G.O.A.T.AI Treasury

Documentation for Security Audit Request

Summary

Name:
G.O.A.T.AI (GOAT Athletics Inc.) Treasury

Contract:
GOATATreasury

Contract Types:
- Custom Upgradeable, Pausable

Solidity Version:
0.8.28

Blockchain Network on which contracts will be deployed:
Base chain - Ethereum Layer 2.

Upgradeability:
Upgradeable

License:
MIT

GitHub Repository:
https://github.com/GOAT-Athletics-Inc/contracts

Security Contact:
dev@goatathletics.ai

Project Overview

Project Description

GOAT Athletics Inc, from here on called G.O.A.T.AI, is a charity, sports and meme governance token. The G.O.A.T.AI token powers a running team and a charity.

Team G.O.A.T.AI is an official team of RunGP, a brand new competitive global running league. The team is owned by our community of token holders. G.O.A.T.AI raises funds for the Global Running Foundation (GRF), the charitable arm of RunGP, from buy and sell trading fees.

The GOATAI token is an ERC20 token, which charges buy and sell trading fees. The collected fees go to GRF for charity projects, and to G.O.A.T.AI Operations and Marketing.

For the operational purpose of streamlining the withdrawal of the fees for both G.O.A.T.AI and GRF, the smart contract **GOATAITreasury** has been created (file: **contracts/GOATAITreasury.sol**). Both GRF and G.O.A.T.AI teams' have set up multisigs to receive funds.

Periodic swapping and withdrawal of tokens via multisig's can be very cumbersome, inefficient and time-consuming, if done manually, and SaaS solutions normally do not support fee-on-transfer tokens and outright remove them from the list of supported tokens, even if the recipient address is exempt from such fees. Moreover GRF as a foundation will have highly simplified accounting by receiving funds directly in stablecoins.

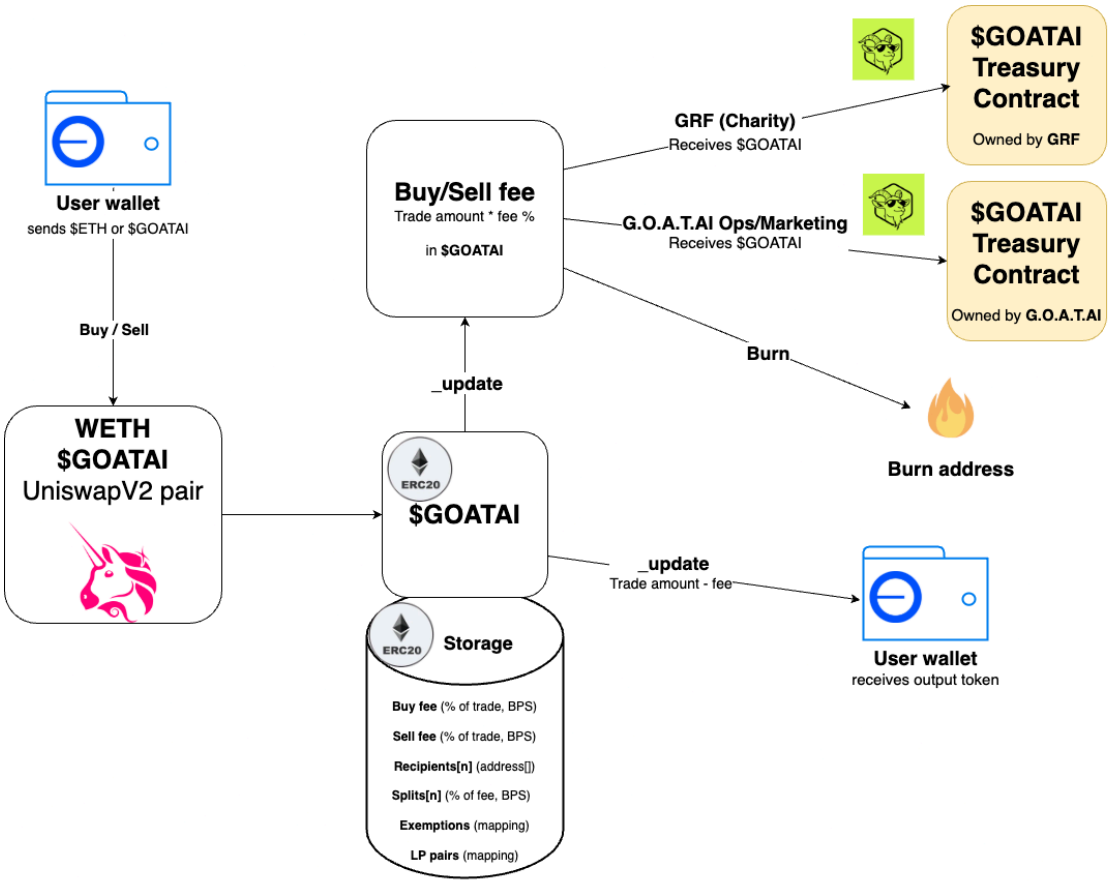
It is foreseen by the withdrawal with swap function to be executable by an Executor account, whose purpose is solely to execute the transaction and pay for the gas. The funds will never transit via the Executor account and will only go to the recipient. The Executor account role can be given to a cloud-based function, programmed to run periodically (e.g. every hour or every few hours) and withdraw funds to the recipient.

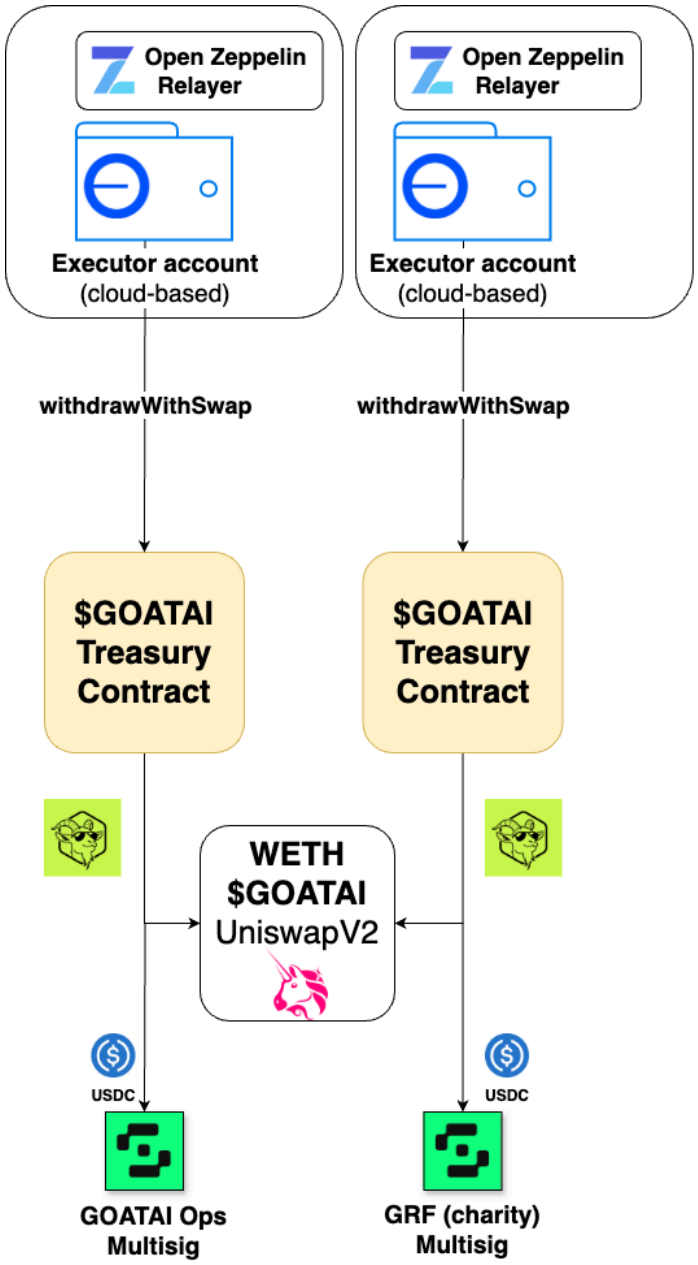
GOATAITreasury is **meant for internal use** for the GRF and G.O.A.T.AI teams and will not be opened to public use. Supporting any other token aside from $GOATAI and stablecoins on Uniswap is not a requirement.

Core Features

1. **Withdrawal of $GOATAI token to recipient address**
    - Withdrawal via swap to output ERC20 token (initially set to USDC, and probably not changed for foreseeable future)
    - Direct Withdrawal to recipient address - backup function, which will be used
2. **Withdrawal Parameters management**
    - Set recipient address
    - Set base & output tokens (not foreseen to be called, but available methods if necessary)
    - Set router address (not foreseen to be called, but available methods if necessary)
3. **Pause/Unpause**
4. **Contract Upgrade(s)**

Architecture




Open Zeppelin Relayer
Executor account (cloud-based)
Open Zeppelin Relayer
Executor account (cloud-based)
withdrawWithSwap
withdrawWithSwap
$GOATAI Treasury Contract
$GOATAI Treasury Contract
WETH
$GOATAI
UniswapV2
USDC
USDC
GOATAI Ops Multisig
GRF (charity) Multisig

Core Components

GOATAITreasury Contract

- Withdrawal of $GOATAI tokens (baseToken) swapped to stablecoin (outputToken) on UniswapV2
- Direct Withdrawal of $GOATAI tokens
- Controls access through role-based system
- Pausability
- Upgradeability

Modules used: OpenZeppelin Contracts Upgradeable v5.2.0

- OpenZeppelin Contracts Upgradeable v5.2.0
  - Initializable
  - AccessControlUpgreadeable
  - PausableUpgradeable
  - ReentrancyGuardUpgradeable
  - UUPSUpgradeable
- UniswapV2 Router

External Integrations

$GOATAI Token

The $GOATAI Token has already been audited by QuillAudits

UniswapV2

WETH / GOATAI liquidity is deployed to Uniswap as a V2 pool to support the fee-on-transfer nature of the GOATAI token.

Roles and Permissions

GOATATreasury's permission model is based on OpenZeppelin's AccessControl. Here is a breakdown of the roles and their permissions.

EXECUTOR_ROLE

ExecutorsFee managers will be capable of setting:

- Buy fee - as a portion of the total trade
- Sell fee - as a portion of the total trade
- Recipient addresses & number of recipients
- Fee Split for each of the recipients
- Burn amount - implicitly - as a remainder of the fee splits for all other recipients

This role will be initially taken by the team's multisig and will be eventually moved to the DAO.

DEFAULT_ADMIN_ROLE

Admins will be capable of:

- Upgrading the contract
- Granting and revoking other accounts the role of *DEFAULT_ADMIN_ROLE*
- Granting and revoking other accounts the role of *EXECUTOR_ROLE*
- Update swap parameters:
    - recipient address
    - Base token address (method available just in case, but base token should stay $GOATAI token address)
    - Output token address
    - Uniswap V2 Router address - this is not expected to change

This role will be initially taken by the team's multisig and will be eventually moved to the DAO.

Recipient

In addition to these, the recipient has no formal role in the access control model - although it is initially given the DEFAULT_ADMIN role.
The recipient will be the receiving address of the token withdrawal operations.

Considerations

Known Assumptions & Risks

- **Potential Attack Vectors:**
    - Compromising Admin account. Mitigation
        - usage of Safe multisig as Admin account & recipient
    - Compromise of Executor account. This might cause unfavorable trades. Mitigations:
        - Maximum slippage hard-coded to 10%
        - Limited privilege of executor role, which can easily be revoked.
    - Setting malicious tokens as base token or output token
        - We consider this risk to be very limited, as the internal teams will be the only ones accessing the contract.

Testing and Coverage

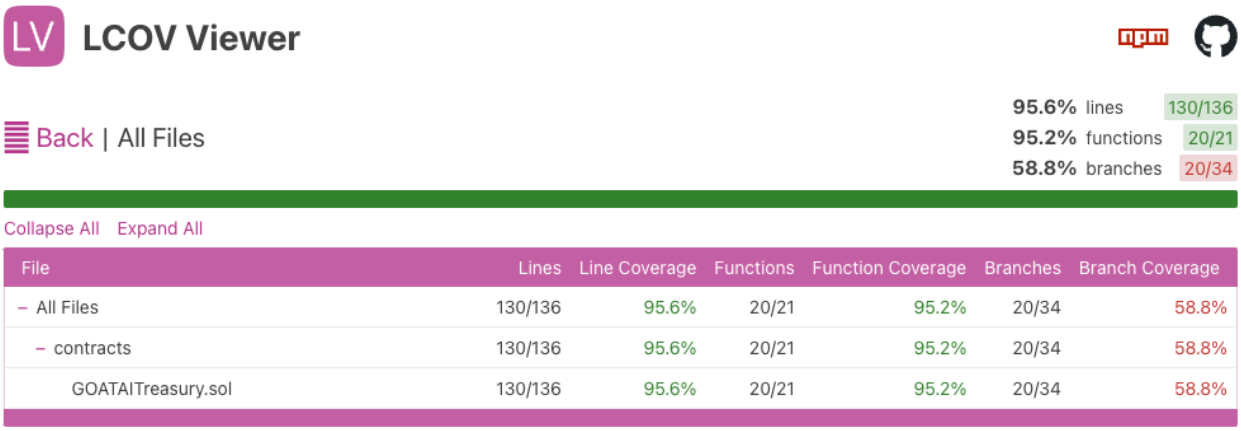
Checks performed

✅ Ran **Forge unit-tests & forge coverage:**

```
forge coverage \
  --report lcov \
  --report summary \--no-match-coverage "(script|test)"
```

Lines	Statements	Branches	Functions
95.59% (130/136)	89.22% (149/167)	58.82% (20/34)	95.24% (20/21)

The lower coverage in the branches is largely due to hard to replicate Swap Failures.



✅ Ran **Slither** (static analysis)

No results, excluding issues relating to OpenZeppelin modules, and informational issues due to casing or assembly.

Audit Goals

What We Want Auditors to Focus On:

- Ensure security of contract, focusing on internal usage by teams, rather than general use by others and other tokens.
- Assess security of architectural setup.

Additional Notes:

The Cloud-based function which will trigger the withdrawal with swap is outside of the scope of this audit, although high level architectural considerations could be considered.

Already audited modules and contract templates, as well as GOATAI_ERC20 are to be considered outside the audit scope. Although their integration in the GOATAI_ERC20 contract and in the Governance architecture can be considered within scope. These are the modules and integrations we are referring to for exclusion:

- All OpenZeppelin contracts and modules used for GOATAITreasury
- GOATAI_ERC20