Cyberscope

Audit Report

Snark Launch

April 2023

Network ZKSYNC

Audited by © cyberscope

Table of Contents

Review

Contract Name	SnrkPrivate
Compiler Version	v0.8.17
Optimized	Yes
Explorer	https://explorer.zksync.io
Address	
Network	ZKSYNC

Audit Updates

Initial Audit	08 Apr 2023

Source Files

Filename	SHA256
SnrkPrivate.sol	c40cfa9bf845eadecf5118d404337b0ae94583c4c8c76ab4da57196803c93f1e

Findings Breakdown

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	RSV	Redundant State Variable	Unresolved
	CR	Code Repetition	Unresolved
	MEM	Misleading Error Messages	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

RSV - Redundant State Variable

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L38
Status	Unresolved

Description

The contract uses a private state variable called `inserted` that determines if a user has participated in the contribution. On the other hand, the state variable called `allocationAmount` stores the total allocation of every user. That means that if the `allocationAmount` of a specific user is not zero, then the user has participated in the contribution. Hence, the usage of the `inserted` variable is redundant. The redundant usage of a state variable unnecessarily increases gas consumption and decreases code readability.

```
if (inserted[msg.sender]) {
    allocationAmount[msg.sender] += _amount;
} else {
    inserted[msg.sender] = true;
    allocationAmount[msg.sender] = _amount;
    participants.push(msg.sender);
}
```

Recommendation

The team is advised to remove the state variable `inserted` and check if the `allocationAmount` is not zero in order to achieve the same result with less gas consumption.

CR - Code Repetition

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L38
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

The `allocationAmount` in both cases is increased by the amount. If it is the first time for the user, then the `allocationAmount` is by default zero.

```
if (inserted[msg.sender]) {
    allocationAmount[msg.sender] += _amount;
} else {
    inserted[msg.sender] = true;
    allocationAmount[msg.sender] = _amount;
    participants.push(msg.sender);
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. The expression that involves the `allocationAmount` could be moved from both if-else statements to one expression outside the if-else.

A suggested implementation could be:

```
if (!inserted[msg.sender]) {
    inserted[msg.sender] = true;
    participants.push(msg.sender);
}

allocationAmount[msg.sender] += _amount;
```

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L36
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

The contract reverts the execution if the deposited amount is not between 0.1 and 10 native tokens. The error message is "Over the allowed range" but if the user deposits an amount less than 0.1 then the error is misleading since the amount is under the allowed range.

```
require((_amount >= 0.1 * 10 **18) && (_amount <= 10 * 10 **18),
"Over the allowed range");
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L30
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
owner
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L21
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address public recipient =
0xEab8573343887E16efCfc6bD9C31b4f28e80ba84
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L67
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint _amount
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	SnrkPrivate.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.17;
```

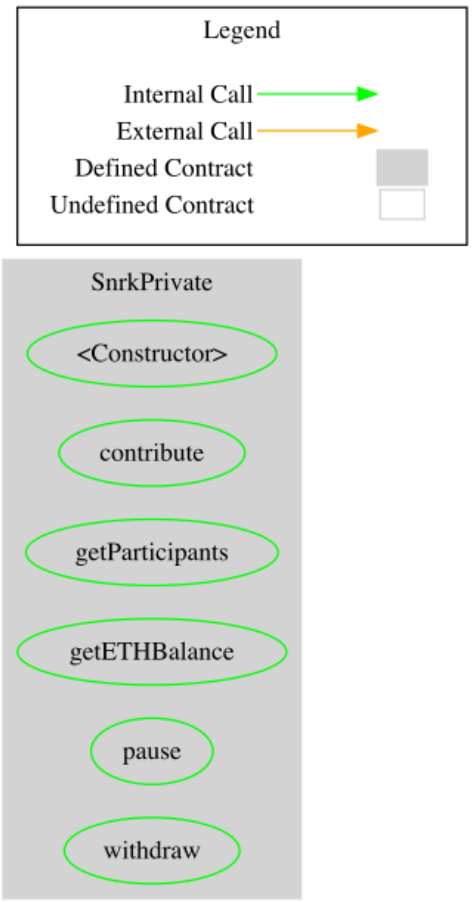
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SnrkPrivate	Implementation			
		Public	✓	-
	contribute	Public	Payable	-
	getParticipants	Public		-
	getETHBalance	Public		-
	pause	Public	✓	-
	withdraw	Public	✓	-

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SnrkPrivate
<Constructor>
contribute
getParticipants
getETHBalance
pause
withdraw

Summary

Snark Launch contract implements a fund raising mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error or critical issues.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope are subject to dependencies and are 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io