Shiryo

Smart Contract Security Assessment

December 13, 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 Deposit.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:

Shiryo.sol	// Core contract for receiving investor deposits
TokenSwap.sol	// Allows users to swap their tokens

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 this document.

https://github.com/krypt0ape/shiryo-contracts/tree/main

Github hash: 9bf968c

Deployment network: Ethereum

Project wallet address:

Links to verified contracts (2):

Name	Address	Network	Matched
shiryo.sol	ETH: https://etherscan.io/address/0x47037D765bE953e6DE1D446c15851D89F9033607#code	Ethereum,	**YES**
TokenSwap.sol	ETH:	Ethereum,	**NO**

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	1	1	0	0
• Medium	1	1	0	0
• Low	4	3	0	1
• Informational	0	0	0	0
Total	6	5	0	1

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	Shiryo.sol	MEDIUM	The `takeETHback` function uses `.transfer`, which risks transaction failures due to its 2300 gas limit. This method can cause silent failures, especially if the recipient's fallback function requires more gas.	RESOLVED
02	Shiryo.sol	HIGH	swapExactTokensForETHSupportingFeeOnTransferTokens() at L697 is invoked without setting amountOutMin, exposing the transaction to potential manipulation by bots through sandwich attacks.	RESOLVED
03	Shiryo.sol	LOW	The _name and _symbol variables are declared but not initialized, leading to incomplete token metadata.	RESOLVED
04	TokenSwap.sol	LOW	The if-else condition checking startingSupply to determine _decimals and _decimalsMul is unnecessary as the outcome is constant and predictable.	RESOLVED
05	TokenSwap.sol	LOW	The claim functions lack a deadline parameter, leaving it vulnerable to arbitrage during prolonged execution windows.	RESOLVED
06	TokenSwap.sol	LOW	The `TokenSwap` contract does not emit events for critical operations, such as token swaps or claiming bonuses, making it difficult to monitor or verify activities.	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 **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.

2.1.1 Privileged Roles

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

- intializeContract
- transferOwner
- renounceOwnership
- setNewRouter
- setLpPair
- setTaxesBuy
- setTaxesSell
- setTaxesTransfer
- setValues
- setRatios
- MaxTx
- WalletSize
- setSwapSettings
- NewMarketWallet
- NewDevWallet
- setSwapAndLiquifyEnabled
- setExcludedFromReward
- enableTrading
- takeETHback

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	10%
Max Liquidity Fee	10%
Max Marketing Fee	22%

2.1.4 Shiryo Issues & Recommendations

Issue Number: 1

Title: Replace `.transfer` with `.call` for Ether Transfers

Severity: Medium

Files: `Shiryo.sol`

Proposed Fix:
Use `.call` instead of `.transfer` for sending Ether to provide higher flexibility and explicit error handling. Update the function as follows:

For takeETHback() L857:

```
(bool success, ) = payable(owner()).call{value:
address(this).balance}("");
require(success, "Failed to send Ether");
```

For swapAndLiquify() L724-726:

```
((bool marketSuccess, ) = _marketWallet.call{value:
marketFee}("");
require(marketSuccess, "Market wallet transfer failed");
(bool devSuccess, ) = _devWallet.call{value: devfeeshare}("");
require(devSuccess, "Dev wallet transfer failed");
```

Resolution:

Proposed fix implemented.

Issue Number: 2

Title: Absence of Minimum Return Amount in Token Swap

Severity: High

File: `Shiryo.sol`

Summary:
`swapExactTokensForETHSupportingFeeOnTransferTokens()` at L697 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:

Proposed fix implemented.

Issue Number: 3

Title: Uninitialized _name and _symbol Variables

Severity: Low

Files: `Shiryo.sol`

Summary:
The _name and _symbol variables are declared but not initialized, leading to incomplete token metadata.

Description:
These variables provide the token's name and symbol, and their absence may cause issues with token interfaces like wallets or explorers.

Recommendation:
Initialize _name and _symbol with constant values like :

```
string private constant _name = "Shiryo-Inu-v2";
string private constant _symbol = "Shiryo-Inu-v2";
```

Resolution:

Proposed fix implemented.

2.2 TokenSwap

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 TokenSwap contract:

- emergencyWithdrawTokenIn
- emergencyWithdrawTokenOut

2.2.2 Initial Token Allocation

No tokens are allocated on initialization..

2.2.3 Taxes, Rules, Initial Variables.

Not applicable for TokenSwap.

2.2.4 TokenSwap Issues & Recommendations

Issue Number: 4

Title: Redundant Condition for _decimals and _decimalsMul Initialization

Severity: Low

File: `TokenSwap.sol`

Summary:
The if-else condition checking startingSupply to determine _decimals and _decimalsMul is unnecessary as the outcome is constant and predictable.

Description:
The startingSupply is hard coded as 150_000_000, which is always less than 10000000000000. This makes the if-else condition redundant since only the first branch will execute. Additionally, the operation _decimalsMul = _decimals; is identical in both branches.

Recommendation:
Remove the if-else statement and directly assign the values as follows:

```
startingSupply = 150_000_000;
_decimals = 18;
_decimalsMul = _decimals;
```

Resolution:

Proposed fix implemented.

Issue Number: 5

Title: Recommendation to add Deadline Parameter in Claim function.

Severity: Low

File: `TokenSwap.sol`

Summary:
The claim functions lack a deadline parameter, leaving it vulnerable to arbitrage during prolonged execution windows.

Description:
If the price of the new token is high enough compared to the old token, users can buy the cheap old one and swap to get the 10% stakeBonus that could be worth more thus extracting value through arbitrage. This effect can be amplified with a flash loan.

Recommendation:
Include a deadline parameter in the claim function.

Resolution:

Proposed fix implemented.

Issue Number: 6

Title: Lack of Events

Severity: Low

File: `TokenSwap.sol`

Summary:
The `TokenSwap` contract does not emit events for critical operations, such as token swaps or claiming bonuses, making it difficult to monitor or verify activities.

Description:
The absence of events reduces transparency and prevents the use of analytical tools to track key operations. This could hinder integration efforts and diminish user trust in the contract.

Recommendation:
Add events for significant actions like claim() to enhance transparency and compatibility with monitoring tools.

Resolution:

Issue acknowledged.