CERTIK

Security Assessment

Lifty

CertiK Verified on Apr 21st, 2023

CertiK Verified on Apr 21st, 2023

Lifty

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Marketplace	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 04/21/2023	N/A

CODEBASE

https://etherscan.io/address/0xa4b71b99eef99ea217adb7e3641cb3e779eaa529

Github: https://github.com/lifty-io/marketplace

...View All

COMMITS

Github commit: feb1cb8b324f0456e61590ee4e55e105ad3427da

...View All

Vulnerability Summary

11	5	1	0	5	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Mitigated	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.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 Resolved	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.
7	Informational	2 Resolved, 5 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | LIFTY

CODEBASE | LIFTY

Repository

https://etherscan.io/address/0xa4b71b99eef99ea217adb7e3641cb3e779eaa529

Github: https://github.com/lifty-io/marketplace

Commit

Github commit: feb1cb8b324f0456e61590ee4e55e105ad3427da

AUDIT SCOPE | LIFTY

3 files audited ● 2 files with Acknowledged findings ● 1 file with Mitigated findings

ID	Repo	File	SHA256 Checksum
● MAR	mainnet	contracts/Marketplace.sol	49fc123040906a147780ee1da62a303a3565392509ea4e02c5d0c77b767f304a
● OFH	mainnet	contracts/utils/OrderFulfiller.sol	d62e14ad2d4f80d4edbe9ed51ab41ac5714ee399dbefcb80a81c73487d6fde16
● FPH	mainnet	contracts/utils/fees/FeeProvider.sol	519d95ac5d44408a1f578d76f4f5cc37991d1298ac722b1e167f3a92c14457b9

APPROACH & METHODS | LIFTY

This report has been prepared for Lifty to discover issues and vulnerabilities in the source code of the Lifty 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.

REVIEW NOTES | LIFTY

Overview

`Lifty` is an NFT Ecosystem for Gamers and Games. `Lifty` created an NFT marketplace decentralized with features like creating a collection of NFT or exchanging the NFTs on the secondary market. `Lifty` marketplace is relying upon the OpenZepplin standard (`ERC721` , `ERC20`).

The `Lifty` marketplace has been deployed to the following address : 0xa4b71b99eef99ea217adb7e3641cb3e779eaa529.

External Dependencies

The `Lifty` contract relies on the *frontend* of the project that will generate the *hashes* to create the `orders` for the marketplace. Since the *frontend* is not in the current audit scope the *frontend* is considered as an **external dependency** (blackbox) and the auditors assume this is working as expected. In particular, the *frontend* decides certain fields of the orders, such as the `askAny` and `bidAny` flags, which are important in determining how an order is hashed.

The `Lifty` project uses OpenZeppelin libraries and contracts for contract format and functionality as well as for functions such as security and verification.

The following contracts & libraries are referenced in various contracts:

- `ERC721` , `Context` , `Ownable` , `ERC165`
- `ECDSA` , `ReentrancyGuard`
- `Operator` , `MerkleTree` , `TransferHelper` , `TokenTransferrer`

The following address interacts at some point with the specified contract, making it an external dependency:

- `collection` in `FeeProvider`

Privileged Functions

In the `Lifty` project, the `_owner` is adopted to ensure the dynamic runtime updates of the project, which were specified in the findings *FPH-01/ Centralization Risks in FeeProvider.sol*.

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worthy of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

FINDINGS | LIFTY

11	0	1	1	2	7
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Lifty. Through this audit, we have uncovered 11 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

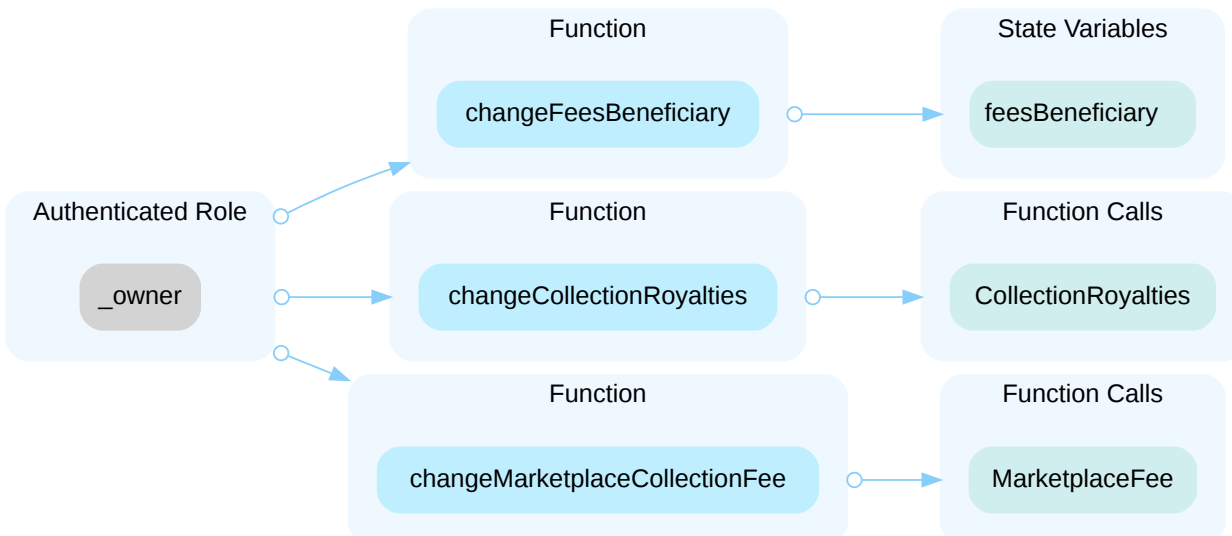
ID	Title	Category	Severity	Status
FPH-01	**Centralization Risks In FeeProvider.Sol**	**Centralization / Privilege**	**Major**	**Mitigated**
OFH-01	Possible Denial Of Service	Logical Issue	Medium	Resolved
FPH-02	Insufficient Check On Fees	Logical Issue	Minor	Resolved
MAR-02	Susceptible To Signature Malleability	Volatile Code	Minor	Resolved
FPH-03	Missing Zero Address Validation	Volatile Code	Informational	Resolved
MAR-03	Order Taker Decides Order Type	Logical Issue	Informational	Acknowledged
MAR-04	Lack Of Check On Expiration Time	Logical Issue	Informational	Resolved
MAR-05	Implementation Of Merkle Tree	Logical Issue	Informational	Acknowledged
MAR-06	Payable Buy Function	Coding Style	Informational	Acknowledged
OFH-02	Asset Used To Calculate Fees	Logical Issue	Informational	Acknowledged
OFH-03	Lack Of Checks On Royalty Fees	Logical Issue	Informational	Acknowledged

FPH-01 | CENTRALIZATION RISKS IN FEEPROVIDER.SOL

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/utils/fees/FeeProvider.sol: 139, 148, 172**	**Mitigated**

Description

In the contract `FeeProvider` the role `_owner` has authority over the functions shown in the diagram below:



- `changeFeesBeneficiary()` : Change the address of the beneficiary of the fees.
- `changeMarketplaceCollectionFee()` : Change the marketplace fees for a collection given.
- `changeCollectionRoyalties()` : Change the amounts and recipients of the royalties for a collection given

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the amount fees collected for a an arbitrary collection.

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

[Lifty - 03/07/2023] :
The team heeded the advice and mitigate the centralization risk by implementing multi-signature wallets (Gnosis Safe - 2/3 requirements) at the address 0xDDbB16DEB2ac7e8131CB9969130A028eBF32B52f.
The team is using a Timelock contract (48 hours delay) deployed at the address 0x6b8110EB8D06c88f5Aa1f836D92Bc9958454C59D as the `owner` of the marketplace.

Related information on the centralization risk can be found on the medium page of Lifty at this address : https://lifty.medium.com/liquidifty-is-set-to-enhance-security-and-decentralization-with-new-timelock-smart-contract-and-e21ea5fa1f0e

OFH-01 | POSSIBLE DENIAL OF SERVICE

Category	Severity	Location	Status
Logical Issue	Medium	contracts/utils/OrderFulfiller.sol: 116	Resolved

Description

The function `transferPaymentAndFees()` is checking if the royalties are set using the boolean `royaltiesApplied`. The `royaltiesApplied` boolean will be set to `true` if the condition `assets.length == 1` is met.
This is likely during some calls to the function `buy()`.

```
54  transferPaymentAndFees(from,to,payment[i],assets[0],assets.length == 1,amount);
```

In the case of the boolean `royaltiesApplied` is `true`, the contract `OrderFulfiller` will call the following statement:

```
115  IERC165(collection).supportsInterface(INTERFACE_ID_FEES);
```

This will check if the `collection` support the interface `INTERFACE_ID_FEES`, but because the ERC20 standard does not have the `supportsInterface()` function, this will revert and can lead to a Denial of Service during the calls to the `buy()` function.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Script.sol";
import "forge-std/Test.sol";

import "src/eth-mainnet-
0xa4b71b99eef99ea217adb7e3641cb3e779eaa529/contracts/Marketplace.sol";

contract PoC_revert {
    fallback() external payable {
    }
}

contract PoC is Script, Test {
    function setUp() public {}

    function run() public {
        PoC_revert P = new PoC_revert();
        IERC165(address(P)).supportsInterface(0xb7799584); /* This will revert even
if the contract has a fallback function /*
```

Results

```
[9079256848778921206] PoC::run()
    ├─ [25281] → new PoC_revert@0xC7f2Cf4845C6db0e1a1e91ED41Bcd0FcC1b0E141
    │   └─ ← 115 bytes of code
    ├─ [202] PoC_revert::supportsInterface(0xb7799584) [staticcall]
    │   └─ ← "EvmError: NotActivated"
    └─ ← "EvmError: Revert"
```

Recommendation

We recommend using a low-level call to avoid reverting when the signature `supportsInterface()` is not present in the contract, for example in the ERC20 standard, and handling the result of the low-level call.

Alleviation

[Lifty - 04/21/2023] :

The team heeded the advice and resolved this issue by adding a `try-catch` statement in the commit feb1cb8b324f0456e61590ee4e55e105ad3427da.

FPH-02 | INSUFFICIENT CHECK ON FEES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/utils/fees/FeeProvider.sol: 154	Resolved

Description

When changing the fees of a token, there is a check to ensure that the seller fee and the buyer fee are both less than 100%.

```
148     function changeMarketplaceCollectionFee(
149         address collection,
150         uint16 buyerFee,
151         uint16 sellerFee
152     ) external onlyOwner {
153         require(
154             sellerFee < 10000 && buyerFee < 10000,
155             "FeeProvider: wrong fee amount"
156         );
```

However, this check is insufficient as both seller and buyer fees are applied to an order amount, meaning that the sum of the fees should be less than 100%.

```
 92             uint256 buyerFee = calculateFee(
 93                 payment.amount,
 94                 marketplaceFee.buyerFee
 95             );
 96             uint256 sellerFee = calculateFee(
 97                 payment.amount,
 98                 marketplaceFee.sellerFee
 99             );
100
101             fee = (buyerFee + sellerFee) * amount;
```

Recommendation

We recommend changing the check to `sellerFee + buyerFee < 10000` .

Alleviation

[Lifty - 03/07/2023] :

The team heeded the advice and resolved this issue by changing the fees requirement mechanism in the `changeMarketplaceCollectionFee()` function in the commit ac1f7ad2317e172a4b09fc982830465131a4c692.

MAR-02 | SUSCEPTIBLE TO SIGNATURE MALLEABILITY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Marketplace.sol: 97, 126, 251	Resolved

Description

The functions `ECDSA.recover` and `ECDSA.tryRecover` are vulnerable to a kind of signature malleability due to accepting EIP-2098 compact signatures in addition to the traditional 65-byte signature format.

This is only an issue for the functions that take a single bytes argument, and not the functions that take `r, v, s` or `r, vs` as separate arguments.

Reference: https://github.com/advisories/GHSA-4h98-2769-gh6h

Recommendation

Recommend using the latest stable version of the OpenZeppelin library during deployment to avoid the risk of potential vulnerabilities in an outdated version.

Alleviation

[Lifty - 03/07/2023] :
The team heeded the advice and resolved this issue by upgrading the version of the OpenZeppelin library to 4.8.0 in the commit 1878e8d8721d2b7c23da319a8a79ed9f56aaf9b9.

FPH-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Informational	contracts/utils/fees/FeeProvider.sol: 142	Resolved

Description

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

```
142         feesBeneficiary = newFeesBeneficiary;
```

- `newFeesBeneficiary` 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

[Lifty - 03/07/2023] :

The team heeded the advice and resolved this issue by adding the missing zero validation requirement in the function `changeFeesBeneficiary()` in the commit 93e5bed053f5bc46e43f3cb400015e7b0e439127.

MAR-03 | ORDER TAKER DECIDES ORDER TYPE

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Marketplace.sol: 142	Acknowledged

Description

The order type determines how fees are implemented, where the order maker pays fees for an `OFFER` and the order taker pays fees for a `SALE`. In the case of a `SWAP`, no fees are applied.

Since the order type is not part of an order's hash, this allows the order taker to always state the order's type is a `SWAP` to avoid fees.

Recommendation

We advise hashing all the necessary information to avoid malleability or managing edge cases independently.

Alleviation

[Lifty - 03/07/2023] :
The team acknowledged the finding and decided not to make any related changes at the moment, however the team mentioned they will fix this issue in the next release of the protocol.

MAR-04 | LACK OF CHECK ON EXPIRATION TIME

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Marketplace.sol: 75	Resolved

Description

There is no explicit check that each order meant to be filled in a `buy()` call has not expired yet. The only check involving expiration is that an operator has signed the order hashes along with an `expiredAt` time that is in the future.

```
73          require(expiredAt > block.timestamp, "Buy: the deal is expired.");
74          require(
75              verifyDealSign(expiredAt, dealSign, ordersHashes),
76              "Buy: wrong deal signature"
77          );
```

Since this `expiredAt` time is not compared with an order's expiration time, an operator is able to approve a deal involving an already expired order.

Recommendation

We recommend adding a requirement on the `order.expirationDate` to ensure the expiration is correct.

Alleviation

[Lifty - 03/07/2023] :

The team heeded the advice and resolved this issue by adding a requirement on the expiration order function `proceedOrder()` in commit 6ae6b4292770bae97b8dccc05fdd0134a3fb4304.

MAR-05 | IMPLEMENTATION OF MERKLE TREE

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Marketplace.sol: 132	Acknowledged

Description

When processing an order, there is a check requiring proof that the order's hash is compatible with the root of a Merkle tree.

```
125         require(
126             order.signer == recoverSigner(order.rootSign, order.root),
127             "ProceedOrder: wrong root sign"
128         );
129 
130         // check order merkle tree proof
131         require(
132             MerkleTree.verify(order.proof, order.root, orderHash),
133             "ProceedOrder: invalid proof"
134         );
```

The root of the Merkle tree is also checked to be signed by `order.signer`.

Since `order.amount` is not included in an order's hash, this allows the same order, with the same `order.root` and the same `order.rootSign`, but with a different `order.amount` to be processed multiple times.

Recommendation

We recommend ensuring that the `order.amount` cannot be processed multiple times.

Alleviation

[Lifty - 03/07/2023] :
The team acknowledged the finding and decided not to make any related changes at the moment.

MAR-06 | PAYABLE BUY FUNCTION

Category	Severity	Location	Status
Coding Style	Informational	contracts/Marketplace.sol: 72	Acknowledged

Description

The function `buy()` is used to fill orders involving `ERC20`, `ERC721`, `ERC1155`, and native tokens.

```
67    function buy(
68          Order[] calldata orders,
69          uint expiredAt,
70          bytes calldata dealSign,
71          bytes32[] calldata ordersHashes
72      ) external payable nonReentrant {
73          require(expiredAt > block.timestamp, "Buy: the deal is expired.");
74          require(
75              verifyDealSign(expiredAt, dealSign, ordersHashes),
76              "Buy: wrong deal signature"
77          );
78
79          for (uint256 i = 0; i < orders.length; ) {
80              proceedOrder(orders[i], ordersHashes[i]);
81              unchecked {
82                  ++i;
83              }
84          }
85      }
```

Since there is no check that `msg.value` is the correct amount in the case of native token transfers, it is possible for native tokens to forever be locked within the contract.

Recommendation

We recommend ensuring the `msg.value` is correct or providing a refund if it is higher than expected.

Alleviation

[Lifty - 03/07/2023] :
The team acknowledged the finding and decided not to make any related changes at the moment. The team mentioned they are checking the amount manually using the front-end application and also fix this issue in the next release of the protocol.

OFH-02 | ASSET USED TO CALCULATE FEES

Category	Severity	Location	Status
Logical Issue	Informational	contracts/utils/OrderFulfiller.sol: 58	Acknowledged

Description

When fees are applied to a transfer, the fees used are determined by the first asset in the `assets` array, which is the first bid asset in a sale and the first ask asset in an offer.

```
54              transferPaymentAndFees(
55                  from,
56                  to,
57                  payment[i],
58                  assets[0],
59                  assets.length == 1,
60                  amount
61              );
```

This allows the order maker to arrange the bid and ask arrays such that the first token in each array has the lowest fees, resulting in possibly fewer fees collected by the protocol.

Recommendation

We recommend covering all cases for ensuring the fees are distributed correctly and cannot be bypassed.

Alleviation

[Lifty - 03/07/2023] :
The team acknowledged the finding and decided not to make any related changes at the moment, however the team mentioned they will fix this issue in the next release of the protocol.

OFH-03 | LACK OF CHECKS ON ROYALTY FEES

Category	Severity	Location	Status
Logical Issue	Informational	contracts/utils/OrderFulfiller.sol: 120	Acknowledged

Description

In the function `changeMarketplaceCollectionFee()`, upper fee checks are implemented on the `marketplaceCollectionFee`.

```
148     function changeMarketplaceCollectionFee(
149         address collection,
150         uint16 buyerFee,
151         uint16 sellerFee
152     ) external onlyOwner {
153         require(
154             sellerFee < 10000 && buyerFee < 10000,
155             "FeeProvider: wrong fee amount"
156         );
```

However, for the *royalties* that are using `customCollectionRoyalties` the royalties are calculated using an external call `IGetRoyalties(collection)` without upper checks requirements.

```
115   if (IERC165(collection).supportsInterface(INTERFACE_ID_FEES)) {
116             IGetRoyalties collection = IGetRoyalties(collection);
117             return (collection.getFeeRecipients(id), collection.getFeeBps(id));
118         }
```

Recommendation

We recommend not relying on external calls. If this is necessary for the business logic then the values should be checked to ensure that unexpected values are not returned.

Alleviation

[Lifty - 03/07/2023] :

The team acknowledged the finding and decided not to make any related changes in the codebase. However, the team mentioned they will fix this issue in the next release of the protocol.

APPENDIX | LIFTY

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.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

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

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

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.