CHAIN TROOPERS

Zircon Finance

Protocol 2

Security Assessment Report

August 31th, 2022

Version 1.1

CONFIDENTIAL

Table of Contents

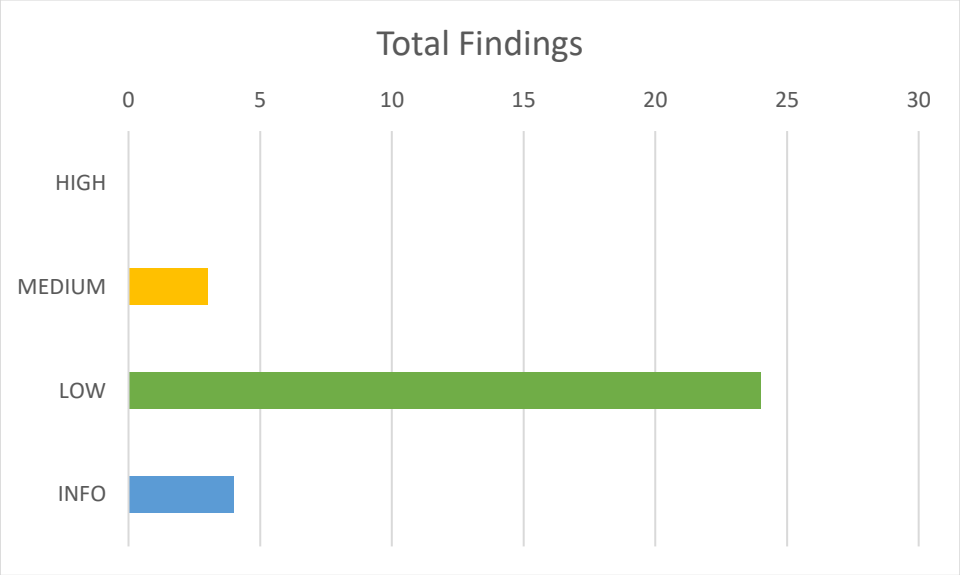
1 Executive Summary

1.1 Introduction

The report contains the results of Zircon Finance Protocol 2 security assessment that took place from July 18th, 2022, to August 4th, 2022, and from August 24th, 2022, to August 26th, 2022. The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

In total, the team identified thirty-three (33) vulnerabilities. There were also four (4) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity. A retesting phase was carried out on August 25th, 2022, and the results are presented in Section 6.

1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to five (5) Business Logic and Data Validation issues of MEDIUM risk. Regarding the Business Logic issues, it was identified that the order of operations in the function that calculates the pending rewards and the fee by gamma can introduce a loss of precision due to round-off errors *('5.1.1* - Order of operations can introduce round-off errors at "PsionicFarmInitializable.sol"*', '5.1.4* - Order of operations can introduce round-off errors at "ZirconEnergy.sol"*', '5.1.5* - *Order of operations can* introduce *round-off errors at "ZirconEnergyRevenue.sol"')*, which occur because the division of a number takes place before a follow-up multiplication. Furthermore, it was found that the contract was minting more fees than the design *('5.1.3* - Double *minting in "_mintFee" at "ZirconPair.sol"*). In reference to the Data Validation issues, the team identified that one deposit function is using the "*tx.origin*" to perform its operations and withdraw the reward tokens *('5.1.2* - Deposit through tx.origin in "routerDeposit()" at "PsionicFarmInitializable.sol"*')*. This issue can be abused by an adversary who can trick a legitimate user to call this deposit functionality through a contract's fallback function, and as a result perform the deposit using the "*tx.origin*" of the legitimate user.

The in-scope components were also affected by twenty-four (24) Data Validation, Administration, Network Layer, Access Control and Error Handling vulnerabilities of MEDIUM risk. Regarding the LOW-risk Administration issues, the team identified that many admin functionalities do not emit the appropriate event when an emergency action is taken *('5.2.15* - Event not emitted in "emergencyRewardWithdraw" and "stopReward()" functionalities at "PsionicFarmInitializable.sol"*),* potentially affecting the credibility and the confidence in the system. Regarding the Error Handling LOW-risk issues, it was found that many functionalities do not check the return value of "create2" ('*5.2.24* - Unchecked create2 call return value in "deployPool()" at "PsionicFarmFactory.sol"*', '5.2.25* - Unchecked create2 call return value in "createPair()" at "ZirconFactory.sol"*', '5.2.26* - Unchecked create2 call return value in "createPTAddress()" at "ZirconPTFactory.sol"*', '5.2.27* - Unchecked create2 call return value in "createPylon()" at "ZirconPylonFactory.sol"*'),* allowing the execution to resume even if the creation fails.

In reference to the Data Validation issues of LOW risk, it was identified that certain functions are not validating the provided addresses to not be the zero address *('5.2.1* - Unvalidated _token_address in constructor at "ZirconDrop.sol"*', '5.2.2* - Unvalidated _to address in "burn" function at "PsionicFarmVault.sol"*', '5.2.9* - Unvalidated addresses in functions "setFeeToSetter", "setMigrator" and the constructor at "ZirconEnergyFactory.sol"*', '5.2.10* - Unvalidated addresses in functions "mint", "burn", "initialize", "changePylonAddress" and the constructor at "ZirconPoolToken.sol"*', '5.2.11* - Unvalidated addresses in functions "initialize", "mint", "mintOneSide", "burnOneSide", "burn", "skim" and "changeEnergyRevAddress" at "ZirconPair.sol"*', '5.2.12* - Unvalidated *addresses in functions "setMigrator" at "Migrator.sol").* An accidental transfer of several tokens to the zero address is equivalent to burning that number of tokens. A number of integer overflows were also found to be feasible in certain functionalities *(`5.2.13* - *Integer* overflow *in "calculate()" at "ZirconEnergyRevenue.sol" `, `5.2.14* - Integer *overflow* using *"_liquidityfee" at "ZirconEnergyRevenue.sol" and "ZirconPylon.sol" `).* However, a privileged user had to set specially crafted values to exploit these issues. Moreover, the team identified that many smart contracts are using a floating pragma or support deprecated and outdated Solidity versions *('5.2.19* - Floating pragma in multiple contracts*', '5.2.20* - Outdated compiler version specified at multiple contracts*').* In Solidity programming, multiple APIs are only supported in some specific versions.

In addition to the previous Data Validation LOW-risk issues, the team also identified that the "permit()" functionality at ZirconERC20 was vulnerable to replay attacks when the nonces counter overflows *('5.2.3* - Replay attack in permit() function if nonces counter overflows at "ZirconERC20.sol"*'),* allowing adversaries to reuse previous permits without the owner's knowledge. Moreover, it was found that the initialization function in many contacts can be called multiple times *('5.2.4* - Multiple initializations are allowed in "ZirconPair.sol"*', '5.2.5* - Multiple initializations are allowed in "ZirconPoolToken.sol"*', '5.2.6* - Multiple initializations are allowed in "ZirconPylon.sol"*', '5.2.7* - Multiple initializations are allowed in "ZirconEnergy.sol"*', '5.2.8* - Multiple initializations are allowed in "ZirconEnergyRevenue.sol"*'),* even if the contract is active an operational.

Furthermore, it was identified that the function which is used by administrators or the farm to burn and remove tokens, contains a potentially costly loop that makes the function inefficient for using it in emergency cases *('5.2.18* - Excessive loop iterations allowed in "burn", "remove" and constructor functions at "PsionicFarmVault.sol"*').* If the farm contains a significantly large array of tokens symbols, it is possible that the function will not be fully executed neither in the

current nor in the following blocks, allowing adversaries who monitor the transactions to front run and circumvent this action. Regarding the Network Layer LOW-risk issues, it was found that the "transfer()" function is used to transfer the claimed tokens *('5.2.17* - Gas limited function is used for token transfer in "claim" function at "ZirconDrop.sol"*'),* which is always forwarding a fixed amount of gas and may be incompatible with certain contracts.

In reference to the Access Control LOW-risk issues, it was found that the contracts do not have a dedicated circuit breaker control that can be used in case of emergency to pause the transactions *('5.2.22* - Lack of circuit breaker for emergency stop at "ZirconDrop.sol"*, '5.2.23* - Lack of circuit breaker for emergency stop at "PsionicFarmInitializable.sol" and "PsionicFarmVault.sol"*').* Furthermore, no multisign protection is used in many admin functionalities *('5.2.21* - No multisig protection in "onlyOwner" modifier at "ZirconDrop.sol"*, '5.2.28* - No multisig protection in "onlyOwner" modifier at "PsionicFarmInitializable.sol"*').* Finally, it was found that the Uniswap V2 skim function is exposed ('*5.2.16* - The *Uniswap V2 skim functionality is exposed at "ZirconPair.sol"'),* and can be abused by malicious users to withdraw tokens.

There were also four (4) findings of no-risk (INFORMATIONAL).

Chaintroopers recommend the immediate mitigation of all MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

1.2.1 Retesting Results

Results from retesting carried out on August 2022, determined that two out of three (2 of 3) MEDIUM risk issues (see sections 5.1.2, and 5.1.4), eight out of twenty-four (8 of 24) reported LOW-risk issues (see sections 5.2.1, 5.2.2, 5.2.7, 5.2.8, 5.2.22, 5.2.23, 5.2.24, and 5.2.25) and two out of four (2 of 4) INFORMATIONAL issues (see sections 5.3.1, and 5.3.3) were sufficiently addressed (12 out of 31 findings).

1.3 Summary of Findings

The following findings were identified in the examined source code:

Vulnerability Name	Status	Retest Status	Page
Order of operations can introduce round-off errors at "PsionicFarmInitializable.sol"	MEDIUM	MEDIUM	18
Deposit through tx.origin in "routerDeposit()" at "PsionicFarmInitializable.sol"	MEDIUM	CLOSED	21
Double minting in "_mintFee" at "ZirconPair.sol"	N/A	MEDIUM	26
Order of operations can introduce round-off errors at "ZirconEnergy.sol"	MEDIUM	CLOSED	28
Order of operations can introduce round-off errors at "ZirconEnergyRevenue.sol"	N/A	MEDIUM	30
Unvalidated _token_address in constructor at "ZirconDrop.sol"	LOW	CLOSED	33
Unvalidated _to address in "burn" function at "PsionicFarmVault.sol"	LOW	CLOSED	35
Replay attack in permit() function if nonces counter overflows at "ZirconERC20.sol"	LOW	LOW	37
Multiple initializations are allowed in "ZirconPair.sol"	LOW	LOW	39
Multiple initializations are allowed in "ZirconPoolToken.sol"	LOW	LOW	41
Multiple initializations are allowed in "ZirconPylon.sol"	LOW	LOW	43
Multiple initializations are allowed in "ZirconEnergy.sol"	LOW	CLOSED	47

Multiple initializations are allowed in "ZirconEnergyRevenue.sol"	LOW	CLOSED	49
Unvalidated addresses in functions "setFeeToSetter", "setMigrator" and the constructor at "ZirconEnergyFactory.sol"	LOW	LOW	51
Unvalidated addresses in functions "mint", "burn", "initialize", "changePylonAddress" and the constructor at "ZirconPoolToken.sol"	LOW	LOW	53
Unvalidated addresses in functions "initialize", "mint", "mintOneSide", "burnOneSide", "burn", "skim" and "changeEnergyRevAddress" at "ZirconPair.sol"	LOW	LOW	55
Unvalidated addresses in functions "setMigrator" at "Migrator.sol"	N/A	LOW	58
Integer overflow in "calculate()" at "ZirconEnergyRevenue.sol"	N/A	LOW	59
Integer overflow using "_liquidityfee" at "ZirconEnergyRevenue.sol" and "ZirconPylon.sol"	N/A	LOW	62
Event not emitted in "emergencyRewardWithdraw" and "stopReward()" functionalities at "PsionicFarmInitializable.sol"	LOW	LOW	66
The Uniswap V2 skim functionality is exposed at "ZirconPair.sol"	N/A	LOW	68
Gas limited function is used for token transfer in "claim" function at "ZirconDrop.sol"	LOW	LOW	70
Excessive loop iterations allowed in "burn", "remove" and constructor functions at "PsionicFarmVault.sol"	LOW	LOW	72
Floating pragma in multiple contracts	LOW	LOW	76

Outdated compiler version specified at multiple contracts	LOW	LOW	78
No multisig protection in "onlyOwner" modifier at "ZirconDrop.sol"	LOW	LOW	81
Lack of circuit breaker for emergency stop at "ZirconDrop.sol"	LOW	CLOSED	83
Lack of circuit breaker for emergency stop at "PsionicFarmInitializable.sol" and "PsionicFarmVault.sol"	LOW	CLOSED	85
Unchecked create2 call return value in "deployPool()" at "PsionicFarmFactory.sol"	LOW	CLOSED	87
Unchecked create2 call return value in "createPair()" at "ZirconFactory.sol"	LOW	CLOSED	89
Unchecked create2 call return value in "createPTAddress()" at "ZirconPTFactory.sol"	LOW	LOW	91
Unchecked create2 call return value in "createPylon()" at "ZirconPylonFactory.sol"	LOW	LOW	93
No multisig protection in "onlyOwner" modifier at "PsionicFarmInitializable.sol"	LOW	LOW	95
Unchecked transfer call return value in "claim" functionality at "ZirconDrop.sol" (needs further examination)	INFO	CLOSED	98
Raw data left in published repository at "ZirconDrop"	INFO	INFO	100
Debugging utility in "PsionicFarmVault.sol"	INFO	CLOSED	102
It is impossible to use "withdraw()" or "deposit()" if user balance is below a certain level at "PsionicFarmInitializable.sol"	INFO	INFO	103

2 Assessment Description

2.1 Target Description

Zircon is a DeFi ecosystem focused on unlocking previously untapped yield generation opportunities and simplifying them for the average DeFi user. Its core product is Zircon Pylon, a protocol that builds on the basic Uniswap V2 architecture to enable single-sided liquidity management.

Zircon Pylon is a risk tranching solution that enables high-risk and low-risk asset holders to stay in the same pool without any compromises to their portfolio. It keeps track of the assets that were supplied to the protocol through virtual variables, that are also used to define the relative share of the pool for each side: the high-risk Float and the low-risk Anchor. There are two main rules that define the relationship between the two sides:

- The Anchor side can only claim as many assets as were originally supplied, plus any fees they accrued.
- The Float side can claim the excess that is not claimed by the Anchor, but it must compensate Anchors when the Anchor virtual balance is higher than their underlying reserve in the pool.

Each Zircon pool has its Pylon vault, which holds a portion of the pool's liquidity through the pair's pool tokens. The pool has some reserves that come both from Pylon users and any external liquidity provider. Liquidity can be provided in two ways:

- The Sync liquidity provisioning (LPing), in which the protocol collects deposits from the two sides and send them into the pool when there is a match.
- The Async liquidity provisioning, in which the protocol sells the tokens against the pool to form the 50-50 split required by Uniswap. Users still get only Float or Anchor tokens depending on which one they chose.

2.2 In-Scope Components

The following list specifies the repository that was subject to this audit. Tests are excluded.

- *https://github.com/Zircon-Finance/zircon-protocol-2/tree/develop*

Component	Commit Identifier
zircon-protocol-2 (develop)	*e3e5b8a93a73b598d602bc482282de6e25dabb61*
zircon-protocol-2 (develop) - Retest	*fa04a132a81600612756397927dd52847b673f74*

3 Methodology

3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

3.2 Smart Contracts

The testing methodology used is based on the empirical study "Defining Smart Contract Defects on Ethereum" by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo and T. Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in "Security Considerations" section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement
- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment
- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

In Substrate Pallets, the list of vulnerabilities that are identified also includes:

- Static or Erroneously Calculated Weights
- Arithmetic Overflows
- Unvalidated Inputs
- Runtime Panic Conditions
- Missing Storage Deposit Charges
- Non-Transactional Dispatch Functions
- Unhandled Errors &Unclear Return Types
- Missing Origin Authorization Checks

4 Scoring System

4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

Rating	CVSS Score
None/Informational	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

5 Identified Findings

5.1 Medium Severity Findings

5.1.1 Order of operations can introduce round-off errors at "PsionicFarmInitializable.sol"

Description	MEDIUM

The team identified that the order of operations in "*pendingReward*()" function can introduce a loss of precision due to round-off errors. A roundoff error, also called rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic. Rounding errors are due to inexactness in the representation of real numbers and the arithmetic operations done with them. These types of errors occur when division of a number takes place before a multiplication.

In the specific case the *adjustedTokenPerShare* is calculated by dividing with *stakedTokenSupply*, and then multiplied with *user.amount* before dividing again with *PRECISION_FACTOR*.

The issue exists at the following location:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
296:     /*
297:      * @notice View function to see pending reward on frontend.
298:      * @param _user: user address
299:      * @return Pending reward for a given user
300:      */
301:     function pendingReward(address _user) external view  returns
(uint256) {
302:         UserInfo storage user = userInfo[_user];
303:         uint256 stakedTokenSupply =
stakedToken.balanceOf(address(this));
304:         if (block.number > lastRewardBlock  stakedTokenSupply != 0)
{
305:             uint256 multiplier = _getMultiplier(lastRewardBlock,
block.number) * 1e18;
306:             uint256 adjustedTokenPerShare = accTokenPerShare +
```

```
(multiplier * PRECISION_FACTOR) / stakedTokenSupply;
307:             return (user.amount * adjustedTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
308:         } else {
309:             return (user.amount * accTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
310:         }
311:     }
```

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
296:     /*
297:      * @notice View function to see pending reward on frontend.
298:      * @param _user: user address
299:      * @return Pending reward for a given user
300:      */
301:     function pendingReward(address _user) external view  returns
(uint256) {
302:         UserInfo storage user = userInfo[_user];
303:         uint256 stakedTokenSupply =
stakedToken.balanceOf(address(this));
304:         if (block.number > lastRewardBlock  stakedTokenSupply != 0)
{
305:             uint256 multiplier = _getMultiplier(lastRewardBlock,
block.number) * 1e18;
306:             uint256 adjustedTokenPerShare = accTokenPerShare +
(multiplier * PRECISION_FACTOR) / stakedTokenSupply;
307:             return (user.amount * adjustedTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
308:         } else {
309:             return (user.amount * accTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
310:         }
311:     }
```

Impact

The current implementation introduces a small loss on precision due to the order of multiplication and division. This can affect the pendingReward values.

Recommendation

It is recommended to rearrange the mathematical operations and try to perform all multiplications before any divisions, to increase precision.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.2 Deposit through tx.origin in "routerDeposit()" at "PsionicFarmInitializable.sol"

Description	MEDIUM

It was found that the "*routerDeposit*()" function is using the "*tx.origin*" to perform a deposit operation and withdraw the reward tokens. The "*tx.origin*" is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract.

For example, let's assume that a victim A, is tricked by an adversary to make a transaction towards their malicious contract. This contract can use a fallback function to call the "*routerDeposit*()" at PsionicFarmInitializable contract with the victim's A address since the authorized address is stored in "*tx.origin*:"

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
159:     /*
160:      * @notice Deposit staked tokens and collect reward tokens (if
any)
161:      * @param _amount: amount to withdraw (in rewardToken)
162:      */
163:     function routerDeposit(uint256 _amount) external nonReentrant {
164:         deposit(_amount, tx.origin);
165:     }
166:
```

Then, the "deposit()" function will perform the following operations:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
127:     function deposit(uint256 _amount, address sender) internal {
128:         UserInfo storage user = userInfo[sender];
129:         userLimit = hasUserLimit();
130:         require(!userLimit || ((_amount + user.amount) =
poolLimitPerUser), "Deposit: Amount above limit");
131:
132:         _updatePool();
133:
134:         if (user.amount > 0) {
135:             uint256 pending = (user.amount * accTokenPerShare) /
```

```
PRECISION_FACTOR - user.rewardDebt;
136:              if (pending > 0) {
137:                  _sendTokens(pending, address(sender));
138:              }
139:          }
140:
141:          if (_amount > 0) {
142:              user.amount = user.amount + _amount;
143:              stakedToken.safeTransferFrom(address(sender),
address(this), _amount);
144:          }
145:
146:          user.rewardDebt = (user.amount * accTokenPerShare) /
PRECISION_FACTOR;
147:
148:          emit Deposit(msg.sender, _amount);
149:      }
150:
```

Impact

An adversary can trick a legitimate user to call the "*routerDeposit*()" function through another fallback function, and as a result perform an action using the "tx.orign" of the legitimate user.

The issue is marked as MEDIUM since the "*safeTransferFrom*()" function is used. As a result, since the legitimate user ("from" field) is not the caller of the function, it must have been allowed to perform this transfer using "approve" earlier. However, if the approve for a large amount has been made, the adversary can still force the victim to do transfers in arbitrary times.

Recommendation

To prevent this kind of attack, it is advisable to restrict access to this functionality or use the msg.sender.

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.3 Double minting in "_mintFee" at "ZirconPair.sol"

Description	MEDIUM

The team identified that the "_mintFee" functionality at the "ZirconPair" is minting more energy revenue than expected based on the documentation.

The issue exists at the following location:

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconPair.sol
112:
113:     // if fee is on, mint liquidity equivalent to 1/6th of the
growth in sqrt(k)
114:     function _mintFee(uint112 _reserve0, uint112 _reserve1) private
{
115:     uint _kLast = kLast; // gas savings
116:         if (_kLast != 0) {
117:             uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
118:             uint rootKLast = Math.sqrt(_kLast);
119:             if (rootK > rootKLast) {
120:                 uint dynamicRatio =
IZirconFactory(factory).dynamicRatio();
121:                 uint numerator = (rootK.sub(rootKLast)).mul(1e18);
122:                 uint denominator =
rootK.mul(dynamicRatio).add(rootKLast);
123:                 uint liquidityPercentage = numerator / denominator;
124:
125:                 if (liquidityPercentage > 0) {
126: //                    console.log("C ore: liqPercentage",
liquidityPercentage);
127:                     _mint(energyRevenueAddress,
liquidityPercentage.mul(totalSupply)/1e18);
128:                     _mint(energyRevenueAddress,
liquidityPercentage.mul(totalSupply)/1e18);
129:                     uint totalPercentage =
((rootK.sub(rootKLast)).mul(1e18))/rootKLast;
130:
IZirconEnergyRevenue(energyRevenueAddress).calculate(totalPercentage.sub(
liquidityPercentage));
```

```
131:                     }
132:                 }
133:             }
134:
135:         }
```

Impact

The contract is minting more energy revenue than the expected design.

Recommendation

It is advisable to remove one of the minting operations.

CVSS Score

AV:N/AC:L/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.4 Order of operations can introduce round-off errors at "ZirconEnergy.sol"

Description	MEDIUM

The team identified that the order of operations in "*getFeeByGamma*()" function can introduce a loss of precision due to round-off errors. A roundoff error, also called rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic. Rounding errors are due to inexactness in the representation of real numbers and the arithmetic operations done with them. These types of errors occur when division of a number takes place before a multiplication.

In the specific case the fee amount is calculated by dividing the multiplication of *_minFee* with *x*, with *1e18*, and then multiplied *x* and *36* before dividing again with *1e18.*

The issue exists at the following location:

```
File: /packages/zircon-core/contracts/energy/ZirconEnergy.sol
151:   //Returns the fee in basis points (0.01% units, needs to be
divided by 10000)
152:   //Uses two piece-wise parabolas. Between 0.45 and 0.55 the fee is
very low (minFee).
153:   //After the cutoff it uses a steeper parabola defined by a max fee
at the extremes (very high, up to 1% by default).
154:   function getFeeByGamma(uint gammaMulDecimals) external view
returns (uint amount) {
155:     (uint _minFee, uint _maxFee) = getFee();
156:     uint _gammaHalf = 5e17;
157:     uint x = (gammaMulDecimals > _gammaHalf) ? (gammaMulDecimals -
_gammaHalf).mul(10) : (_gammaHalf - gammaMulDecimals).mul(10);
158:     if (gammaMulDecimals = 45e16 || gammaMulDecimals >= 55e16) {
159:       amount = (_maxFee.mul(x)/1e18).mul(x)/(25e18); //25 is a
reduction factor based on the 0.45-0.55 range we're using.
160:     } else {
161:       amount =
((_minFee.mul(x)/1e18).mul(x).mul(36)/(1e18)).add(_minFee); //Ensures
minFee is the lowest value.
```

```
162:     }
163:   }
164:
```

Impact

The current implementation introduces a small loss on precision due to the order of multiplication and division. This can affect the fee value.

Recommendation

It is recommended to rearrange the mathematical operations and try to perform all multiplications before any divisions, to increase precision.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:L/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.1.5 Order of operations can introduce round-off errors at "ZirconEnergyRevenue.sol"

Description	MEDIUM

The team identified that the order of operations in "calculate()" function at "ZirconEnergyRevenue.sol" can introduce a loss of precision due to round-off errors. A roundoff error, also called rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic. Rounding errors are due to inexactness in the representation of real numbers and the arithmetic operations done with them. These types of errors occur when division of a number takes place before a multiplication.

In the specific case the fee amount is calculated by dividing the multiplication of amount with pylonBalance0, with totalSupply, and then multiply it with (100 - feePercentageForRev) before dividing again with 100.

The issue exists at the following location:

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/energy/ZirconEnergyRevenue.sol

```
073:     function calculate(uint percentage) external _onlyPair
nonReentrant _initialize {
074:         uint balance =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(address(this));
075:         require(balance > reserve, "ZER: Reverted");
076:
077:         //Percentage is feeValue/TPV, percentage of pool reserves
that are actually fee
078:         //It's reduced by mint fee already
079:         uint totalSupply =
IUniswapV2ERC20(zircon.pairAddress).totalSupply();
080:         //These are the PTBs, balance of pool tokens held by each
pylon vault
081:         uint pylonBalance0 =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(zircon.pylon0);
082:         uint pylonBalance1 =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(zircon.pylon1);
```

```
083:         {
084:             (uint112 _reservePair0, uint112 _reservePair1,) =
IZirconPair(zircon.pairAddress).getReserves();
085:
086:             //Increments the contract variable that stores total
fees acquired by pair. Multiplies by each Pylon's share
087:
088:             feeValue0 +=
percentage.mul(_reservePair1).mul(2).mul(pylonBalance0)/totalSupply.mul(1
e18);
089:             feeValue1 +=
percentage.mul(_reservePair0).mul(2).mul(pylonBalance1)/totalSupply.mul(1
e18);
090:         }
091:
092:         {
093:             uint feePercentageForRev =
IZirconEnergyFactory(energyFactory).feePercentageRev();
094:             uint amount = balance.sub(reserve);
095:             uint pylon0Liq =
(amount.mul(pylonBalance0)/totalSupply).mul(100 -
feePercentageForRev)/(100);
096:             uint pylon1Liq =
(amount.mul(pylonBalance1)/totalSupply).mul(100 -
feePercentageForRev)/(100);
..
100:         }
```

Impact

The current implementation introduces a small loss on precision due to the order of multiplication and division. This can affect the fee value.

Recommendation

It is recommended to rearrange the mathematical operations and try to perform all multiplications before any divisions, to increase precision.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:L/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2 Low Severity Findings

5.2.1 Unvalidated _token_address in constructor at "ZirconDrop.sol"

Description	LOW

The team identified that the constructor of ZirconDrop is not validating the provided address to not be the zero address. Transferring a number of tokens to the zero address during the "recharge" operation, is equivalent to burning that number of tokens. The transaction should not revert (there's no contract that would throw an exception on the 0x0 address) and should not return any data (again, no contract that would return any data).

```
File: /zircon-airdrop/contracts/ZirconDrop.sol
34:     constructor (address _token_address, bytes32 _merkleRoot, uint256
_start_time, uint256 _end_time) {
35:         require(validRange(48, _start_time), "Invalid Start Time");
36:         require(validRange(48, _end_time), "Invalid End Time");
37:
38:         merkleRoot = _merkleRoot;
39:         info.start_time = uint48(_start_time);
40:         info.end_time = uint48(_end_time);
41:         info.token_address = _token_address;
42:         creator = msg.sender;
43:     }
```

Impact

The owner can use the "recharge" function to transfer a number of tokens to the zero address (either accidentally or on purpose set during the construction of the contract), effectively burning that number of tokens.

Recommendation

It is advisable to verify that the address is not the zero address. The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

if (_token_address == address(0)) revert InvalidAddress();

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.2 Unvalidated _to address in "burn" function at "PsionicFarmVault.sol"

Description	LOW

The team identified that the "burn" function of PsionicFarmVault is not validating the provided address to not be the zero address. Transferring an amount of tokens to the zero address is equivalent to burning that amount of tokens.

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
115:     /*
116:     *   @notice Burn Function to withdraw tokens
117:     *   @param _to: the address to send the rewards token
118:     */
119:     function burn(address _to, uint _liquidity) external onlyFarm
nonReentrant {
120:         require(_liquidity > 0, "PFV: Not enough");
121:         address[] memory tokensToWithdraw = rewardTokens;
122:         uint tokenLength = tokensToWithdraw.length;
123:         uint ts = totalSupply();
124:
125:         for (uint256 i = 0; i  tokenLength; i++) {
126:             IERC20 token = IERC20(tokensToWithdraw[i]);
127:             uint balance = token.balanceOf(address(this));
128:             uint amount = (balance * (_liquidity))/ts;
129:             token.safeTransfer(_to, amount);
130:         }
131:         _burn(farm, _liquidity);
132:     }
133:
```

Impact

The farm can use the "*burn*" function to send the reward tokens to the zero address (either accidentally or on purpose), effectively burning that number of tokens.

Recommendation

It is advisable to verify that the address is not the zero address. The functions assert and require can be used to check for conditions and throw an exception

if the condition is not met. The control can also be implemented with a simple check:

if (_to == address(0)) revert InvalidAddress();

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.3 Replay attack in permit() function if nonces counter overflows at "ZirconERC20.sol"

Description	LOW

It was identified that the "*permit*()" functionality can be vulnerable to replay attacks if the nonces counter overflows. A replay attack is a valid data transmission that is maliciously or fraudulently repeated or delayed. In the examined functionality an anti-replay mechanism that is based on incremental nonces is deployed. However, no protection against integer overflows is implemented. Sn integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits – either higher than the maximum or lower than the minimum representable value. An integer overflow can cause the value to wrap, which violates the program's assumption and may lead to unexpected behavior.

Currently, ZirconERC20 is using an outdated solidity version that does not implement *safemath* by default. Instead, the developers are required to use function calls such as ".*add*()", and ".*sub*()" in order to protect from overflow attacks.

```
File: /packages/zircon-core/contracts/ZirconERC20.sol
1: pragma solidity =0.5.16;
2: import './libraries/SafeMath.sol';
3: import '@uniswap/v2-core/contracts/interfaces/IUniswapV2ERC20.sol';
4:
5: contract ZirconERC20 is IUniswapV2ERC20 {
6:     using SafeMath for uint;
```

However, it was found that in "*permit*()" functionality, the nonces counter is incremented with a simple addition, which can be affected by an integer overflow.

```
File: /packages/zircon-core/contracts/ZirconERC20.sol
80:     function permit(address owner, address spender, uint value, uint
deadline, uint8 v, bytes32 r, bytes32 s) external {
81:         require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
82:         bytes32 digest = keccak256(
```

```
83:             abi.encodePacked(
84:                 '\x19\x01',
85:                 DOMAIN_SEPARATOR,
86:                 keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender,
value, nonces[owner]++, deadline))
87:             )
88:         );
89:         address recoveredAddress = ecrecover(digest, v, r, s);
90:         require(recoveredAddress != address(0)  recoveredAddress ==
owner, 'UniswapV2: INVALID_SIGNATURE');
91:         _approve(owner, spender, value);
92:     }
```

An adversary, who possess already a valid permit, will be able to use it when the counter overflows. Unfortunately, the counter can be incremented only if there are valid permits that do not fail at the digest validation and this requirement can only be fulfilled by the owner.

Impact

A malicious user could leverage this issue when the counter overflows, take valid signatures and content hashes from a previous permit, and create new valid permits for them without the owner's knowledge.

Since an adversary is not able to overflow the nonces counter on-demand, and the previous permits expire after a specific timestamp, the issue is marked as LOW.

Recommendation

It is advisable to use *safemath* to increment the nonces counter.

Furthermore, it is recommended to examine if it is possible to use a more updated solidity version.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.4 Multiple initializations are allowed in "ZirconPair.sol"

Description	LOW

It was identified that the ZirconPair is using an initialization function that can be called multiple times. In general, Solidity provides a constructor declaration inside the smart contract, and it invokes only once when the contract is deployed and is used to initialize the contract state, to avoid changes in the storage space of other contracts. In cases that the contract is using a proxy, then the initialize function is used to perform this operation.

In the examined case it was found that the functionality does not have any protection to limit multiple calls. However, the purpose of the initializer (as well as the constructor) is to be called as the first function before using the contract, and never be called back a second time.

The issue exists at:

```
File: /packages/zircon-core/contracts/ZirconPair.sol
77:     // called once by the factory at time of deployment
78:     function initialize(address _token0, address _token1, address
_energy) external {
79:         require(msg.sender == factory, 'ZirconPair: FORBIDDEN'); //
sufficient check
80:         token0 = _token0;
81:         token1 = _token1;
82:         energyRevenueAddress = _energy;
83:     }
```

Impact

The "initialize" function can be called multiple times, even during the normal operation of a published ZirconPair.

Since only the factory can call this function, the issue is marked as LOW.

Recommendation

It is advisable to use a pattern that blocks multiple initializations such as the following:

```
bool isInitialized = false;

function initialize(
    uint256 _param1,
    ...) public {
    require(!isInitialized, 'Contract is already initialized!');
    ...
}
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.5 Multiple initializations are allowed in "ZirconPoolToken.sol"

Description	LOW

It was identified that the ZirconPoolToken is using an initialization function that can be called multiple times. In general, Solidity provides a constructor declaration inside the smart contract, and it invokes only once when the contract is deployed and is used to initialize the contract state, to avoid changes in the storage space of other contracts. In cases that the contract is using a proxy, then the initialize function is used to perform this operation. In the examined case it was found that the functionality does not have any protection to limit multiple calls. However, the purpose of the initializer (as well as the constructor) is to be called as the first function before using the contract, and never be called back a second time.

The issue exists at:

```
File:  /packages/zircon-core/contracts/ZirconPoolToken.sol
32:     // called once by the factory at time of deployment
33:     function initialize(address _token0, address _pair, address
_pylon, bool _isAnchor) external {
34:         require(msg.sender == pylonFactory, 'ZPT: FORBIDDEN');
35:         // sufficient check
36:         token = _token0;
37:         pair = _pair;
38:         isAnchor = _isAnchor;
39:         pylon = _pylon;
40:     }
```

Impact

The "initialize" function can be called multiple times, even during the normal operation of a published ZirconPoolToken.

Since only the factory can call this function, the issue is marked as LOW.

Recommendation

It is advisable to use a pattern that blocks multiple initializations such as the following:

```
bool isInitialized = false;

function initialize(
    uint256 _param1,
    ...) public {
    require(!isInitialized, 'Contract is already initialized!');
    ...
}
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.6 Multiple initializations are allowed in "ZirconPylon.sol"

Description	LOW

It was identified that the ZirconPylon is using initialization functions that can be called multiple times. In general, Solidity provides a constructor declaration inside the smart contract, and it invokes only once when the contract is deployed and is used to initialize the contract state, to avoid changes in the storage space of other contracts. In cases that the contract is using a proxy, then the initialize function is used to perform this operation. In the examined case it was found that the functionality does not have any protection to limit multiple calls. However, the purpose of the initializer (as well as the constructor) is to be called as the first function before using the contract, and never be called back a second time.

The issue exists at:

```
File: /packages/zircon-core/contracts/ZirconPylon.sol
184:     // ***** INIT ******
185:     // @notice Called once by the factory at time of deployment
186:     // @_floatPoolTokenAddress -> Contains Address Of Float PT
187:     // @_anchorPoolTokenAddress -> Contains Address Of Anchor PT
188:     // @floatToken -> Float token
189:     // @anchorToken -> Anchor token
190:     function initialize(address _floatPoolTokenAddress, address
_anchorPoolTokenAddress, address _floatToken, address _anchorToken,
address _pairAddress, address _pairFactoryAddress, address _energy)
external nonReentrant {
191:         onlyFactory();
192:         floatPoolTokenAddress = _floatPoolTokenAddress;
193:         anchorPoolTokenAddress = _anchorPoolTokenAddress;
194:         pairAddress = _pairAddress;
195:         isFloatReserve0 = IZirconPair(_pairAddress).token0() ==
_floatToken;
196:         pylonToken = PylonToken(_floatToken, _anchorToken);
197:         pairFactoryAddress = _pairFactoryAddress;
198:         energyAddress = _energy;
199:     }
200:
201:     function initMigratedPylon(uint _gamma, uint _vab) external {
```

```
202:         onlyFactory(); // sufficient check
203:         gammaMulDecimals = _gamma;
204:         virtualAnchorBalance = _vab;
205:         _update();
206:         initialized = 1;
207:     }
208:
209:     // @notice On init pylon we have to handle two cases
210:     // The first case is when we initialize the pair through the
pylon
211:     // And the second one is when initialize the pylon with a pair
already existing
212:     function initPylon(address _to) external nonReentrant returns
(uint floatLiquidity, uint anchorLiquidity) {
213:         require(initialized == 0, "ZP: AI");
214:         require(!IZirconPylonFactory(factoryAddress).paused(), 'ZP:
P');
215:         uint balance0 =
IUniswapV2ERC20(pylonToken.float).balanceOf(address(this));
216:         uint balance1 =
IUniswapV2ERC20(pylonToken.anchor).balanceOf(address(this));
217:
218:         notZero(balance0);
219:         notZero(balance1);
220:
221:         // Let's get the balances so we can see what the user send
us
222:         // As we are initializing the reserves are going to be null
223:         // Let's see if the pair contains some reserves
224:         (uint112 _reservePair0, uint112 _reservePair1) =
getPairReservesNormalized();
225:         // If pair contains reserves we have to use the ratio of the
Pair so...
226:         virtualAnchorBalance = balance1;
227:
228:         if (_reservePair0 > 0  _reservePair1 > 0) {
229:             uint tpvAnchorPrime =
(virtualAnchorBalance.add(balance0.mul(_reservePair1)/_reservePair0));
230:
231:             if (virtualAnchorBalance  tpvAnchorPrime/2) {
232:                 gammaMulDecimals = 1e18 -
```

```
(virtualAnchorBalance.mul(1e18)/(tpvAnchorPrime));
233:             } else {
234:                 gammaMulDecimals =
tpvAnchorPrime.mul(1e18)/(virtualAnchorBalance.mul(4)); // Subflow
already checked by if statement
235:             }
236:             // console.log("ZP: GAMMA_MUL_DECIMALS",
gammaMulDecimals);
237:             // This is gamma formula when FTV = 50%
238:         } else {
239:             // When Pair is not initialized let's start gamma to 0.5
240:             gammaMulDecimals = 500000000000000000;
241:         }
242:
243:
244:         // Time to mint some tokens
245:         (anchorLiquidity) = _calculateSyncLiquidity(balance1, 0,
_reservePair1, anchorPoolTokenAddress, true);
246:         (floatLiquidity) = _calculateSyncLiquidity(balance0, 0,
_reservePair0, floatPoolTokenAddress, false);
247:
248:         lastPoolTokens = IZirconPair(pairAddress).totalSupply();
249:         lastK = _reservePair0.mul(_reservePair1);
250:
251:         _syncMinting();
252:
253:         IZirconPoolToken(anchorPoolTokenAddress).mint(_to,
anchorLiquidity);
254:         IZirconPoolToken(floatPoolTokenAddress).mint(_to,
floatLiquidity);
255:
256:         muMulDecimals = gammaMulDecimals; //Starts as gamma,
diversifies over time. Used to calculate fee distribution
257:         muBlockNumber = block.number; //block height of last mu
update
258:         muOldGamma = gammaMulDecimals; //gamma value at last mu
update
259:
260:
261:         //Here it updates the state and throws liquidity into the
pool if possible
```

```
262:
263:          _update();
264:          initialized = 1;
265:      }
266:
```

Impact

The "initialize" function can be called multiple times, even during the normal operation of a published ZirconPylon.

Since only the factory can call this function, the issue is marked as LOW.

Recommendation

It is advisable to use a pattern that blocks multiple initializations such as the following:

```
bool isInitialized = false;

function initialize(
    uint256 _param1,
    ...) public {
    require(!isInitialized, 'Contract is already initialized!');
    ...
}
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.7 Multiple initializations are allowed in "ZirconEnergy.sol"

Description	LOW

It was identified that the ZirconEnergy is using initialization functions that can be called multiple times. In general, Solidity provides a constructor declaration inside the smart contract, and it invokes only once when the contract is deployed and is used to initialize the contract state, to avoid changes in the storage space of other contracts. In cases that the contract is using a proxy, then the initialize function is used to perform this operation. In the examined case it was found that the functionality does not have any protection to limit multiple calls. However, the purpose of the initializer (as well as the constructor) is to be called as the first function before using the contract, and never be called back a second time.

The issue exists at:

```
File: /packages/zircon-core/contracts/energy/ZirconEnergy.sol
76:
77:   function initialize(address _pylon, address _pair, address _token0,
address _token1) external {
78:     require(msg.sender == energyFactory, 'Zircon: FORBIDDEN'); //
sufficient check
79:     bool isFloatToken0 = IZirconPair(_pair).token0() == _token0;
80:     (address tokenA, address tokenB) = isFloatToken0 ? (_token0,
_token1) : (_token1, _token0);
81:     pylon = Pylon(
82:       _pylon,
83:       _pair,
84:       tokenA,
85:       tokenB,
86:       1,
87:       1
88:     );
89:     // Approving pylon to use anchor tokens
90:     IUniswapV2ERC20(tokenB).approve(_pylon, 2^256 - 1);
91:     IUniswapV2ERC20(_pair).approve(_pylon, 2^256 - 1);
```

```
92:
93:    }
```

Impact

The "initialize" function can be called multiple times, even during the normal operation of a published ZirconEnergy.

Since only the energy factory can call this function, the issue is marked as LOW.

Recommendation

It is advisable to use a pattern that blocks multiple initializations such as the following:

```
bool isInitialized = false;

function initialize(
    uint256 _param1,
    ...) public {
    require(!isInitialized, 'Contract is already initialized!');
    ...
}
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.8 Multiple initializations are allowed in "ZirconEnergyRevenue.sol"

Description	LOW

It was identified that the ZirconEnergyRevenue is using initialization functions that can be called multiple times. In general, Solidity provides a constructor declaration inside the smart contract, and it invokes only once when the contract is deployed and is used to initialize the contract state, to avoid changes in the storage space of other contracts. In cases that the contract is using a proxy, then the initialize function is used to perform this operation. In the examined case it was found that the functionality does not have any protection to limit multiple calls. However, the purpose of the initializer (as well as the constructor) is to be called as the first function before using the contract, and never be called back a second time.

The issue exists at:

```
File: /packages/zircon-core/contracts/energy/ZirconEnergyRevenue.sol
50:     function initialize(address _pair, address _tokenA, address
_tokenB, address energy0, address energy1, address pylon0, address
pylon1) external {
51:         require(energyFactory == msg.sender, "ZER: Not properly
called");
52:         zircon = Zircon(
53:             _pair,
54:             _tokenA,
55:             _tokenB,
56:             energy0,
57:             energy1,
58:             pylon0,
59:             pylon1
60:         );
61:
62:     }
```

Impact

The "initialize" function can be called multiple times, even during the normal operation of a published ZirconEnergyRevenue.

Since only the energy factory can call this function, the issue is marked as LOW.

Recommendation

It is advisable to use a pattern that blocks multiple initializations such as the following:

```
bool isInitialized = false;

function initialize(
    uint256 _param1,
    ...) public {
    require(!isInitialized, 'Contract is already initialized!');
    ...
}
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.9 Unvalidated addresses in functions "setFeeToSetter", "setMigrator" and the constructor at "ZirconEnergyFactory.sol"

Description	LOW

The team identified that functions "*setFeeToSetter*", "*setMigrator*" and the constructor of ZirconEnergyFactory are not validating the provided address to not be the zero address.

The issue exists at:

```
File: /packages/zircon-core/contracts/energy/ZirconEnergyFactory.sol
36:     constructor(address _feeToSetter, address _migrator) public {
37:         fee = Fee({minPylonFee: 1, maxPylonFee: 50});
38:         insurancePerMille = 100;
39:         feePercentageRev = 20;
40:         feeToSetter = _feeToSetter;
41:         migrator = _migrator;
42:     }
43:
170:     function setFeeToSetter(address _feeToSetter) external
onlyFeeToSetter {
171:         feeToSetter = _feeToSetter;
172:     }
173:
174:     function setMigrator(address _migrator) external onlyMigrator {
175:         migrator = _migrator;
176:     }
```

Impact

The Migrator can use the "*setMigrator*" function to move the ownership to the zero address (either accidentally or on purpose set during the construction of the contract), effectively losing access at the contract. A similar issue can occur with *feeToSetter* role.

Recommendation

It is advisable to verify that the address is not the zero address.

The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

if (_token_address == address(0)) revert InvalidAddress();

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.10 Unvalidated addresses in functions "mint", "burn", "initialize", "changePylonAddress" and the constructor at "ZirconPoolToken.sol"

Description	LOW

The team identified that functions "*mint*", "*burn*", "*initialize*", "*changePylonAddress*" and the constructor of ZirconPoolToken are not validating the provided address to not be the zero address.

The issue exists at:

```
File: /packages/zircon-core/contracts/ZirconPoolToken.sol
14:     constructor(address _pylonFactory) public {
15:         pylonFactory = _pylonFactory;
16:         ...
17:     }
23:
24:     function mint(address account, uint256 amount) external
onlyPylon{
25:         _mint(account, amount);
26:     }
27:
28:     function burn(address account, uint256 amount) external
onlyPylon{
29:         _burn(account, amount);
30:     }
31:
32:     // called once by the factory at time of deployment
33:     function initialize(address _token0, address _pair, address
_pylon, bool _isAnchor) external {
34:         require(msg.sender == pylonFactory, 'ZPT: FORBIDDEN');
35:         // sufficient check
36:         token = _token0;
37:         pair = _pair;
...
39:         pylon = _pylon;
40:     }
41:
42:     function changePylonAddress(address _pylon) external {
...
```

```
44:          pylon = _pylon;
45:      }
```

Impact

The owner can use the "mint" and "burn" function to transfer a number of tokens to the zero address (either accidentally or on purpose set during the construction of the contract), or change the pylon address to the zero contract.

Recommendation

It is advisable to verify that the address is not the zero address.

The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

if (account == address(0)) revert InvalidAddress();

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.11 Unvalidated addresses in functions "initialize", "mint", "mintOneSide", "burnOneSide", "burn", "skim" and "changeEnergyRevAddress" at "ZirconPair.sol"

Description	LOW

The team identified that many functions of ZirconPair are not validating the provided address to not be the zero address. Transferring a number of tokens to the zero address during the "mint" operation, is equivalent to burning that number of tokens. The transaction should not revert (there's no contract that would throw an exception on the 0x0 address) and should not return any data (again, no contract that would return any data).

```
File: /packages/zircon-core/contracts/ZirconPair.sol
77:     function initialize(address _token0, address _token1, address
_energy) external {
78:         require(msg.sender == factory, 'ZirconPair: FORBIDDEN'); //
sufficient check
79:         token0 = _token0;
80:         token1 = _token1;
81:         energyRevenueAddress = _energy;
82:     }
125:
126:     // this low-level function should be called from a contract
which performs important safety checks
127:     function mint(address to) external lock returns (uint liquidity)
{
...
143:         _mint(to, liquidity);
...
148:     }
149:
150:     // this low-level function should be called from a contract
which performs important safety checks
151:     // TODO: will be better if we pass the output amount
152:     function mintOneSide(address to, bool isReserve0) external lock
returns (uint liquidity, uint amount0, uint amount1) {
...
184:         _mint(to, liquidity);
...
```

```
188:     }
189:
192:     // TODO: maybe allow burning both sides to one
193:     function burnOneSide(address to, bool isReserve0) external lock
returns (uint amount) {
1...
219:         if (isReserve0) {
220:             _safeTransfer(_token0, to, amount);
221:         }else{
222:             _safeTransfer(_token1, to, amount);
223:         }
...
230:     }
231:
232:     // this low-level function should be called from a contract
which performs important safety checks
233:     function burn(address to) external lock returns (uint amount0,
uint amount1) {
...
246:         _safeTransfer(_token0, to, amount0);
247:         _safeTransfer(_token1, to, amount1);
...
254:     }
...
291:     function skim(address to)  external lock {
292:         address _token0 = token0; // gas savings
293:         address _token1 = token1; // gas savings
294:         _safeTransfer(_token0, to,
IUniswapV2ERC20(_token0).balanceOf(address(this)).sub(reserve0));
295:         _safeTransfer(_token1, to,
IUniswapV2ERC20(_token1).balanceOf(address(this)).sub(reserve1));
296:     }
297:
302:
303:     function changeEnergyRevAddress(address _revAddress) external {
304:         require(msg.sender == factory, 'UniswapV2: NOT_ALLOWED');
305:         energyRevenueAddress = _revAddress;
306:     }
```

Impact

The users can use the "*burnOneSide*" and "*skim"* functionalities to transfer a number of tokens to the zero address (either accidentally or on purpose set during the construction of the contract), effectively burning that number of tokens.

Recommendation

It is advisable to verify that the address is not the zero address.

The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

if (account == address(0)) revert InvalidAddress();

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.12 Unvalidated addresses in functions "setMigrator" at "Migrator.sol"

Description	LOW

The team identified that the "setMigrator" function of Migrator is not validating the provided address to not be the zero address.

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/governance/Migrator.sol

```
38:     function setMigrator(address migrator_) public onlyOwner {
39:         IZirconPylonFactory(pylonFactory).setMigrator(migrator_);
40:         IZirconEnergyFactory(energyFactory).setMigrator(migrator_);
41:         IZirconFactory(pairFactory).setMigrator(migrator_);
42:         IZirconPTFactory(ptFactory).setMigrator(migrator_);
43:     }
44:
```

Impact

The owner can set the migrator as the zero address.

Recommendation

It is advisable to verify that the address is not the zero address.

The functions assert and require can be used to check for conditions and throw an exception if the condition is not met. The control can also be implemented with a simple check:

```
if (migrator_ == address(0)) revert InvalidAddress();
```

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:H/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.13 Integer overflow in "calculate()" at "ZirconEnergyRevenue.sol"

Description	LOW

The team identified that the "calculate" functionality of the "ZirconEnergyRevenue.sol" contract, is affected by integer overflow vulnerability. An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.

More precisely, it was identified that if "calculate" function is used multiple times, and the "getBalanceFromPair()" is not used to collect the fees, the feeValue1 and feeValue2 can wrap before the fees are collected. The issue exists because a simple addition is used to calculate the fees instead of a safemath operation (e.g. " feeValue0 += "). However, to exploit this issue a significant amount of fees is required, reducing the risk.

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/energy/ZirconEnergyRevenue.sol

```
073:     function calculate(uint percentage) external _onlyPair
nonReentrant _initialize {
074:         uint balance =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(address(this));
075:         require(balance > reserve, "ZER: Reverted");
076:
077:         //Percentage is feeValue/TPV, percentage of pool reserves
that are actually fee
078:         //It's reduced by mint fee already
079:         uint totalSupply =
IUniswapV2ERC20(zircon.pairAddress).totalSupply();
080:         //These are the PTBs, balance of pool tokens held by each
pylon vault
081:         uint pylonBalance0 =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(zircon.pylon0);
082:         uint pylonBalance1 =
IUniswapV2ERC20(zircon.pairAddress).balanceOf(zircon.pylon1);
083:         {
084:             (uint112 _reservePair0, uint112 _reservePair1,) =
IZirconPair(zircon.pairAddress).getReserves();
085:
```

```
086:             //Increments the contract variable that stores total
fees acquired by pair. Multiplies by each Pylon's share
087:
088:             feeValue0 +=
percentage.mul(_reservePair1).mul(2).mul(pylonBalance0)/totalSupply.mul(1
e18);
089:             feeValue1 +=
percentage.mul(_reservePair0).mul(2).mul(pylonBalance1)/totalSupply.mul(1
e18);
090:         }
091:
092:         {
093:             uint feePercentageForRev =
IZirconEnergyFactory(energyFactory).feePercentageRev();
094:             uint amount = balance.sub(reserve);
095:             uint pylon0Liq =
(amount.mul(pylonBalance0)/totalSupply).mul(100 -
feePercentageForRev)/(100);
096:             uint pylon1Liq =
(amount.mul(pylonBalance1)/totalSupply).mul(100 -
feePercentageForRev)/(100);
097:             _safeTransfer(zircon.pairAddress, zircon.energy0,
pylon0Liq);
098:             _safeTransfer(zircon.pairAddress, zircon.energy1,
pylon1Liq);
099:             reserve = balance.sub(pylon0Liq.add(pylon1Liq));
100:         }
101:     }
```

Impact

In case that multiple fees are calculated before these fees are collected, the local variable can wrap, losing all the revenue.

Recommendation

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.14 Integer overflow using "_liquidityfee" at "ZirconEnergyRevenue.sol" and "ZirconPylon.sol"

Description	LOW

The team identified that the "calculate" functionality of the "ZirconEnergyRevenue.sol" contract, is affected by integer overflow vulnerability. An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.

First the _onlyFeeToSetter has to provide a specially crafted fee (e.g. 10001):

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconFactory.sol
88:     function setLiquidityFee(uint _liquidityFee) external
_onlyFeeToSetter {
89:         liquidityFee = _liquidityFee;
90:     }
91:
```

Then, a function that uses the "liquidityFee" with unsafe math operations has to be used, such as the "burnOneSide" at "ZirconPair.sol"

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconPair.sol
213:     function burnOneSide(address to, bool isReserve0) external lock
returns (uint amount) {
214:         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); //
gas savings
215:         address _token0 = token0;                                //
gas savings
216:         address _token1 = token1;                                //
gas savings
217:         uint amount0;
218:         uint amount1;
219:         uint balance0 =
IUniswapV2ERC20(_token0).balanceOf(address(this));
220:         uint balance1 =
IUniswapV2ERC20(_token1).balanceOf(address(this));
```

```
221:         uint liquidity = balanceOf[address(this)];
222:         uint _liquidityFee = IZirconFactory(factory).liquidityFee();
223:
224:         _mintFee(_reserve0, _reserve1);
225:         uint _totalSupply = totalSupply; // gas savings, must be
defined here since totalSupply can update in _mintFee
226:         amount0 = liquidity.mul(balance0) / _totalSupply; // using
balances ensures pro-rata distribution
227:         amount1 = liquidity.mul(balance1) / _totalSupply; // using
balances ensures pro-rata distribution
228:         if (isReserve0) {
229:             amount0 += getAmountOut(amount1, _reserve1 - amount1,
_reserve0 - amount0, _liquidityFee);
230:             amount = amount0;
231:             require(amount < balance0, "UniswapV2:
EXTENSION_NOT_ENOUGH_LIQUIDITY");
232:         }else{
233:             amount1 += getAmountOut(amount0, _reserve0 - amount0,
_reserve1 - amount1, _liquidityFee);
234:             amount = amount1;
235:             require(amount < balance1, "UniswapV2:
EXTENSION_NOT_ENOUGH_LIQUIDITY");
236:         }
```

Finally, the overflow will occur at "getAmountOut" where the 10000-fee calculatation will take place

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/ZirconPair.sol

```
201:     function getAmountOut(uint amountIn, uint reserveIn, uint
reserveOut, uint fee) internal pure returns (uint amountOut) {
202:         require(amountIn > 0, 'UV2: IIA');
203:         require(reserveIn > 0 && reserveOut > 0, 'UV2: IL');
204:         uint amountInWithFee = amountIn.mul(10000-fee);
205:         uint numerator = amountInWithFee.mul(reserveOut);
206:         uint denominator =
reserveIn.mul(10000).add(amountInWithFee);
207:         amountOut = numerator / denominator;
208:     }
```

```
209:
```

A similar issue exists at:

```
File: .zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconPylon.sol
642:     function payFees(uint amountIn, uint feeBps, bool isAnchor)
private returns (uint amountOut){
643:         uint fee = amountIn * feeBps/10000;
...
651:             uint amountInWithFee = fee.mul(10000-
IZirconPylonFactory(factoryAddress).liquidityFee());
652:             uint amountSwapped = amountInWithFee.mul(_reservePair1)
/ _reservePair0.mul(10000).add(amountInWithFee);
653:
655:             IZirconPair(pairAddress).swap(isFloatReserve0 ? 0 :
amountSwapped, isFloatReserve0 ? amountSwapped : 0, energyAddress, "");
656:         }
657:         IZirconEnergy(energyAddress).registerFee();
658:         amountOut =  amountIn - fee;
659:     }
```

and

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconPylon.sol
728:     function calculateLiquidity(uint amountIn, bool isAnchor) view
private returns (uint amount, uint liquidity) {
....
736:         uint amountInWithFee = amountIn.mul(10000-
(IZirconPylonFactory(factoryAddress).liquidityFee()/2 + 1))/10000;
```

Impact

The local variable amountInWithFee can wrap, affecting the expected results.

Recommendation

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.

An alternative solution would be to validate that liquidityFee is always less than 10000.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:H/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.15 Event not emitted in "emergencyRewardWithdraw" and "stopReward()" functionalities at "PsionicFarmInitializable.sol"

Description	LOW

It was identified that the authorized command "*emergencyRewardWithdraw*" and "*stopReward*()" do not emit an event. A contract can emit events when it wants to notify external entities like users, chain explorers, or dApps about changes or conditions in the blockchain. When an event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain

The issue exists at:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
211:     /*
212:      * @notice Stop rewards
213:      * @dev Only callable by owner. Needs to be for emergency.
214:      */
215:     function emergencyRewardWithdraw(uint256 _amount) external
onlyOwner {
216:         _sendTokens(_amount, address(msg.sender));
217:     }
218:
```

and at:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
235:
236:     /*
237:      * @notice Stop rewards
238:      * @dev Only callable by owner
239:      */
240:     function stopReward() external  onlyOwner {
241:         bonusEndBlock = block.number;
242:     }
```

Impact

Events are necessary to notify the off-chain world of successful state transitions. Administration functionalities should emit the corresponding events throughout the system's life cycle to provide credibility and confidence in the system.

In the specific case, if the authorized command "*emergencyRewardWithdraw*" or "*stopReward*" is called, no event will be emitted.

Recommendation

It is recommended to emit an event related to this functionality.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.16 The Uniswap V2 skim functionality is exposed at "ZirconPair.sol"

Description	LOW

The team identified that the "skim" functionality is exposed at "ZirconPair.sol". In Uniswap V2 contracts, it is possible for the real balances to get out of sync with the reserves that the pair exchange thinks it has. While there is no way to withdraw tokens without the contract's consent, it is still possible by foreign wallets or contracts to make deposits that will affect the balance. For example, an account can transfer tokens to the exchange without calling either mint or swap. Uniswap V2 provides two solutions to resolve this issue:

- Either use "sync", and update the reserves to the current balances
- Or use "skim" and withdraw the extra amount. Note that any account is allowed to call skim because the contract doesn't know who deposited the tokens. This information is emitted in an event, but events are not accessible from the blockchain.

However, there are malicious actors searching for these cases using automated tools (e.g. https://github.com/nicholashc/uniswap-skim), and trying to withdraw the extra tokens by calling skim. Currently, there is an increasing number of bots searching for these opportunities.

The issue exists at the following location:

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/ZirconPair.sol

```
310:     // force balances to match reserves
311:     function skim(address to)  external lock {
312:         address _token0 = token0; // gas savings
313:         address _token1 = token1; // gas savings
314:         _safeTransfer(_token0, to,
IUniswapV2ERC20(_token0).balanceOf(address(this)).sub(reserve0));
315:         _safeTransfer(_token1, to,
IUniswapV2ERC20(_token1).balanceOf(address(this)).sub(reserve1));
316:     }
```

Impact

Bots may search and find a discrepancy in ZirconPair, and use "skim" to force a token withdrawal.

Recommendation

It is recommended to allow only the "sync()" functionality to balance the discrepancies.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.17 Gas limited function is used for token transfer in "claim" function at "ZirconDrop.sol"

Description	LOW

The team found that the "*claim*()" function at "ZirconDrop.sol", is using the "*transfer*()" function to send the requested amount of tokens to the user. When the "*transfer*()" function is used, the receiving smart contract should have a fallback function defined. However, the "*transfer*()" function is always forwarding a fixed amount of gas (2300) and may be incompatible with certain contracts.

```
File: /zircon-airdrop/contracts/ZirconDrop.sol
66:     function claim(uint256 index, uint256 amount, bytes32[] calldata
merkleProof) external nonReentrant {
67:         require(!if_claimed(index), "Already Claimed");
68:         require(block.timestamp > info.start_time, "Not Started");
69:         require(block.timestamp  info.end_time  (block.timestamp -
info.start_time) / 86400  5, "Expired");
70:         bytes32 leaf = keccak256(abi.encodePacked(index, msg.sender,
amount));
71:         require(MerkleProof.verify(merkleProof, merkleRoot, leaf),
'Not Verified');
72:         amount *= (10 ** 18);
// 18 decimals
73:         IERC20(info.token_address).transfer(msg.sender, amount);
74:         set_claimed(index);
75:         emit Claimed(amount, block.timestamp);
76:     }
```

Impact

Functions such as *send*() or *transfer*() can prevent reentrancy in very specific cases but will be incompatible with any contract whose fallback function requires more than 2,300 gas.

As a result, it is possible that the users may not be able to claim the available tokens due to gas exhaustion.

Recommendation

It is advisable to use the "*call*()" function, while at the same take check the return value and evaluate that all the necessary actions have been implemented to prevent *reentracy* attacks (e.g. *nonReentrant* guard or check-effects pattern)

CVSS Score

AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.18 Excessive loop iterations allowed in "burn", "remove" and constructor functions at "PsionicFarmVault.sol"

Description	LOW

It was identified that the function "burn", contains a potentially costly loop. Computational power on blockchain environments is paid, thus reducing the computational steps required to complete an operation is not only a matter of optimization but also cost efficiency. Loops are a great example of costly operations: as many elements an array has, more iterations will be required to complete the loop.

Excessive loop iterations may exhaust all available gas. For example, if an attacker can influence the element array's length, then they will be able to cause a denial of service, preventing the execution to jump out of the loop.

In the specific case, the function "burn" iterates over the "*tokensToWithdraw*" array and is of unspecified length:

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
115:     /*
116:     *   @notice Burn Function to withdraw tokens
117:     *   @param _to: the address to send the rewards token
118:     */
119:     function burn(address _to, uint _liquidity) external onlyFarm
nonReentrant {
120:         require(_liquidity > 0, "PFV: Not enough");
121:         address[] memory tokensToWithdraw = rewardTokens;
122:         uint tokenLength = tokensToWithdraw.length;
123:         uint ts = totalSupply();
124:
125:         for (uint256 i = 0; i  tokenLength; i++) {
126:             IERC20 token = IERC20(tokensToWithdraw[i]);
127:             uint balance = token.balanceOf(address(this));
128:             uint amount = (balance * (_liquidity))/ts;
129:             token.safeTransfer(_to, amount);
130:         }
131:         _burn(farm, _liquidity);
132:     }
```

A similar issue exists in "remove" and constructor functions, although these can only be used by the Owner.

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
085:
086:     // @notice Admin method to remove reward token
087:     function remove(address token) external onlyOwner {
088:         require(tokens[token], "Vault: Token not added");
089:         require(rewardTokens.length > 1, "At least 1 token is
required");
090:         address[] memory _tokens = rewardTokens;
091:         uint tokenLength = _tokens.length;
092:         for (uint256 i = 0; i  tokenLength; i++) {
093:             address rewardToken = _tokens[i];
094:             if (token == rewardToken) {
095:                 tokens[token] = false;
096:                 rewardTokens[i] = rewardTokens[rewardTokens.length-
1];
097:                 rewardTokens.pop();
098:                 break;
099:             }
100:         }
101:     }
```

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
44:     function initialize(
45:         address[] memory _rewardTokens,
46:         uint _initialSupply,
47:         address _farm,
48:         address _admin
49:     )  external  {
50:         require(!isInitialized, "Already initialized");
51:         require(msg.sender == PSIONIC_FACTORY, "Not factory");
52:         require(_rewardTokens.length > 0, "Empty reward tokens");
53:
54:         // Make this contract initialized
55:         isInitialized = true;
56:
57:         // Saving tokens
```

```
58:         rewardTokens = _rewardTokens;
59:
60:         uint tokenLength = _rewardTokens.length;
61:
62:         for (uint256 i = 0; i  tokenLength; i++) {
63:             address token = _rewardTokens[i];
64:             bool isToken = tokens[token];
65:             if (isToken) {
66:                 revert("Tokens must be unique");
67:             }else{
68:                 tokens[token] = true;
69:             }
70:         }
71:         // Minting initial Supply to Psionic Farm
72:         _mint(_farm, _initialSupply);
73:
74:         // Transfer ownership to the admin address who becomes owner
of the contract
75:         transferOwnership(_admin);
76:         farm = _farm;
77:     }
```

Impact

The legitimate owners may provide a significantly large "tokensToWithdraw" array that will cause the loop iterations to not be fully executed neither in the current nor in the following blocks, although all the provided gas will still be consumed. As a result, this may lead to a gas limit DoS scenario.

The issue is marked as LOW since only the owner can provide new tokens:

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
79:     // @notice Admin method to add reward token
80:     function add(address token) external onlyOwner {
81:         require(!tokens[token], "Vault: Token already added");
82:         tokens[token] = true;
83:         rewardTokens.push(token);
84:     }
```

Recommendation

It is advisable to refactor the logic to perform the operation in multiple batch transactions (e.g., an index on the *rewardTokens*), or to insert an upper limit that will allow the operation to be performed without failing due to insufficient gas.

CVSS Score

AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.19 Floating pragma in multiple contracts

Description	LOW

It was found that many smart contracts are using a floating pragma. In Solidity programming, multiple APIs only be supported in some specific versions. In each contract, the pragma keyword is used to enable certain compiler features or checks. If a contract does not specify a compiler version, developers might encounter compile errors in the future code reuse because of the version gap.

The issue exists at:

- zircon-airdrop/contracts/test_tokenA.sol:2:pragma solidity >= 0.8.0;
- *zircon-airdrop/contracts/ZirconDrop.sol:4:pragma solidity >= 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:5:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:167:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:194:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:274:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:582:pragma solidity >=0.6.2 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:750:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:827:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:864:pragma solidity >= 0.6.0 = 0.8.0;*
- *zircon-core/contracts/energy/interfaces/IZirconEnergyRevenue.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/energy/interfaces/IZirconEnergy.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/ZirconPoolToken.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/libraries/ZirconLibrary.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/libraries/console.sol:2:pragma solidity >= 0.4.22 0.9.0;*

- *zircon-core/contracts/libraries/TransferHelper.sol:3:pragma solidity >=0.6.0;*
- *zircon-core/contracts/ZirconPylon.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/governance/Migrator.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/governance/FeeToSetter.sol:1:pragma solidity ^0.5.16;*

For example, in case of "*FeeToSetter.so*" where the following directive exists "*pragma solidity ^0.5.16;",* the source file with the line above does not compile with a compiler earlier than version *0.5.16*, and it also does not work on a compiler starting from version *0.6.0* (this second condition is added by using ^). The exact version of the compiler is not fixed, so that bugfix releases are still possible.

In other cases, such as "*zircon-core/contracts/libraries/TransferHelper.sol*", where the "p*ragma solidity >=0.6.0*" is defined, any future version will be accepted.

Impact

Since different versions of Solidity may contain different APIs, developers might encounter compile errors in the future code reuse.

Recommendation

Source files should be annotated with a version pragma to reject compilation with previous or future compiler versions that might introduce incompatible changes.

It is advisable to remove the ">=" directives for the compiler version. Furthermore, it is recommended to avoid using the "^" directive to avoid using nightly builds.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.20 Outdated compiler version specified at multiple contracts

Description	LOW

The team identified that multiple contracts specify a less secure and outdated Solidity compiler version. In Solidity programming, multiple APIs only be supported in some specific versions. In each contract, the pragma keyword is used to enable certain compiler features or checks.

The issue exists at:

- *zircon-airdrop/contracts/airdrop_flattened:5:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:167:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:194:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:274:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:582:pragma solidity >=0.6.2 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:750:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:827:pragma solidity >=0.6.0 0.8.0;*
- *zircon-airdrop/contracts/airdrop_flattened:864:pragma solidity >= 0.6.0 = 0.8.0;*
- *zircon-core/contracts/ZirconFactory.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/energy/ZirconEnergyFactory.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/energy/libraries/SafeMath.sol:3:pragma solidity =0.5.16;*
- *zircon-core/contracts/energy/ZirconEnergyRevenue.sol:1:pragma solidity =0.5.16;*
- *zircon-core/contracts/energy/ZirconEnergy.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/energy/interfaces/IZirconEnergyRevenue.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/energy/interfaces/IZirconEnergy.sol:1:pragma solidity ^0.5.16;*

- *zircon-core/contracts/energy/interfaces/IZirconEnergyFactory.sol:1:pragma solidity =0.5.16;*
- *zircon-core/contracts/ZirconPoolToken.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/libraries/Token.sol:1:pragma solidity =0.5.16;*
- *zircon-core/contracts/libraries/ZirconLibrary.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/libraries/SafeMath.sol:3:pragma solidity =0.5.16;*
- *zircon-core/contracts/libraries/UQ112x112.sol:3:pragma solidity =0.5.16;*
- *zircon-core/contracts/libraries/Math.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/libraries/console.sol:2:pragma solidity >= 0.4.22 0.9.0;*
- *zircon-core/contracts/libraries/TransferHelper.sol:3:pragma solidity >=0.6.0;*
- *zircon-core/contracts/ZirconERC20.sol:1:pragma solidity =0.5.16;*
- *zircon-core/contracts/ZirconPylon.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/ZirconPTFactory.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/governance/Migrator.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/governance/FeeToSetter.sol:1:pragma solidity ^0.5.16;*
- *zircon-core/contracts/ZirconPylonFactory.sol:2:pragma solidity =0.5.16;*
- *zircon-core/contracts/ZirconPair.sol:2:pragma solidity =0.5.16;*

Impact

Developers are allowed to use outdated and less secure compilers to compile the specific file. For example, if the compiler *0.4.22* is selected, *safemath* will not be enabled by default.

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. For example:

https://docs.soliditylang.org/en/v0.8.15/bugs.html

Recommendation

It is recommended to specify s specific, more updated compiler version.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.21 No multisig protection in "onlyOwner" modifier at "ZirconDrop.sol"

Description	LOW

The team identified that the admin role (owner) of the "ZirconDrop.sol" contract is not protected with *multisig*. Smart contracts have privileged roles that are responsible to perform operations such as withdrawing, pausing, and upgrading, which are necessary in the lifecycle of a project. The best practice for securing admin accounts is to use a *multisig*. A *multisig* is a contract that can execute actions, as long as a predefined number of trusted members agree upon it. A *multisig* has a number of owners (N) and requires some of them (M) to approve a transaction. This configuration is referred to as M of N.

In the specific case, the admin role (owner) of the contract is responsible to set the merkle root, to recharge funds and to withdraw funds.

The issue exists at:

```
File /zircon-protocol-2-develop/packages/zircon-
airdrop/contracts/ZirconDrop.sol
45:     function recharge (uint256 _total) public onlyOwner {
46:         Info memory _info = info;
47:         require(IERC20(_info.token_address).allowance(msg.sender,
address(this)) >= _total, "WE NEED MORE!!!!!");
48:         IERC20(_info.token_address).safeTransferFrom(msg.sender,
address(this), _total);
49:         emit Recharged(_total, block.timestamp);
50:     }
86:     function withdraw() public onlyOwner {
87:         Info memory _info = info;
88:         require(block.timestamp > _info.end_time, "Not Expired");
89:         uint256 left =
IERC20(_info.token_address).balanceOf(address(this));
90:         require(left > 0, "What?");
91:         IERC20(_info.token_address).transfer(msg.sender, left);
92:         emit Withdrawed(left, block.timestamp);
93:     }
94:
95:     function set_root(bytes32 root) external onlyOwner {
96:         emit RootChanged(merkleRoot, root);
```

```
97:          merkleRoot = root;
98:      }
```

As a result, if the private key of this owner is compromised, an adversary would be able to withdraw all funds.

Impact

In case that the owner's private key is compromised, an adversary would be able to withdraw all the funds.

Recommendation

It is advisable to protect the owner functions with multisig.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.22 Lack of circuit breaker for emergency stop at "ZirconDrop.sol"

Description	LOW

It was identified that the "ZirconDrop.sol" does not support a circuit breaker control. A circuit breaker, also referred to as an emergency stop, can stop the execution of functions inside the smart contract. A circuit breaker can be triggered manually by trusted parties included in the contract like the contract admin or by using programmatic rules that automatically trigger the circuit breaker when the defined conditions are met. Applying the Emergency Stop pattern to a contract adds a fast and reliable method to halt any sensitive contract functionality as soon as a bug or another security issue is discovered. This leaves enough time to weigh all options and possibly upgrade the contract to fix the security breach.

However, it should be noted that the negative consequence of having an emergency stop mechanism from a user's point of view is, that it adds unpredictable contract behavior. There is always the possibility that the stop is abused in a malicious way by the authorized entity.

Impact

There is no way to halt the "*Claim*()" function, such as in case of a security breach.

The only method that can be used by the admin is to change the Merkle root or to withdraw the funds.

Recommendation

It is advisable to add a circuitBreaker. For example, the following code can be used to set a modifer:

```
bool public contractPaused = false;

function circuitBreaker() public onlyOwner { // onlyOwner can call
    if (contractPaused == false) {
        contractPaused = true;
    }
    else { contractPaused = false; }
```

```
}
// If the contract is paused, stop the modified function
// Attach this modifier to all public functions
modifier checkIfPaused() {
    require(contractPaused == false);
    _;
}
```

And then:

```
function claim(uint256 index, uint256 amount, bytes32[] calldata
merkleProof) external nonReentrant checkIfPaused
```

This approach is similar to openzeppelin pausable contract which can be found in the following URL:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/Pausable.sol

In both cases, a *multisig* Owner address must be used to ensure a decentralization strategy.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.23 Lack of circuit breaker for emergency stop at "PsionicFarmInitializable.sol" and "PsionicFarmVault.sol"

Description	LOW

It was identified that the "*PsionicFarmInitializable.sol*" and "*PsionicFarmVault.sol*" does not support a circuit breaker control. A circuit breaker, also referred to as an emergency stop, can stop the execution of functions inside the smart contract. A circuit breaker can be triggered manually by trusted parties included in the contract like the contract admin or by using programmatic rules that automatically trigger the circuit breaker when the defined conditions are met. Applying the Emergency Stop pattern to a contract adds a fast and reliable method to halt any sensitive contract functionality as soon as a bug or another security issue is discovered. This leaves enough time to weigh all options and possibly upgrade the contract in order to fix the security breach.

However, it should be noted that the negative consequence of having an emergency stop mechanism from a user's point of view is, that it adds unpredictable contract behavior. There is always the possibility that the stop is abused in a malicious way by the authorized entity.

Impact

There is no way to halt the "*withdraw*()" and "*deposit*()" functions, such as in case of a security breach.

Recommendation

It is advisable to add a circuitBreaker. For example, the following code can be used to set a modifer:

```
bool public contractPaused = false;

function circuitBreaker() public onlyOwner { // onlyOwner can call
    if (contractPaused == false) {
        contractPaused = true;
    }
    else { contractPaused = false; }
```

```
}
// If the contract is paused, stop the modified function
// Attach this modifier to all public functions
modifier checkIfPaused() {
    require(contractPaused == false);
    _;
}
```

And then:

```
function withdraw (uint256 _amount) external nonReentrant checkIfPaused
```

This approach is similar to openzeppelin pausable contract which can be found in the following URL:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/Pausable.sol

In both cases, a *multisig* Owner address must be used to ensure a decentralization strategy.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.24 Unchecked create2 call return value in "deployPool()" at "PsionicFarmFactory.sol"

Description	LOW

It was found that the "*deployPool*()" functionality does not check the return value of *create2.* Execution will resume even if the creation fails. Smart contracts can be created both by other contracts (using Solidity's new keyword) and by regular accounts (e.g., deploy) and in both cases the address for the new contract is computed as a function of the sender's own address and a nonce. With *create2* opcode it is possible to create a contract and make the resulting address independent of future events. Regardless of what may happen on the blockchain, it will always be possible to deploy the contract at the precomputed address. However, in case that the contract fails to deploy, the return address will be zero.

In the examined case, it was found that the contract does not validate the generated address.

The issue exists in the following location:

```
File: /packages/zircon-farming/contracts/PsionicFarmFactory.sol
26:     function deployPool(
27:         IERC20Metadata _stakedToken,
28:         address[] memory rewardTokens,
29:         uint256 _startBlock,
30:         uint256 _bonusEndBlock,
31:         uint256 _poolLimitPerUser,
32:         uint256 _numberBlocksForUserLimit,
33:         address _admin)
34:     external onlyOwner returns (address psionicFarmAddress, address
psionicVault) {
35:         ...
41:         assembly {
42:             psionicVault := create2(0, add(bytecodeVault, 32),
mload(bytecodeVault), saltVault)
43:         }
44:
```

```
...
47:
48:         assembly {
49:             psionicFarmAddress := create2(0, add(bytecode, 32),
mload(bytecode), salt)
50:         }
```

Impact

In case that the "create2" fails to create and deploy the requested new contract, the examined contract will resume execution and this can lead to unexpected results.

For example, in "*deployPool*()" function, even if the "*psionicVault*()" contract fails to be deployed, the "*psionicFarmAddress*()" may be deployed normally.

Recommendation

It is recommended to require that the return value of *create2* is not 0, otherwise revert. For example:

```
ret := create2(...);
require(ret != address(0), "...");
```

Alternatively, you may use a library that implements the check by itself such as the Create2 contract by openzeppelin.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.25 Unchecked create2 call return value in "createPair()" at "ZirconFactory.sol"

Description	LOW

It was found that the "*createPair*()" functionality does not check the return value of *create2.* Execution will resume even if the creation fails. Smart contracts can be created both by other contracts (using Solidity's new keyword) and by regular accounts (e.g., deploy) and in both cases the address for the new contract is computed as a function of the sender's own address and a nonce. With *create2* opcode it is possible to create a contract and make the resulting address independent of future events. Regardless of what may happen on the blockchain, it will always be possible to deploy the contract at the precomputed address. However, in case that the contract fails to deploy, the return address will be zero.

In the examined case, it was found that the contract does not validate the generated address.

```
File: /packages/zircon-core/contracts/ZirconFactory.sol
47:     function createPair(address tokenA, address tokenB, address
_pylonFactory) external returns (address pair) {
48:         ...
54:         assembly {
55:             pair := create2(0, add(bytecode, 32), mload(bytecode),
salt)
56:         }
57:         address energyRev = createEnergy(pair, token0, token1,
_pylonFactory);
58:         ...
59:         getPair[token0][token1] = pair;
60:         getPair[token1][token0] = pair; // populate mapping in the
reverse direction
61:         allPairs.push(pair);
62:         emit PairCreated(token0, token1, pair, allPairs.length);
63:     }
```

Impact

In case that the "*create2*" fails to create and deploy the requested new contract, the examined contract will resume execution and this can lead to unexpected results.

For example, in "*createPair*()" function, even if the "pair" contract fails to be deployed, the energy function may be executed normally.

Recommendation

It is recommended to require that the return value of *create2* is not 0, otherwise revert. For example:

```
ret := create2(...);
require(ret != address(0), "...");
```

Alternatively, you may use a library that implements the check by itself such as the Create2 contract by openzeppelin.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.26 Unchecked create2 call return value in "createPTAddress()" at "ZirconPTFactory.sol"

Description	LOW

It was found that the "*createPTAddress*()" functionality does not check the return value of *create2.* Execution will resume even if the creation fails. Smart contracts can be created both by other contracts (using Solidity's new keyword) and by regular accounts (e.g., deploy) and in both cases the address for the new contract is computed as a function of the sender's own address and a nonce. With *create2* opcode it is possible to create a contract and make the resulting address independent of future events. Regardless of what may happen on the blockchain, it will always be possible to deploy the contract at the precomputed address. However, in case that the contract fails to deploy, the return address will be zero.

In the examined case, it was found that the contract does not validate the generated address.

The issue exists in the following location:

```
File: /packages/zircon-core/contracts/ZirconPTFactory.sol
36:     function createPTAddress(address _token, address pylonAddress)
external returns (address poolToken) {
37:         // Creating Token
...
40:         assembly {
41:             poolToken := create2(0, add(bytecode, 32),
mload(bytecode), salt)
42:         }
43:     }
44:
```

Impact

In case that the "*create2*" fails to create and deploy the requested new contract, the examined contract will resume execution and this can lead to unexpected results.

For example, in "*createPTAddress*()" function, even if the "*poolToken*" contract fails to be deployed, the "*addPylon*()" may be executed normally.

Recommendation

It is recommended to require that the return value of *create2* is not 0, otherwise revert. For example:

```
ret := create2(...);
require(ret != address(0), "...");
```

Alternatively, you may use a library that implements the check by itself such as the Create2 contract by openzeppelin.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.27 Unchecked create2 call return value in "createPylon()" at "ZirconPylonFactory.sol"

Description	LOW

It was found that the "*createPylon*()" functionality does not check the return value of *create2.* Execution will resume even if the creation fails. Smart contracts can be created both by other contracts (using Solidity's new keyword) and by regular accounts (e.g., deploy) and in both cases the address for the new contract is computed as a function of the sender's own address and a nonce. With *create2* opcode it is possible to create a contract and make the resulting address independent of future events. Regardless of what may happen on the blockchain, it will always be possible to deploy the contract at the precomputed address. However, in case that the contract fails to deploy, the return address will be zero.

In the examined case, it was found that the contract does not validate the generated address.

The issue exists in the following location:

```
File: /packages/zircon-core/contracts/ZirconPylonFactory.sol
65:     function createPylon( address _tokenA, address _tokenB, address
_pair) private returns (address pylon) {
66:         // Creating Token
67:         bytes memory bytecode = type(ZirconPylon).creationCode;
68:         bytes32 salt = keccak256(abi.encodePacked(_tokenA, _tokenB,
_pair));
69:         assembly {
70:             pylon := create2(0, add(bytecode, 32), mload(bytecode),
salt)
71:         }
72:     }
```

Impact

In case that the "create2" fails to create and deploy the requested new contract, the examined contract will resume execution and this can lead to unexpected results.

For example, in "*createPair*()" function, even if the "pylon" fails to be deployed, the "*addPylonCustomPT*()" function may be executed normally.

Recommendation

It is recommended to require that the return value of *create2* is not 0, otherwise revert. For example:

```
ret := create2(...);
require(ret != address(0), "...");
```

Alternatively, you may use a library that implements the check by itself such as the Create2 contract by openzeppelin.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.28 No multisig protection in "onlyOwner" modifier at "PsionicFarmInitializable.sol"

Description	LOW

The team identified that the admin role (owner) of the "*PsionicFarmInitializable*.sol" contract is not protected with *multisig*. Smart contracts have privileged roles that are responsible to perform operations such as withdrawing, pausing, and upgrading, which are necessary in the lifecycle of a project. The best practice for securing admin accounts is to use a *multisig*. A *multisig* is a contract that can execute actions, as long as a predefined number of trusted members agree upon it. A *multisig* has a number of owners (N) and requires some of them (M) to approve a transaction. This configuration is referred to as M of N.

In the specific case, the admin role (owner) of the contract is able to withdraw reward tokens in emergency situation, to recover tokens sent to the contract by mistake, to update the per-user limits and the start and end blocks.

The issue exists at:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
211:     /*
212:      * @notice Stop rewards
213:      * @dev Only callable by owner. Needs to be for emergency.
214:      */
215:     function emergencyRewardWithdraw(uint256 _amount) external
onlyOwner {
216:         ...
217:     }
218:
219:     /**
220:      * @notice Allows the owner to recover tokens sent to the
contract by mistake
221:      * @param _token: token address
222:      * @dev Callable by owner
223:      */
224:     function recoverToken(address _token) external onlyOwner {
225:        ...
234:     }
```

```
235:
236:     /*
237:      * @notice Stop rewards
238:      * @dev Only callable by owner
239:      */
240:     function stopReward() external  onlyOwner {
241:       ...
242:     }
243:
244:     /*
245:      * @notice Update pool limit per user
246:      * @dev Only callable by owner.
247:      * @param _userLimit: whether the limit remains forced
248:      * @param _poolLimitPerUser: new pool limit per user
249:      */
250:     function updatePoolLimitPerUser(bool _userLimit, uint256
_poolLimitPerUser)  external onlyOwner {
..
260:     }
261:
262:
263:     /**
264:      * @notice It allows the admin to update start and end blocks
265:      * @dev This function is only callable by owner.
266:      * @param _startBlock: the new start block
267:      * @param _bonusEndBlock: the new end block
268:      */
269:     function updateStartAndEndBlocks(uint256 _startBlock, uint256
_bonusEndBlock) external  onlyOwner {
270:        ...
294:     }
```

As a result, if the private key of this owner is compromised, an adversary would be able to withdraw all funds.

Impact

In case that the owner's private key is compromised, an adversary would be able to withdraw all reward tokens.

Recommendation

It is advisable to protect the owner functions with *multisig*.

CVSS Score

AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3 Informational Findings

5.3.1 Unchecked transfer call return value in "claim" functionality at "ZirconDrop.sol"

Description	INFO

It was found that the "*Claim*()" function at the "*ZirconDrop*.sol" contract is using the "*transfer*()" function is used and the result is not checked. According to the specification (*https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md, https://eips.ethereum.org/EIPS/eip-20*), the *transfer*() function in ERC20 compatible tokens, moves an amount of tokens from the caller's account to the recipient address, returns a Boolean value indicating whether the operation succeeded and finally emits a Transfer event. Furthermore, the function SHOULD throw if the balance of the user does not have enough tokens.

As a result, it is expected that transfer will revert if the external transaction reverts. However, it is still recommended to use the *safeTransfer* if non-compliant ERC20 tokens from other vendors are used.

The issue exists at:

```
File: /zircon-protocol-2-develop/packages/zircon-
airdrop/contracts/ZirconDrop.sol
14:     using SafeERC20 for IERC20;
15:
...
66:     function claim(uint256 index, uint256 amount, bytes32[] calldata
merkleProof) external nonReentrant {
67:         require(!if_claimed(index), "Already Claimed");
68:         require(block.timestamp > info.start_time, "Not Started");
69:         require(block.timestamp  info.end_time  (block.timestamp -
info.start_time) / 86400  5, "Expired");
70:         bytes32 leaf = keccak256(abi.encodePacked(index, msg.sender,
amount));
71:         require(MerkleProof.verify(merkleProof, merkleRoot, leaf),
'Not Verified');
72:         amount *= (10 ** 18);
// 18 decimals
73:         IERC20(info.token_address).transfer(msg.sender, amount);
```

```
74:          set_claimed(index);
75:          emit Claimed(amount, block.timestamp);
76:      }
```

Impact

Since the token is currently deployed by the team, and it is expected to be a standard ERC20 token, the issue is marked as INFORMATIONAL.

Recommendation

It is advisable to use SafeERC20 and *safeTransfer*. SafeERC20 is a wrapper around the interface that checks the Boolean return values of ERC20 operations and revert the transaction if they fail and allows the of support some non-standard ERC20 tokens that don't have Boolean return values. To use this library, the contract has to add a "using SafeERC20 for ERC20;" statement, which will then allow the devs to call the safe operations as *token.safeTransfer(...),* etc. In the examined case, it was found that the specific statement is indeed added in the contract code. However, the safe operations are not used.

An alternative recommendation is to adopt a withdrawal pattern, in which, each user is burdened with calling an isolated function (withdraw) which handles the sending of ether out of the contract and therefore independently deals with the consequences of failed send transactions, while the tokens have been marked as claimed in "*claim*()" function.

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.2 Raw data left in published repository at "ZirconDrop"

Description	INFO

It was identified that all the list of the allowed addresses and the calculated proofs is included in the repository.

The application uses the Airdrop functionality for a number of initial users, in order to increase the number of token holders as well as to promote the upcoming token more widely. To reduce the storage and transaction fees of deploying the whole list of allowed accounts, the Merkle tree is used. The Merkle Tree is simply a binary tree data structure where the values of the nodes and leaves are the hash of the data. However, only the Merkle proof is required to verify that the addresses belong to the allowed list.

The issue exists at:

- */packages/zircon-airdrop/data/**
- */packages/zircon-airdrop/src/rawData.js*

Impact

All the allowed addresses will be published, even if the corresponding tokens are not claimed.

Since this repository is still under development the issue is marked as INFORMATIONAL.

Recommendation

It is recommended to remove the files from the repository before being published.

More details on how to remove sensitive information from a GitHub repository can be found in the following URL:

https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository

CVSS Score

AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.3 Debugging utility in "PsionicFarmVault.sol"

Description	INFO

It was identified that the *PsionicFarmVault* contract contains a debugging utility. Hardhat comes built-in with Hardhat Network, a local Ethereum network designed for development. When running contracts and tests on Hardhat Network it is possible to print logging messages and contract variables calling *console.log()* from the Solidity code. To perform this operation, it is required to *import hardhat/console.sol* in the contract code.

```
File: /packages/zircon-farming/contracts/PsionicFarmVault.sol
4: import "hardhat/console.sol";
```

Impact

Since no logging functionality is used, the issue is marked as INFORMATIONAL

Recommendation

It is recommended to remove hardhat console from production code.

CVSS Score

AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.4 It is impossible to use "withdraw()" or "deposit()" if user balance is below a certain level at "PsionicFarmInitializable.sol"

Description	INFO

It was identified that the "*deposit*()" and "*withdraw*()" functions might not be accessible if for any reason, the *user.rewardDebt* balance is incorrect.

The *user.amount* needs to be at least:

```
require(user.amount > (user.rewardDebt * PRECISION_FACTOR) /
accTokenPerShare,"Not enough user balance")
```

instead of just greater than zero:

```
134:         if (user.amount > 0) {
```

The issue exists at:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
127:     function deposit(uint256 _amount, address sender) internal {
128:         UserInfo storage user = userInfo[sender];
129:         userLimit = hasUserLimit();
130:         require(!userLimit || ((_amount + user.amount) =
poolLimitPerUser), "Deposit: Amount above limit");
131:
132:         _updatePool();
133:
134:         if (user.amount > 0) {
135:             uint256 pending = (user.amount * accTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
136:             if (pending > 0) {
137:                 _sendTokens(pending, address(sender));
138:             }
139:         }
140:
141:         if (_amount > 0) {
142:             user.amount = user.amount + _amount;
143:             stakedToken.safeTransferFrom(address(sender),
address(this), _amount);
```

```
144:         }
145:
146:         user.rewardDebt = (user.amount * accTokenPerShare) /
PRECISION_FACTOR;
147:
148:         emit Deposit(msg.sender, _amount);
149:     }
```

and in:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
171:     function withdraw(uint256 _amount) external  nonReentrant {
172:         UserInfo storage user = userInfo[msg.sender];
173:
174:         require(user.amount >= _amount, "Amount to withdraw too
high");
175:
176:         _updatePool();
177:
178:         uint256 pending = (user.amount * accTokenPerShare) /
PRECISION_FACTOR - user.rewardDebt;
179:
180:         if (_amount > 0) {
181:             user.amount = user.amount - _amount;
182:             stakedToken.safeTransfer(address(msg.sender), _amount);
183:         }
184:
185:         if (pending > 0) {
186:             _sendTokens(pending, address(msg.sender));
187:         }
188:
189:         user.rewardDebt = (user.amount * accTokenPerShare) /
PRECISION_FACTOR;
190:
191:         emit Withdraw(msg.sender, _amount);
192:     }
```

Impact

Since each time that a deposit takes place the *rewardDept* is updated, this is not expected to happen:

```
File: /packages/zircon-farming/contracts/PsionicFarmInitializable.sol
127:     function deposit(uint256 _amount, address sender) internal {
..
146:         user.rewardDebt = (user.amount * accTokenPerShare) /
PRECISION_FACTOR;
..
149:     }
```

As a result, the issue is marked as INFORMATIONAL

Recommendation

It is recommended to update the function to require the correct *user.amount*

```
require(user.amount > (user.rewardDebt * PRECISION_FACTOR) /
accTokenPerShare,"Not enough user balance")
```

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

6 Retest Results

6.1 Retest of Medium Severity Findings

The following MEDIUM-risk vulnerabilities were found to be successfully addressed:

- *5.1.2 - Deposit through tx.origin in "routerDeposit()" at "PsionicFarmInitializable.sol"*
- *5.1.4 - Order of operations can introduce round-off errors at "ZirconEnergy.sol"*

In the first case, the “msg.sender” can only be the “PSIONIC_FACTORY”:

```
File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-farming/contracts/PsionicFarmInitializable.sol
171:     function routerDeposit(uint256 _amount) external nonReentrant notPaused {
172:         require(IPsionicFarmFactory(address(PSIONIC_FACTORY)).PYLON_ROUTER() == msg.sender, "ONLY PYLON ROUTER");
173:         deposit(_amount, tx.origin);
174:     }
175:
```

While in the second case, the order of operation was changed:

```
File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-core/contracts/energy/ZirconEnergy.sol
104:   function getFeeByGamma(uint gammaMulDecimals) _initialize external view returns (uint amount) {
105:     (uint _minFee, uint _maxFee) = getFee();
106:     uint _gammaHalf = 5e17;
107:     uint x = (gammaMulDecimals > _gammaHalf) ? (gammaMulDecimals - _gammaHalf).mul(10) : (_gammaHalf - gammaMulDecimals).mul(10);
108:     if (gammaMulDecimals <= 45e16 || gammaMulDecimals >= 55e16) {
109:       amount = (_maxFee.mul(x).mul(x))/(25e36); //25 is a reduction factor based on the 0.45-0.55 range we're using.
110:     } else {
```

```
111:        amount = (_minFee.mul(x).mul(x).mul(36)/(1e36)).add(_minFee);
//Ensures minFee is the lowest value.
112:      }
113:    }
```

The rest MEDIUM-risk findings remained OPEN (see sections 5.1.1).

6.2 Retest of Low Severity Findings

The following LOW-risk vulnerabilities were found to be sufficiently mitigiated, since the affected functionalities have either been fixed or removed:

- *5.2.1 – Unvalidated _token_address in constructor at "ZirconDrop.sol"*
- *5.2.2 – Unvalidated _to address in "burn" function at "PsionicFarmVault.sol"*
- *5.2.7 - Multiple initializations are allowed in "ZirconEnergy.sol"*
- *5.2.8 - Multiple initializations are allowed in "ZirconEnergyRevenue.sol"*
- *5.2.22 -Lack of circuit breaker for emergency stop at "ZirconDrop.sol"*
- *5.2.23 - Lack of circuit breaker for emergency stop at "PsionicFarmInitializable.sol" and "PsionicFarmVault.sol"*
- *5.2.24 - Unchecked create2 call return value in "deployPool()" at "PsionicFarmFactory.sol"*
- *5.2.25 - Unchecked create2 call return value in "createPair()" at "ZirconFactory.sol"*

Regarding *5.2.1* and *5.2.2*, checks have been added to validate the inputs:

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-airdrop/contracts/ZirconDrop.sol

```
35:     constructor (address _token_address, bytes32 _merkleRoot, uint256
_start_time, uint256 _end_time) {
36:         require(validRange(48, _start_time), "Invalid Start Time");
37:         require(validRange(48, _end_time), "Invalid End Time");
38:         require(_token_address != address(0), "Invalid Token Address");
```

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-farming/contracts/PsionicFarmVault.sol

```
117:     function burn(address _to, uint _liquidity) external onlyFarm
nonReentrant {
```

```
118:          require(_to != address(0), "PFV: Invalid address");
```

In reference to *5.2.7* and *5.2.8,* a modifier has been added:

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/energy/ZirconEnergy.sol
37:   modifier _initialize() {
38:     require(initialized == 1, 'Zircon: FORBIDDEN');
39:     _;
40:   }
....
49:   function initialize(address _pylon, address _pair, address _token0,
address _token1) external {
50:     require(initialized == 0, "ZER: AI");
51:     require(msg.sender == energyFactory, 'Zircon: FORBIDDEN'); //
sufficient check
…
63:
64:     initialized = 1;
65:
66:   }
```

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/energy/ZirconEnergyRevenue.sol
36:     modifier _initialize() {
37:         require(initialized == 1, 'Zircon: FORBIDDEN');
38:         _;
39:     }
...
49:     function initialize(address _pair, address _tokenA, address
_tokenB, address _energy0, address _energy1, address _pylon0, address
_pylon1) external {
50:         require(initialized == 0, "ZER: Not Factory");
51:         require(energyFactory == msg.sender, "ZER: Not Factory");
52:         zircon = Zircon(
53:             _pair,
54:             _tokenA,
55:             _tokenB,
```

```
56:             _energy0,
57:             _energy1,
58:             _pylon0,
59:             _pylon1
60:         );
61:
62:         initialized = 1;
63:
64:     }
```

For 5.2.22 and *5.2.23, a pause* function was added:

File /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-airdrop/contracts/ZirconDrop.sol

```
47:     function pause() external onlyOwner {
48:         _pause();
49:     }
50:
51:     function unpause() external onlyOwner {
52:         _unpause();
53:     }
```

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-airdrop/contracts/ZirconDrop.sol

```
76:     function claim(uint256 index, uint256 amount, bytes32[] calldata
merkleProof) external nonReentrant whenNotPaused {
77:         require(!if_claimed(index), "Already Claimed");
```

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-farming/contracts/PsionicFarmFactory.sol

```
24:     function isPaused() external view returns (bool) {
25:         return _paused;
26:     }
27:
28:     function switchPause() external onlyOwner {
```

```
29:         _paused = !_paused;
30:     }
```

File: //zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-farming/contracts/PsionicFarmInitializable.sol

```
69:     modifier notPaused {
70:
require(!IPsionicFarmFactory(address(PSIONIC_FACTORY)).isPaused(), "The
factory is paused");
71:         _;
72:     }
..
163:     function deposit(uint256 _amount) external nonReentrant notPaused
{
164:         deposit(_amount, msg.sender);
...
180:     function withdraw(uint256 _amount) external nonReentrant
notPaused{
181:         UserInfo storage user = userInfo[msg.sender];
```

In reference to *5.2.24 and 5.2.25,* a check has been added:

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-farming/contracts/PsionicFarmFactory.sol

```
44:     function deployPool(
45:         …)…
57:         bytes32 saltVault = keccak256(abi.encodePacked(_stakedToken,
_startBlock));
58:
59:         assembly {
60:             psionicVault := create2(0, add(bytecodeVault, 32),
mload(bytecodeVault), saltVault)
61:         }
62:         require(psionicVault != address(0), "Vault creation failed");
63:
…
```

```
66:
67:         assembly {
68:             psionicFarmAddress := create2(0, add(bytecode, 32),
mload(bytecode), salt)
69:         }
70:         require(psionicFarmAddress != address(0), "Vault creation
failed");
```

```
File: /zircon-protocol-2-
fa04a132a81600612756397927dd52847b673f74/packages/zircon-
core/contracts/ZirconFactory.sol
47:
48:     function createPair(address tokenA, address tokenB, address
_pylonFactory) external returns (address pair) {
        ...
55:         assembly {
56:             pair := create2(0, add(bytecode, 32), mload(bytecode),
salt)
57:         }
58:         require(pair != address(0), 'ZF: PCF');
```

The rest LOW-risk findings remained OPEN (see sections 5.2.3, 5.2.4, 5.2.5, 5.2.6, 5.2.9, 5.2.10, 5.2.11, 5.2.15, 5.2.17, 5.2.18, 5.2.19, 5.2.20, 5.2.21, 5.2.26, 5.2.27 and 5.2.28).

6.3 Retest of Informational Findings

The following issues were found to be sufficiently addressed:

- *5.3.1 – Unchecked transfer call return value in "claim" functionality at "ZirconDrop.sol"*
- *5.3.3 - Debugging utility in "PsionicFarmVault.sol"*

Regarding *5.3.1*, the "transfer" function was replaced by "safeTransfer" which validates the result:

File: /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-airdrop/contracts/ZirconDrop.sol

```
096:     function withdraw() public onlyOwner {
097:         Info memory _info = info;
098:         require(block.timestamp > _info.end_time, "Not Expired");
099:         uint256 left =
IERC20(_info.token_address).balanceOf(address(this));
100:         require(left > 0, "What?");
101:         IERC20(_info.token_address).safeTransfer(msg.sender, left);
102:         emit Withdrawed(left, block.timestamp);
103:     }
104:
```

File /zircon-protocol-2-fa04a132a81600612756397927dd52847b673f74/packages/zircon-airdrop/contracts/ZirconDrop.sol

```
76:     function claim(uint256 index, uint256 amount, bytes32[] calldata
merkleProof) external nonReentrant whenNotPaused {
77:         require(!if_claimed(index), "Already Claimed");
78:         require(block.timestamp > info.start_time, "Not Started");
79:         require(block.timestamp < info.end_time && (block.timestamp -
info.start_time) / 86400 < 5, "Expired");
80:         bytes32 leaf = keccak256(abi.encodePacked(index, msg.sender,
amount));
81:         require(MerkleProof.verify(merkleProof, merkleRoot, leaf), 'Not
Verified');
82:         amount *= (10 ** 18);
// 18 decimals
83:         IERC20(info.token_address).safeTransfer(msg.sender, amount);
84:         set_claimed(index);
85:         emit Claimed(amount, block.timestamp);
86:     }
```

In reference to *5.3.3, the import was removed.*

The rest INFORMATIONAL findings remained OPEN (see sections 5.3.2 and 5.3.4)

References & Applicable Documents

Ref.	Title	Version
N/A	*N/A*	*N/A*

Document History

Revision	Description	Changes Made By	Date
0.2	*Initial Draft*	*Chaintroopers*	*August 4th, 2022*
1.0	*First Version*	*Chaintroopers*	*August 4th, 2022*
1.1	*Added retest results*	*Chaintroopers*	*August 31th , 2022*