

ZAP · SECURITY REVIEW

Zap Seasons.

A security review of Seasons.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Zap	Seasons	Guardian	November 29 – December 9, 2024	December 9, 2024

§ 00

Contents.

31 pages: overview, confidence ranking, scope, 20 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
§ 04	Findings	P. 06
	Main Review · 19 findings	P. 07
	Remediation Review · 1 finding	P. 29
§ 05	Disclaimer	P. 30
§ 06	About Guardian	P. 31

§ 01

Overview.

Guardian reviewed Seasons across 2 review rounds. The final report includes 2 High, 1 Medium, 17 Low findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/Lithium-Ventures/zap-memecoin-launcher
CLIENT	METHODOLOGY
Zap	Smart contract · Manual · Main Review · Remediation Review
PROJECT	MAIN REVIEW COMMIT
Seasons	7f1c47880d43
PREPARED BY	REMEDIATION REVIEW COMMIT
Guardian	e65332769957
AUDIT WINDOW	
November 29 – December 9, 2024	
FINAL REPORT	
December 9, 2024	

1.1 Confidence Ranking

4 / 5

High Confidence

Guardian assigns a Confidence Ranking of 4 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED	FIX REVIEW
● Critical	0	0	0	0	0	0
● High	2	0	0	0	2	0
● Medium	1	0	0	0	1	0
● Low	17	0	7	0	9	1
● Informational	0	0	0	0	0	0
All severities	20	0	7	0	12	1

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence ASSIGNED	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed Seasons in `https://github.com/Lithium-Ventures/zap-memecoin-launcher`.

IN SCOPE

LiquidityFreeLaunch.sol

Season.sol

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
H-01	Cumulative Tickets Can Be Inaccurate	Logical Error	HIGH	RESOLVED	p. 7
H-02	No validation winners claiming prizes multiple times	—	HIGH	RESOLVED	p. 8
M-01	More Than amountOfWinners May Be Selected	Gaming	MEDIUM	RESOLVED	p. 9
L-01	Lacking SafeTransfer Usage	Best Practices	LOW	RESOLVED	p. 10
L-02	Lacking OFFCHAIN Prize Logic	Warning	LOW	RESOLVED	p. 11
L-03	Malicious Token Name Risk	Validation	LOW	ACKNOWLEDGED	p. 12
L-04	Prizes Claimed Before All Winners Selected	Unexpected Behavior	LOW	RESOLVED	p. 13
L-05	Winners Should Always Be Able To Receive NFTs	Warning	LOW	ACKNOWLEDGED	p. 14
L-06	Typo	Typo	LOW	RESOLVED	p. 15
L-07	uploadEntrants Called After Winner Selection	Validation	LOW	RESOLVED	p. 16
L-08	isValidRequestHash Hardcoded To True	Warning	LOW	ACKNOWLEDGED	p. 17
L-09	Unnecessary If Case	Optimization	LOW	ACKNOWLEDGED	p. 18
L-10	Some Seasons May Be Hard To Complete	Warning	LOW	RESOLVED	p. 19
L-11	Requested Hash Uncleared	Best Practices	LOW	ACKNOWLEDGED	p. 20
L-12	Modulo Bias	Warning	LOW	ACKNOWLEDGED	p. 21
L-13	Lacking Nonzero Tickets Validation	Validation	LOW	RESOLVED	p. 24
L-14	Typo	Typo	LOW	RESOLVED	p. 25
L-15	Treasury Fee Taken Before Seed Amount	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 26
L-16	Unused Constant	Optimization	LOW	RESOLVED	p. 28
REMEDIATION REVIEW					
L-01	_fulfillRandomness Gas Expenditure	Warning	LOW	REPORTED	p. 29

HIGH **H-01**

RESOLVED

Cumulative Tickets Can Be Inaccurate

CATEGORY	Logical Error
LOCATION	https://github.com/Lithium-Ventures/zap-memecoin-launcher/blob/7f1c47880d43e67a2c73eb8a3a9f4cb864820670/contracts/Season.sol#L77
REVIEW	Main Review

DESCRIPTION

In the `uploadEntrants` function if the first user in the provided `users` array is being processed then the `previousTickets` amount used to write to the `cumulativeTickets` array is 0. This will be inaccurate if the `uploadEntrants` function is called multiple times, which it is likely to be in the event that a large number of entrants must be uploaded.

Additionally when processing subsequent users the `cumulativeTickets` value is read using the current iteration `index - 1`, which would also yield an inaccurate `previousTickets` value when the `uploadEntrants` function is called multiple times.

This will perturb the ability for users with higher ticket IDs to win the season as the random value is modulo by a `totalTickets` amount which will be smaller than the actual total tickets.

RECOMMENDATION

Instead of basing the `previousTickets` on the current users array index, base it off of the length of the `cumulativeTickets` array length + the current user array index.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/83015bf606b2f40345b1afd a3626db4a9ea81620>

HIGH **H-02**

RESOLVED

No validation winners claiming prizes multiple times

REVIEW

Main Review

— DESCRIPTION

A winner should only be able to claim once. I have added a patch in the linked commit

— RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/ac34f174e2f598dceda6fa39d5c0277eaa297f05>

MEDIUM **M-01**

RESOLVED

More Than amountOfWinners May Be Selected

CATEGORY Gaming

REVIEW Main Review

DESCRIPTION

**The **`selectWinner` function implements a validation against a winner being selected when the desired `amountOfWinners` has already been met, however since the winner is not pushed to the winners array until fulfillment of the random request the admin could intentionally or accidentally call the `selectWinner` function more than the `amountOfWinners` number of times and allow the system to accrue more than the `amountOfWinners` in the `winners` array.

RECOMMENDATION

Consider also including the `winners` length validation in the `_fulfillRandomness` function.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/926f8e55f42bbe9e34261446ca665098e7ba0848>

LOW **L-01**

RESOLVED

Lacking SafeTransfer Usage

CATEGORY Best Practices

LOCATION <https://github.com/Lithium-Ventures/zap-memecoin-launcher/blob/7f1c47880d43e67a2c73eb8a3a9f4cb864820670/contracts/Season.sol#L115C32-L115C44>

REVIEW Main Review

DESCRIPTION

In the `initSeason` function the price ERC20 tokens are transferred in using `transferFrom` without validating the return value. Instead `safeTransfer` should be used to safely handle any arbitrary price tokens.

Additionally in the `claimPrizes` function transfer is used without checking the return value when sending ERC20 tokens to the winner.

RECOMMENDATION

Use `safeTransferFrom` for the transferring of ERC20 prize tokens in the `initSeason` function and `safeTransfer` when sending ERC20 prize rewards to the winner in the `claimPrizes` function.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/fc5ba8f98ad31ae556097ad8d48a33cb73da9f83>

LOW **L-02**

RESOLVED

Lacking OFFCHAIN Prize Logic

CATEGORY Warning
LOCATION Season.sol
REVIEW Main Review

— DESCRIPTION

In the `Season` contract there is no logic in the `claimPrizes` function for the `PrizeType.OFFCHAIN` prize type. This may be the expected behavior and all logic might be computed off-chain, however it may be useful to emit an event in the `claimPrizes` function to indicate that off-chain rewards should be delivered, for instance it may be useful to see which off-chain item `uuid` should be received by the user.

— RECOMMENDATION

Consider whether the `claimPrizes` function should include any logic, such as an event to indicate that off chain prizes have been claimed.

— RESOLUTION

Status: Resolved.

LOW **L-03**

ACKNOWLEDGED

Malicious Token Name Risk

CATEGORY	Validation
LOCATION	LiquidityFreeLaunch.sol: 46
REVIEW	Main Review

— DESCRIPTION

The `LiquidityLaunchFactory` contract allows users to deploy a custom token including the ability to set a `name` and `symbol` for the new token.

However, it is possible for an attacker to craft a `name` such that it includes markup that can contain Javascript code. If loaded into a frontend without XSS protection, this can cause potential harm to users of the platform as was the case in the EtherDelta exploit.

For more details on the EtherDelta exploit please refer to this article:

[<https://medium.com/@zhongqiangc/code-injection-attacks-using-blockchains-part-1-db4e121890bb>]

(<https://medium.com/@zhongqiangc/code-injection-attacks-using-blockchains-part-1-db4e121890bb>)

— RECOMMENDATION

Consider adding a length limit to the token name in addition to the pre-existing validation for the symbol.

— RESOLUTION

Status: Acknowledged.

LOW **L-04**

RESOLVED

Prizes Claimed Before All Winners Selected

CATEGORY Unexpected Behavior

LOCATION Season.sol: 156

REVIEW Main Review

DESCRIPTION

In the `claimPrizes` function there is no requirement that all winners have been chosen when the claim is taking place. This may be unexpected for the Zap team.

RECOMMENDATION

Consider whether all winners should be chosen before any claims are allowed to occur, and if so then implement the corresponding validation in the `claimPrizes` function.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/0ab5705bf3f0471aa6c1a9654d2cb5067ee9e95d>

LOW **L-05**

ACKNOWLEDGED

Winners Should Always Be Able To Receive NFTs

CATEGORY Warning
LOCATION Season.sol: 178
REVIEW Main Review

— DESCRIPTION

In the `claimPrizes` function when awarding a winner with an ERC721 token they have won `safeTransferFrom` is used which will revert if the receiver is a smart contract and does not implement the `onErc721Received` function.
For participants that use a smart contract address this will prevent them from claiming all of their rewards.

— RECOMMENDATION

Be sure to document this risk to users so this issue does not arise.

— RESOLUTION

Status: Acknowledged.

LOW **L-06**

RESOLVED

Typo

CATEGORY Typo
LOCATION Season.sol: 218
REVIEW Main Review

DESCRIPTION

In the comment for the `onERC721Received` function the word receiver is misspelled as `reciever`.

RECOMMENDATION

Correct the spelling to `receiver`.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/ea904df78f538c383d8d86496917400277562aad>

LOW **L-07**

RESOLVED

uploadEntrants Called After Winner Selection

CATEGORY Validation
LOCATION Season.sol: 66
REVIEW Main Review

— DESCRIPTION

In the Season contract the `uploadEntrants` function can be called after winners have already been chosen. This action would not normally be taken by an admin, however it should be explicitly prevented to avoid any wasted gas if the transaction were to be accidentally sent.

— RECOMMENDATION

Add validation to the `uploadEntrants` function that asserts that no winners have been chosen.

— RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/c4487183552d16064052dd20753567867e580167>

LOW **L-08**

ACKNOWLEDGED

isValidRequestHash Hardcoded To True

CATEGORY Warning
LOCATION GelatoVRFConsumerBase.sol: 78
REVIEW Main Review

— DESCRIPTION

The `isValidRequestHash` variable is currently hardcoded to true with the comment that `This is a temporary fix to make the test pass`.

— RECOMMENDATION

Be sure to resolve this comment and use the actual logic for the `isValidRequestHash` value.

— RESOLUTION

Status: Acknowledged.

LOW **L-09**

ACKNOWLEDGED

Unnecessary If Case

CATEGORY Optimization
LOCATION GelatoVrfConsumerBase.sol: 84
REVIEW Main Review

DESCRIPTION

In the `fulfillRandomness` function there is an if case which relies on the `isValidRequestHash` value after the `isValidRequestHash` value has already been required to be true.

RECOMMENDATION

Either remove the require statement or the if case as both are unnecessary together.

RESOLUTION

Status: Acknowledged.

LOW **L-10**

RESOLVED

Some Seasons May Be Hard To Complete

CATEGORY	Warning
LOCATION	Season.sol
REVIEW	Main Review

— DESCRIPTION

In the event that one or just a couple users have purchased the majority of tickets in a season it may be a very high probability that a previous winner is selected in the `_fulfillRandomness` Function. If the odds are sufficiently large for a small group of winners then it may be unrealistic to be able to complete a Season. Additionally this will cost an unnecessary amount of gas fees to the operator and additional fees to cover the Gelato cost.

— RECOMMENDATION

Be aware of this risk and ensure Seasons have a wide distribution of users with no single large ticket holders. Furthermore if this behavior should be avoided, consider adding an escape hatch to end a Season early or even removing winners from the `entrantAddresses` and `cumulativeTickets`, though this will likely introduce more complexity than the resolution is worth.

— RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/e65332769957221394d06af3544cc73ccf335f31>

LOW **L-11**

ACKNOWLEDGED

Requested Hash Uncleared

CATEGORY Best Practices
LOCATION GelatoVRFConsumerBase.sol
REVIEW Main Review

— DESCRIPTION

In the `_requestRandomness` function the `requestedHash` entry for the request is not cleared. This diverts from the latest `GelatoVRFConsumerBase` contract implementation here:

[<https://github.com/gelatodigital/vrf-contracts/blob/main/contracts/GelatoVRFConsumerBase.sol>]

(<https://github.com/gelatodigital/vrf-contracts/blob/main/contracts/GelatoVRFConsumerBase.sol>)

— RECOMMENDATION

Consider adding logic to delete the `requestedHash` in the event that the random number was accepted.

— RESOLUTION

Status: Acknowledged.

LOW **L-12**

ACKNOWLEDGED

Modulo Bias

CATEGORY Warning

LOCATION <https://github.com/Lithium-Ventures/zap-memecoin-launcher/blob/7f1c47880d43e67a2c73eb8a3a9f4cb864820670/contracts/Season.sol#L197>

REVIEW Main Review

DESCRIPTION

The `randomTicketNumber` value is computed using modulo for the final randomized value, however this can lead to slightly unfair probability distributions due to what is known as modulo bias.

For example, consider the simplified scenario where the `totalTickets` is 3 and the random number is generated from 0 to 10 inclusive. There are three outcomes for the `randomTicketNumber`, however the probabilities which are shown below are not equal.

Random	Result Mod 3
0	0
1	1
2	2
3	0
4	1
5	2
6	0
7	1
8	2
9	0
10	1

L-12 Modulo Bias

CONTINUED

```

</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</table>

```

Probabilities for `randomTicketNumber` :

```

<table>
<tr>
<td>randomTicketNumber</td>
<td>Probability</td>
</tr>
<tr>
<td>0</td>
<td>4/11 = 36.36%</td>
</tr>
<tr>
<td>1</td>
<td>4/11 = 36.36%</td>

```

L-12 Modulo Bias

CONTINUED

```
</tr>
<tr>
<td>2</td>
<td>3/11 = 27.27%</td>
</tr>
</table>
```

In actuality the `totalTickets` will be much larger and the random number is generated in the entire range of a `uint256` word, however this simplified example demonstrates how the probabilities may not be even amongst all ticket numbers when the modulo does not divide `type(uint256).max + 1`.

— RECOMMENDATION

In actuality the modulo bias experienced by Zap will be reduced notably as higher ticket amounts are minted, so it is likely that this bias can be safely ignored for the purposes of Zap's Season contract. However if modulo bias should be a priority to remove, one approach could be to require that tickets purchased always result in a modulo divisor that divides the cardinality of possible random numbers, e.g. `type(uint256).max + 1` (all positive integers in a `uint256` and 0).

— RESOLUTION

Status: Acknowledged.

LOW **L-13**

RESOLVED

Lacking Nonzero Tickets Validation

CATEGORY Validation
LOCATION Season.sol: 66
REVIEW Main Review

DESCRIPTION

**In the **`uploadEntrants` function there is no validation that the provided `user.tickets` value is greater than 0. If a `user.tickets` value of 0 were to be used then a user can exist in the `entrantAddresses` array multiple times which would be unexpected behavior.

RECOMMENDATION

The `uploadEntrants` function is performed by a trusted party, however to avoid any unintentional mistakes it may be prudent to add validation which prevents a `user.tickets` value of 0.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/7b68c96921d39d5f9899a619abd5ca0d9c0a70c2>

LOW **L-14**

RESOLVED

Typo

CATEGORY	Typo
LOCATION	LiquidityFreeLaunch.sol
REVIEW	Main Review

— DESCRIPTION

In the `LiquidityFreeLaunch` contract the `zapDegenTokenReciever` variable misspells receiver as `reciever`.

— RECOMMENDATION

Correct the spelling to `receiver`.

— RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/8c6d3c91f0553a6216077b3206c3ca80fdea5277>

LOW **L-15**

ACKNOWLEDGED

Treasury Fee Taken Before Seed Amount

CATEGORY Unexpected Behavior
 LOCATION LiquidityFreeLaunch.sol: 149
 REVIEW Main Review

DESCRIPTION

In the `seedWithFeeAndAcceptToken` function the `feeForZapTreasury` is bought with the `getTokensOutForEthDeposit` and the results are added to the `totalSold` and `balance` values before the initial seed amount is added.

This results in the fee being larger and the seeded `amountOut` being smaller.

Using explicative numbers, if the `feeForZapTreasury` is 0.01 ETH and the `seedAmount` is 0.1 ETH then:

For the `feeForZapTreasury`:

$$A = (1 + k * 0.01)^n = 1.00593$$

$$\text{startSupply} = 0$$

$$\text{endSupply} = \text{totalTokens} * (1 - 1/A) = 910_374_363 * (1 - 1/1.00593) = 5_366_695$$

$$\text{amountOutForZapTreasury} = 5_366_695 \text{ tokens}$$

For the `amountOut`:

$$A = (1 + k * (0.1 + 0.01))^n = 1.06696$$

$$\text{startSupply} = 5_366_695 \text{ tokens}$$

$$\text{endSupply} = \text{totalTokens} * (1 - 1/A) = 910_374_363 * (1 - 1/1.06696) = 57_133_039$$

$$\text{amountOut} = 57_133_039 - 5_366_695 = 51_766_344 \text{ tokens}$$

However if the ordering were reversed the `amountOut` is larger and the zap treasury fee is smaller.

For the `amountOut`:

$$A = (1 + k * (0.1))^n = 1.06072$$

$$\text{startSupply} = 0 \text{ tokens}$$

$$\text{endSupply} = \text{totalTokens} * (1 - 1/A) = 910_374_363 * (1 - 1/1.06072) = 52_113_593$$

$$\text{amountOut} = 52_113_593 \text{ tokens}$$

L-15 Treasury Fee Taken Before Seed Amount

CONTINUED

For the `feeForZapTreasury` :

$$A = (1 + k * (0.1 + 0.01))^n = 1.06696$$

$$\text{startSupply} = 52_113_593 \text{ tokens}$$

$$\text{endSupply} = \text{totalTokens} * (1 - 1/A) = 910_374_363 * (1 - 1/1.06696) = 57_133_039$$

$$\text{amountOutForZapTreasury} = 57_133_039 ** - ** 52_113_593 = 5_019_446 \text{ tokens}$$

— RECOMMENDATION

This may be the expected behavior since the fee can be reduced for larger initial seed amounts, however it should be confirmed whether this ordering is expected and clearly documented for users.

— RESOLUTION

Status: Acknowledged.

LOW **L-16**

RESOLVED

Unused Constant

CATEGORY Optimization
LOCATION LiquidityFreeLaunch.sol
REVIEW Main Review

DESCRIPTION

In the `LiquidityFreeLaunch` contract the `DUST_THRESHOLD` constant is unused.

RECOMMENDATION

Remove or implement the use case for the `DUST_THRESHOLD`.

RESOLUTION

Status: Resolved.

<https://github.com/Lithium-Ventures/zap-memecoin-launcher/commit/2948a4c3e9aa9d2e0c56d216a102ae6ae7435e70>

LOW **L-01**

_fulfillRandomness Gas Expenditure

CATEGORY	Warning
LOCATION	Season.sol: 220
REVIEW	Remediation Review

— DESCRIPTION

In the `_fulfillRandomness` function a `while(true)` loops is now used to re-draw random numbers to avoid selecting a winner that has already been chosen. In cases where a single user has the majority of tickets in a raffle this loop may execute many times before finding a unique winner, and may even need more gas than the block gas limit.

This will incur a large gas fee and thus a large gelato fee to Zap and may not result in a unique winner being chosen. To limit the gas expenditure for cases where the loop will execute many times a while loop iteration limit can be introduced.

— RECOMMENDATION

Consider introducing an iteration limit in the `_fulfillRandomness` function to ensure that unexpectedly large gas fees are not levied on the protocol.

— RESOLUTION

Status: Reported during the fix review period.

§ 05

Disclaimer.

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

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

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

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.