Security Audit Report

MemeGoat

September 2024

V202408

Executive Summary

CoinFabrik was asked to audit the contracts for the **MemeGoat** project.

During this audit we found three critical issues, one high issue, one medium issue and several minor issues. Also, several enhancements were proposed.

The three critical issues were resolved. The high issue is still to be audited, given the massive changes in the code. The medium issue was acknowledged. The minor issues were either acknowledged or resolved. One of the enhancement proposals was not implemented, the other one was not checked given the massive changes in code.

Given the numerous critical issues, incomplete functionalities, and overall lack of code quality, more security issues could be hidden and not arise during this audit. In addition to the fixes of the issues found in this audit, new functionality, which was not audited, was added[1]. Therefore, we strongly recommend addressing the structural issues within the smart contracts and conducting a subsequent audit before deploying this code to a production environment.

The development team informed us that they intend to conduct a new security audit in the next month.

Scope

The audited files are from the git repository located at https://github.com/Meme-Goat-Finance/Audit-Contracts.git, in the **`contracts`** directory. The audit is based on commit `e18e2b45a91504ff1304bdb4f2d1efeef1196d9b`. Fixes were checked on commit `b805c9368592cb343e19c81a2b370fd027d3327a`.

The scope for this audit includes and is limited to the following files:

- **`contracts/memegoat-launchpad.clar`**: Launchpad contract.
- **`contracts/memegoat-stake-pool.clar`**: Staking pool contract.
- **`contracts/vault-v1-2.clar`**: Vault contract. This contract is intended to store the tokens belonging to other contracts.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

[1] See Unaudited Changes for details.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

Each severity label is detailed in the Severity Classification section. Additionally, the statuses are explained in the Issues Status section.

ID	Title	Severity	Status
CR-01	Empty Vault through Staking	Critical	Resolved
CR-02	Cannot Claim Rewards in Launchpad	Critical	Resolved
CR-03	Empty Vault Through Launchpad	Critical	Resolved
HI-01	Authentication with tx-sender	High	To be audited
ME-01	Arbitrary Contract Call	Medium	Acknowledged
MI-01	Broken Principals	Minor	Acknowledged
MI-02	Lack of Vault Isolation	Minor	Resolved
MI-03	Use of non-Whitelisted Traits	Minor	Resolved
MI-04	Use of block-height for Time Windows	Minor	Acknowledged

Critical Severity Issues

CR-01 Empty Vault through Staking

Location

- `contracts/memegoat-stake-pool.clar`

Classification

- CWE-862: Missing Authorization[2]

Description

An attacker may create a new pool where the `stake-token` calls the vault's `transfer-ft` and/or `transfer-stx` on some transfers, and then trigger it by calling `stake` and `unstake`.

On the `unstake` activation, it executes the `contract-call?` on line 443:

```
(as-contract (try! (contract-call? .memegoat-stakepool-vault-v1
transfer-ft stake-token (from-precision precison-amount stake-token)
user-addr)))
```

Allowing the attacker to run arbitrary code with `tx-sender` set to the staking pool. On this arbitrary code, the attacker can call the vault's `transfer-ft` and `transfer-stx` as it fancies, taking all the funds in the pool.

A similar attack can be conducted with the `reward-token`, taking advantage of the call in line 480, which is executed when the `claim-reward` function is called:

```
(as-contract (try! (contract-call? .memegoat-stakepool-vault-v1
transfer-ft reward-token reward-amount user-addr)))
```

Notice that while the staking pool has a function intended to verify pools, as shown by the existence of the `verify-pool` function in line 168, its verification status is not used to do any check in the `stake`, `unstake` and/or `claim-reward` functions.

This issue's impact has been bigger because of MI-02 Lack of Vault Isolation.

This attack would not be possible if the recommendation in HI-01 Authentication with tx-sender is followed.

[2] https://cwe.mitre.org/data/definitions/862.html

This attack would not be possible if the recommendation in ME-01 Arbitrary Contract Call is followed.

Recommendation

Check that the pool has been verified in the `stake`, `unstake` and `claim-reward` functions.

Status

Resolved. The staking pools are whitelisted now.

CR-02 Cannot Claim Rewards in Launchpad

Location

- `contracts/memegoat-launchpad.clar: 652-673`

Description

In the launchpad, rewards can never be claimed, given that the `campaign-rewards-set` field in the launch obtained in lines 658-659 as part of a `let` clause

```
(token-launch (try! (get-token-launch-by-id token-launch-id)))

(campaign-rewards-set (get campaign-rewards-set token-launch))
```

and checked for `true` in line 663

```
(asserts! campaign-rewards-set ERR-REWARDS-NOT-SET)
```

is never set to `true`.

Recommendation

Finish the implementation. Automated tests should check both the happy path and the different possible errors.

Status

Resolved. The `claim-rewards` function was removed.

CR-03 Empty Vault Through Launchpad

Location

- `contracts/memegoat-launchpad.clar`

Classification

- CWE-862: Missing Authorization[3]

Description

An attacker may register a new token launch using the `register-token-launch` function passing an attacking contract in the `token` parameter, then fund it with the `deposit-stx` function and finally steal the vault funds by calling `claim-token`. This will activate this call in line 639

```
(as-contract (try! (contract-call? .memegoat-launchpad-vault
transfer-ft token-trait user-allocation sender)))
```

allowing the attacker to withdraw the funds in a similar fashion as explained in CR-01 Empty Vault through Staking.

A similar call in the `claim-rewards` function (line 667) cannot be exploited only because, as explained in CR-02 Cannot Claim Rewards in Launchpad, the function always fails.

This issue's impact has been bigger because of MI-02 Lack of Vault Isolation.

This attack would not be possible if the recommendation in HI-01 Authentication with tx-sender is followed.

Recommendation

The tokens need to be whitelisted by the owner.

Status

Resolved. The token used in `claim-token` is whitelisted now.

[3] https://cwe.mitre.org/data/definitions/862.html

High Severity Issues

HI-01 Authentication with tx-sender

Location

- `contracts/memegoat-launchpad.clar`
- `contracts/memegoat-stake-pool.clar`
- `contract/vault-v1-2.clar`

Classification

- CWE-303: Incorrect Implementation of Authentication Algorithm[4]

Description

Using `tx-sender` to authenticate may lead to phishing attacks and exposure to other contract's bugs[5]. This is especially impactful when no tokens are transferred, as the Stacks' postconditions mechanism does not work, such as when an administrator changes a contract setting. Post-conditions also do not make any difference when the `tx-sender` is not the EOA that initiated the transaction.

Recommendation

Use `contract-caller` to authenticate instead. As an example of attacks that are not possible if `contract-caller` is used, if the approved contract would have been authenticated with this method then the attacks described in CR-01 Empty Vault through Staking and CR-03 Empty Vault Through Launchpad would not be possible.

Status

To be audited. The development team claims to have fixed it. Checking for this would require a new security audit, as a new authorization logic was introduced.

Medium Severity Issues

ME-01 Arbitrary Contract Call

Location

- `contracts/memegoat-launchpad.clar: 428, 430, 543`

[4] https://cwe.mitre.org/data/definitions/303.html
[5] https://www.coinfabrik.com/blog/tx-sender-in-clarity-smart-contracts-is-not-adviced/.

- `contracts/memegoat-stake-pool.clar: 279, 403`
- `contracts/vault-v1-2.clar: 28, 67`

Classification

- CWE-829: Inclusion of Functionality from Untrusted Control Sphere[6]

Description

Besides the CR-01 and CR-03 critical issues, there are other problems with allowing a contract to call an arbitrary contract passed as a parameter, as it may be used as part of an attack. This can potentially be made with contracts outside of the scope of this audit.

Recommendation

All contract calls need to be made with whitelisted contracts or hardcoded contracts[7]. Following this recommendation would have made both CR-01 Empty Vault through Staking and CR-03 Empty Vault Through Launchpad impossible.

Status

Acknowledged. This issue is still present in the code.

Minor Severity Issues

MI-01 Broken Principals

Location

- `contracts/memegoat-launchpad.clar: 46, 293, 300, 430, 437`
- `contracts/memegoat-stake-pool.clar: 282`
- `Clarinet.toml`

Description

The principals `.memegoat-dead-wallet`, `.memegoat-treasury`, `.memegoat-treasury-v1` and `.memegoatstx` are being used in the code but not defined in the Clarinet.toml file.

The first 3 are just recipients of a transfer. The `.memegoatstx` is being used to make a `contract-call?` the following let expression in `contracts/memegoat-launchpad.clar:568` (`goat-token-trait` can only point to `.memegoatstx`).

[6] https://cwe.mitre.org/data/definitions/829.html
[7] This means that they need to be of the form `'AAAAA.bbbbbb` or `.cccccc`, no variables involved.

```
(user-goat-balance (try! (contract-call? goat-token-trait get-balance sender)))
```

But `user-goat-balance` is not being used in any part of the code (aside from the commented code in `contracts/memegoat-launchpad.clar:586`).

Having this call in the code may allow for additional security issues without any upside.

Recommendation

State the location of `.memegoat-dead-wallet`, `.memegoat-treasury` and `.memegoat-treasury-v1` in the `Clarinet.toml` file. Remove the references to `.memegoatstx` and the transitive closure of the code where it is used.

Status

Acknowledged. New unresolved principals, such as `sip-10-token` were added.

MI-02 Lack of Vault Isolation

Location

- `contracts/vault-v1-2.clar`

Classification

- CWE-653: Improper Isolation or Compartmentalization[8]

Description

The funds deposited in the vault by the different approved contracts are not distinguished, so if a single one of those contracts is compromised all the funds of all the contracts are jeopardized.

This issue made CR-01 Empty Vault through Staking more impactful.

Recommendation

Allow each of the approved contracts to only transfer away their own funds. In order to know which funds belong to which approved contract add two functions `receive-ft` and `receive-stx` with a parameter indicating the owner of the received funds.

[8] https://cwe.mitre.org/data/definitions/653.html

Status

Resolved. This issue does not apply anymore as the `contracts/vault-v1-2.clar` file was removed.

MI-03 Use of non-Whitelisted Traits

Location

- `contracts/vault-v1-2.clar: 26, 63`

Description

In the vault contract, the `get-balance` and `transfer-ft` functions receive a trait parameter (`the-token`) and then invoke a function in the corresponding contract. This may lead the contract to be used in attacks, by passing an attacking contract. A similar concept is used in the CR-01 Empty Vault through Staking issue.

Recommendation

Whitelist the tokens used in the vault contract.

Status

Resolved. This issue does not apply anymore as the `contracts/vault-v1-2.clar` file was removed.

MI-04 Use of block-height for Time Windows

Location

- `contracts/memegoat-launchpad.clar`
- `contracts/memegoat-stake-pool.clar`

Description

In both the staking contract and the launchpad contract `block-height` is used to calculate time windows. But after theNakamoto Release it will be unreliable[9].

[9] For a timeline of this release see https://docs.stacks.co/nakamoto-upgrade/nakamoto-rollout-plan.

Recommendation

Use `burn-block-height` instead, as it corresponds to Bitcoin blocks.

Status

Acknowledged. The issue is still present in the code.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Lack of Code Quality	To be audited
EN-02	No Automated Tests	Not implemented

EN-01 Lack of Code Quality

Location

- `contracts/memegoat-launchpad.clar`
- `contracts/memegoat-stake-pool.clar`
- `contracts/vault-v1-2.clar`

Description

The audited code has lots of very easily fixable coding problems including:

a. commented out code.
b. unused let mappings.
c. data variables that do not change.
d. unused private functions.
e. incorrect function descriptions.
f. unfinished functionality.
g. repeated functionality.

Having those problems impacts in several ways including:

1. adding additional attack surface for no benefit.
2. obscuring the intent of the code.
3. masking security issues and other bugs.
4. paying extra for the contract storage in the blockchain[10].

The following issues can be tracked to the problem stated here:

- CR-01 Empty Vault through Staking
- CR-02 Cannot Claim Rewards in Launchpad
- MI-01 Broken Principals

These are some examples for each of the problems stated. This is not an exhaustive list.

a. Commented code:
   - `contracts/memegoat-launchpad.clar: 248, 384`
   - `contracts/memegoat-stake-pool.clar: 185-248`
b. Unused let mappings:
   - `contracts/memegoat-launchpad.clar: 271, 272, 568, 576`
   - `contracts/memegoat-stake-pool.clar: 506, 523`
c. Data variables that do not change:
   - `contracts/memegoat-launchpad.clar: 46`
d. Unused private functions:
   - `contracts/memegoat-launchpad.clar: 692`
   - `contracts/memegoat-stake-pool.clar: 563`
e. Incorrect function descriptions:
   - `contracts/memegoat-launchpad.clar: 382`
f. Unfinished functionality:
   - The pools can be verified by the owner (see `contracts/memegoat-stake-pool.clar:168`), but its verification status is not used in the staking pool.
   - Cannot claim rewards in the launchpad. See CR-02 Cannot Claim Rewards in Launchpad.
g. Repeated functionality:
   - See `contracts/memegoat-launchpad.clar:279`, where the check-is-owner functionality is implemented again just besides the function call.

[10] In stacks, the source code of the smart contracts is stored in the blockchain.

Recommendation

Follow the EN-02 No Automated Tests recommendation and then do the code changes required to improve the quality of the code while keeping the tests passing and adding and removing tests as functionality changes.

Status

To be audited. Assessing if this enhancement was implemented requires a new audit.

EN-02 No Automated Tests

Description

There are no automated tests in the repository and commit corresponding to this audit.

Recommendation

Add automated tests. Make sure that the tests:

1. can be run with a single command.
2. are deterministic.
3. have the proper coverage of the audited contracts.

Adding the tests should be made before the rest of the fixes, in order to minimize the risk of adding new issues while doing the rest of the fixes.

Status

Not implemented. No tests were provided.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Unaudited Changes

The commit provided by the development team to check the fixes of the issues found during this audit includes significant changes that require a new audit to fully assess its security, as it includes new functionality such as the one in the `supporting-contracts/dao.clar` file.

The development team informed us that they intend to conduct a new security audit in the next month.

Upgrades

The audited contracts have no provisions to upgrade its code.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

Launchpad Contract

Owner

The owner of the contract can:

- set a new owner via the `set-contract-owner` function.
- set a new fee percentage via the `set-launchpad-fee` function.
- set the minimum memegoat balance via the `set-min-goat-balance` function[11].
- finalize a listing in Velar via the `finalize-listing-velar` function.
- make a listing request to Alex via the `make-listing-request-alex` function.
- finalize a listing in Alex via the `finalize-listing-alex` function.
- pause or resume the contract operation via the `pause` function.

The initial owner is the deployer of the contract

Staking Contract

Owner

The owner of the contract can:

[11] But this setting is not enforced. The check in `contracts/memegoat-launchpad.clar:586` is commented out. See EN-01 Lack of Code Quality.

- set a new contract owner via the `check-is-owner` function.
- pause or resume contract operations via the `pause` function.
- mark a pool as verified via the `verify-pool` function[12].

The initial owner is the deployer of the contract.

Vault Contract

Approved Contract

An approved contract can:

- withdraw fungible tokens via the `transfer-ft` function.
- withdraw STX via the `transfer-stx` function.

Initially there are no approved contracts. The set of approved contracts is managed by the contract owner.

Owner

The owner of the contract can:

- do all the actions that can be performed by an approved contract.
- set a new contract owner via the `set-contract-owner` function.
- add or remove approved contracts via the `set-approved-contract` function.
- pause or resume contract operations via the `pause` function.

The initial owner is the deployer of the contract.

About CoinFabrik

CoinFabrik is a research and development company specialized in Web3, with a strong background in cybersecurity. Founded in 2014, we have worked on over 500 decentralization projects, including EVM-based and other platforms like Solana, Algorand, and Polkadot. Beyond development, we offer security audits through a dedicated in-house team of senior cybersecurity professionals, working on code in languages such as Substrate, Solidity, Clarity, Rust, TEAL, and Stellar Soroban.

Our team has an academic background in computer science, software engineering, and mathematics, with accomplishments including academic publications, patents turned into

[12] But this setting is not enforced. See CR-01 Empty Vault through Staking and EN-01 Lack of Code Quality

products, and conference presentations. We actively research in collaboration with universities worldwide, such as Cornell, UCLA, and École Polytechnique in Paris, and maintain an ongoing collaboration on knowledge transfer and open-source projects with the University of Buenos Aires, Argentina. Our management and people experience team has extensive expertise in the field.

Methodology

CoinFabrik was provided with the source code. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Severity Classification

Security risks are classified as follows:

Critical	These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
High	These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.

Medium	These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
Minor	These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**

Issue Status

An issue detected by this audit has one of the following statuses:

- **Unresolved:** The issue has not been resolved.
- **Resolved:** Adjusted program implementation to eliminate the risk.
- **Partially Resolved:** Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Acknowledged:** The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Mitigated:** Implemented actions to minimize the impact or likelihood of the risk.
- **To be audited**: The issues' fix needs to be assessed in a future audit.

Disclaimer

This audit report has been conducted on a **best-effort basis within a tight deadline defined by time and budget constraints**. We reviewed only the specific smart contract code provided by the client at the time of the audit, detailed in the Scope section, which does not include other components like front-end, back-end or data management.

While we have employed the latest tools, techniques, and methodologies to identify potential vulnerabilities, **this report does not guarantee the absolute security of the contracts, as undiscovered vulnerabilities may still exist.** Our findings and recommendations are

suggestions to enhance security and functionality and are not obligations for the client to implement.

The results of this audit are valid solely for the code and configurations reviewed, and any modifications made after the audit are outside the scope of our responsibility. CoinFabrik disclaims all liability for any damages, losses, or legal consequences resulting from the use or misuse of the smart contracts, including those arising from undiscovered vulnerabilities or changes made to the codebase after the audit.

This report is intended exclusively for the **MemeGoat project** and should not be relied upon by any third party without the explicit consent of CoinFabrik. Blockchain technology and smart contracts are inherently experimental and involve significant risk; users and investors should fully understand these risks before deploying or interacting with the audited contracts.

Changelog

Date	Description
2024-09-13	Initial report based on commit `e18e2b45a91504ff1304bdb4f2d1efeef1196d9b`.
2024-10-14	Fixes checked on commit `b805c9368592cb343e19c81a2b370fd027d3327a`.