Shiryo

Smart Contract Security Assessment

December 23, 2024

VERACITY

Disclaimer

Veracity Security ("Veracity") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature.

The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by the Veracity team.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Veracity is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Veracity or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for **Shiryo project** Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective. The scope of this audit is the initial raise contract Shiryov2.sol, which includes industry standard libraries from OpenZepplin.

1.1 Summary

Name	Shiryo
URL	https://shiryo.com
Platform	Ethereum
Language	Solidity

Shiryo consists of 2 contracts:

Shiryov2.sol	// Core contract for receiving investor deposits
ShiryoTokenClaim.sol	//Contract for managing token claims by investors

1.2 Testing

Following an initial pass on all contracts, we performed a series of tests. However it is not possible to catch all scenarios with these tests. Veracity has implemented a suite of audit tests that also exercise the primary functions of each contract to ensure that no transaction or fund locking occurs.

Tests have been implemented with the Foundry fuzz testing framework and some of the issues discovered are listed in the tables below. No further critical issues were discovered during this secondary process.

1.3 Final Contracts Assessed

Following deployment of the contracts assessed, Veracity compares the contracts that have been deployed, and wired with the contracts that have been audited to guarantee no tampering has been possible between audit report issue and project start.

This gives project owners and community members confidence that what has been deployed matches the findings and resolution status described in these documents.

https://github.com/krypt0ape/shiryo-contracts/blob/main/contracts/shiryov2.sol
https://github.com/krypt0ape/shiryo-contracts/blob/main/contracts/ShiryoTokenClaim.sol

Github hash: `3d03310`

Deployment network: Ethereum

Project wallet address:

Links to verified contracts (2):

Name	Address	Network	Matched
Shiryov2.sol	ETH: https://etherscan.io/address/0x5bd03ed7d4cd9048b95eceacb862becfdbd86ec2#code	Ethereum	**YES**
ShiryoTokenClaim.sol	ETH: https://etherscan.io/address/0x8Eb9B0B0737D9D75108842Dfc6820c32FF0622d1#code	Ethereum	**YES**

There are 2 contracts deployed.

1.4 Findings Summary

Individual issues found have been categorised based on criticality as high, medium, low or informational. The client is required to respond to each issue individually, although it may be by design and therefore simply acknowledged. Additional recommendations may apply to all contracts, but are replicated for each for resolution.

For example an issue relating to centralisation of financial risk may apply to all administration functions, but will be included only once per contract. The table below shows the collected number of issues found and the resolution statuses across all contracts in the project.

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change)
• High	2	2	0	0
• Medium	2	2	0	0
• Low	3	1	0	2
• Informational	0	0	0	0
Total	7	5	0	2

1.4.1 Status Classifications

Severity	Description
• High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
• Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
• Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
• Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.
• Optimization	Suboptimal implementations that may result in additional gas consumption, unnecessary computation or avoidable inefficiencies.

1.4.2 Collected Issues and Statuses

ID	Contract	Severity	Summary	Status
01	Shiryov2.sol	MEDIUM	The code performs Ether transfers using `.call` but does not validate the `success` return value. Ignoring this can lead to silent failures if the transfer fails.	RESOLVED
02	Shiryov2.sol	LOW	The state variables `router` (Line 1136) and `taxWallet` (Line 1137) are not updated after deployment. Declaring them as `constant` will save gas during contract execution.	RESOLVED
03	Shiryov2.sol	HIGH	`swapBack()` causes tokens to be locked in contract including ETH and token.	RESOLVED
04	Shiryov2.sol	HIGH	Absence of Minimum Return Amount in Token Swap	RESOLVED
05	ShiryoTokenClaim.sol	MEDIUM	The `registerClaim` function includes an unnecessary `_address` parameter, which introduces a risk of misuse. The function should always register claims for `msg.sender` instead of relying on an externally provided `_address`. Allowing `_address` to be passed as a parameter could enable malicious actors to spoof claims on behalf of other users, leading to unauthorized or incorrect claims being set.	RESOLVED
06	ShiryoTokenClaim.sol	LOW	The contract should emit events for state-changing actions like	ACKNOWLEDGED

			`registerClaim` and `claim` to provide transparency and enable efficient tracking of these operations.	
07	ShiryoTokenClaim.sol	LOW	The current contract allows users to claim tokens without any time restriction. It is recommended to implement an expiry or deadline for the claims (even with a very long duration) to ensure claims are only valid for a certain period. This helps to prevent users from claiming tokens indefinitely or doing arbitrage.	ACKNOWLEDGED

2. Findings

The contract(s) assessed have been largely authored from scratch rather than using industry tested implementations for ERC20. Standard interfaces have been included inline which adds risk of errors, however code comparison shows no errors have been introduced during this process. This can result in the introduction of vulnerabilities or bugs that have not been seen or addressed in previous projects. However our team has made recommendations and several code sweeps to mitigate the effect of not using industry standard libraries. The following sections outline issues found with individual contracts.

2.1 Shiryo

This report has been prepared for the **Shiryo project**. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

2.1.1 Privileged Roles

The following functions can be called by the admin or manager role of the Shiryov2.sol contract:

- renounceOwnership
- transferOwnership
- enableTrading
- removeLimits
- disableTransferDelay

- updateSwapTokensAtAmount
- updateMaxTxnAmount
- updateMaxWalletAmount
- excludeFromMaxTransaction
- updateSwapEnabled
- updateBuyFees
- updateSellFees
- excludeFromFees
- blacklistAccount
- setAutomatedMarketMakerPair
- updateFeeWallet
- setSlippage

2.1.2 Initial Token Allocation

No tokens are allocated on initialization.

2.1.3 Taxes, Rules, Initial Variables.

Name	Shiryo
Symbol	SHIRYO
Initial Supply	150 Million
Decimals	18
Router	Uniswap V2: 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
Reflection Fee	0
Liquidity Fee	0
Marketing Fee	0
Max Reflection Fee	
Max Buy Fee	40%
Max Sell Fee	40%

2.1.4 Shiryo Issues & Recommendation

Issue Number: 1

Title: Missing `success` check after `.call` execution

Severity: Medium

Files: `Shiryov2.sol`

Summary:
The code performs Ether transfers using `.call` but does not validate the `success` return value. Ignoring this can lead to silent failures if the transfer fails.

Proposed Fix:
Add a `require` statement to check the `success` result and handle failures explicitly.

- Update the Code as Follows:
  For `devWallet` transfer:

```
(bool success, ) = address(devWallet).call{value: ethForDev}("");
require(success, "Transfer to devWallet failed");
```

- For `marketingWallet` transfer:

```
(bool success, ) = address(marketingWallet).call{value:
address(this).balance}("");
require(success, "Transfer to marketingWallet failed");
```

Resolution: Issue fixed.

Issue Number: 2

Title: `router` and `taxWallet` should be declared `constant`

Severity: Low

File: `Shiryov2.sol`

Summary:
The state variables `router` (Line 1136) and `taxWallet` (Line 1137) are not updated after deployment. Declaring them as `constant` will save gas during contract execution.

Proposed Fix:
Update the variable declarations to include the `constant` keyword:

For `router` (L1136):

```
address public constant router = <address>;
```

For `taxWallet` (L1137):

```
address public constant taxWallet = <address>;
```

Resolution: Issue fixed.

Issue Number: 3

Title: `swapBack()` causes tokens to be locked in contract including ETH and token.

Severity: High

File: `Shiryov2.sol L1518`

Summary:
The deployed token contract includes a mechanism to swap tokens for ETH when specific conditions are satisfied. However, with fees set to 0 and ownership renounced, the following condition in the swapBack() function blocks the swap operation:

if (contractBalance == 0 || totalTokensToSwap == 0) { return; }

Proposed Fix:
Adjust the condition above in `swapBack()` .

Resolution: Issue fixed.

Issue Number: 4

Title: Absence of Minimum Return Amount in Token Swap

Severity: High

File: `shiryov2.sol`

Summary:
swapTokensForEth() at L1417 is invoked without setting amountOutMin, exposing the transaction to potential manipulation by bots through sandwich attacks.

Description:
The function allows for token to ETH swaps with no lower limit on the ETH received (amountOutMin = 0). This causes unfavorable rates due to market manipulation such as front-running and sandwich attacks.

Recommendation:
Implement slippage protection by setting an amountOutMin based on currentPrice as with getAmountsOut.

Resolution: Issue fixed.

2.2 TokenClaim

This report has been prepared for the **Shiyo project**. Veracity provides an examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

2.2.1 Privileged Roles

The following functions can be called by the admin or manager role of the TokenClaim contract:

- emergencyWithdrawTokenOut
- udpateMaxClaimableAmount
- updateSigner

2.2.2 Initial Token Allocation

No tokens are allocated on initialization.

2.2.3 Taxes, Rules, Initial Variables.

Not applicable for TokenClaim.

2.2.4 TokenClaim Issues & Recommendations

Issue Number: 5

Title: Unnecessary `_address` Parameter in `registerClaim` Function

Severity: Medium

File: `ShiryoTokenClaim.sol`

Summary:
The `registerClaim` function includes an unnecessary `_address` parameter, which introduces a risk of misuse. The function should always register claims for `msg.sender` instead of relying on an externally provided `_address`. Allowing `_address` to be passed as a parameter could enable malicious actors to spoof claims on behalf of other users, leading to unauthorized or incorrect claims being set.

Proposed Fix:
Remove the `_address` parameter from the function and replace all references to `_address` with `msg.sender`.

Resolution: Issue fixed.

Issue Number: 6

Title: Missing Event Emissions in Key Functions

Severity: Low

File: `ShiryoTokenClaim.sol`

Summary:
The contract should emit events for state-changing actions like `registerClaim` and `claim` to provide transparency and enable efficient tracking of these operations.

Resolution: Issue acknowledged.

Issue Number: 7

Title: Lack of Expiry or Deadline for Claims

Severity: Low

File: `ShiryoTokenClaim.sol`

Summary:
The current contract allows users to claim tokens without any time restriction. It is recommended to implement an expiry or deadline for the claims (even with a very long duration) to ensure claims are only valid for a certain period. This helps to prevent users from claiming tokens indefinitely or doing arbitrage.

Proposed Fix:
Add a `claimDeadline` variable that limits the time period during which claims can be made.

Resolution: Issue acknowledged.