PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Sage

28 March 2024

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 Sage 6

2 Findings 7

2.1 Sage 7

2.1.1 Privileged Functions 7

2.1.2 Issues & Recommendations 8

Disclaimer

Paladin Blockchain Security ("Paladin") 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 advocation 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 Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons 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 Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin 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 therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Sage on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Sage
URL	https://www.sageterminal.io/#/terminal
Platform	Ethereum
Language	Solidity
Preliminary Contracts	`0xEa277346e91F35B254232A07d9A08757e4c92564`
Resolution 1	https://github.com/Sageterminal/Sage-Contracts/commit/0142fa7f9b0b9b1ee185ccf50e71bdada6366df2

1.2 Contracts Assessed

Name	Contract	Live Code Match
Sage	`N/A`	UNDEPLOYED

Note: The client has not deployed the contract with the fixes. Users should be aware of the issues in this report when interacting with the original contract.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	0	-	-	-
High	1	1	-	-
Medium	1	-	-	1
Low	2	-	-	2
Informational	2	-	-	2
Total	**6**	**1**	**-**	**5**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
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 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.

1.3.1 Sage

ID	Severity	Summary	Status
01	HIGH	Users overpay for tokens in `buy()`	✓ RESOLVED
02	MEDIUM	Tokens held by users' addresses can exceed `maxWallet`	ACKNOWLEDGED
03	LOW	`fee` can be set to more than 5% in constructor	ACKNOWLEDGED
04	LOW	Insufficient check for `_timeTillUnlockLiquidity`	ACKNOWLEDGED
05	INFO	Hardcoded values prevents customisation	ACKNOWLEDGED
06	INFO	Adding liquidity has the side effect of enabling trading	ACKNOWLEDGED

2 Findings

2.1 Sage

Sage is a contact that allows users to buy and sell the underlying token. It has similar functions to ERC20, except it does not have the functions `transferFrom`, `approve`, and `allowance`.

The assigned liquidity provider will transfer Ether into the contract to initialize and declare the start of trading, and users can buy or sell the underlying token. When liquidity is added, the liquidity provider can choose a timestamp before they can remove liquidity.

It is paramount to note that **every token will become worthless** once the timestamp is reached and liquidity is removed.

2.1.1 Privileged Functions

- `enableTrading`
- `enableMaxWallet`
- `setTradingFee`
- `setMaxWallet`
- `setFeeCollector`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Users overpay for tokens in buy()
Severity	HIGH SEVERITY
Description	The crux of this issue lies in how fees are included as part of the calculation for a user's received `tokenAmount`. Following is a scenario to illustrate this issue, with the assumption that `fee` is 50% and the starting `reserveETH = 1000 ether`. 1. User calls `buy()` with `msg.value == 200 ether` 2. `ETHafterFee = 100 ETH` as fee is 50% 3. `reserveETH = 1200 ETH` as 1000 + 200 = 1200 4. `tokenAmount = 100/reserveETH * reserveToken = 100/1200 * reserveToken` However, step 4 is incorrect. The fee should not be part of `reserveETH` in this step. The correct calculation for `tokenAmount` at this step should be `100/1100 * reserveToken`. Users receive less tokens than they should, as the amount of `reserveETH` is incorrectly calculated to be higher than it should be. Note that `accruedFeeAmount` is only increased after the calculation for `tokenAmount`. The function `getAmountOut(true)` is based on the value in which the fee is incorrectly added into the calculation and should be fixed too.
Recommendation	Consider incrementing `accruedFeeAmount` before retrieving `reserveETH` using `getReserves()`. As the protocol has already been deployed, we recommend slightly reducing the fees to compensate for the miscalculation.
Resolution	RESOLVED

Issue #02	Tokens held by users' addresses can exceed `maxWallet`
Severity	MEDIUM SEVERITY
Description	The check for whether the balance of `msg.sender` exceeds `maxWallet` is only enforced in `buy()`. Users can always use `buy()` on a different address and then use `transfer()` to transfer the tokens to the wallet used to do the purchasing and bypass the maximum amount declared in maxWallet.
Recommendation	Consider adding the `maxWallet` check to `_transfer`, and remove it from `buy()`.
Resolution	ACKNOWLEDGED

Issue #03	`fee` can be set to more than 5% in constructor
Severity	LOW SEVERITY
Description	There is no `require` check for `fee` in the constructor, unlike in `setTradingFee()`. `fee` can be set to more than the expected 5% within the constructor.
Recommendation	Consider using `setTradingFee()` in the constructor to set the fees to ensure the same check is made.
Resolution	ACKNOWLEDGED

Issue #04	Insufficient check for `_timeTillUnlockLiquidity`
Severity	LOW SEVERITY
Description	A liquidity provider adds liquidity by calling `addLiquidity()` with a chosen timestamp. However, it only checks the lower bound of `_timeTillUnlockLiquidity` and the upper bound is not checked. If `_timeTillUnlockLiquidity` is set to an unreasonably high value, the liquidity provider will not be able to call `removeLiquidity()`.
Recommendation	Consider setting an upper bound for `_timeTillUnlockLiquidity` to a reasonable and expected timestamp. This check should be added to both `addLiquidity()` and `extendLiquidityLock()`.
Resolution	ACKNOWLEDGED

Issue #05	Hardcoded values prevents customisation
Severity	INFORMATIONAL
Description	Several values in the contract are hardcoded, which prevents any kind of customisation by the owner. For example: 1. `totalSupply` is hardcoded to `1,000,000e18`. 2. Owner always receives 10% of `totalSupply`. 3. Initial liquidity is always 90% of `totalSupply`.
Recommendation	Consider allowing owner to choose their own values for the above variables since they are not changeable.
Resolution	ACKNOWLEDGED

Issue #06	Adding liquidity has the side effect of enabling trading
Severity	INFORMATIONAL
Description	When the liquidity provider adds liquidity, trading is enabled at the same time. This seems unnecessary and possibly unintended since owner may not want to enable trading at the same time as when liquidity is added. While the owner can choose to set `tradingEnable` to `false`, some users could have already executed buys.
Recommendation	Consider removing the line `tradingEnable = true` in `addLiquidity()`.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY