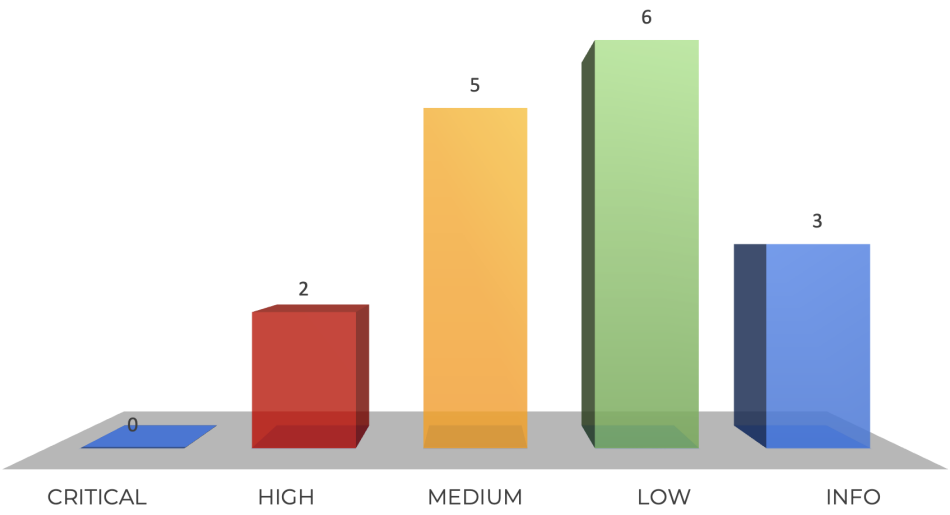
The retest scope included the same repositories, on different commits.

- Braintrust Token API
  - Repository: https://git.hexocean.com/braintrust-token-api/token-api-backend
  - Commit: 61d701494de373b0e479811a5467e385dd258825
- Braintrust App
  - Repository: https://git.hexocean.com/mb/braintrust
  - Commit: 844203658d8f65d39785ab1ff28e780ec416e3d8
  - Note: Some fixes were present in not yet merged Merge Requests. In those cases, the retest summary includes commit ID in which the fix is present.

2. Current findings status

ID	Severity	Vulnerability	Status
BTRS-ee59003-H01	HIGH	Race condition on BTRST payments and withdrawals	FIXED
BTRS-ee59003-H02	HIGH	Improper validation of Coinbase transaction	FIXED
BTRS-ee59003-M01	MEDIUM	Invalid Safe wallet selection	FIXED
BTRS-ee59003-M02	MEDIUM	Reverts on Safe wallet multi transfers	FIXED
BTRS-ee59003-M03	MEDIUM	Improper event logs parsing	FIXED
BTRS-ee59003-M04	MEDIUM	Improper validation of the signature	FIXED
BTRS-ee59003-M05	MEDIUM	Withdrawal limit bypass	FIXED
BTRS-ee59003-L01	LOW	Exceeding reward cap per member	ACKNOWLEDGED
BTRS-ee59003-L02	LOW	Invalid error message	FIXED
BTRS-ee59003-L03	LOW	Use one key per wallet	ACKNOWLEDGED
BTRS-ee59003-L04	LOW	Invalid BTRST balance when Token API is not working	ACKNOWLEDGED
BTRS-ee59003-L05	LOW	Session ID leakage	FIXED
BTRS-ee59003-L06	LOW	Long session age	ACKNOWLEDGED
ID	**Severity**	**Recommendation**	**Status**
BTRS-ee59003-R01	INFO	Use Onramp Session Token	IMPLEMENTED
BTRS-ee59003-R02	INFO	Consider improvements for secure key management	ACKNOWLEDGED
BTRS-ee59003-R03	INFO	Remove redundant filtering	ACKNOWLEDGED

3. Security review summary (2024-08-30)



3.1. Client project

The **Braintrust** project is a decentralized talent network that connects workers with the reputable brands. The main flow is based on job posting, candidate review and collaboration. However, the entire platform currently has much broader applications such as reference systems, career help or AI support.

The project focused on secure expansion to the Base network, secure setup of Safe wallets generated for each user, and properly integrating with Coinbase Ramp.

3.2. Results

The **Freelance Labs, Inc.** engaged Composable Security to review security of **Braintrust**. Composable Security conducted this assessment over 5 person-weeks with 2 engineers.

The summary of findings is as follows:

- 2 vulnerability with a **high** impact on risk was identified. Their potential consequences are:
  - Spending more tokens than the user have.
  - Increasing the token balance with a cheaper token.
- 5 vulnerabilities with a **medium** impact on risk were identified.
- 6 vulnerabilities with a **low** impact on risk were identified.
- 3 **recommendations** have been proposed that can improve overall security and help implement best practice.
- The most important issues detected concern Freelance Labs, Inc.integration with 3rd party components and cross-chain communication. All data received from external components should be subject to validation.

- During the security review, several additional issues were detected outside the scope of the tests (BTRS-ee59003-L05, BTRS-ee59003-L06). They concern the web application, its full audit is recommended.
- Findings were reported as issues on an ongoing basis, and questions were asked in a specially designated space in Confluence.
- A thorough analysis did not reveal any irregularities in smart contract.

Composable Security recommends that **Freelance Labs, Inc.** complete the following:

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Consider a full audit for web2 application.

3.3. Scope

The scope of the tests included the following:

Smart contracts security review

Deployed contract: *BTRST Token on BASE*

Penetration test based on security consultation for:

- Braintrust Token API
  - Repository: https://git.hexocean.com/braintrust-token-api/token-api-backend
  - Commit: af8c5f1e6cd88dc4745da7774ed410d15baf65af
- Braintrust App
  - Repository: https://git.hexocean.com/mb/braintrust
  - Commit: ee590032ae0bba712660fb0fc2d221f39571c876

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 audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Freelance Labs, Inc.** and their users.
- Finding answers to the questions that are the subject of the consultation.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The scope of the tests included the following:

Smart contracts security review

Deployed contract: https://basescan.org/address/0xa7d68d155d17cb30e311367c2ef1e82ab6022b67

Penetration test based on security consultation for:

- Braintrust Token API
  - Repository: https://git.hexocean.com/braintrust-token-api/token-api-backend
  - Commit: af8c5f1e6cd88dc4745da7774ed410d15baf65af
  - Scope: “safe” application (other applications are out of scope)
- Braintrust App
  - Repository: https://git.hexocean.com/mb/braintrust
  - Commit: ee590032ae0bba712660fb0fc2d221f39571c876
  - Scope:
    - Integration with API microservices (managing transfers) - token_api_client, safe_wallet app
    - Integration with Coinbase - posts, onramp
    - Handling events
- The consultation is focused to answer the following questions:
  - Is it possible to break the application and make it unresponsive?
  - Is it possible to bypass access control of the API and make transfers on behalf of other users?
  - Is it possible to take over the Safe wallet and transfer out tokens without noticing the backend system?
  - Is it possible to block the detection of transfers by the backend system?
  - Is it possible to bypass access control of the payment API and make transfers on

behalf of other users?
    - Can the payment token stay locked in the escrow mechanism?
    - Is it possible to withdraw tokens from the escrow mechanism on behalf of another user?
    - Is it possible for the receiver to unlock tokens in the escrow mechanism before the job is finished?
    - Is it possible for the asker to unlock tokens in the escrow mechanism after the job is finished?
    - Is it possible to use a different token than BTRST to pay for service?
    - Is it possible to refund a payment for delivered work?
    - Is it possible to pay for the service without the fee?
    - Does the integration with Coinbase Pay follow best security practices?
- Excluded from the scope:
    - Risk of Fraudulent Behavior by Help Offer Authors
        - Analyze patterns of behavior to identify potential fraud.
        - Implement and test mechanisms to prevent unjustified token transfers.

Documentation: https://drive.google.com/file/d/1PoBBf5p5Kw46o-IB_gNMSAdArSu03C1D/view

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- CDP Key and API Keys
- BTRST tokens

Potential attackers goals:

- Unauthorized transfer of tokens via API.
- Unauthorized mint of tokens.
- Takeover of the Safe wallet by the user.
- Takeover of the Safe waller by the attacker using a pre-generated wallet.
- Fake events.
- Unauthorized transfer of tokens within BTRST Payments.
- Lock of BTRST tokens in payment escrow mechanism.
- Token theft from escrow mechanism.
- Payment with a fake token.
- Refunding delivered work.
- Bypassing the fee.
- Insecure integration with Coinbase Ramp.

- Denial of Service.

Potential scenarios to achieve the indicated attacker's goals:

- Forging signer's signature.
- Using less valued token to increase the balance.
- Omitting events.
- Spoofing Safe wallet address.
- DoS through low gas limit.
- DoS through multiple small txs.
- DoS through spaming wallet creation actions.
- Lack of compliance with Base tokens standard deployment process.
- Inability to bridge the token.
- Deposit of fake token.
- Front-running wallet creation.
- Influence or bypass the business logic of the system.
- Take advantage of arithmetic errors.
- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Existence of known vulnerabilities (e.g., front-running, re-entrancy).
- Design issues.
- Excessive power, too much in relation to the declared one.
- Poor security against taking over the managing account.
- Private key compromise, rug-pull.
- Withdrawal of more funds than expected.
- Modifying or executing submitted transactions.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Freelance Labs, Inc.** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

4.5. Disclaimer

Smart contract security review **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 smart contract platforms 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. Vulnerabilities

[BTRS-ee59003-H01] Race condition on BTRST payments and withdrawals

HIGH FIXED

Retest (2024-09-30)

The vulnerability has been removed as recommended.
Commit ID: *48ba4b94e04090e08d6b88423601aff2a9d50ec2*.

Affected files

- views.py#L731-L737
- utils.py#L507
- utils.py#L566
- views.py#L2782
- models.py#L54

Description

To determine the user's BTRST token balance, the `available_tokens` function performs the following steps:

- retrieves the current balance of the Safe wallet,
- adds the amounts of transactions that have been mined but not yet processed in the application,
- subtracts any payments that are still being processed (such as withdrawals or payments for help offers).

To prevent multiple withdrawals from being initiated simultaneously, users are restricted from submitting new withdrawal requests until the current one has been fully processed and confirmed by the application.

However, certain parts of the code bypass this locking mechanism. Instead of `available_tokens` function, they directly call the `get_balance` function, which does not account for subtracting the payments being processed. This issue affects scenarios such as creating a balance item for Coinbase transactions, paying for pending upgrades, and purchasing boosts.

Attack scenario

The attackers might take the following steps in turn:

① Submit a withdrawal request for all 30 tokens to their address.
② The withdrawal is executed on the blockchain but has not yet been confirmed in the application.
③ Due to the withdrawal lock, no additional withdrawal requests can be submitted.
④ The attacker purchases a boost upgrade, taking advantage of the bypassed checks.

Result of the attack: Spending more tokens than the user have.

Recommendation

Do not use the `get_balance` function directly. Always use `available_tokens`. Consider using a lock for all operations on user's Safe wallet.

References

1. SCSVS G4: Business Logic

[BTRS-ee59003-H02] Improper validation of Coinbase transaction

HIGH FIXED

Retest (2024-09-30)

The vulnerability has been removed as recommended. Both the `purchase_currency` is validated and the BTRS-ee59003-R01 recommendation has been implemented. Commit ID: *03abbfe0137aded5fc96f9f4baab75f727c98dfd*.

Affected files

- models.py#L33-L62
- utils.py#L46-L73

Description

The users can buy BTRST tokens using the Coinbase Onramp. During this process, the application redirects user to the Coinbase where they pay for the token. Then, the token is transferred to the user's Safe wallet. The application specified the address where tokens should be sent after payment (the user's Safe wallet address) and the token to be purchased.

However, when the Coinbase transaction is processed, the token that was bought is not verified by the application whether it is a BTRST token. The Safe wallet address is not verified either.

This issue combined with the BTRS-ee59003-L04 allows an attacker to buy any other cheaper token and trick the application to create the balance item with a BTRST token as the currency.

Note: To exploit this vulnerability, the Token API must be off so that the balance is calculated based on the balance items.

Attack scenario

The attacker might take the following steps in turn:

① Initiate the Coinbase Onramp process.
② Change BTRST to some other, cheaper token.
③ Buy the other token on Coinbase exchange.
④ Coinbase transfers the token to user's Safe wallet.
⑤ The application creates a balance item with BTRST as currency.
⑥ The Token API is off.
⑦ The application calculates balance using balance items where one item is created with other token.

Result of the attack: Increasing the token balance with a cheaper token.

Recommendation

Implement recommendation described in BTRS-ee59003-R01.
Alternatively, you should:

- Validate the `purchase_currency` being bought and make sure it is BTRST token.
- Make sure the tokens were transferred to the user's Safe wallet. Otherwise, do not create a balance item.

References

1. SCSVS G5: Access control

[BTRS-ee59003-M01] Invalid Safe wallet selection

MEDIUM FIXED

Retest (2024-09-30)

The vulnerability has been removed as recommended.
Commit ID: *724cd88cd5460ca6a4c4011ed53daf2354598b0e*.

Affected files

- models.py#L1374-L1374

Description

The `get_live_token_balance` function uses the `self.safe_wallets.first()` instruction to get user's Safe wallet instance. However, there might be a deleted wallet which is not active anymore. In such situation the balance would be invalid.

Vulnerable scenario

The following steps lead to the described result:

① The user get their Safe wallet created.
② The Safe wallet is marked as inactive and new Safe wallet is created and assigned to the user.
③ The application gets the live token balance for user's Safe wallet.
④ The balance is retrieved for the deleted wallet.

Result: The live token balance is returned for inactive Safe wallet.

Recommendation

Use `get_active_safe_wallet` function to get user's Safe Wallet.

References

1. SCSVS G4: Business Logic

[BTRS-ee59003-M02] Reverts on Safe wallet multi transfers

MEDIUM FIXED

Retest (2024-09-30)

The vulnerability has been removed. The `safe_tx_gas` parameter is updated: the constant 75_000 limit is multiplied by the number of transfers.
The number of transfers is limited by the env variable, currently set to 10. Additionally, the gas prices is not intended to be changed in the code and always set to 0.
Regarding the monitoring, the team responded: *Our logic does not need to monitor the status of multitransaction, as it monitors individual transfers separately. There is no need to check the ExecutionSuccess event.*
Commit ID: *bd5cbf2f2f13032aeccc092e6415004b2669a440*.

Affected files

- safe_services.py#L28

Description

The Token API can execute a single transfer using Safe wallet's `execTransaction` function (using CALL operation) or multiple transfers using the same `execTransaction` function, but calling the MultiSend contract via DELEGATECALL.

In both cases the `safe_tx_gas` parameter is set to 75_000, which is enough for a single transfer but might be too low for multi transfer. If you look at the execution trace below, the MultiSend call consumes more than 75_000, that is about 84_000 for 9 transfers in it.

This is not an issue at the moment, because the `gas_price` parameter is set to 0 and the transaction can consume more than `safe_tx_gas`. However, it may lead to reverts if the `gas_price` parameter is changed. What is more, the internal operation (the transfer made by Safe wallet) can revert while the whole transaction does not.

Vulnerable scenario

The following steps lead to the described result:

① The value of `gas_price` parameter is changed.
② The application schedules a multi transfer with the number of transfer that consume more than 75_000 gas.
③ The transaction is not reverted, but the transfers are.

Result: The multi transfer transaction can execute successfully with a reverted transfer in it.

Recommendation

Make sure that the number of transfers within multi transfer is limited and calculate the correct gas limit for such operation.
If you do not plan to change the `gas_price` parameter add an assert with information that will appear during testing.

Additionally, monitor the multi transfer transactions and make sure they don't contain reverted internal calls. You can check whether the event `ExecutionSuccess` was emitted by Safe wallet.

References

1. SCSVS G10: Gas

[BTRS-ee59003-M03] Improper event logs parsing

MEDIUM FIXED

Retest (2024-09-30)

The vulnerability has been removed as recommended.
Commit ID: *7cc0e7fbc2b3ddfc9693b1d003bf07d9163d735e*.

Affected files

- retrieve_safe_wallet_events_coordinator.py#L17-L31
- retrieve_safe_wallet_events_coordinator.py#L54-L65

Description

The Token API backend application parses multiple events read from the Ethereum and BASE blockchains, including BTRST token transfers, Safe wallet creations, and more. For Safe wallet creation events, the address of the created Safe wallet is taken from the first event in the transaction.

Such approach works in the current state of the application, but if the creation process changes (e.g. include a payment supported by the `setup` function) the first event can be a different from the `SafeSetup` and the wallet address would be incorrect, blocking further confirmation of the legitimate wallet.

Note: There is a further validation of events emitted by the Safe wallet address in the `_retrieve_events_for_safe` function, and depending on whether a specific event is found (an edge case) the user is either assigned a wrong wallet address or the wallet address is never confirmed.

Vulnerable scenario

The following steps lead to the described result:

① The wallet creation includes the payment within the setup process.
② The creation process emits the `Transfer` event as the first event.
③ The application gets the address of the payment token as user's wallet address.
④ Further processing of the wallet creation is blocked because no valid events are returned for the token's address in `_retrieve_events_for_safe`.

Result: Depending on whether there is an event matching the filter from `_retrieve_events_for_safe` function the result may be that the wallet creation process cannot be processed, or an invalid wallet address is assigned to the user and all tokens sent to it will be blocked.

Recommendation

Iterate over all events in the transaction receipt and find the `SafeSetup` event to retrieve address. Additionally, in the `_retrieve_events_for_safe` function, specify correct event topic hash for `SafeSetup` event.

References

1. SCSVS G1: Architecture, design and threat modeling

[BTRS-ee59003-M04] Improper validation of the signature

MEDIUM FIXED

Retest (2024-09-30)

The vulnerability has been removed. Now each signature validation requires correct prefix, depending on the context.
Commit ID: *f80160250296c404079647a14e624893d85960b2*.

Affected files

- utils.py#L297-L304

Description

The application uses signatures by `signing.Signer` to authorize multiple operations, such as e-mail change confirmation, withdrawal confirmation, or using an invitation. However, there are cases where the signed confirmation data is not validated properly and some signatures can be used in a different context.

It is possible to generate many signed strings using the `prepare_upload` endpoint that returns a signed UUIDv4. Later, it can be used as a valid invitation key as described below in the attack scenario.

HTTP Request to register new account using the forged signature with valid invitation ID:

```
POST /api/user/registration/?new_employer_invitation_key=07a3d5c6-11-48c3-8f65
    -75b0f273f6b2.csv:smBSGot3rE117BgJMVWPn5UcsomeMvNFrmOTRewy0fI&
    amplitude_device_id=dNM54FSLocbHHFc2mO4QRI HTTP/2
Host: app3.bthexocean.com
sessionid=zcfp4t6bpsh38atj7ig1sv3hwiloaewj;
Content-Length: 2496
(...)

{"first_name":"AAA","last_name":"AAA","email":"company2@example.com","
    organization_size":"LESS_THAN_10","accept_terms_of_use":true,"password":"
    password","g-recaptcha-response":"(...)"}
```

HTTP Response:

```
HTTP/2 201 Created
Date: Wed, 07 Aug 2024 11:32:44 GMT
Content-Type: application/json
Content-Length: 115
(...)

{"email":"company2@example.com","finish_signup_url":"/onboarding/organization/",
    "id":11}
```

Attack scenario

The attackers might take the following steps in turn:

① Use `prepare_upload` endpoint to generate a filename signature where the second part of UUIDv4 is an ID of valid invitation (e.g. `07a3d5c6-8802-48c3-8f65-75b0f273f6b2.csv:sLikahRf9zioS6e-FM_86qPXTvRees5ERCVqvsQgVgU` for invitation with ID 8802).

② Use `user/registration` endpoint with the signature passed as `new_employer_invitation_key` parameter (as presented above).

③ Change the account e-mail to a malicious one to block the password reset. This step requires the password, but it's defined during registration.

Result of the attack: Registering an account based on the taken over invitation.

Recommendation

Validate the format of the signed key, especially the prefix (e.g. `invitation`).

References

1. CWE-347: Improper Verification of Cryptographic Signature

[BTRS-ee59003-M05] Withdrawal limit bypass

MEDIUM FIXED

Retest (2024-09-30)

The vulnerability has been removed.
Commit ID: *3b861ae04a0b887a428bbde60c2f6e2b4208e67b*.

Affected files

- views.py#L721-L739
- utils.py#L309-L332

Description

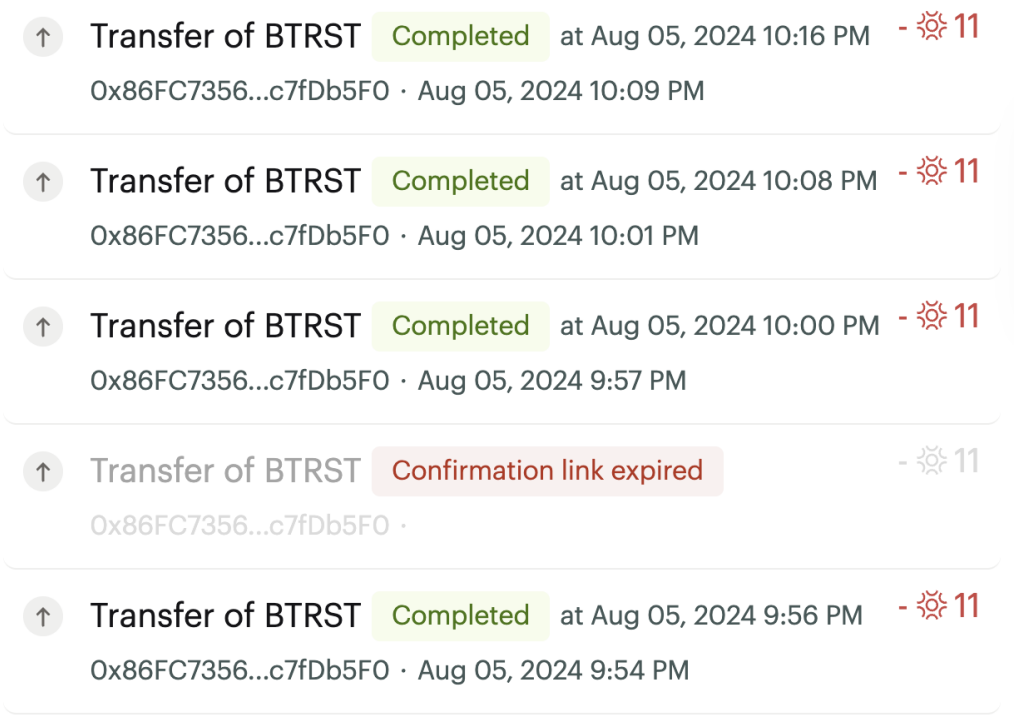
The application introduces a withdrawals limit (3 withdrawals per 7 days). However, the limit is not enforced when submitting new withdrawals. It allows user to submit multiple withdrawals. The `token_withdrawals_limit_exceeded` function only counts the withdrawals that have been successfully completed.

Later, when confirming the withdrawal using the link sent in the e-mail, the application does not enforce the limit anymore as shown in the code snippet below.

```
721 def confirm(self, request, pk=None, serializer_class=
        TransactionConfirmationSerializer):
722     with get_redis_lock(f'confirm_withdrawal_transaction_{request.user.id}'):
723         serializer = TransactionConfirmationSerializer(data=request.data)
724         serializer.is_valid(raise_exception=True)
725         transaction_id = self.get_transaction_id(serializer.validated_data.
                get('code'))
726         transaction = self.get_object(transaction_id)
727         bt_wt = TokenAPIExternalWithdrawalTransactionWrapper(transaction)
728         bt_wt.is_confirmation_valid()
729         bt_wt.validate_withdrawal_amount()
730
731         if bt_wt.can_be_scheduled():
732             bt_wt.schedule_withdrawal()
733         else:
734             raise NonFieldValidationError(
```

```
735                 'Only one transaction can be processed at a time and another
                        transaction is in '
736                 'progress right now. Please wait for it to be completed before
                         ordering another one.'
737             )
738
739         return Response(status=status.HTTP_204_NO_CONTENT)
```

Here is the proof of more withdrawals than the limit:



Note: The only limit is that there can be only one withdrawal processed at a time so the number of withdrawals executed by the user is limited by processing time, which was from 3 to 7 minutes during tests.

Attack scenario

The attackers might take the following steps in turn:

① The user submits many withdrawals to different addresses and with different amounts.
② The user confirms withdrawals (one by one) using links sent via e-mail.

Result of the attack: Executing more withdrawals than the limit set (currently 3 withdrawals per week).

Recommendation

Either include the pending withdrawals when counting withdrawals when submitting them or add validation in the confirmation process.

References

1. SCSVS G4: Business logic

[BTRS-ee59003-L01] Exceeding reward cap per member

LOW ACKNOWLEDGED

Retest (2024-09-30)

The vulnerability has been acknowledged.
The team has responded: *After reviewing the specs, we confirmed that the reward cap and the reward cap per member function as intended, serving as a soft limit. Therefore, the current behavior is correct. We've updated our Handbook to reflect this and added clear comments to the RewardRule model to indicate that this is a soft limit.*

Affected files

- models.py#L356-L367
- models.py#L90-L98
- models.py#L158-L159

Description

When a user is eligible for a reward (e.g. for referring someone) they get rewarded according to a specific reward rule. Before they receive the reward, some parameters are checked, e.g. whether they have already reached the reward cap. If they reached the cap, the reward is not credited to them.

However, when the user is close to the cap they receive the reward and the sum of all received rewards can exceed the cap.

Vulnerable scenario

The following steps lead to the described result:

① There is a reward rule with 10 000 cap per member.
② The user receives 3 rewards 3 000 each and has already received 9 000 in rewards.
③ The user is going to be rewarded next 3 000.

④ At this moment the user has not reached the cap.
⑤ The reward is credited to user.
⑥ The user has received 12 000 rewards which is 2 000 more than the cap.

Result: The user receives more rewards than the cap per member.

Recommendation

Check whether the current reward would not exceed the cap and if it does, decrease it to exactly reach the cap.

References

1. SCSVS G4: Business Logic

[BTRS-ee59003-L02] Invalid error message

LOW FIXED

Retest (2024-09-30)

The vulnerability has been removed.
Commit ID: *2c837507cbbbf7dd9893ff57449cb4ba23bebb86*.

Affected files

- wrappers.py#L38-L48

Description

The `is_confirmation_valid` function is used to check whether the token withdrawal link is valid. The first check is the status check and if the status is different from `AWAITING`, it returns the message: `Transaction has already been confirmed.` which is not correct if the status is `LINK_EXPIRED`.

Vulnerable scenario

The following steps lead to the described result:

① The user submits withdrawal.
② The time passes and the withdrawal becomes expired, unfreezing the tokens and setting the status to `LINK_EXPIRED`.
③ The user tries to confirm the withdrawal but gets the message: `Transaction has already been confirmed`.

Result: Invalid error message presented to the user.

Recommendation

Firstly check whether the status is `LINK_EXPIRED` and return appropriate message and then perform other checks.

References

1. SCSVS G5: Access control

[BTRS-ee59003-L03] Use one key per wallet

LOW ACKNOWLEDGED

Retest (2024-10-03)

The team plans to implement the recommendation. At this moment, they are evaluating the code footprint and other technical aspects before deploying it.

Affected files

Description

All Safe wallets in current implementation are protected by the same address, including the HOT wallet. This design can lead to the loss of all tokens managed by the application if the key is lost or leaked.

Recommendation

Set up the Safe wallets in the following way:

- Each wallet should have either one owner whose key is kept in a COLD wallet and multiple places (backup) or a 1-of-2 multisig composed of two owners whose keys are kept in COLD wallets and different places. Those keys are going to be used to change the owner in the future if you plan to migrate them to the users.
- Each wallet should use the AllowanceModule to allow a single address to execute transfers of BTRST. The key assigned to that address is used by the application to execute transfers and ideally, it's different for each wallet, especially for the HOT wallet.

References

1. SCSVS G1: Architecture, design and threat modeling

2. Safe AllowanceModule

[BTRS-ee59003-L04] Invalid BTRST balance when Token API is not working

LOW ACKNOWLEDGED

Retest (2024-09-30)

The vulnerability has been acknowledged.
The team has responded: *In both cases, the user will see a "too low" balance. This doesn't pose any risk of excessive transfer; it simply means the balance will appear slightly different.*

Affected files

- models.py#L1340-L1351

Description

The `get_balance` function is used to retrieve the user's balance for a specific currency. If the currency is the BTRST token it checks whether the "live" balance should be taken from the Safe wallet through the Token API. If the Token API is off, or there is no active wallet, the balance is returned based on the items stored in database which can be stale.

Additionally, due to the bug described in BTRS-ee59003-H02, this vulnerability can lead to a higher severity issue.

Vulnerable scenario

The following steps lead to the described result:

① The user buys tokens using Coinbase Onramp.
② The Token API is off.
③ The Coinbase transaction is confirmed by the application, but no balance item is created yet.
④ The user sends BTRST tokens to their Safe wallet.
⑤ The application does not show the correct balance and omits those 2 incomes.

Result: Invalid user's BTRST balance retrieved by the application.

Recommendation

Consider returning zero balance when the Token API is off or no active Safe wallet exists for the user.
If you plan to keep the balance calculation based on the balance items, inform user that the balance might be a bit stale due to the fallback calculation mechanism.

References

1. SCSVS G1: Architecture, design and threat modeling

[BTRS-ee59003-L05] Session ID leakage

LOW FIXED

Retest (2024-09-30)

The vulnerability has been removed. The session ID is being encrypted before returning in the response. The leaked encrypted session ID could be used only to logout user.
Commit ID: *b95228cb84e6d95b3bd00de2e908ea35c231c388*.

Affected files

- serializers.py#L831-L838

Description

The endpoint `/user_sessions` returns the session ID of logged in user which should not be accessible by the frontend app (in JavaScript) because the `sessionid` Cookie has `HttpOnly` flag.

HTTP Request:

```
1 GET /api/user_sessions/ HTTP/2
2 Host: app3.bthexocean.com
3 Cookie: marker_id_62877feed16c2c9a7f23e537=c1bb7680-0ef5-464a-9bbb-3b8b15b1c345;
     __stripe_mid=bcd9b891-8d44-4af6-8274-652ac31bb5d0ac46fa; sessionid=
    thor9c86f7xjeu823eggh4umel39jb5l; messages=.
    eJy9zTEOwjAMAMCvWJ5AmAiB2Bn6gnasqsikpgokDqqTob-
    nr2C95cYRvX9bUZ_FjBdButD1Tji0EHZ5tZQ2sLiozBAV2KDroO_hMHOOrG5tFlU-
    pypWb49Q8rcYP5OcTUJbY93cbkeHhDjRP7PpB36HS3k:1sbHMU:
    C6RqyThHm8D1M72411IPlQFnflD6JY3GfHopmykABgY; csrftoken=
    G7tzYABVIDrpxx1RDDRHvimnPXWueWmr;
4 (...)
```

HTTP Response:

```
HTTP/2 200 OK
Date: Wed, 07 Aug 2024 07:57:34 GMT
Content-Type: application/json
Content-Length: 237
(...)

[{"is_current_session":true,"last_activity":"2024-08-07T09:57:10.690382+02:00","
   localization":"<redacted>","device_details":"PC / Windows 10 / Chrome
   112.0.5615","id":"thor9c86f7xjeu823eggh4umel39jb5l","ip_address":"<redacted>"
   }]
```

Attack scenario

The attackers might take the following steps in turn:

① The attacker finds an XSS vulnerability.
② The attacker makes the call to `/user_sessions` and reads the `id` parameter.
③ The attacker executes an operation that is not 2FA protected (e.g. purchasing a boost upgrade).

Result of the attack: An XSS vulnerability would allow to retrieve the session ID of the logged in user and execute some operations on their behalf.

Recommendation

Do not return session ID in the session's data.

References

1. CWE-201: Insertion of Sensitive Information Into Sent Data

[BTRS-ee59003-L06] Long session age

LOW ACKNOWLEDGED

Retest (2024-09-30)

The vulnerability has been acknowledged.
The team has responded: *We're currently in discussion with the team about whether to implement this, given the low severity and potential UX/business impact. We're leaning towards shortening the cookie duration to potentially 1 day.*

Affected files

- django.py#L379-L379

Description

Currently, the session cookie is stored for 2 weeks which means that the session is active for 2 weeks without user interaction.

```
377 SESSION_CACHE_ALIAS = 'session'
378 SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
379 SESSION_COOKIE_AGE = get_seconds(weeks=2)
```

Long session age can lead to take over of the session (and account) in case the user shares the device with others.

Attack scenario

The attacker might take the following steps in turn:

① The victims logs in to the application.
② The victim leaves the application and the device.
③ Less than 2 weeks later the attacker gets access to the device and reads the session ID.
④ The attacker can execute actions not protected by 2FA.

Result of the attack: Reuse of old session IDs to perform actions, not protected by 2FA, on behalf of user.

Recommendation

Set the cookie age to 1 hour as it is updated on each request.

References

1. CWE-613: Insufficient Session Expiration

6. Recommendations

[BTRS-ee59003-R01] Use Onramp Session Token

INFO IMPLEMENTED

Retest (2024-09-30)

The recommendation has been implemented as recommended.

Description

The Onramp Session Token is used to protect Onramp URLs from modification that can lead to buy different token or transfer it to different address than specified in the original URL. The case where the modification is used to exploit the integration has been described in (BTRS-ee59003-H02).

The token is requested by the application and generated by Coinbase API. Later, it must be included in the Onramp URL because Coinbase API will require it and verify whether the parameters are the same as in the token. Any change in parameters will be detected and the process will be halted.

Recommendation

- Turn on "Require secure initialization" to require the session token.
- Generate the token on each Onramp initialization and pass it as `sessionToken` parameter.

References

1. (Coinbase) Initializing Onramp

[BTRS-ee59003-R02] Consider improvements for secure key management

INFO ACKNOWLEDGED

Retest (2024-09-30)

The vulnerability has been acknowledged.
The team has responded: *We're currently blocked by Coinbase, as they have disabled the organizations feature, preventing us from adding a second person to share the account and keys. According to their response, they are in the process of reinstating this feature.*

Description

Secure integration with Coinbase Onramp relies heavily on CDP key security and its configuration. After the analysis, several opportunities to further improve security have been identified.

Recommendation

The following actions are recommended:

- more than one person should be able to generate keys (the so-called bus factor should be taken into account). If this is not actively used, an emergency procedure should be implemented to allow for regaining access to credentials,
- rotate all API keys provided during testing,
- consider using additional security feature such as IP whitelisting,
- additionally, in the future, treat any even short key leaks as their complete compromise and regenerate them immediately.

References

1. SCSVS G2: Policies and procedures

[BTRS-ee59003-R03] Remove redundant filtering

INFO ACKNOWLEDGED

Retest (2024-09-30)

The vulnerability has been acknowledged.
The team has responded: *Given the long ETA (Oct. 9) and low severity, we won't be re-testing this.*

Description

The `get_data_from_coinbase` function is responsible for retrieving transactions from the Token API. They are filtered by the dates, from the `start_date` till the `end_date`.

After the transactions are retrieved, additional filtering occurs. It removes transactions that were created after the `end_date`. However, it is redundant as the received transactions already met this condition.

Recommendation

Remove the redundant filtering in utils.py#L34.

References

1. SCSVS G11: Code Clarity

7. Smart contract security review

During the security review of the indicated smart contract, the team focused on following key threats:

- Vulnerabilities resulting from environment change (Ethereum -> Base),
- Changes to the original contract,
- Inconsistency with business logic,
- Lack of compliance with Base tokens standard deployment process.

A thorough analysis did not reveal any irregularities in this regard. The process was verified against the documentation provided here https://docs.base.org/docs/tokens/list/.

8. Answers to the consultation questions

The consultation was focused to answer the following questions:

- **Is it possible to break the application and make it unresponsive?**
  - **Yes.** The following vulnerabilities (BTRS-ee59003-M03) have been detected that influence availability of the application for a particular user.
    However, none of the attempts resulted in any long-term and irreversible disruption of the application. Overload attacks such as DDoS can still succeed, but this is a problem directly related to bandwidth and firewalls, which is beyond the scope of this endeavor.
- **Is it possible to bypass access control of the API and make transfers on behalf of other users?**
  - **No.** Access control issues were detected, but none of them allowed a transfer to be made on behalf of another user.
- **Is it possible to take over the Safe wallet and transfer out tokens without noticing the backend system?**
  - **No.** However, a vulnerability (BTRS-ee59003-M01) has been detected that causes the return of live token balance for inactive Safe wallet.
- **Is it possible to block the detection of transfers by the back-end system?**
  - **No.** There were no cases of blocking the detection of transfers identified.
- **Is it possible to bypass access control of the payment API and make transfers on behalf of other users?**
  - **No.** Access control issues were detected, but none of them allowed a transfer to be made on behalf of another user.
- **Can the payment token stay locked in the escrow mechanism?**
  - **No.** During testing, no vulnerability was detected that would allow this.
- **Is it possible to withdraw tokens from the escrow mechanism on behalf of another user?**
  - **No.** During testing, no vulnerability was detected that would allow this.
- **Is it possible for the receiver to unlock tokens in the escrow mechanism before the job is finished?**
  - **No.** During testing, no vulnerability was detected that would allow this.
- **Is it possible for the asker to unlock tokens in the escrow mechanism after the job is finished?**
  - **No.** During testing, no vulnerability was detected that would allow this.
- **Is it possible to use a different token than BTRST to pay for service?**
  - **Yes.** The following vulnerabilities should be removed by following the recommendations: (BTRS-ee59003-H02).
- **Is it possible to refund a payment for delivered work?**

- **No.** During testing, no vulnerability was detected that would allow this.

- **Is it possible to pay for the service without the fee?**
  - **No.** During testing, no vulnerability was detected that would allow this.
- **Does the integration with Coinbase Onramp follow best security practices?**
  - **No.** The following vulnerabilities should be removed by following the recommendations: (BTRS-ee59003-H02), (BTRS-ee59003-R01).

9. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

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

10. Long-term best practices

10.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

10.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

10.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

10.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

10.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com