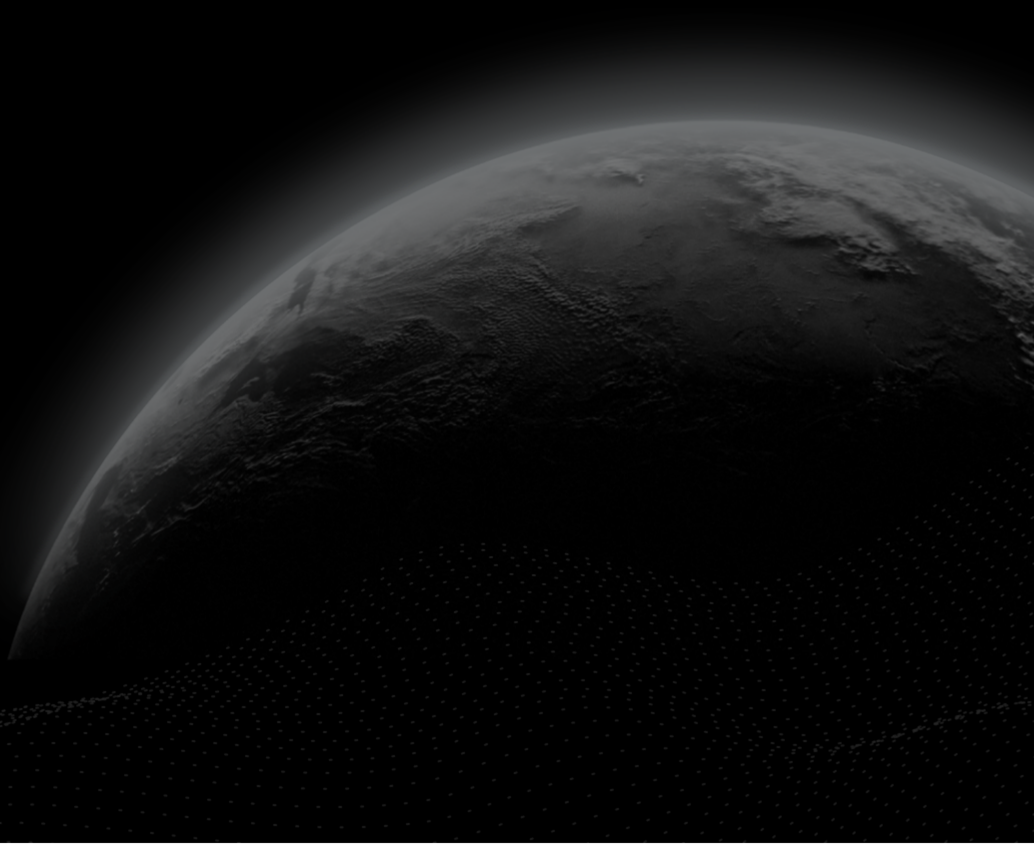
front-cover-logo

Preliminary Comments

Draft (Internal Use Only)

solidity.io - Triflex

CertiK Verified on Aug 5th, 2022



CertiK Certified

CertiK Verified on Aug 5th, 2022

solidity.io - Triflex

These preliminary comments were prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeflationaryToken	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 08/05/2022	N/A

CODEBASE

https://github.com/solidity-io/Triflex-Certik

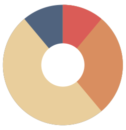
...View All

COMMITS

de6b0b59dfeb95fee84d98507951a6cde1b4c34d

...View All

Vulnerability Summary



18	12	0	0	6	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

Count	Severity	Status	Description
2	Critical	1 Resolved, 1 Acknowledged	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.
5	Major	1 Resolved, 4 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.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
9	Minor	8 Resolved, 1 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.
2	Informational	2 Resolved	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.
0	Discussion		The impact of the issue is yet to be determined, hence requires further clarifications from the project team.

TABLE OF CONTENTS | SOLIDITY.IO - TRIFLEX

CODEBASE | SOLIDITY.IO - TRIFLEX

Repository

https://github.com/solidity-io/Triflex-Certik

Commit

de6b0b59dfeb95fee84d98507951a6cde1b4c34d

AUDIT SCOPE | SOLIDITY.IO - TRIFLEX

24 files audited • 6 files with Acknowledged findings • 2 files with Resolved findings • 16 files without findings

ID	File	SHA256 Checksum
BBT	contracts/BuyBack.sol	8134b532c340f212d5efbdc9fdbdd565b440d1c7f4e90037fab7ba49e1dc0d12
DPT	contracts/DividendPayingToken.sol	0fd142ef0dcb298048a4b2525577fc883c29e26e7c8dd2dbb4df022f74fed9e3
GTC	contracts/Governance.sol	645e542468e739bb1909385266359d8f83097514371984e9f71057e595773cd7
RTT	contracts/RewardsTracker.sol	aee63e75601dbe354e38894401a4611426020b7da2529f4a6f2de57d5f81797e
TTC	contracts/Triflex.sol	00bd1cb7326499a055e4a2d2863d29ef5c31bfbff5fabfbe5e2da2166ccf300a
TTW	contracts/TriflexTWAP.sol	c9da7b74ae3da0d8ce514af217c6267ed0015f3ba04deea9d82da3e2cd7ae08d
ERC	contracts/ERC20PermitUpgradeable.sol	65e21b22021dd18aeef3784fb692d120fca08ceb11fe0bb8ebb54264d42c8504
IMT	contracts/IterableMapping.sol	aed3a444a4a1079a9fc1e2e1a400317d3ac9bf18a0c616bfea129545e9727d2f
BTC	contracts/libraries/Babylonian.sol	64467a06f69fd5da24e53975e72d91001562ae68062cd4ba3c3534621e3c6c23
FPT	contracts/libraries/FixedPoint.sol	1477cc85bea47ae89537d2d953dfaaa9460a5e163e10afd9d8522c9a3fc182e3
SMT	contracts/libraries/SafeMath.sol	2db002b8a8c8357b7f98348b03710aa308b79cf96d27432cb2ec54c3948e5217
SMI	contracts/libraries/SafeMathInt.sol	04936ade70e50b165fd378e21e2888a5828e1f2754a5cfd89a32106f1c553742
SMU	contracts/libraries/SafeMathUint.sol	d94df3e5335ad628931be92b2bf6b7f17907720798f4b7f273d3b3585b54b119
UVL	contracts/libraries/UniswapV2Library.sol	ccde0c1b148c8dfa912ceb44ffc1904f8bdf31a8861d5029e96ddbdfc02632c4

ID	File	SHA256 Checksum
UVO	contracts/libraries/UniswapV2OracleLibrary.sol	79384bd261cbdfec37f6e27708d9ca4d46c049ff36725582e22b89d128d81fd0
DSS	contracts/interface/.DS_Store	913f41bb417b33a6d852f6d0efa14608898a363628cbaa78e5e8e321d6a06603
IDP	contracts/interface/IDividendPayingToken.sol	0970c8afd69c7d3b658ebcbf0323f51041946772853e050ec257784c94ffb4d6
IGT	contracts/interface/IGovernance.sol	d426df03bc3f52461bb38f22958466b81d5e31d1f5116c738d978911dd7eefda
IRT	contracts/interface/IRewardsTracker.sol	c4fad5459e3c867d1989441a3161f9074d60d2229ab537d4df1422b192c22645
IUV	contracts/interface/IUniswapV2Factory.sol	ad27dc554913b6d6a693f620c9b7bfb1b61acaf58fa4fbf41b6931fefca236c8
IUP	contracts/interface/IUniswapV2Pair.sol	8eef00638c3fa3f1a1b1d8d0b835b4d89bfba69384445640d8e476a91dd81335
IUR	contracts/interface/IUniswapV2Router01.sol	06c6dddf24363cbdbec33ff67b289d1a3c3667be600041faff836d13abaec2a5
IUT	contracts/interface/IUniswapV2Router02.sol	73c588138d5ff8150074e84aed66e5d4b7c05ea16042fcab9c16d60fda6bb5fb
IWB	contracts/interface/IWBNB.sol	555de967fe1e1595843698b41cbfde4e57811960d43ab814b5ef0250e09a1af6

APPROACH & METHODS | SOLIDITY.IO - TRIFLEX

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

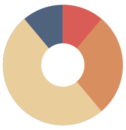
The auditing process pays special attention to the following considerations:

- 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.

FINDINGS | SOLIDITY.IO - TRIFLEX



18	2	5	0	9	2	0
Total Findings	Critical	Major	Medium	Minor	Informational	Discussion

This report has been prepared to discover issues and vulnerabilities for solidity.io - Triflex. Through this audit, we have uncovered 18 issues ranging from different severity levels. Utilizing Static Analysis techniques to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
DPT-01	**Centralization Risks In DividendPayingToken.Sol**	**Centralization / Privilege**	**Major**	**Acknowledged**
DPT-02	Missing Upper Bound	Logical Issue	Minor	Resolved
GTC-01	**Centralization Risks In Governance.Sol**	**Centralization / Privilege**	**Major**	**Acknowledged**
RTT-01	**Centralization Risks In RewardsTracker.Sol**	**Centralization / Privilege**	**Major**	**Acknowledged**
TCB-01	Implementation Contract Is Not Initialized Automatically	Language Specific	Critical	Resolved
TCB-02	A Reverting Fallback Function Will Lock Up All Rewards	Logical Issue	Critical	Acknowledged
TCB-03	Potential Reentrancy Attack (Involving Ether)	Volatile Code	Major	Resolved
TCB-04	Third Party Dependency	Volatile Code	Minor	Acknowledged
TCB-05	Missing Zero Address Validation	Volatile Code	Minor	Resolved
TCB-06	Potential Reentrancy Attack (Events)	Volatile Code	Minor	Resolved
TCB-07	Potential Reentrancy Attack (Benign)	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
TCB-08	Unused Return Value	Volatile Code	Minor	Resolved
TTC-01	**Centralization Risks In Triflex.Sol**	**Centralization / Privilege**	**Major**	**Acknowledged**
TTC-02	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
TTC-03	Code Is Commented Out	Inconsistency	Minor	Resolved
TTC-04	Missing Lower Bound	Logical Issue	Minor	Resolved
TCB-10	Debugging Tool Imported	Language Specific	Informational	Resolved
TCB-11	Missing Error Messages	Coding Style	Informational	Resolved

DPT-01 | CENTRALIZATION RISKS IN DIVIDENDPAYINGTOKEN.SOL

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/DividendPayingToken.sol: 99, 103, 107, 120, 129, 139, 148, 158, 168, 177, 186, 195, 385, 477, 501, 509, 517, 525, 539**	**Acknowledged**

Description

In the contract `DividendPayingToken,` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below:

- `setMaxTokenSendAmount()`
- `setGovToken()`
- `setGovTokenRate()`
- `setTriflex()`
- `setTriflexTokenRate()`
- `setPromoTokenRate()`
- `setTriflexPair()`
- `setBuyBackAddress()`
- `setGasForTransfer()` Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the governance token to an attacker-controlled contract by calling the function `setGovToken().` , which may drain users' funds.

The compromised account also can adjust the rates of any of the listed above willingly.

In the contract `DividendPayingToken,` the role `PAUSER_ROLE` has authority over the functions shown below:

- `_pause()`
- `_unpaused()` Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause the contract indefinitely.

In the contract `DividendPayingToken,` the role `TRIFLEX_ROLE` has authority over the functions shown below:

- `_setTotalPendingDividends()`
- `_setUserCustomRewardToken()`
- `_setTokenAvailable()`
- `_setTokenShouldSwap()` Any compromise to the `TRIFLEX_ROLE` account may allow the hacker to take advantage of this authority and add attacker-controlled contracts into the mapping `tokenShouldSwap` by calling the function `_setTokenShouldSwap()` , which may drain users' funds.

The attacker can also halt transactions by setting users custom reward tokens to a reverting 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

[Triflex] - Issue acknowledged. I won't make any changes to the current version. The contract will be moved to a gnosis with more than three signers after deployment.

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged.

DPT-02 | MISSING UPPER BOUND

Category	Severity	Location	Status
Logical Issue	Minor	contracts/DividendPayingToken.sol: 129~130, 148~149, 158~159	Resolved

Description

In the current implementation, rates can be set as much as 2^256-1. There should abound limits on how high the rates can be set in the linked functions.

Recommendation

We recommend including a bound to minimize how high the reward can be set.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by adding an upper for each rate.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 at line 175, 195 and 205 of OnRay.sol.

GTC-01 | CENTRALIZATION RISKS IN GOVERNANCE.SOL

Category	Severity	Location	Status
Centralization / Privilege	● Major	contracts/Governance.sol: 28, 32, 36, 40	● Acknowledged

Description

In the contract `Otzi` the role `MINTER_ROLE` has authority over the functions shown below:

- `mint()`
- `` Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint a large number of tokens to attacker-controlled addresses.

In the contract `Otzi` the role `PAUSER_ROLE` has authority over the functions shown below:

- `pause()`
- `unpause()` Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause the contract indefinitely.

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

[Triflex] - Issue acknowledged. I won't make any changes to the current version. The contract will be moved to a gnosis with more than three signers after deployment.

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged.

RTT-01 | CENTRALIZATION RISKS IN REWARDSTRACKER.SOL

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**contracts/RewardsTracker.sol: 56, 67, 150, 167 , 222, 235**	● **Acknowledged**

Description

In the contract `TriflexRewardsTracker` the role `TRIFLEX_ROLE` has authority over the functions shown in the diagram below.

- `excludeFromRewards()`
- `setBalance()`
- `process()`
- `processAccount()`

Any compromise to the `TRIFLEX_ROLE` account may allow the hacker to take advantage of this authority and burn anyone's reward by calling the function `setBalance()` .

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

[Triflex] - Issue acknowledged. I won't make any changes for the current version. TRIFLEX_ROLE will be transferred to a gnosis after deployment.

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged.

TCB-01 | IMPLEMENTATION CONTRACT IS NOT INITIALIZED AUTOMATICALLY

Category	Severity	Location	Status
Language Specific	● Critical	contracts/DividendPayingToken.sol: 73~74; contracts/RewardsTracker.sol: 44; contracts/Triflex.sol: 124	● Resolved

Description

When you deploy the implementation contract and call the function `initialize()` from the Proxy contract, it will execute in the context of the Proxy contract. Hence the state variable `_initialized` and `_initializing` will be 0 and false, respectively. Attackers can directly call the function `initialize(),` bypass the `initializer` modifier, and feed some malicious inputs. Please check the following link for more information about this attack: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract.

Recommendation

We recommend adding the following code into your implementation contract.

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

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by following the recommendation above.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 at line 84 of OnRye.sol and line 46 of Ham.sol. Contract Triflex.sol is no longer upgradeable, hence this issue is resolved.

TCB-02 | A REVERTING FALLBACK FUNCTION WILL LOCK UP ALL REWARDS

Category	Severity	Location	Status
Logical Issue	Critical	contracts/DividendPayingToken.sol: 345~346; contracts/RewardsTracker.sol: 167~168	Acknowledged

Description

The function `process()` processes a list of transfers. If any of the recipients of a BNB transfer is a smart contract that reverts, then the entire payout will fail. This will prevent users from receiving their dividends.

Recommendation

We recommend implementing a queuing mechanism to allow users to initiate the withdrawal on their own using a 'pull-over-push pattern.' For more information about this pattern, please check this link.

Alleviation

[CertiK] - The Triflex team acknowledged the issue but choose to leave the source code unchanged.

TCB-03 | POTENTIAL REENTRANCY ATTACK (INVOLVING ETHER)

Category	Severity	Location	Status
Volatile Code	● Major	contracts/BuyBack.sol: 64~69, 70; contracts/Triflex.sol: 401~403, 406~408, 409~411, 441~443, 456~458, 470~472, 486~488, 503~509, 515~522	● Resolved

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)

```
64        swapRouter.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
ethAmount}(
65          1,
66          path,
67          address(0xdeaDDeADDEaDdeaDdEAddEADDEAdDeadDEADDEaD),
68          block.timestamp
69        );
```

State variables written after the call(s)

```
70        inSwap = false;
```

External call(s)

```
377                     swapAndsendEth();
```

- This function call executes the following external call(s).
- In `Triflex.addLiquidity`,
    - `uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)`
- In `Triflex.swapTokensForEth`,

  - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

- In `Triflex.swapAndsendEth`,
  - `(dSuccess) = address(devWallet).call{value: devAllocation}()`
- In `Triflex.swapAndsendEth`,
  - `(mSuccess) = address(treasuryWallet).call{value: treasuryAllocation}()`
- In `Triflex.swapAndsendEth`,
  - `(rSuccess) = address(dividendToken).call{value: rewardETHAllocation}()`
- In `Triflex.swapAndsendEth`,
  - `(mSuccess) = address(treasuryWallet).call{value: amountETH}()`
- This call sends Ether.

```
380         _transferStandard(from, to, amount, currenttotalFee);
```

- This function call executes the following external call(s).
- In `Triflex._transferStandard`,
  - `dividendToken._setTotalPendingDividends(currentDividends + calculatedDividends)`
- In `Triflex._transferStandard`,
  - `rewardsTracker.setBalance(address(sender),balanceOf(sender))`
- In `Triflex._transferStandard`,
  - `rewardsTracker.setBalance(address(recipient),balanceOf(recipient))`

State variables written after the call(s)

```
380         _transferStandard(from, to, amount, currenttotalFee);
```

- This function call executes the following assignment(s).
- In `ERC20Upgradeable._transfer`,
  - `_balances[from] = fromBalance - amount`
- In `ERC20Upgradeable._transfer`,

- `_balances[to] += 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

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by following the recommendation above.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol and Triflex.sol.

TCB-04 | THIRD PARTY DEPENDENCY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BuyBack.sol: 13; contracts/DividendPayingToken.sol: 46, 55, 57; contracts/Triflex.sol: 29, 32; contracts/TriflexTWAP.sol: 14	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume 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.

```
13    IUniswapV2Router02 public swapRouter;
```

- The contract `BuyBack` interacts with third party contract with `IUniswapV2Router02` interface via `swapRouter`.

```
46      mapping(address => address) public userCurrentRewardToken;
```

- The contract `DividendPayingToken` interacts with third party contract with `IGovernance` interface via `userCurrentRewardToken`.

```
55      IUniswapV2Router02 public swapRouter;
```

- The contract `DividendPayingToken` interacts with third party contract with `IUniswapV2Router02` interface via `swapRouter`.

```
57      IWBNB public wBNB;
```

- The contract `DividendPayingToken` interacts with third party contract with `IWBNB` interface via `wBNB`.

```
29      IUniswapV2Router02 public uniswapV2Router;
```

- The contract `Triflex` interacts with third party contract with `IUniswapV2Router02` interface via `uniswapV2Router`.

```
    ITriflexRewardsTracker public rewardsTracker;
```

- The contract `Triflex` interacts with third party contract with `ITriflexRewardsTracker` interface via `rewardsTracker`.

```
    IUniswapV2Pair public immutable pair;
```

- The contract `TriflexTWAP` interacts with third party contract with `IUniswapV2Pair` interface via `pair`.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Triflex] - Issue acknowledged. I won't make any changes for the current version.

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged.

TCB-05 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BuyBack.sol: 27, 44; contracts/DividendPayingToken.sol: 79, 125, 144, 173, 182; contracts/Triflex.sol: 137, 629	Resolved

Description

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

```
27      triflex = _triflex;
```

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

```
44      triflex = _triflex;
```

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

```
79          triflex = _triflex;
```

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

```
125         govToken = _govToken;
```

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

```
144         triflex = _triflex;
```

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

```
173         triflexPair = _triflexPair;
```

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

```
182        buyBackAddress = _buyBackAddress;
```

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

```
137        treasuryWallet = payable(_treasuryWallet); // EOA tax
```

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

```
629        (bool success, ) = to.call{value: balance}("");
```

- `to` 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

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by following the recommendation above.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol, OnRay.sol and Triflex.sol.

TCB-06 | POTENTIAL REENTRANCY ATTACK (EVENTS)

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BuyBack.sol: 64~69, 71; contracts/RewardsTracker.sol: 62, 64, 158, 161, 164, 212, 216; contracts/Triflex.sol: 175~176, 180, 181, 192, 194, 377, 380, 401~403, 406~408, 409~411, 423, 441~443, 445~449, 456~458, 460~464, 470~472, 474~478, 482, 486~488, 490, 503~509, 515~522, 526~530, 531~538, 569, 571, 582~585, 590, 601, 616, 617, 622, 623, 629, 631, 680, 681, 715, 716, 717, 722, 723	Resolved

Description

This finding has a minor impact because the reentrancy only causes out-of-order events.

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)

```
64        swapRouter.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
ethAmount}(
65            1,
66            path,
67            address(0xdeaDDeADDEaDdeaDdEAddEADDEAdDeadDEADDEaD),
68            block.timestamp
69        );
```

Events emitted after the call(s)

```
71        emit BoughtAndBurned(ethAmount);
```

External call(s)

```
62            dividendToken._setBalance(account, 0);
```

Events emitted after the call(s)

```
        emit ExcludedFromDividends(account);
```

External call(s)

```
            dividendToken._setBalance(account, newBalance);
```

```
            dividendToken._setBalance(account, 0);
```

```
        _processAccount(account, true);
```

- This function call executes the following external call(s).
- In `TriflexRewardsTracker._processAccount`,
  - `amount = dividendToken._withdrawDividendOfUser(account)`

Events emitted after the call(s)

```
            emit Claim(token, account, amount, automatic);
```

- Executed via the following function call(s):
  - `_processAccount(account,true)`

External call(s)

```
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(_uniswapV2Router.WETH(), address(this));
```

Events emitted after the call(s)

```
        emit Transfer(address(0), _msgSender(), TOTAL_SUPPLY);
```

```
        emit ExcludeFromFees(account);
```

- Executed via the following function call(s):
  - `excludeFromFees(_msgSender())`

- `excludeFromFees(address(this))`

```
269         emit Transfer(address(0), account, amount);
```

- Executed via the following function call(s):
    - `_mint(_msgSender(),TOTAL_SUPPLY)`

External call(s)

```
377                 swapAndsendEth();
```

- This function call executes the following external call(s).
- In `Triflex.addLiquidity`,
    - `uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)`
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`
- In `Triflex.swapAndsendEth`,
    - `(dSuccess) = address(devWallet).call{value: devAllocation}()`
- In `Triflex.swapAndsendEth`,
    - `(mSuccess) = address(treasuryWallet).call{value: treasuryAllocation}()`
- In `Triflex.swapAndsendEth`,
    - `(rSuccess) = address(dividendToken).call{value: rewardETHAllocation}()`
- In `Triflex.swapAndsendEth`,
    - `(mSuccess) = address(treasuryWallet).call{value: amountETH}()`
- This call sends Ether.

```
380         _transferStandard(from, to, amount, currenttotalFee);
```

- This function call executes the following external call(s).
- In `Triflex._transferStandard` ,
    - `dividendToken._setTotalPendingDividends(currentDividends + calculatedDividends)`
- In `Triflex._transferStandard` ,
    - `rewardsTracker.setBalance(address(sender),balanceOf(sender))`
- In `Triflex._transferStandard` ,
    - `rewardsTracker.setBalance(address(recipient),balanceOf(recipient))`

Events emitted after the call(s)

```
        emit Transfer(from, to, amount);
```

- Executed via the following function call(s):
    - `_transferStandard(from,to,amount,currenttotalFee)`

External call(s)

```
        swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth` ,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

```
                (bool dSuccess, ) = address(devWallet).call{
                    value: devAllocation
                }("");
```

- This call sends Ether.

Events emitted after the call(s)

```
                emit Transfer(
                    address(this),
                    devWallet,
                    devAllocation
                );
```

External call(s)

```
        swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

```
            (bool dSuccess, ) = address(devWallet).call{
                value: devAllocation
            }("");
```

- This call sends Ether.

```
            (bool mSuccess, ) = address(treasuryWallet).call{
                value: treasuryAllocation
            }("");
```

- This call sends Ether.

Events emitted after the call(s)

```
                emit Transfer(
                    address(this),
                    treasuryWallet,
                    treasuryAllocation
                );
```

External call(s)

```
        swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

```
                (bool dSuccess, ) = address(devWallet).call{
                    value: devAllocation
                }("");
```

- This call sends Ether.

```
                (bool mSuccess, ) = address(treasuryWallet).call{
                    value: treasuryAllocation
                }("");
```

- This call sends Ether.

```
                (bool rSuccess, ) = address(dividendToken).call{
                    value: rewardETHAllocation
                }("");
```

- This call sends Ether.

Events emitted after the call(s)

```
                    emit Transfer(
                        address(this),
                        address(dividendToken),
                        rewardETHAllocation
                    );
```

External call(s)

```
            swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

```
            (bool dSuccess, ) = address(devWallet).call{
                value: devAllocation
            }("");
```

- This call sends Ether.

```
            (bool mSuccess, ) = address(treasuryWallet).call{
                value: treasuryAllocation
            }("");
```

- This call sends Ether.

```
            (bool rSuccess, ) = address(dividendToken).call{
                value: rewardETHAllocation
            }("");
```

- This call sends Ether.

```
            addLiquidity(amountToLiquify, amountETHLiquidity);
```

- This function call executes the following external call(s).
- In `Triflex.addLiquidity`,
    - `uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)`
- This call sends Ether.

Events emitted after the call(s)

```
        emit Approval(owner, spender, amount);
```

- Executed via the following function call(s):
    - `addLiquidity(amountToLiquify,amountETHLiquidity)`

External call(s)

```
        swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`

```
                (bool mSuccess, ) = address(treasuryWallet).call{
                    value: amountETH
                }("");
```

- This call sends Ether.

Events emitted after the call(s)

```
                    emit Transfer(address(this), treasuryWallet, amountETH);
```

External call(s)

```
        (
            uint256 iterations,
            uint256 claims,
            uint256 lastProcessedIndex
        ) = rewardsTracker.process(gas);
```

Events emitted after the call(s)

```
        emit ProcessedDividendTracker(
            iterations,
            claims,
            lastProcessedIndex,
            false,
            gas,
            tx.origin
        );
```

External call(s)

```
        rewardsTracker.excludeFromRewards(newPair);
```

Events emitted after the call(s)

```
        emit SetAutomatedMarketMakerPair(uniswapV2Pair);
```

External call(s)

```
            uniswapV2Pair =
IUniswapV2Factory(_newRouter.factory()).createPair(
                address(this),
                _newRouter.WETH()
            );
```

Events emitted after the call(s)

```
        emit SetNewRouter(newRouter);
```

External call(s)

```
        rewardsTracker.updateClaimWait(claimWait);
```

Events emitted after the call(s)

```
        emit UpdateClaimWait(claimWait);
```

External call(s)

```
        token.transfer(to, balance);
```

Events emitted after the call(s)

```
        emit Transfer(address(this), to, balance);
```

External call(s)

```
        (bool success, ) = to.call{value: balance}("");
```

Events emitted after the call(s)

```
        emit BNBWithdrawn(to, balance);
```

External call(s)

```
680         dividendToken._setUserCustomRewardToken(holder, rewardTokenAddress);
```

Events emitted after the call(s)

```
681         emit RewardsTokenChosen(holder, rewardTokenAddress);
```

External call(s)

```
715         dividendToken._deleteTokenAvailable(rewardTokenAddress);
```

```
716         dividendToken._deleteTokenShouldSwap(rewardTokenAddress);
```

Events emitted after the call(s)

```
717         emit RewardTokenRemoved(rewardTokenAddress);
```

External call(s)

```
722         rewardsTracker.excludeFromRewards(account);
```

Events emitted after the call(s)

```
723         emit ExcludeFromRewards(account);
```

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

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by applying Checks-Effects-Interactions Pattern.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol, OnRay.sol and Triflex.sol.

TCB-07 | POTENTIAL REENTRANCY ATTACK (BENIGN)

Category	Severity	Location	Status
Volatile Code	Minor	contracts/DividendPayingToken.sol: 87~90, 91, 92, 93, 94, 95, 96, 247~255, 257, 258~260, 264, 267~268, 315~317, 330~334, 345, 347~348; contracts/RewardsTracker.sol: 158, 161, 164, 212, 215; contracts/Triflex.sol: 175~176, 178, 180, 181, 183, 184, 186, 188, 189, 190, 192, 423, 441~443, 456~458, 470~472, 482, 503~509, 515~522, 582~585, 589, 600	Resolved

Description

This issue is considered benign because the reentrancy only acts as a double call.

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)

```
87          IERC20Upgradeable(_triflex).safeApprove(
88              address(this),
89              0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
90          );
```

- This call sends Ether.

State variables written after the call(s)

```
95          demoninator = 1000;
```

```
91          gasForTransfer = 3000;
```

```
92          govTokenRate = 100;
```

Note: Only a sample of 3 assignments (out of 6) are shown above.

External call(s)

```
try
    swapRouter.swapExactETHForTokensSupportingFeeOnTransferTokens{
        value: ethAmount
    }(1, path, address(recipient), block.timestamp + 360)
{
    swapSuccess = true;
} catch {
    swapSuccess = false;
}
```

- This call sends Ether.

```
wBNB.deposit{value: ethAmount}();
```

- This call sends Ether.

```
try wBNB.transfer(recipient, ethAmount) {
    swapSuccess = true;
} catch {}
```

- This call sends Ether.

```
(bool success, ) = recipient.call{value: ethAmount}("");
```

- This call sends Ether.

State variables written after the call(s)

```
    withdrawnDividends[recipient] = withdrawnDividends[recipient]
        .sub(ethAmount);
```

External call(s)

```
try IGovernance(token).mint(recipient, amountToSend) {
    sendSuccess = true;
} catch {}
```

- This call sends Ether.

```
            IERC20Upgradeable(token).safeTransferFrom(
                address(this),
                recipient,
                amountToSend
            );
```

- This call sends Ether.

```
            (bool success, ) = recipient.call{value: ethAmount}("");
```

- This call sends Ether.

State variables written after the call(s)

```
                withdrawnDividends[recipient] = withdrawnDividends[recipient]
                    .sub(ethAmount);
```

External call(s)

```
            dividendToken._setBalance(account, newBalance);
```

- This call sends Ether.

```
            dividendToken._setBalance(account, 0);
```

- This call sends Ether.

```
        _processAccount(account, true);
```

- This function call executes the following external call(s).
- In `TriflexRewardsTracker._processAccount` ,
    - `amount = dividendToken._withdrawDividendOfUser(account)`

- This call sends Ether.

State variables written after the call(s)

```
        _processAccount(account, true);
```

- This function call executes the following assignment(s).
- In `TriflexRewardsTracker._processAccount`,
    - `lastClaimTimes[account] = block.timestamp`

External call(s)

```
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(_uniswapV2Router.WETH(), address(this));
```

- This call sends Ether.

State variables written after the call(s)

```
        _canTransferBeforeOpenTrading[address(this)] = true;
```

```
        _canTransferBeforeOpenTrading[_msgSender()] = true;
```

```
        maxWalletAmount = (TOTAL_SUPPLY * 15) / 10000; //.15% of TOTAL_SUPPLY
```

Note: Only a sample of 3 assignments (out of 11) are shown above.

External call(s)

```
        swapTokensForEth(amountToLiquify);
```

- This function call executes the following external call(s).
- In `Triflex.swapTokensForEth`,
    - `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,1,path,address(this),block.timestamp)`
- This call sends Ether.

```
                (bool dSuccess, ) = address(devWallet).call{
                    value: devAllocation
                }("");
```

- This call sends Ether.

```
                (bool mSuccess, ) = address(treasuryWallet).call{
                    value: treasuryAllocation
                }("");
```

- This call sends Ether.

```
                (bool rSuccess, ) = address(dividendToken).call{
                    value: rewardETHAllocation
                }("");
```

- This call sends Ether.

```
                addLiquidity(amountToLiquify, amountETHLiquidity);
```

- This function call executes the following external call(s).
- In `Triflex.addLiquidity`,
    - `uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp)`
- This call sends Ether.

State variables written after the call(s)

```
                addLiquidity(amountToLiquify, amountETHLiquidity);
```

- This function call executes the following assignment(s).
- In `ERC20Upgradeable._approve`,
    - `_allowances[owner][spender] = amount`

External call(s)

```
            uniswapV2Pair =
IUniswapV2Factory(_newRouter.factory()).createPair(
                address(this),
                _newRouter.WETH()
            );
```

- This call sends Ether.

State variables written after the call(s)

```
589         uniswapV2Router = _newRouter;
```

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

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by applying Checks-Effects-Interactions Pattern.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol, OnRay.sol, Ham.sol and Triflex.sol.

TCB-08 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/DividendPayingToken.sol: 532~535; contracts/Triflex.sol: 293, 515~522	Resolved

Description

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

```
532             IERC20(rewardToken).approve(
533                 address(this),
534
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
535             );
```

```
293         rewardsTracker.processAccount(payable(_msgSender()), false);
```

```
515         uniswapV2Router.addLiquidityETH{value: ethAmount}(
516             address(this),
517             tokenAmount,
518             0,
519             0,
520             address(this),
521             block.timestamp
522         );
```

Recommendation

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

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by checking the return values.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in OnRay.sol and Triflex.sol.

TTC-01 | CENTRALIZATION RISKS IN TRIFLEX.SOL

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**contracts/Triflex.sol: 197, 205, 211, 219, 224, 229, 237, 247, 525, 541, 551, 557, 566, 574, 593, 597, 604, 614, 620, 626, 634, 654, 692, 706, 720**	● **Acknowledged**

Description

In the contract `Triflex` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and implement a few attacks.

Assume the owner role was compromised by a hacker named Oscar,

1st Attack Oscar can block all transfers.

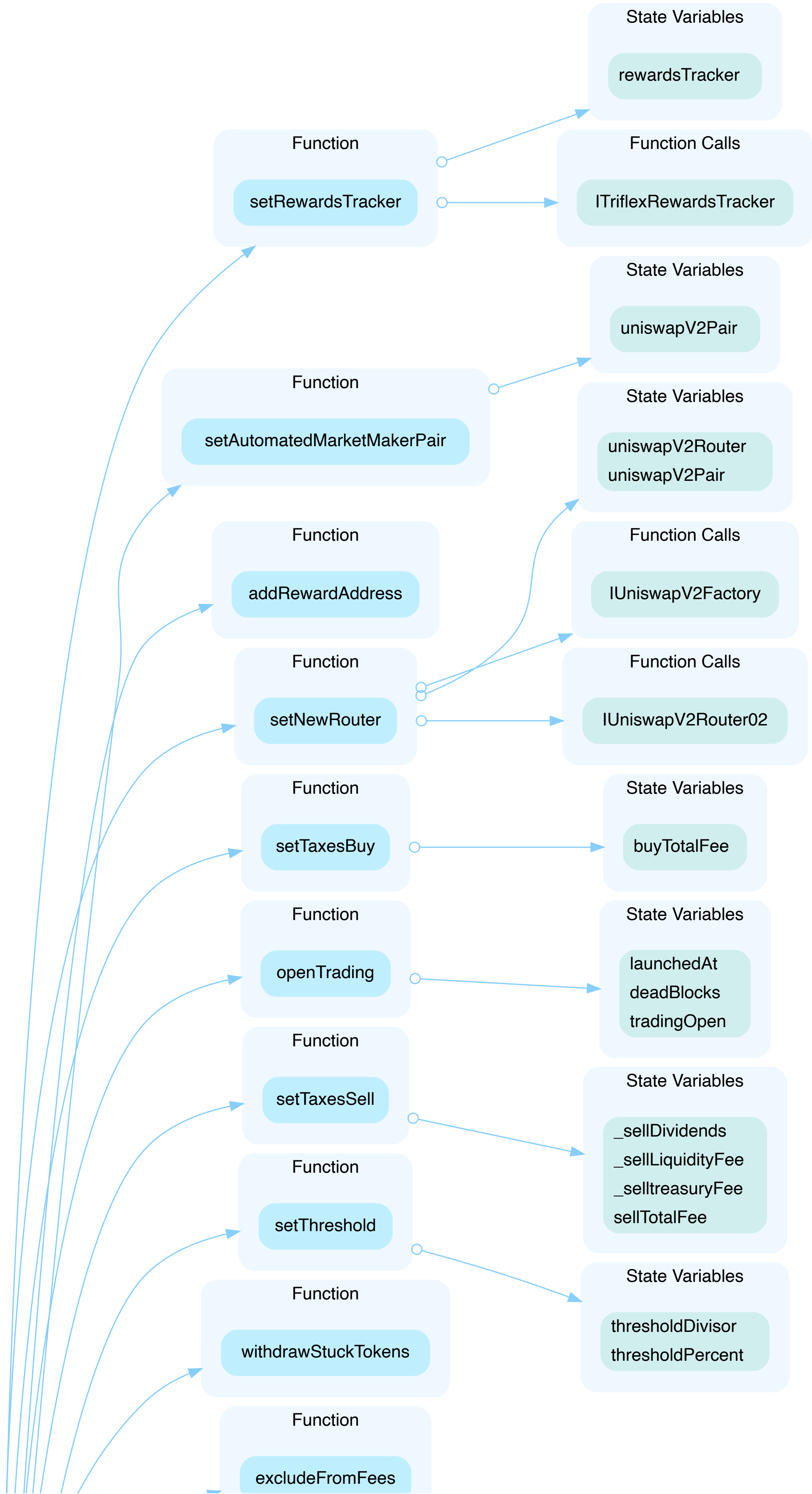
This attack can easily be done by invoking the function `setMaxWalletAmount` and setting the variable `maxWalletAmount` to 1. This would prevent every call to `_transfer` to fail because line 322 will always fail.

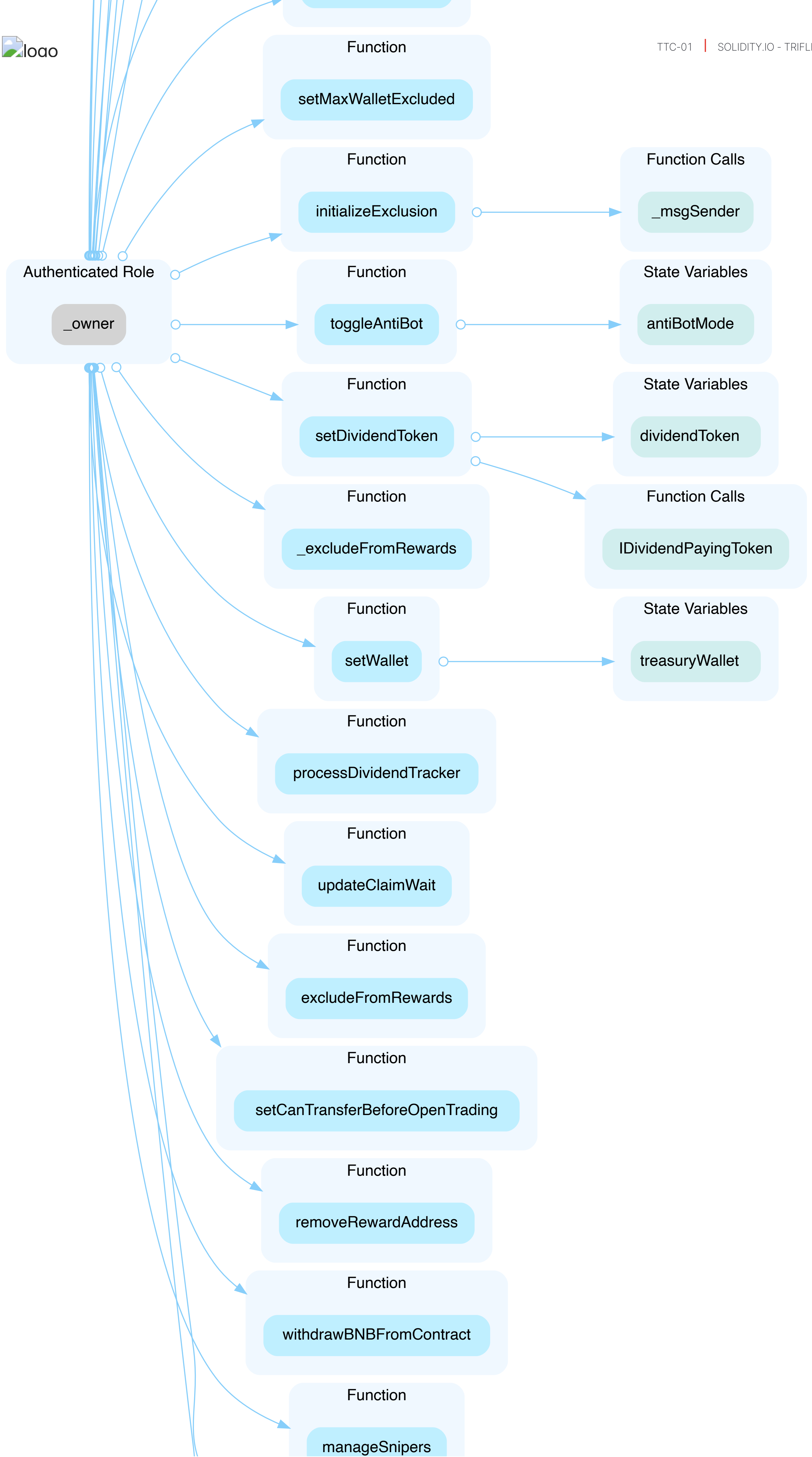
2nd Attack Oscar can drain tokens from the contract by calling function `withdrawStuckTokens()` and `withdrawBNBFromContract()`.

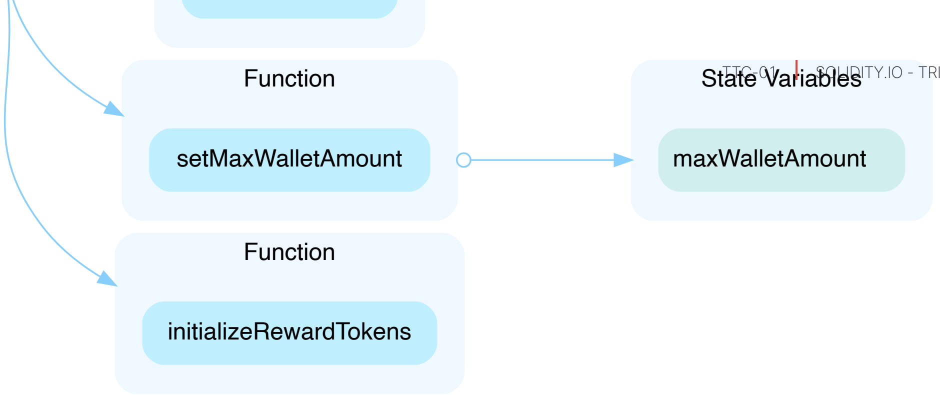
3rd Attack Oscar can halt all users from withdrawing their dividends.

1. Oscar will remove all reward addresses via the function `removeRewardAddress()`.
2. Oscar then creates a malicious contract that contains a fallback() function which called upon will revert any call.
3. Pass the address of the malicious contract as reward token via `addRewardAddress()`.

This attack will prevent any user from withdrawing their dividends.


State Variables
rewardsTracker
Function
setRewardsTracker
Function Calls
ITriflexRewardsTracker
State Variables
uniswapV2Pair
Function
setAutomatedMarketMakerPair
State Variables
uniswapV2Router
uniswapV2Pair
Function
addRewardAddress
Function Calls
IUniswapV2Factory
Function
setNewRouter
Function Calls
IUniswapV2Router02
Function
setTaxesBuy
State Variables
buyTotalFee
Function
openTrading
State Variables
launchedAt
deadBlocks
tradingOpen
Function
setTaxesSell
State Variables
_sellDividends
_sellLiquidityFee
_selltreasuryFee
sellTotalFee
Function
setThreshold
State Variables
thresholdDivisor
thresholdPercent
Function
withdrawStuckTokens
Function
excludeFromFees


Function
setMaxWalletExcluded
Function
initializeExclusion
Function Calls
_msgSender
Authenticated Role
_owner
Function
toggleAntiBot
State Variables
antiBotMode
Function
setDividendToken
State Variables
dividendToken
Function
_excludeFromRewards
Function Calls
IDividendPayingToken
Function
setWallet
State Variables
treasuryWallet
Function
processDividendTracker
Function
updateClaimWait
Function
excludeFromRewards
Function
setCanTransferBeforeOpenTrading
Function
removeRewardAddress
Function
withdrawBNBFromContract
Function
manageSnipers



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

[Triflex] - Issue acknowledged. I won't make any changes for the current version. Ownership of the contract will be transferred to a gnosis after deployment

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged.

TTC-02 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Triflex.sol: 622	Resolved

Description

The return value of the transfer() call is not checked.

```
622        token.transfer(to, balance);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by checking the returned `bool` value.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 at line 541 of Triflex.sol.

TTC-03 | CODE IS COMMENTED OUT

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Triflex.sol: 54~57, 138~140	Resolved

Description

The linked statements are commented out.

Recommendation

We recommend reviewing the commented out code. If it is not needed we recommend removing the commented code.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by deleting the commented out code.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in Triflex.sol.

TTC-04 | MISSING LOWER BOUND

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Triflex.sol: 205~207	Resolved

Description

The lower bound is set to 0 which is an ineffective bound.

Suppose the owner role is compromised by a hacker. The hacker can block all transfers by setting the `maxWalletAmount` to 1.

Recommendation

We recommend increasing the lower bound.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by adding a lower bound.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 at line 166 of Triflex.sol.

TCB-10 | DEBUGGING TOOL IMPORTED

Category	Severity	Location	Status
Language Specific	Informational	contracts/BuyBack.sol: 6; contracts/RewardsTracker.sol: 7	Resolved

Description

The following line of code should be deleted when deploying your contracts:

```
import "hardhat/console.sol";
```

Recommendation

We recommend deleting the debugging tool.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by following the recommendation above.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol and Triflex.sol.

TCB-11 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	contracts/RewardsTracker.sol: 60; contracts/Triflex.sol: 206, 292, 316~318, 321, 545, 552, 567, 575, 598, 630, 700, 721	● Resolved

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 recommend including a message that details the potential error in the linked statements.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by including an error message for each require statement. The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in Ham.sol and Triflex.sol.

OPTIMIZATIONS | SOLIDITY.IO - TRIFLEX

ID	Title	Category	Severity	Status
GTC-02	User-Defined Getters	Gas Optimization	Optimization	Acknowledged
TCB-09	Improper Usage Of `public` And `external` Type	Gas Optimization	Optimization	Resolved
TTC-05	Unnecessary Use Of SafeMath	Language Specific	Optimization	Resolved

GTC-02 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Governance.sol: 46~48	Acknowledged

Description

The linked function is equivalent to the compiler-generated getter function for the respective variable.

Recommendation

We advise that the linked variable is instead declared as `public` as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

[Triflex] - Issue acknowledged. I won't make any changes for the current version.

[CertiK] - The `Triflex` team acknowledged the issue but chose to leave the source code in regards to this finding unchanged. We note that not resolving this finding does not pose any risk to this project. Therefore the `Triflex` community should not be concerned that this finding is not resolved.

TCB-09 | IMPROPER USAGE OF `public` AND `external` TYPE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/BuyBack.sol: 39, 43, 47; contracts/DividendPayingToken.sol: 73, 107, 120, 129, 139, 148, 158, 168, 177, 413, 462; contracts/ERC20PermitUpgradeable.sol: 92, 130; contracts/IterableMapping.sol: 13, 17, 24, 28, 32, 43; contracts/RewardsTracker.sol: 44, 235; contracts/Triflex.sol: 124, 197, 205, 211, 219, 224, 287, 302, 574, 720	Resolved

Description

`public` functions that are never called by the contract could be declared as `external`. `external` functions are more efficient than `public` functions.

Recommendation

Consider using the external attribute for public functions that are never called within the contract.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by following the recommendation above.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in BuyBack.sol, OnRay.sol, Ham.sol, IterableMapping.sol and Triflex.sol.

TTC-05 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Language Specific	Optimization	contracts/Triflex.sol: 13~14	Resolved

Description

The contract `SafeMathUpgradeable` can be removed because SafeMath is no longer needed starting with Solidity 0.8. The compiler now has built-in overflow checking.

Recommendation

We recommend removing this import for gas optimization.

Alleviation

[Triflex] - Issue acknowledged. Changes have been reflected in the commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7.

[CertiK] - The `Triflex` team resolved the issue by deleting the contract `SafeMathUpgradeable`.

The changes can be seen in commit hash fbd96ec36fd218c5f64fdfa374690b14c52d24c7 in Triflex.sol.

APPENDIX | SOLIDITY.IO - TRIFLEX

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)))
```

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.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
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.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

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.

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.