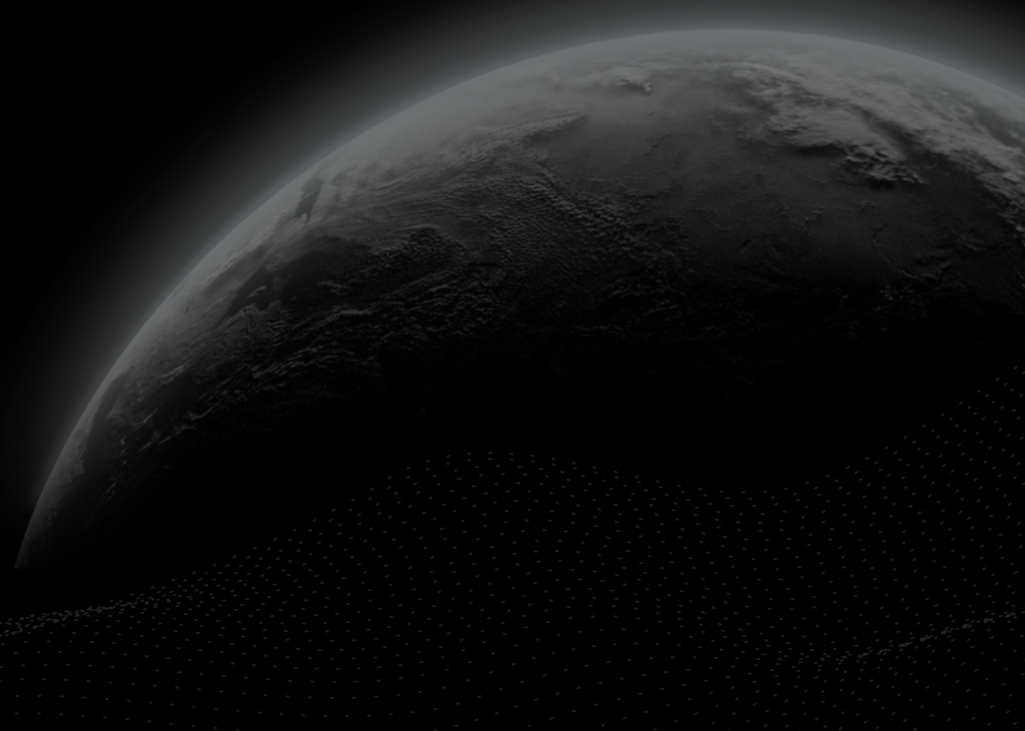
CERTIK

Security Assessment

Tenderize

CertiK Verified on Feb 15th, 2023



CertiK Verified on Feb 15th, 2023

Tenderize

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi, Staking	Ethereum | Other	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 02/15/2023	N/A

CODEBASE

https://github.com/Tenderize/tender-core/tree/ea06140e3eddf18f3e02292793c9865605c40c4b

...View All

COMMITS

ea06140e3eddf18f3e02292793c9865605c40c4b

...View All

Vulnerability Summary

15	2	0	0	13	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	1 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
7	Minor	1 Resolved, 6 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	4 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | TENDERIZE

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Review Notes

Findings

CON-01 : Centralization Risks

CON-02 : Centralized Control of Contract Upgrade

CON-03 : Incompatibility with Deflationary Tokens

MTB-01 : Payable Function Using `delegatecall` Inside a Loop

CON-04 : `initialize()` Is Unprotected

CON-05 : Unknown implementations

INE-01 : Unused Return Value

LTB-01 : Potential Sandwich Attacks

TED-01 : Missing Zero Address Validation

TED-02 : Lacking Share Calculation In `Audius` And `Livepeer`

TTB-01 : `MAX_FEE` is not applied in the function `_initialize()`

CON-06 : Missing Error Messages

CON-07 : Missing Emit Events

TER-01 : Token Economy

TFT-01 : Potential Reentrancy Attack

Optimizations

CON-08 : Function Could Be Declared External

Formal Verification

Considered Functions And Scope

Verification Results

Appendix

Disclaimer

CODEBASE | TENDERIZE

Repository

https://github.com/Tenderize/tender-core/tree/ea06140e3eddf18f3e02292793c9865605c40c4b

Commit

ea06140e3eddf18f3e02292793c9865605c40c4b

AUDIT SCOPE | TENDERIZE

29 files audited ● 12 files with Acknowledged findings ● 17 files without findings

ID	File	SHA256 Checksum
MTB	contracts/helpers/Multicall.sol	5938a0dd12b441205d7082f225d13819eddcaa2f84b265f12c652a504df783f7
RTB	contracts/helpers/Registry.sol	817904805290e6f2218e09465ed157aa058ee22293a8995bc54d80bc5cec2841
TFT	contracts/tenderfarm/TenderFarm.sol	77db5dd9d92ddd1bf72e09c991450f3de5b82969df5cc1e5f6ccce69b4bf072b
ATB	contracts/tenderizer/integrations/audius/Audius.sol	73bd714349c32083f3e6d8617121849b18d214f6408665bda986a37c2005752f
GTB	contracts/tenderizer/integrations/graph/Graph.sol	c1675cd48d35578de74453d1158d4fd23215baa7a8e9f74829e2ece48de83353
LTB	contracts/tenderizer/integrations/livepeer/Livepeer.sol	4b44dfd906f1c08265c140010de05b5594a9fad102d04805c4e4ad495e44a20a
MTU	contracts/tenderizer/integrations/matic/Matic.sol	247d41ee20819582f5fad26aea18e4caef8be133521678f229479b7736ff6e84
TTB	contracts/tenderizer/Tenderizer.sol	d94ffc9a3dfbdbd1e49e0804d9f57332073557cb9deeca68be3131f7a08b3eb8
LPT	contracts/tenderswap/LiquidityPoolToken.sol	788cb0cb550900a21e34684b7bebcc7dd1dd58fd5a3da1ce4f4f45fbeec4d933
SUT	contracts/tenderswap/SwapUtils.sol	7e958e0b2f936efe98c83e43aaa8c127e79af9ce116b0a48be458eabd5e12b54
TST	contracts/tenderswap/TenderSwap.sol	566c5c42f897671143b24f5f1bd4d933e9f86731cee1525e2a102284530b53d5
TER	contracts/token/TenderToken.sol	bb480710e8d9c04fa01cce01e2ebefa92dd8bf28d29a5f035af0a3da426c9114
SPT	contracts/helpers/SelfPermit.sol	2ad7a2faa3aaec0301a9ccdb0712e411b5dbe2a65dc45e425627349ab77ef846
ISR	contracts/interfaces/ISwapRouter.sol	216ab8c57f00c0dfa011c48e270190881ab9242aafca64cc50df8b93b626862c

ID	File	SHA256 Checksum
IWE	contracts/interfaces/IWETH.sol	4fbbfd9c2f2525cdf36bfc40f11a567b6ba5077915113e45ed0add11cfa12e36
MUT	contracts/libs/MathUtils.sol	7d8686c33756e262e272ea2bbc84a89b6cef07c561d05f577e6ab41a4d0d12a2
ITF	contracts/tenderfarm/ITenderFarm.sol	341f9c2cc797a46956fc1f8bbf00592376a25a51867b4f9f5f7484957e4c147b
TFF	contracts/tenderfarm/TenderFarmFactory.sol	d0ef0d368f47748e40ea395402c83cefb580ba091a4a020e39377eec6fbcab61
IAT	contracts/tenderizer/integrations/audius/IAudius.sol	a0117043c27f06c20722b4efb344d35df0d9db7add184991154669351354874e
IGT	contracts/tenderizer/integrations/graph/IGraph.sol	cac6dcc0ae7d3a6769c18d3362eb98e8853500b8aa47e5d37a10fa92f9eddc8d
ILT	contracts/tenderizer/integrations/livepeer/ILivepeer.sol	9ec4df0be721f1859998172665eb7b528db6b3279c2e75120874faae2e3ee812
IMT	contracts/tenderizer/integrations/matic/IMatic.sol	1b9e72549d6c5df71c365d2734ff2a42525d7cfe5a29a4ba8854e315a4b9b3c5
ITT	contracts/tenderizer/ITenderizer.sol	4f18d606f6ef36f26462d03bd4f4181d70c42debba03ceb453607dd2741befed
ITS	contracts/tenderizer/ITotalStakedReader.sol	1abd6b5a04804075f5a01691afc5d9c59bbe4545684d45bd0a6706f22ea85ef7
WLT	contracts/tenderizer/WithdrawalLocks.sol	85675505da89db2ad4294efde96bd7a8688984a78598f63311f3f53067370e0b
WPT	contracts/tenderizer/WithdrawalPools.sol	70a4b14813f34b079e2b13f1222ac23f7c6610a71e882eac5e89d270d98310b2
IST	contracts/tenderswap/ITenderSwap.sol	354e60b01f386a6bae2ad4cf4d59477616c63debc41601b3387c098f394140a5
TSF	contracts/tenderswap/TenderSwapFactory.sol	55b83ef8438ed81249aac2cf5bd221dd35f6e32a66db83f176fbfbb4653a9b64
TTT	contracts/token/ITenderToken.sol	7e0632156c9942c1ed652ca5b558d8cc1c2bb4f2c8db9cf6fe6753c955138371

APPROACH & METHODS | TENDERIZE

This report has been prepared for Tenderize to discover issues and vulnerabilities in the source code of the Tenderize project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | TENDERIZE

The Tenderize protocol enables liquid staking and yield aggregation for various web3 protocols. It provides end-users a way to earn automatically compounding staking rewards without locking up capital or having to keep close track of several stake delegation markets.

Third Party Dependency

The protocol is serving as the underlying entity to interact with one or more third-party protocols, `Audius`, `Graph`, `Livepeer`, `Matic` and `Uniswaprouter` etc. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

We understand that business logic requires interaction with third-party staking/swap protocols. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

The Tenderize team confirmed they do monitor for changes and keep close tabs on contract/protocol upgrades that are pending in the governance processes of said protocols.

FINDINGS | TENDERIZE

15	0	2	2	7	4
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Tenderize. Through this audit, we have uncovered 15 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-01	**Centralization Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
CON-02	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	**Acknowledged**
CON-03	Incompatibility With Deflationary Tokens	Logical Issue	Medium	Resolved
MTB-01	Payable Function Using `delegatecall` Inside A Loop	Volatile Code	Medium	Acknowledged
CON-04	`initialize()` Is Unprotected	Volatile Code	Minor	Acknowledged
CON-05	Unknown Implementations	Volatile Code	Minor	Acknowledged
INE-01	Unused Return Value	Volatile Code	Minor	Acknowledged
LTB-01	Potential Sandwich Attacks	Logical Issue	Minor	Acknowledged
TED-01	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
TED-02	Lacking Share Calculation In `Audius` And `Livepeer`	Logical Issue	Minor	Resolved
TTB-01	`MAX_FEE` Is Not Applied In The Function `_initialize()`	Volatile Code	Minor	Acknowledged

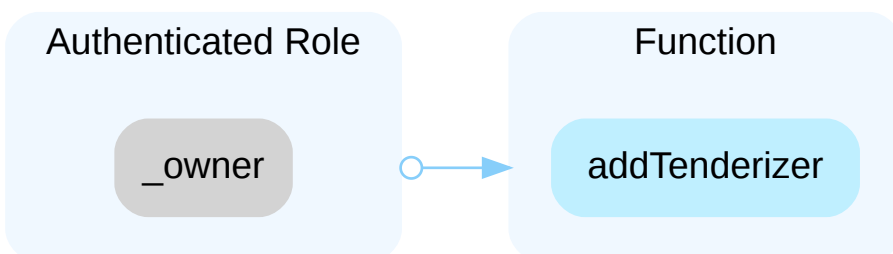
ID	Title	Category	Severity	Status
CON-06	Missing Error Messages	Coding Style	Informational	Acknowledged
CON-07	Missing Emit Events	Coding Style	Informational	Acknowledged
TER-01	Token Economy	Logical Issue	Informational	Acknowledged
TFT-01	Potential Reentrancy Attack	Volatile Code	Informational	Acknowledged

CON-01 | CENTRALIZATION RISKS

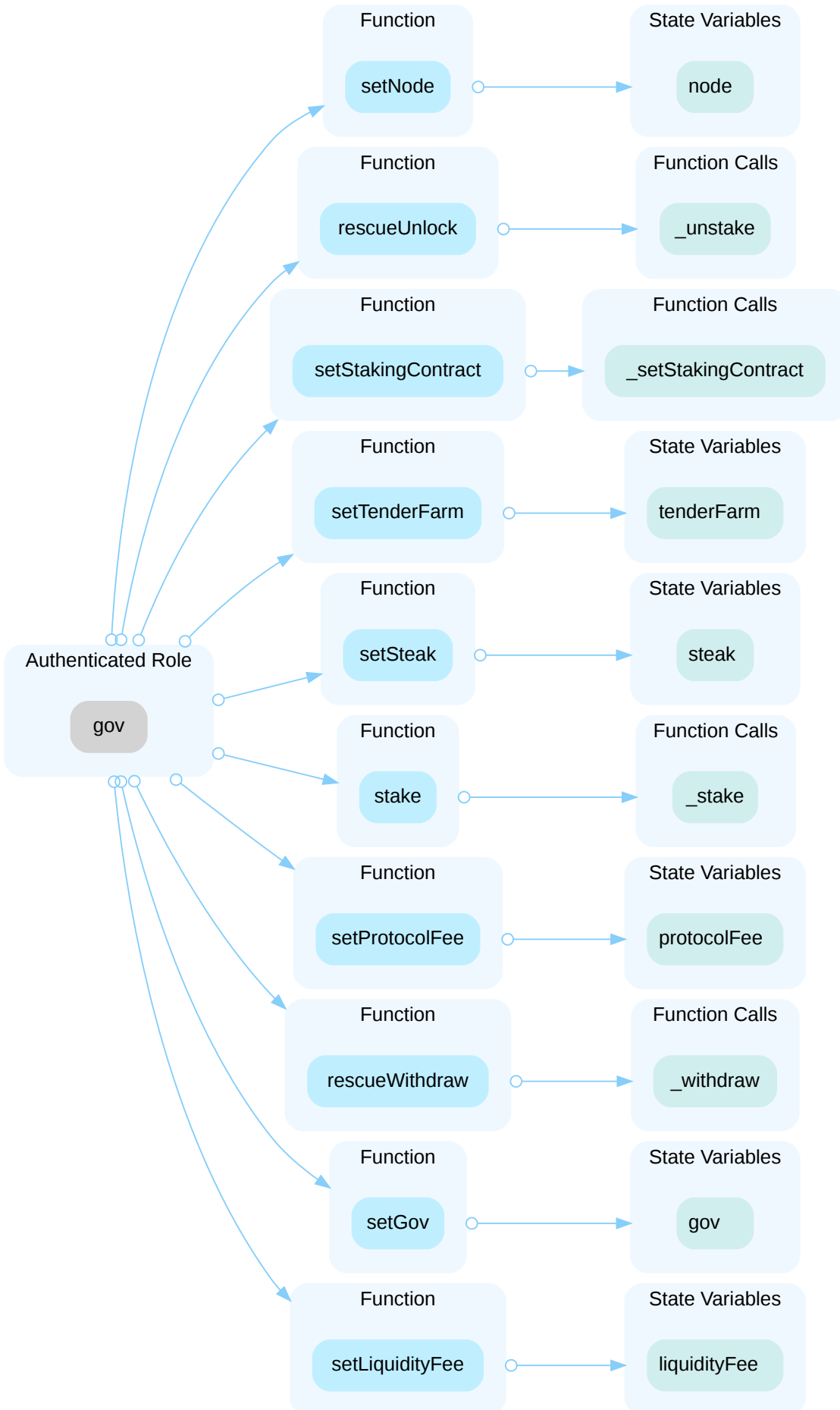
Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/helpers/Registry.sol; contracts/tenderizer/Tenderizer.sol; contracts/tenderizer/integrations/audius/Audius.sol; contracts/tenderizer/integrations/graph/Graph.sol; contracts/tenderizer/integrations/livepeer/Livepeer.sol; contracts/tenderizer/integrations/matic/Matic.sol; contracts/tenderswap/LiquidityPoolToken.sol; contracts/tenderswap/TenderSwap.sol; contracts/token/TenderToken.sol	Acknowledged

Description

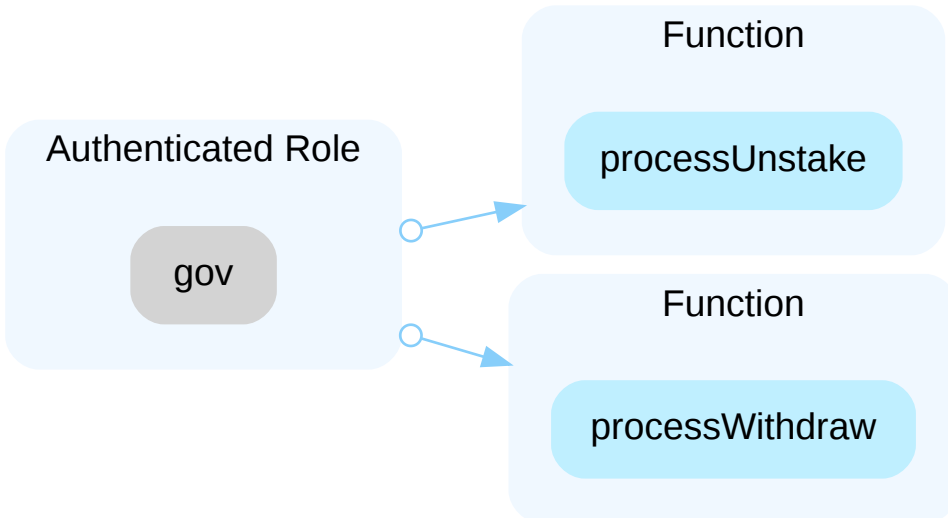
In the contract `Registry` the role `_owner` has authority over the functions shown in the diagram below.



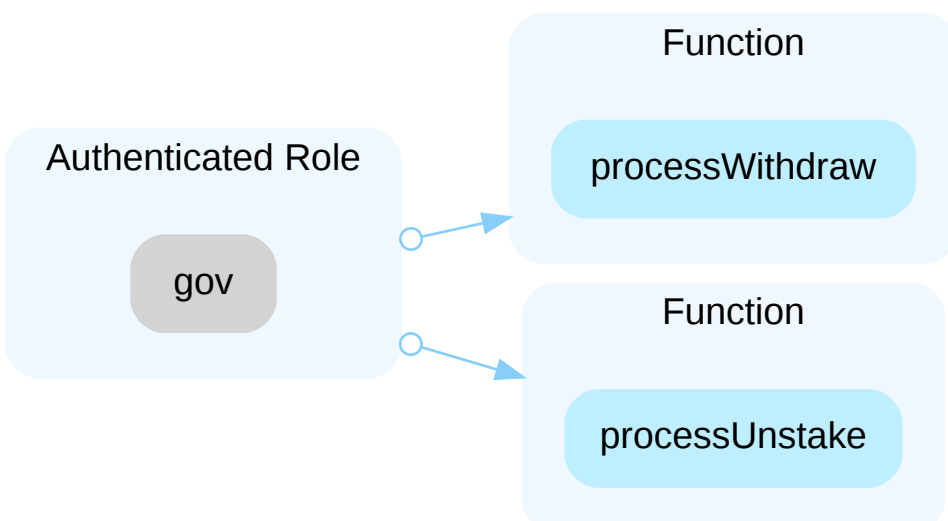
In the contract `Tenderizer` the role `gov` (contract deployer) has authority over the functions shown in the diagram below.


Function
setNode
State Variables
node
Function
rescueUnlock
Function Calls
_unstake
Function
setStakingContract
Function Calls
_setStakingContract
Function
setTenderFarm
State Variables
tenderFarm
Function
setSteak
State Variables
steak
Authenticated Role
gov
Function
stake
Function Calls
_stake
Function
setProtocolFee
State Variables
protocolFee
Function
rescueWithdraw
Function Calls
_withdraw
Function
setGov
State Variables
gov
Function
setLiquidityFee
State Variables
liquidityFee

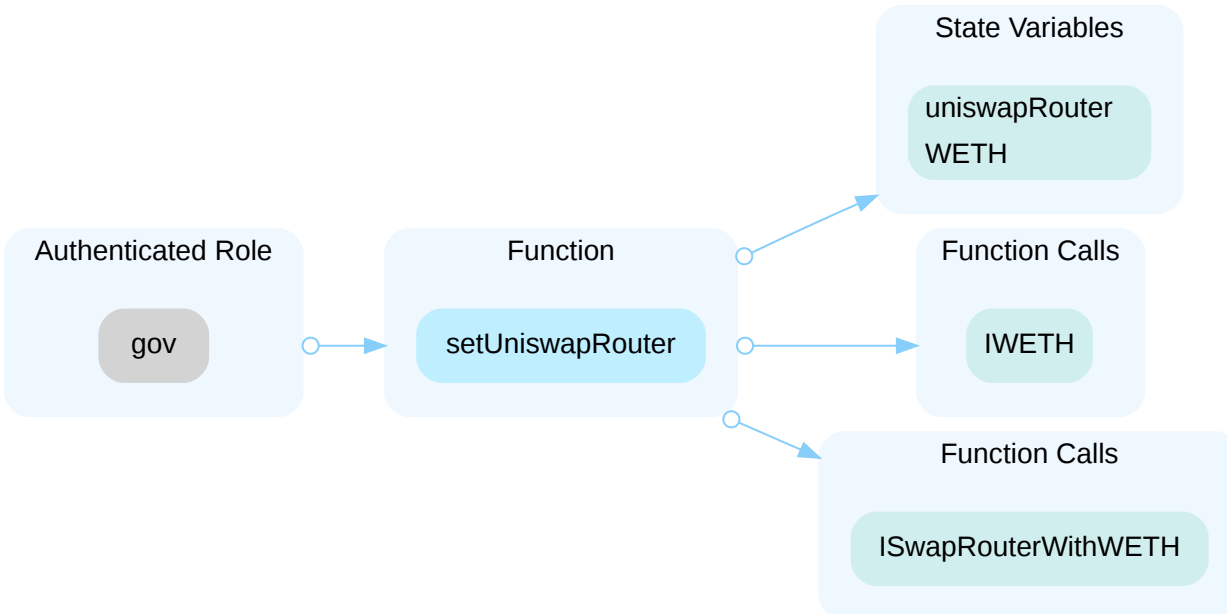
In the contract `Audius` the role `gov` has authority over the functions shown in the diagram below.



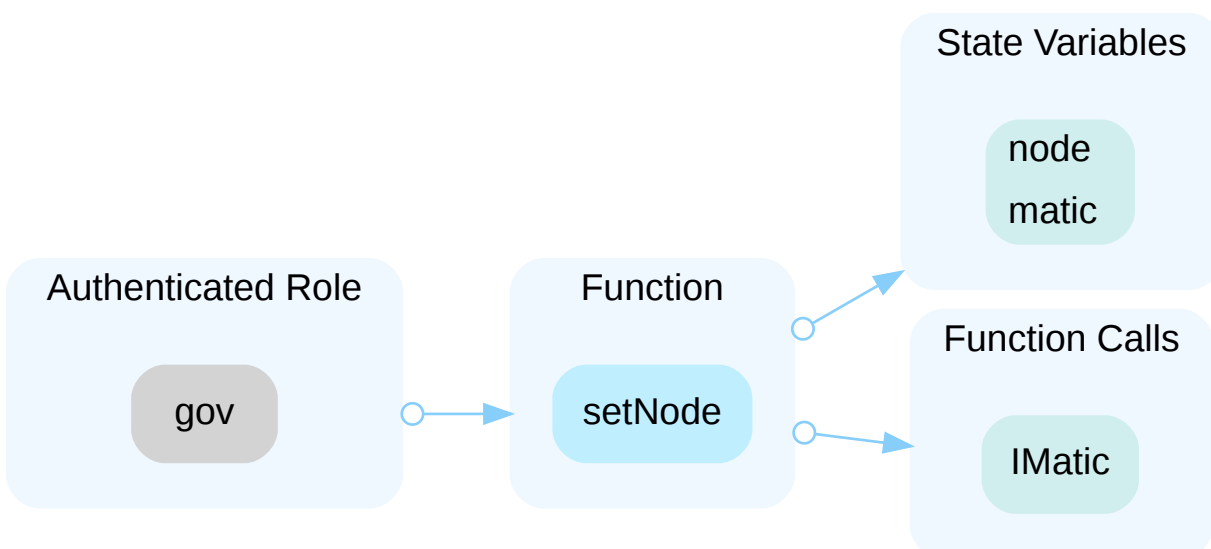
In the contract `Graph` the role `gov` has authority over the functions shown in the diagram below.



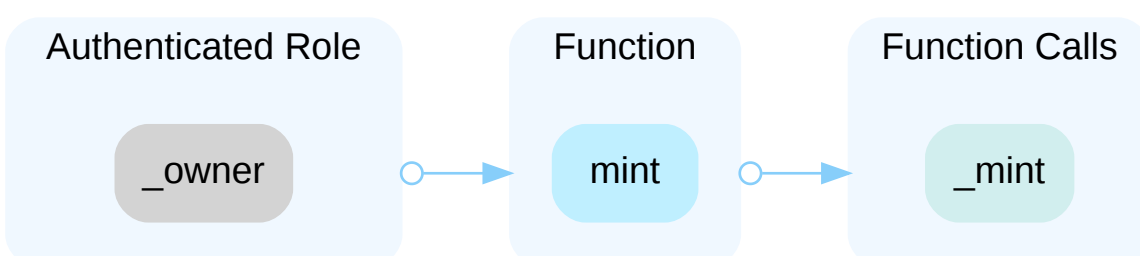
In the contract `Livepeer` the role `gov` has authority over the functions shown in the diagram below.



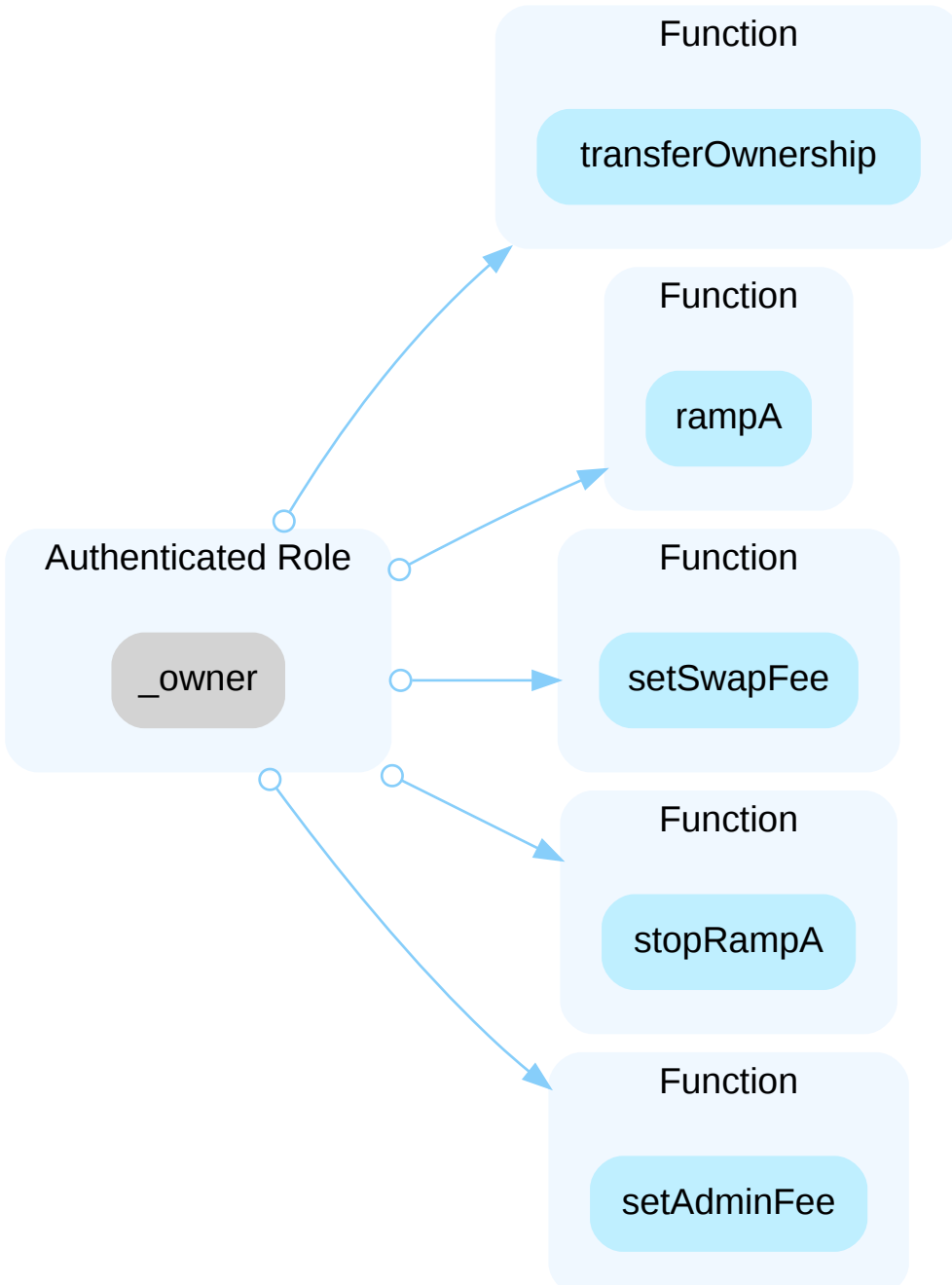
In the contract `Matic` the role `gov` has authority over the functions shown in the diagram below.



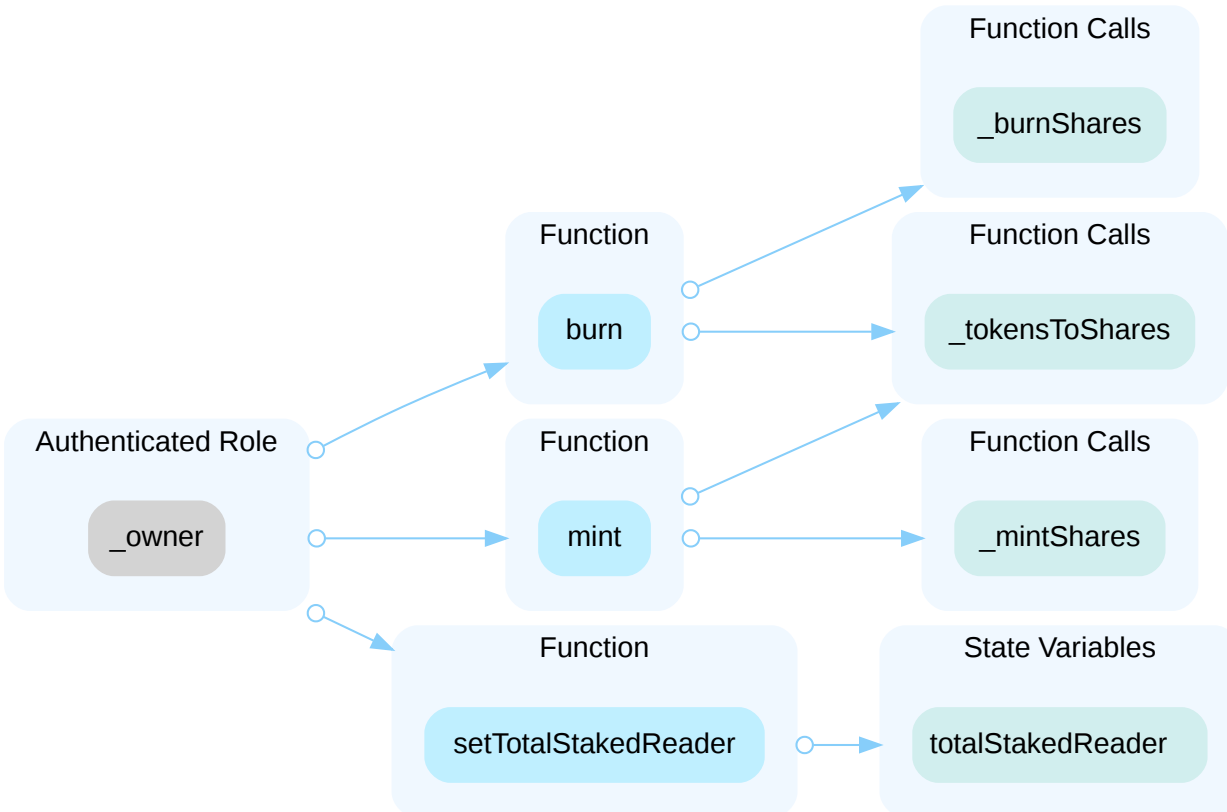
In the contract `LiquidityPoolToken` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `TenderSwap` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `TenderToken` the role `_owner` has authority over the functions shown in the diagram below.



Specifically, in the `Tenderizer` contract and its inheritance contracts(`Audius` , `Graph` , `Livepeer` , and `Matic`), the role `gov` can unstake and withdraw all the contract assets using functions `rescueUnlock()` and `rescueWithdraw()` .

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority and users' assets may suffer loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[Tenderize]`: They will use Multisignature or governance contract to control all the owner functions.

CON-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/tenderswap/LiquidityPoolToken.sol: 11; contracts/tenderswap/TenderSwap.sol: 33; contracts/token/TenderToken.sol: 25**	**Acknowledged**

Description

`LiquidityPoolToken`, `TenderSwap`, and `TenderToken` are upgradeable contracts, the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[Tenderize]`: They will use Multisignature or governance contract to control all the owner functions.

CON-03 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/tenderfarm/TenderFarm.sol: 76, 88, 93, 111, 113, 133, 135; contracts/tenderizer/Tenderizer.sol: 92, 105, 209, 211; contracts/tenderizer/integrations/audius/Audius.sol: 55; contracts/tenderizer/integrations/graph/Graph.sol: 57, 61; contracts/tenderizer/integrations/livepeer/Livepeer.sol: 64; contracts/tenderizer/integrations/matic/Matic.sol: 68	Resolved

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived at the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

```
76          _farmFor(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `require(token.transferFrom(msg.sender, address(this), _amount), "TRANSFERFROM_FAIL");`

```
76          _farmFor(msg.sender, _amount);
```

- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `nextTotalStake += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
88          _farmFor(msg.sender, _amount);
```

- Transferring tokens by `_amount`.

- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `require(token.transferFrom(msg.sender, address(this), _amount), "TRANSFERFROM_FAIL");`

```
88          _farmFor(msg.sender, _amount);
```

- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `nextTotalStake += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
93          _farmFor(_for, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `require(token.transferFrom(msg.sender, address(this), _amount), "TRANSFERFROM_FAIL");`

```
93          _farmFor(_for, _amount);
```

- This function call executes the following operation.
- In `TenderFarm._farmFor`,
    - `nextTotalStake += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
113         require(rewardToken.transferFrom(msg.sender, address(this), _amount),
"TRANSFER_FAILED");
```

- Transferring tokens by `_amount`.

```
111         uint256 shares = rewardToken.tokensToShares(_amount);
```

- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `tokensToShares` is an external function and its behavior wasn't evaluated.

```
92          _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `steak.safeTransferFrom(_for, address(this), _amount);`

```
92          _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `_deposit(_for, _amount);`
- In `Audius._deposit`,
    - `currentPrincipal += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
92          _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `steak.safeTransferFrom(_for, address(this), _amount);`

```
92          _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `_deposit(_for, _amount);`

- In `DummyTenderizer._deposit`,
  - `currentPrincipal += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
92          _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
  - `steak.safeTransferFrom(_for, address(this), _amount);`

```
92          _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
  - `_deposit(_for, _amount);`
- In `Graph._deposit`,
  - `currentPrincipal += _calcDepositOut(_amount);`
- In `Graph._calcDepositOut`,
  - `return _amountIn - ((uint256(graph.delegationTaxPercentage()) * _amountIn) / MAX_PPM);`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
92          _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
  - `steak.safeTransferFrom(_for, address(this), _amount);`

```
92          _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `_deposit(_for, _amount);`
- In `Livepeer._deposit`,
    - `currentPrincipal += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
92          _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `steak.safeTransferFrom(_for, address(this), _amount);`

```
92          _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `_deposit(_for, _amount);`
- In `Matic._deposit`,
    - `currentPrincipal += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
105         _depositHook(msg.sender, _amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `steak.safeTransferFrom(_for, address(this), _amount);`

```
105         _depositHook(msg.sender, _amount);
```

- This function call executes the following operation.
- In `Tenderizer._depositHook`,
    - `_deposit(_for, _amount);`
- In `Audius._deposit`,
    - `currentPrincipal += _amount;`
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

`[Tenderize]`: Issue acknowledged and this is by design. The tokens are not deflationary tokens upon transfer. While slashing in said networks could cause deflation, it is handled through processing rewards.

MTB-01 | PAYABLE FUNCTION USING `delegatecall` INSIDE A LOOP

Category	Severity	Location	Status
Volatile Code	Medium	contracts/helpers/Multicall.sol: 22	Acknowledged

Description

`delegatecall` is used inside a loop in a payable function. If the called function uses `msg.value`, the incoming payment may be processed multiple times unexpectedly.

```
22              (bool success, bytes memory result) =
address(this).delegatecall(_data[i]);
```

Recommendation

We advise making sure that the function called by `delegatecall` is not payable or does not use `msg.value`.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

CON-04 | `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Volatile Code	Minor	contracts/tenderfarm/TenderFarm.sol; contracts/tenderizer/integrations/audius/Audius.sol; contracts/tenderizer/integrations/graph/Graph.sol; contracts/tenderizer/integrations/livepeer/Livepeer.sol; contracts/tenderizer/integrations/matic/Matic.sol; contracts/tenderswap/LiquidityPoolToken.sol; contracts/tenderswap/TenderSwap.sol; contracts/token/TenderToken.sol	Acknowledged

Description

The function `initialize()` is `public` and can be called by anyone as long as the contract is deployed.

Recommendation

We recommend adding a `_disableInitializers()` function similar to Openzeppelin's or using `constructor() initializer {}`.

```
    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() initializer {}
```

This will prevent the calling of `initialize()` directly on the implementation contract. But the proxy will still be able to `initialize()` its storage variables.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

CON-05 | UNKNOWN IMPLEMENTATIONS

Category	Severity	Location	Status
Volatile Code	Minor	contracts/tenderfarm/TenderFarm.sol: 24, 29, 34; contracts/tenderizer/Tenderizer.sol: 29, 31; contracts/tenderswap/TenderSwap.sol: 64, 65; contracts/token/TenderToken.sol: 46	Acknowledged

Description

There are several unknown tokens or changeable contract implementations in this protocol:

- TenderFarm:

1. L24 `IERC20 public token`
2. L29 `ITenderToken public rewardToken`
3. L34 `ITenderizer public tenderizer`

- Tenderizer:

1. L29 `IERC20 public steak`
2. L31 `ITenderFarm public tenderFarm`

- TenderSwap:

1. L64 `IERC20 _token0`
2. L65 `IERC20 _token1`

- TenderToken:

1. L46 `ITotalStakedReader public totalStakedReader`

The scope of the audit treats these entities as black boxes and assumes their functional correctness.

Recommendation

We recommend ensuring the deployed contract addresses are correct. Also, ensure that the contract implementations can meet the requirement.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

INE-01 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/tenderizer/integrations/audius/Audius.sol: 99, 115; contracts/tenderizer/integrations/graph/Graph.sol: 115, 138	Acknowledged

Description

The return value of an external call is not stored in a local or state variable.

```
99          audius.requestUndelegateStake(node_, amount);
```

```
115         audius.undelegateStake();
```

```
115         graph.undelegate(node_, shares);
```

```
138         graph.withdrawDelegated(node, address(0));
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

`[Tenderize]`: The value returned isn't required to check.

LTB-01 | POTENTIAL SANDWICH ATTACKS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/tenderizer/integrations/livepeer/Livepeer.sol: 152~170	Acknowledged

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction is attacked) a transaction to sell the asset.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- `Livepeer._claimSecondaryRewards()`

There are todo comments in the code mentioning setting max slippage to 5%, but it is not implemented in the code and the `amountOutMin` is still 0.

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Alleviation

`[Tenderize]`: This is mainly done for gas cost savings, since the amounts that are swapped are always quite small. They will fix it in the next major protocol version.

TED-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/tenderizer/Tenderizer.sol: 158, 163; contracts/tenderizer/integrations/matic/Matic.sol: 56	Acknowledged

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
158         gov = _gov;
```

- `_gov` is not zero-checked before being used.

```
163         node = _node;
```

- `_node` is not zero-checked before being used.

```
56          maticStakeManager = _matic;
```

- `_matic` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

`[Tenderize]`: These addresses can only be zero due to misconfiguration. They decided to not check for gas cost savings.

TED-02 | LACKING SHARE CALCULATION IN `Audius` AND `Livepeer`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/tenderizer/Tenderizer.sol; contracts/tenderizer/integrations/audius/Audius.sol; contracts/tenderizer/integrations/livepeer/Livepeer.sol	Resolved

Description

In the staking contracts, the amount of staking benefits or losses is recorded in the `int256` type variable `rewards`, which is calculated by the function `_processNewStake()`. Then the `currentPrincipal` will be updated based on the value of `rewards`. However, the calculation of the unstake/withdraw amount does not consider the change of the `currentPrincipal`. Thus, the staked user may only get the staking principle with no benefits.

This issue happens in contracts `Audius` and `Livepeer`. These two contracts do not have the logic to calculate the user's balance with the logic "`shares * currentPrincipal / totalShares`" like the contracts `Graph` and `Matic`. The staking benefits are distributed as claimable fees and recorded in the variable `rewards` by the function `_processNewStake()`. However, the user's balance has no benefits because of the lacking of share calculation.

In the contract `Audius`, the function `_processNewStake()` only updates the `withdrawPool` data when the rewards are less than 0. Thus, the user will suffer the staking loss but cannot get the benefits.

Recommendation

We recommend calculating the unstake/withdraw amount based on the `currentPrincipal` for the contracts `Audius` and `Livepeer`.

Alleviation

`[Tenderize]`:

A user accumulates rewards on the go. When a user would withdraw, the rewards are already included. Acknowledged the naming of 'currentPrincipal' is ambiguous here. When a user has unstaked, he/she should no longer get benefits. However, in some cases, e.g. Matic, a user can still be slashed while unstaking.

The current implementation aligns with the original project design.

TTB-01 `MAX_FEE` IS NOT APPLIED IN THE FUNCTION `_initialize()`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/tenderizer/Tenderizer.sol: 47	Acknowledged

Description

In the contract `Tenderizer`, the `MAX_FEE` limitation is only applied in the set functions `setProtocolFee()` and `setLiquidityFee()`. The function `_initialize()` does not have the limitation checking. Thus, the fee amount can be set arbitrarily in the `_initialize()` function.

Recommendation

We recommend adding `MAX_FEE` limitation checking in the function `_initialize()`.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

CON-06 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	contracts/tenderfarm/TenderFarm.sol: 70; contracts/tenderizer/Tenderizer.sol: 43; contracts/tenderswap/TenderSwap.sol: 84, 91; contracts/token/TenderToken.sol: 180	Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

CON-07 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/tenderfarm/TenderFarm.sol: 181; contracts/tenderizer/integrations/livepeer/Livepeer.sol: 184; contracts/token/TenderToken.sol: 166, 172, 179	Acknowledged

Description

There should be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

TER-01 | TOKEN ECONOMY

Category	Severity	Location	Status
Logical Issue	Informational	contracts/token/TenderToken.sol	Acknowledged

Description

`TenderToken` is used as the staking voucher and can be swapped in the contract `TenderSwap` based on the Curve StableSwap. The user who owns the voucher can call `unstake()` function to get the staking principle and benefit. Thus, it is reasonable to use shares instead of amounts in mint/burn/transfer operations. The owner of the `TenderToken` should be transferred to the `Audius/Graph/Livepper/Matic` contracts immediately before minting any `TenderToken`. Each of these `Tenderizer` contracts needs one instance of the `TenderToken` contract.

Also, although the `TenderToken` is inherited from the ERC20 standard contract, it is not implemented as the normal ERC20. It is not the amount but the user's share that participates in circulation. This will bring uncertainty to the token value. Thus, the token amount owned by the user will be greatly affected by token owners' operations. The price of the `TenderToken` is unpredictable. Therefore, the `TenderToken` may be not suitable for trading in the outside market and put into a trading pair.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

The financial model of this protocol is not in the scope of this audit.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

TFT-01 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	Informational	contracts/tenderfarm/TenderFarm.sol: 130, 132, 144, 146, 164	Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

External call(s)

```
130         _harvest(_for);
```

- This function call executes the following external call(s).
- In `TenderFarm._harvest`,
  - `require(bool,string)(rewardToken.transfer(_for,rewardTokens),TRANSFER_FAIL)`

State variables written after the call(s)

```
132         stakes[_for].stake += _amount;
```

External call(s)

```
144         _harvest(_for);
```

- This function call executes the following external call(s).
- In `TenderFarm._harvest`,
  - `require(bool,string)(rewardToken.transfer(_for,rewardTokens),TRANSFER_FAIL)`

State variables written after the call(s)

```
146         _stake.stake -= _amount;
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

`[Tenderize]`: The potential re-entrancy attack will not do much harm because the amount of reward tokens transferrable by `_harvest()` are capped by the amount returned by `_availableRewardShares(address)`.

The pattern is not optimal and can be improved by making `_availableRewardShares()` a pure function that takes in the required values as arguments rather than reading from storage, but will not change in this existing version.

OPTIMIZATIONS | TENDERIZE

ID	Title	Category	Severity	Status
CON-08	Function Could Be Declared External	Gas Optimization	Optimization	Acknowledged

CON-08 | FUNCTION COULD BE DECLARED EXTERNAL

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/tenderswap/SwapUtils.sol: 383; contracts/token/TenderToken.sol: 64, 69, 84, 114, 124, 130, 144, 154	Acknowledged

Description

The functions which are never called internally within the contract should have external visibility for gas optimization.

```
383     function removeLiquidityImbalance(
```

```
64     function decimals() public pure override(ITenderToken, ERC20Upgradeable)
returns (uint8) {
```

```
69     function totalSupply() public view override(ITenderToken, ERC20Upgradeable)
returns (uint256) {
```

```
106     function balanceOf(address account) public view virtual override returns
(uint256) {
```

```
84     function balanceOf(address _account) public view override(ITenderToken,
ERC20Upgradeable) returns (uint256) {
```

```
114     function transfer(address _recipient, uint256 _amount)
```

```
124     function approve(address _spender, uint256 _amount) public
override(ITenderToken, ERC20Upgradeable) returns (bool) {
```

```
130     function transferFrom(
```

```
144     function increaseAllowance(address _spender, uint256 _addedValue)
```

```
154     function decreaseAllowance(address _spender, uint256 _subtractedValue)
```

Recommendation

We advise to change the visibility of the aforementioned functions to `external`.

Alleviation

`[Tenderize]`: Issue acknowledged. They won't make any changes to the current version.

FORMAL VERIFICATION | TENDERIZE

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-false	If Function `transfer` Returns `false`, the Contract State Has Not Been Changed
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`

Property Name	Title
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-balance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-false	If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State

Property Name	Title
erc20-allowance-correct-value	Function `allowance` Returns Correct Value
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-approve-false	If Function `approve` Returns `false`, the Contract's State Has Not Been Changed
erc20-approve-never-return-false	Function `approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
    - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
    - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
    - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract LiquidityPoolToken (contracts/tenderswap/LiquidityPoolToken.sol)

Verification of ERC-20 compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-change-state	Inconclusive	
erc20-transfer-correct-amount	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-false	Inconclusive	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-change-state	Inconclusive	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-correct-value	True	
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-change-state	Inconclusive	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	Inconclusive	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	Inconclusive	
erc20-allowance-correct-value	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-change-state	Inconclusive	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-false	Inconclusive	
erc20-approve-never-return-false	True	

Detailed Results For Contract TenderToken (contracts/token/TenderToken.sol)

Verification of ERC-20 compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	Inconclusive	
erc20-transfer-succeed-self	Inconclusive	
erc20-transfer-succeed-normal	Inconclusive	
erc20-transfer-correct-amount	Inconclusive	
erc20-transfer-change-state	Inconclusive	
erc20-transfer-correct-amount-self	Inconclusive	
erc20-transfer-exceed-balance	Inconclusive	
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-never-return-false	Inconclusive	
erc20-transfer-false	Inconclusive	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	Inconclusive	
erc20-transferfrom-revert-to-zero	Inconclusive	
erc20-transferfrom-succeed-normal	Inconclusive	
erc20-transferfrom-succeed-self	Inconclusive	
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-correct-allowance	Inconclusive	
erc20-transferfrom-change-state	Inconclusive	
erc20-transferfrom-fail-exceed-balance	Inconclusive	
erc20-transferfrom-fail-exceed-allowance	Inconclusive	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-false	Inconclusive	
erc20-transferfrom-never-return-false	Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	Inconclusive	
erc20-totalsupply-correct-value	Inconclusive	
erc20-totalsupply-change-state	Inconclusive	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	Inconclusive	
erc20-balanceof-correct-value	Inconclusive	
erc20-balanceof-change-state	Inconclusive	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	Inconclusive	
erc20-allowance-correct-value	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-change-state	Inconclusive	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-false	Inconclusive	
erc20-approve-never-return-false	True	

APPENDIX | TENDERIZE

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`.

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                     > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false`, the Contract State Has Not Been Changed.

If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

The transfer function must never return `false` to signal a failure.

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,

- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
               || (_allowances[from][msg.sender]
                   == old(_allowances[from][msg.sender])
                      && (from == msg.sender
                          || old(_allowances[from][msg.sender])
                             == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
    && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
    ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), return
             && (msg.sender == from
                 || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != to
   && _balances[to] + value > type(uint256).max && value <= type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
      ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), _balances[to]
              > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

The `transferFrom` function must never return `false`.

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

The `totalSupply` function in contract contract must not change any state variables.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

Function `allowance` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
               && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

The function `approve` must never returns `false`.

Specification:

```
 [](!(finished(contract.approve, !return)))
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.