CERTIK

Security Assessment

Infinity.xyz

Jun 2nd, 2022

Table of Contents

Summary

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

The contracts audited by CertiK for Infinity are for an NFT marketplace. The exchange adds a new set of features that are not available on similar protocols. These new features allow for a more seamless user experience. Users can place orders within a certain range and wait for the price to hit and purchase an NFT within the collection. This allows for a new depth of liquidity for NFT's. By having dynamic orders, users have access to liquidity on upward and downward trends rather than listing an NFT and waiting for a user to offer or purchase it at X price.

Orders are placed by using a hybrid system of on-chain and off-chain orders. By using EIP-712 and EIP-1271, we sign these orders off-chain to save gas fees on maker side. Orders are allowed to be placed across multiple collections. If the price set by the user is struck, then it will automatically execute X order. These dynamic order features are not available on other protocols. Because of the high fee cost associated with NFT transactions. Being able to batch the orders together across multiple collections allows for a lower gas fee.

Users that stake the Infinity token will receive a discount on the protocol. Discounts are based on how many tokens are staked and penalties can apply by removing longer staking yields. This should allow more tokens to remain inside the protocols.

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:

- 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.

Overview

Project Summary

Project Name	Infinity.xyz
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/mavriklabs/infinity-exchange-contracts
Commit	a6a9ec9123e6de8da94124d0c2889159e091e4dd b6affa33d05f930b2408e0a846491a28f45d943a

Audit Summary

Delivery Date	Jun 02, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

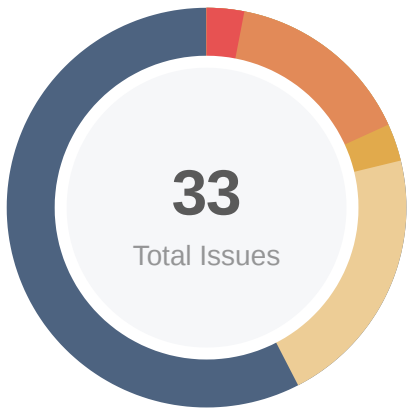
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	1	0	0	1	0	0	0
Major	5	0	0	5	0	0	0
Medium	1	0	0	0	0	0	1
Minor	7	0	0	4	0	0	3
Informational	19	0	0	16	0	1	2
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IEB	contracts/core/InfinityExchange.sol	2ac8cf27f515b66bdcd0e691987d8e4b9abf17ce06dd39f551d8e6ee05325826
IFT	contracts/core/InfinityFeeTreasury.sol	e93e2e93eb3b062daa95b365857f75016bfaa09fe896b67b731004a5ac763324
IOB	contracts/core/InfinityOrderBookComplication.sol	6a81c131429dc0dd7f33ea871c60501e70e4efad45e9bf41af51ab48fff324f6
ISB	contracts/core/InfinityStaker.sol	7b20f4b532c02509eba129bd1d6a89db12a69f3f1a8602b92a108386e5a7277a
ITR	contracts/core/InfinityTradingRewards.sol	6c7cfe262adbaaa692e1247960b3b9ad412286bcff2b6a79e2455a7f22cd1d16

Findings



Critical	**1** (3.03%)
Major	**5** (15.15%)
Medium	**1** (3.03%)
Minor	**7** (21.21%)
Informational	**19** (57.58%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Lack Of Readability With Import Statements	Coding Style	Informational	Acknowledged
GLOBAL-02	Tokenomics	Control Flow	Informational	Acknowledged
GLOBAL-03	Code Readability	Coding Style, Inconsistency	Informational	Acknowledged
CON-01	External Dependency	Logical Issue	Minor	Acknowledged
COR-01	Excessive Owner Permissions On Staked Tokens	Logical Issue	Critical	Acknowledged
COR-02	Outside Token Dependency	Volatile Code	Minor	Acknowledged
COR-03	Usage Of Hardhat's Console	Coding Style	Informational	Acknowledged
COR-04	Unlocked Compiler Version	Language Specific	Informational	Resolved
COR-05	Missing Emit Events	Coding Style	Informational	Acknowledged
COR-06	Missing Input Validation	Volatile Code	Informational	Acknowledged
COR-07	Inefficient Loop Over Memory Array	Gas Optimization	Informational	Acknowledged
COR-08	External Function Arguments Have `memory` Specifier`	Gas Optimization	Informational	Acknowledged
COR-09	Variables That Could Be Declared As `constant`	Gas Optimization	Informational	Acknowledged

ID	Title	Category	Severity	Status
IEB-01	Centralization Risks In InfinityExchange.sol	**Centralization / Privilege**	**Major**	Acknowledged
IEB-02	Third Party Dependencies	Volatile Code	Minor	Acknowledged
IEB-03	Unused `internal` Function	Volatile Code	Minor	Resolved
IEB-04	Potential Error On Canceling Orders/orderValidity	Logical Issue	Informational	Resolved
IEB-05	Treasury Maintanence	Volatile Code	Informational	Acknowledged
IEB-06	Typo	Coding Style	Informational	Acknowledged
IEB-07	Code Readability On Exchange Transfer	Coding Style, Inconsistency	Informational	Acknowledged
IEB-08	Unknown Complication Strategies	Logical Issue	Informational	Partially Resolved
IEB-09	Orders Controlled By Complication	Volatile Code	Informational	Acknowledged
IFT-01	Centralization Risks In InfinityFeeTreasury.sol	**Centralization / Privilege**	**Major**	Acknowledged
IFT-02	Missing Override Specifier	Compiler Error	Minor	Resolved
INF-01	Unknown Merkle Tree And Parameter `proofs`	Volatile Code	Minor	Acknowledged
IOB-01	Centralization Risks In InfinityOrderBookComplication.sol	**Centralization / Privilege**	**Major**	Acknowledged
ISB-01	Centralization Risks In InfinityStaker.sol	**Centralization / Privilege**	**Major**	Acknowledged
ISU-01	Incorrect Update On `_updateUserStakedAmounts()`	Logical Issue	Minor	Resolved
ITR-01	Centralization Risks In InfinityTradingRewards.sol	**Centralization / Privilege**	**Major**	Acknowledged
ITR-02	Potential Approval Risk On `claimReward()`	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
ITR-03	Potential Resource Exhaustion	Gas Optimization	Informational	Acknowledged
ITR-04	Unclear Error Message	Coding Style	Informational	Acknowledged
ITR-05	Incompatibility With Deflationary Tokens	Volatile Code	Informational	Acknowledged

GLOBAL-01 | Lack Of Readability With Import Statements

Category	Severity	Location	Status
Coding Style	Informational		Acknowledged

Description

Because of the high number of import statements, it would be easier for an outside reviewer of the code to break up the imports into openzeppelin sections and custom imports.

Recommendation

We recommend breaking them up with comments to make it easier for any outsider reviewer. For example:

pragma solidity 0.8.1; //openzeppelin contracts import "@openzeppelin" import "@openzeppelin"

//Infinity interfaces ....

//Infinity Libraries ...

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

GLOBAL-02 | Tokenomics

Category	Severity	Location	Status
Control Flow	Informational		Acknowledged

Description

Users of Infinity can lock their tokens for 0, 3, 6, and 12 months. All staking levels get the same discount on fees so there is no incentive past the bronze threshold. For example:

```
uint16 public BRONZE_EFFECTIVE_FEE_BPS = 10000;
uint16 public SILVER_EFFECTIVE_FEE_BPS = 10000;
uint16 public GOLD_EFFECTIVE_FEE_BPS = 10000;
uint16 public PLATINUM_EFFECTIVE_FEE_BPS = 10000;
```

```
function _getUserStakePower(address user) internal view returns (uint256) {
return
((userstakedAmounts[user][Duration.NONE].amount * 1) +
(userstakedAmounts[user][Duration.THREE_MONTHS].amount * 2) +
(userstakedAmounts[user][Duration.SIX_MONTHS].amount * 3) +
(userstakedAmounts[user][Duration.TWELVE_MONTHS].amount * 4)) / (10**18);
  }
```

```
uint16 public BRONZE_STAKE_THRESHOLD = 1000;
uint16 public SILVER_STAKE_THRESHOLD = 5000;
uint16 public GOLD_STAKE_THRESHOLD = 10000;
uint16 public PLATINUM_STAKE_THRESHOLD = 20000;
```

If the user locks up tokens for 12 months, they can easily hit the threshold of bronze.

If it is easy to reach the maximum threshold because the multiple of 4 by locking for one year, this could create a massive downward trend on price if the inflation is relatively high.

Recommendation

Consider revisit the tokenomic and see if it aligns with the design.

Alleviation

`[Infinity]`: The fee discounts can be changed by admin. Initially there is no discount for anyone, hence effective fee bps is the same for all levels. Once we enable fee discounts, these constants will be updated to reflect the right bps amounts.

GLOBAL-03 | Code Readability

Category	Severity	Location	Status
Coding Style, Inconsistency	Informational		Acknowledged

Description

This contract uses many different types of fees such as `CURATOR_FEE_BPS`,BRONZE_EFFECTIVE_FEE_BPS, etc. To make this easier for outsider reviewers to understand what is happening quicker. Labeling what the 250 is equal to.

Abbreviations such as bps might be hard for outside reviewers to understand what the abbreviation stands for.

Recommendation

For example:

```
uint16 public CURATOR_FEE_BPS = 250; // 2.5%
```

Perhaps adding a note at the top to explain what BPS stands for or other technical terms would make this product easier for users to understand.

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

CON-01 | External Dependency

Category	Severity	Location	Status
Logical Issue	Minor	contracts/core/InfinityExchange.sol: 4, 7~8, 13, 520, 521; contracts/interfaces/IInfinityFeeTreasury.sol: 8; contracts/core/InfinityFeeTreasury.sol: 8, 10, 11; contracts/core/InfinityOrderBookComplication.sol: 4; contracts/interfaces/IInfinityExchange.sol: 7	Acknowledged

Description

The implementation of the native and reward token are not included in the audit. Because these are not in the scope, they will be treated as blackbox. We assume that these tokens have correct logical behavior.

Recommendation

We recommend ensuring that these contracts are logically correct to avoid unintended errors.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

COR-01 | Excessive Owner Permissions On Staked Tokens

Category	Severity	Location	Status
Logical Issue	Critical	contracts/core/InfinityStaker.sol: 299; contracts/core/InfinityTradingRewards.sol: 131	Acknowledged

Description

The owner has privilege to withdraw any token inside the `InfinityStaker` and `InfinityTradingRewards` and transfer it to any address they wish. If a malicious actor got control over this function, they could invalidate the entire contract.

For example: Bob stakes 100,000 tokens. Malicious actor takes over dev wallet and calls `rescueTokens()` and takes 100,000 out of the contract. Bob calls to withdraw his tokens after duration and cannot because balance exceeds amount exceeds balance. Bob will never be able to use the contract again due to his account never being able to be reset.

Recommendation

We recommend adding logic to where staked tokens cannot be rescued.

Alleviation

`[Infinity]`: Issue acknowledged. I will fix in the future.

COR-02 | Outside Token Dependency

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/InfinityTradingRewards.sol: 25, 45; contracts/core/InfinityStaker.sol: 19, 35	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party [Staking] and [Reward] protocols. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of [Infinity] requires interaction with their own token. We encourage the team to constantly monitor the statuses of their token to mitigate possible negative side effects due to unforeseen events.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

COR-03 | Usage Of Hardhat's Console

Category	Severity	Location	Status
Coding Style	Informational	contracts/core/InfinityExchange.sol: 19; contracts/core/InfinityFeeTreasury.sol: 13; contracts/core/InfinityOrderBookComplication.sol: 8; contracts/core/InfinityStaker.sol: 10; contracts/core/InfinityTradingRewards.sol: 11	Acknowledged

Description

The contract uses the console contract from Hardhat, which is meant to be used for testing purposes.

Recommendation

It is recommended to remove the import of Hardhat's contract for better code readability and simplicity. We acknowledge it is commented out, but it is a reminder to remove it for deployment purposes.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

COR-04 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/core/InfinityExchange.sol: 2; contracts/core/InfinityFeeTreasury.sol: 2; contracts/core/InfinityOrderBookComplication.sol: 2; contracts/core/InfinityStaker.sol: 2; contracts/core/InfinityTradingRewards.sol: 2	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash c89a1a1c94591a0bfd6101dd168ac9c42586db44.

COR-05 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	contracts/core/InfinityStaker.sol: 307, 312, 334, 338; contracts/core/InfinityTradingRewards.sol: 144, 148, 152, 156	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to [users].

- `updateInfinityTreasury()`
- `updateInfinityRewardsContract()`
- `updateStakeLevelThreshold()`
- `updateupdatePenalties()`
- `updateInfinityStaker()`
- `updateInfinityExchange()`
- `updateInfinityToken()`
- `updateRewardsMap()`

Recommendation

Consider adding events for sensitive actions, and emit them in the function.

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

COR-06 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	contracts/core/InfinityStaker.sol: 334, 338; contracts/core/InfinityTradingRewards.sol: 144, 148, 152; contracts/core/InfinityExchange.sol: 782, 791, 800, 805, 810; contracts/core/InfinityFeeTreasury.sol: 340, 345	Acknowledged

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

COR-07 | Inefficient Loop Over Memory Array

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/core/InfinityExchange.sol: 134, 164, 177, 204, 217, 626, 657, 672, 731, 750; contracts/core/InfinityTradingRewards.sol: 53, 64, 67; contracts/core/InfinityOrderBookComplication.sol: 117, 146, 218, 219, 253, 254; contracts/core/InfinityFeeTreasury.sol: 115, 132, 245, 253	Acknowledged

Description

All for loops could be a more gas efficient by declaring the length into a local variable before calling it.

For example:
The function `cancelMultipleOrders()` loops multiple times for the length of orders cancelled.

```
for (uint256 i = 0; i < orderNonces.length; i++) {
```

Recommendation

We recommended storing the length of the array in a local variable in order to save on the overall cost of gas:

```
uint256 orderNonceCount = orderNonce.length;
for (uint256 i = 0; i < orderNonceCount; i++) {
```

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

COR-08 | External Function Arguments Have `memory` Specifier`

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/core/InfinityStaker.sol: 156; contracts/core/InfinityTradingRewards.sol: 113	Acknowledged

Description

External function accepts `memory` arrays. This will force redundant copying of `calldata` values into memory.

Recommendation

We recommend using `calldata` specifier for external function arguments if the function does not modify them.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

COR-09 | Variables That Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/core/InfinityStaker.sol: 19; contracts/core/InfinityFeeTreasury.sol: 22	Acknowledged

Description

The linked variables could be declared as `constant` since these state variables are never modified.

Recommendation

We recommend to declare these variables as `constant`.

Alleviation

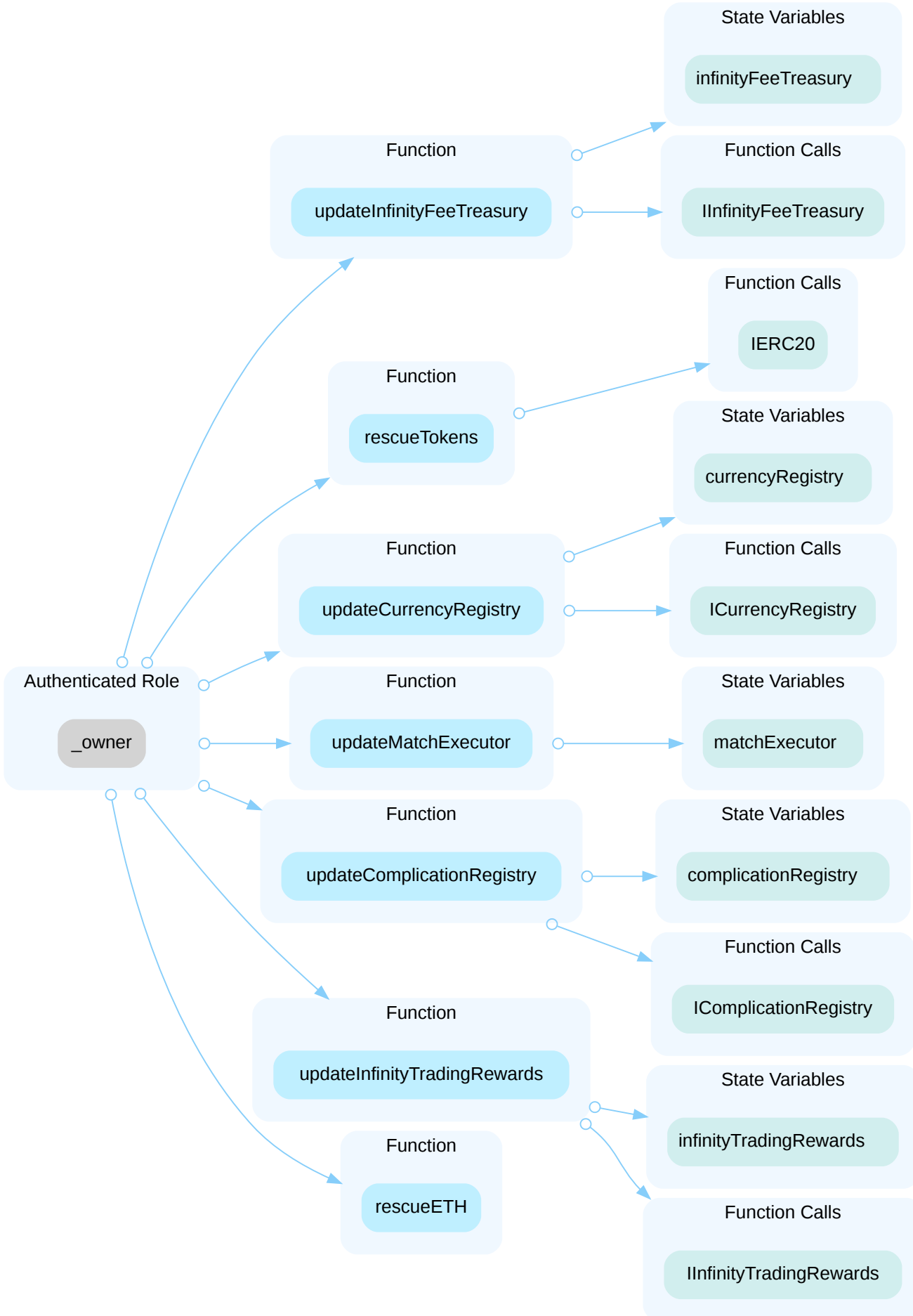
`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IEB-01 | Centralization Risks In InfinityExchange.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/core/InfinityExchange.sol: 765, 773, 782, 791, 800, 805, 810	Acknowledged

Description

In the contract `InfinityExchange` the role `onlyOwner` has authority over the functions shown in the diagram below. Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this authority.


State Variables
infinityFeeTreasury
Function
updateInfinityFeeTreasury
Function Calls
IInfinityFeeTreasury
Function Calls
IERC20
Function
rescueTokens
State Variables
currencyRegistry
Function
updateCurrencyRegistry
Function Calls
ICurrencyRegistry
Authenticated Role
_owner
Function
updateMatchExecutor
State Variables
matchExecutor
Function
updateComplicationRegistry
State Variables
complicationRegistry
Function Calls
IComplicationRegistry
Function
updateInfinityTradingRewards
State Variables
infinityTradingRewards
Function
rescueETH
Function Calls
IInfinityTradingRewards

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

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

IEB-02 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/InfinityExchange.sol: 54, 57, 58	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party [currencyRegistry], [complicationRegistry], [CREATOR_FEE_MANAGER] contracts. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of [Infinity] requires interaction with some of its own contracts such as [currencyRegistry], [complicationRegistry], and [CREATOR_FEE_MANAGER]. However, these contracts are treated as third party dependencies due to being out of the scope of the audit. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IEB-03 | Unused `internal` Function

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/InfinityExchange.sol: 569	Resolved

Description

This internal function is not called anywhere in the contract.

Recommendation

We advise the client to review the functionality of `_getCurrentPrice()` and remove it if unnecessary.

Alleviation

`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash 5f12db3cc8c19ba2cba7002c123a1d5988bf872b.

IEB-04 | Potential Error On Canceling Orders/orderValidity

Category	Severity	Location	Status
Logical Issue	Informational	contracts/core/InfinityExchange.sol: 139, 237, 510	Resolved

Description

```
139 require(orderNonces[i] > userMinOrderNonce[msg.sender], 'nonce too low');
```

```
510 bool orderExpired = isUserOrderNonceExecutedOrCancelled[signer][nonce] || nonce < userMinOrderNonce[signer];
```

```
237 function isNonceValid(address user, uint256 nonce) external view returns (bool) {
238 return !isUserOrderNonceExecutedOrCancelled[user][nonce] && nonce > userMinOrderNonce[user];
239 }
240
241 The function `isNonceValid()` states that nonce should always be greater than `userMinOrderNonce`, but there is a lack of check in `orderValidty()`. This could potentially create unintended results of not being able to cancel orders unless `CancelAllOrders()` is called. However, this is a waste of gas on the users end and could create a negative experience.
242
243 We understand there is no way for the nonce to be less than `userMinOrderNonce`. However, what happens if the nonce is equal to `userMinOrderNonce`? The order will execute properly with no issue, but it could have an issue with canceling it. The user can call `cancelAllOrders()` to cancel if it is equal to the `userMinOrderNonce`. If a user wants to cancel just that equal order, they cannot remove the order off `cancelMultipleOrders()` because it will revert trying to cancel an order that is not greater than `userMinOrderNonce`.
244
245 Because the user has to call `cancelAllOrders()` to raise their `userMinOrderNonce` to cancel it. They waste gas and then have to spend time to redo everything they had setup.
246
247 For example: Bob submits an order that is given a nonce of 20. Bob's userMinOrderNonce is 20. However, Bob wants to cancel his order because the price starts tanking. He goes to call function `cancelMultipleOrders()` the function reverts due it failing for too low of a nonce. The only way for him to cancel this is calling `cancelAllOrders()`.
```

Recommendation

Consider revisit the functionality to fix the potential issue

Alleviation

`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash 8d42fe050961d31b723a72f7463963f01bb4e23e

IEB-05 | Treasury Maintanence

Category	Severity	Location	Status
Volatile Code	Informational	contracts/core/InfinityExchange.sol: 185	Acknowledged

Description

Due to the treasury paying back gas fee to match executor, it should always have enough native tokens such as WETH for Ethereum and other natives for other chains. These are used to cover gas fee cost by match executor.

Recommendation

We recommend ensuring that the contract will never run out to avoid unintended consequences.

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

IEB-06 | Typo

Category	Severity	Location	Status
Coding Style	Informational	contracts/core/InfinityExchange.sol: 613, 625, 658, 673, 693	Acknowledged

Description

Inside the comment on line 613, `transfering` is spelled incorrectly. It should be spelled as `transferring`.

Recommendation

We recommend fixing all spelling errors and grammatical mistakes for clarity.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IEB-07 | Code Readability On Exchange Transfer

Category	Severity	Location	Status
Coding Style, Inconsistency	Informational	contracts/core/InfinityExchange.sol: 621	Acknowledged

Description

As this function is commonly used as a batch transfer in erc1155 tokens, this could be easier explained by adding a comment as to who the from is to avoid confusion. This also could also be achieved by changing the from to exchange.

Recommendation

We recommend adding one of the two changes to help and speed up outside reviewers understanding of the process.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IEB-08 | Unknown Complication Strategies

Category	Severity	Location	Status
Logical Issue	Informational	contracts/core/InfinityExchange.sol: 58, 94	Partially Resolved

Description

There are no listed complications inside the code for reviewers to see what strategies will be potentially executed during the purchase and sale of different assets. Because of the nature of dynamic purchases and other types of purchasing, there should be multiple complications able for people to review to be able to have greater trust.

Recommendation

We advise to have this publicly available before launch because these strategies have strict control over assets.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IEB-09 | Orders Controlled By Complication

Category	Severity	Location	Status
Volatile Code	Informational	contracts/core/InfinityExchange.sol: 146	Acknowledged

Description

If a complication is incorrectly added or used, this could attribute to loss of funds/NFT's. We cannot verify any strategies because this is out of the scope of the audit.

Recommendation

We recommend having strict documentation and testing to ensure loss of funds cannot happen.

Alleviation

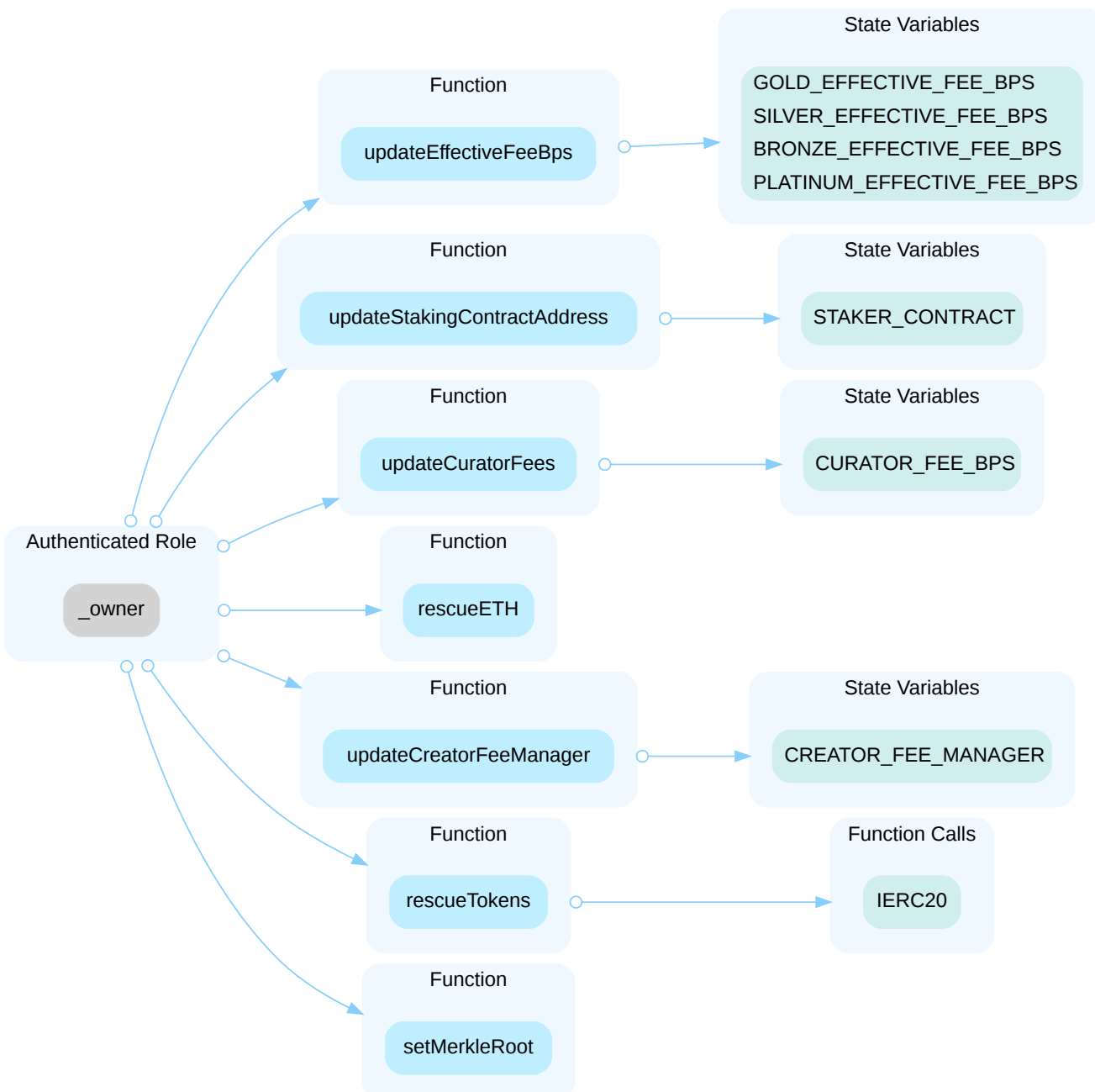
`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

IFT-01 | Centralization Risks In InfinityFeeTreasury.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/core/InfinityFeeTreasury.sol: 327, 335, 340, 345, 350, 355, 368	Acknowledged

Description

In the contract `InfinityFeeTreasury` the role `onlyOwner` has authority over the functions shown in the diagram below. Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this.



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

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

IFT-02 | Missing Override Specifier

Category	Severity	Location	Status
Compiler Error	Minor	contracts/core/InfinityFeeTreasury.sol: 321	Resolved

Description

The contract fails to compile on version 0.8.0 since the linked functions are missing the `override` specifier as these functions are implementations of the `IInfinityFeeTreasury` interface.

Recommendation

For the inheriting functions, it is required to add virtual to every non-interface function intended to override, and to add override to the overriding functions, according to the Solidity 0.6.0 Breaking Changes.

Alleviation

`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash 28ca8b6d910ae35db8395a456dcdc093676c614d.

INF-01 | Unknown Merkle Tree And Parameter `proofs`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/InfinityFeeTreasury.sol (base): 50	Acknowledged

Description

In the contract `InfinityFeeTreasury`, the state variable `merkleRoot` stands for an off-chain Merkle tree, which is used to verify whether a given user has the privilege to claim Curator fees from the treasury. However, the actual Merkle tree used by the contract is uncertain and can be changed. Normally, the Merkle tree is used to prevent arbitrary change to the nodes, but the contracts allow direct updates to the Merkle tree root, which can change the whole tree.

Recommendation

We recommend carefully managing the Merkle tree and the `proofs` data.

Alleviation

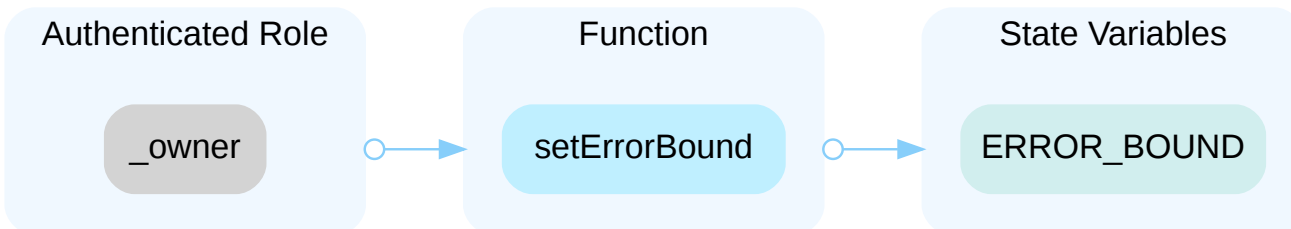
`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

IOB-01 | Centralization Risks In InfinityOrderBookComplication.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/core/InfinityOrderBookComplication.sol: 83	Acknowledged

Description

In the contract `InfinityOrderBookComplication` the role `onlyOwner` has authority over the functions shown in the diagram below. Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this authority.



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

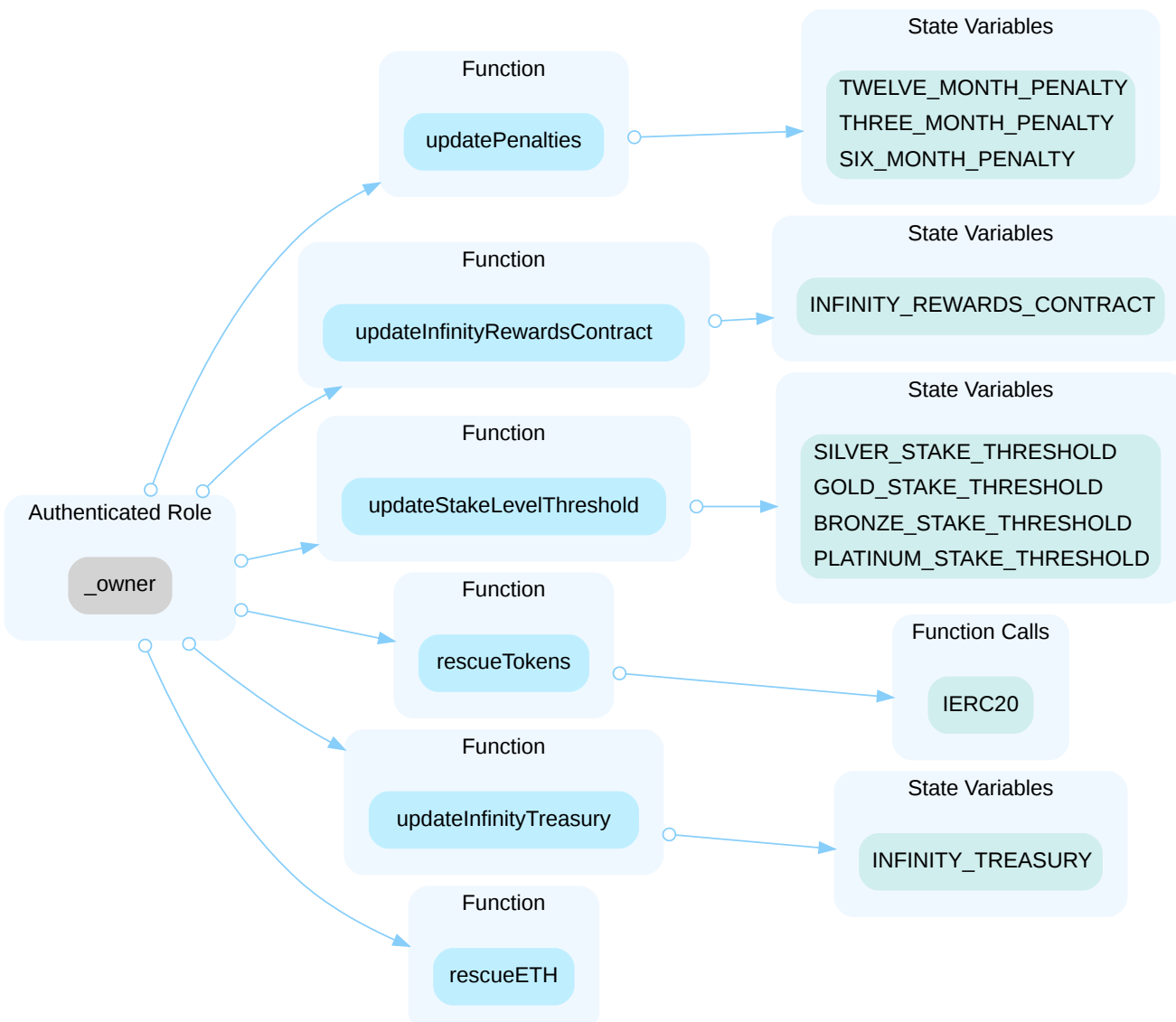
`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

ISB-01 | Centralization Risks In InfinityStaker.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/core/InfinityStaker.sol: 299, 307, 312, 324, 334, 338	Acknowledged

Description

In the contract `InfinityStaker` the role `onlyOwner` has authority over the functions shown in the diagram below. Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this authority and [fixme, describe what hacker can do and the impact].



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

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

ISU-01 | Incorrect Update On `_updateUserStakedAmounts()`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/core/InfinityStaker.sol (base): 104	Resolved

Description

If a user withdraw all their funds, they can call the function `getVestingInfo()`. This will display 0 in the amount in that slot, but it will also still show the block timestamp.

Recommendation

If the amount is emptied from the staking contract for X duration, then the block time stamp should be reset to 0 as well.

Alleviation

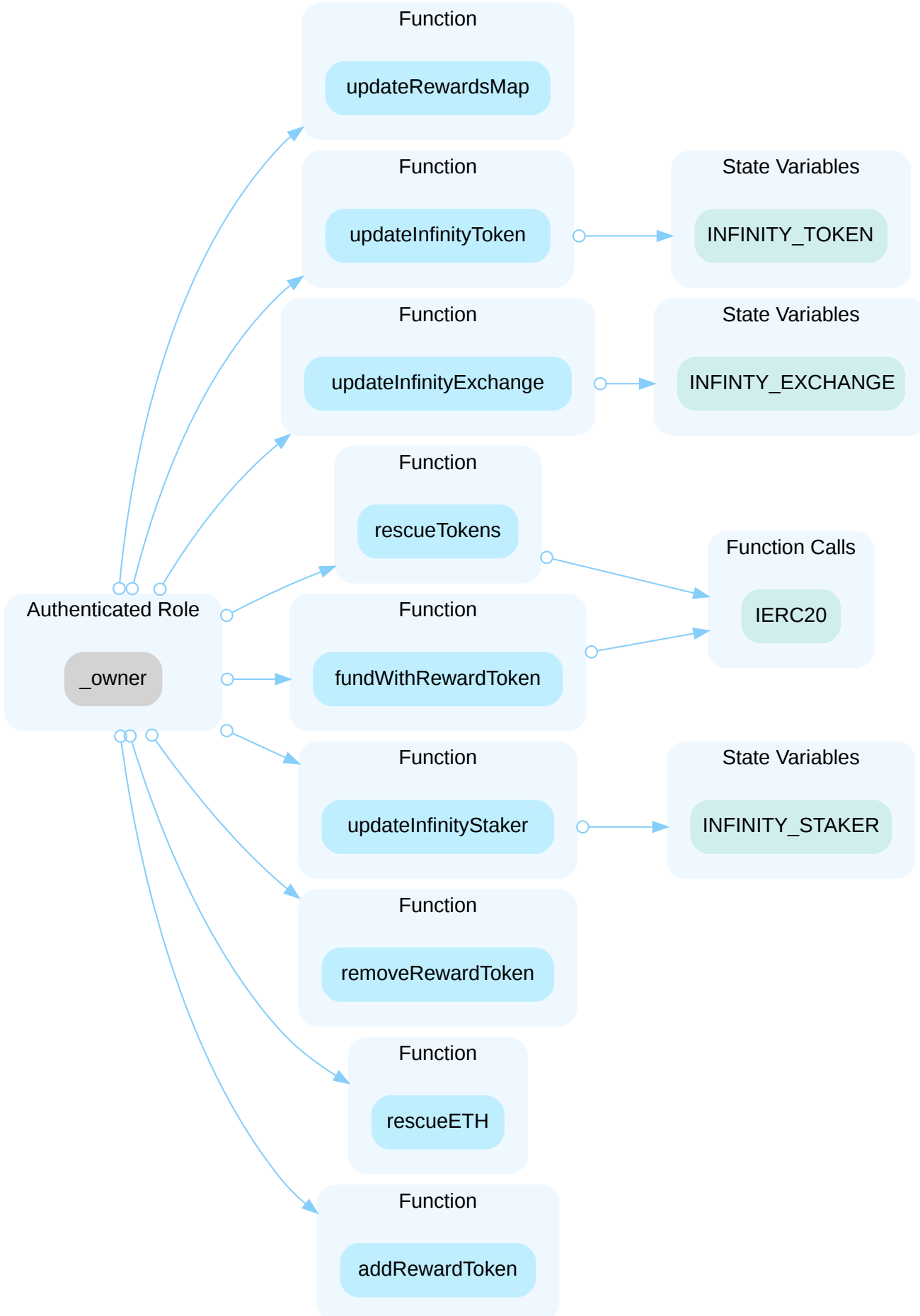
`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash 338f9df328871141c12b30735db7d68ed90df255.

ITR-01 | Centralization Risks In InfinityTradingRewards.sol

Category	Severity	Location	Status
Centralization / Privilege	**Major**	contracts/core/InfinityTradingRewards.sol: 131, 139, 144, 148, 152, 156, 168, 175, 182	Acknowledged

Description

In the contract `InfinityTradingRewards` the role `onlyOwner` has authority over the functions shown in the diagram below. Any compromise to the `onlyOwner` account may allow the hacker to take advantage of this.


Function
updateRewardsMap
Function
updateInfinityToken
State Variables
INFINITY_TOKEN
Function
updateInfinityExchange
State Variables
INFINTY_EXCHANGE
Function
rescueTokens
Function Calls
IERC20
Authenticated Role
_owner
Function
fundWithRewardToken
Function
updateInfinityStaker
State Variables
INFINITY_STAKER
Function
removeRewardToken
Function
rescueETH
Function
addRewardToken

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

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

ITR-02 | Potential Approval Risk On `claimReward()`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/core/InfinityTradingRewards.sol: 81	Resolved

Description

Because this function does not check the message sender for rewards rather it checks if the address has enough to claim. If the malicious actor was pre-approved to trade the token, then they could steal the rewards.

For example: Alice has 100 tokens to claim. Alice approves Bob to trade her tokens. Bob claims her tokens.

Recommendation

We recommend rechecking the logic and ensuring that this cannot happen. It is safer practice to check the message sender and only allow them to claim the tokens.

Alleviation

`[Infinity]`: Issue acknowledged. Changes have been reflected in the commit hash 12187d33964679322bfd01396e12d763aec46f0d.

ITR-03 | Potential Resource Exhaustion

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/core/InfinityTradingRewards.sol: 53	Acknowledged

Description

The function `updateRewards()` loops over a dynamic array. If this loop has to be run too many times it could exceed block limit.

Recommendation

We recommend verifying that the dynamic array inputs cannot cause an out of gas exception.

Alleviation

`[Infinity]`: Issue acknowledged. I will fix the issue in the future.

ITR-04 | Unclear Error Message

Category	Severity	Location	Status
Coding Style	Informational	contracts/core/InfinityTradingRewards.sol: 91	Acknowledged

Description

This error is correct. However, it would be better described as

```
`User withdrawing more than balance`
```

Recommendation

We recommend updating the error message to reflect an easier understanding.

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

ITR-05 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	Informational	contracts/core/InfinityTradingRewards.sol: 188	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee) in this farming, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

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

Recommendation

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

Alleviation

`[Infinity]`: Issue acknowledged. I won't make any changes for the current version.

Appendix

Finding Categories

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.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

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.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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

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.

About

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.