



SAFETIN **AUDIT**

SAMURAI FINANCE

March 1st, 2022



TABLE OF CONTENTS

- I. SUMMARY**
- II. OVERVIEW**
- III. FINDINGS**
- IV. CONCLUSION**
- V. DISCLAIMER**

SUMMARY

This report was written for [Samurai Finance](#) in order to find flaws and vulnerabilities in the [Samurai Finance](#) project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and [Samurai Finance](#) Deployment techniques. The auditing process pays special attention to the following considerations:

- ❖ Testing the smart contracts against both common and uncommon attack vectors
- ❖ Assessing the codebase to ensure compliance with current best practices and industry standards
- ❖ Ensuring contract logic meets the specifications and intentions of the client
- ❖ Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- ❖ Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	SAMURAI FINANCE
Description	The Samurai Finance project is an entire ecosystem (forked from Tomb Finance) operating around an algorithmic token attached to the BNB. The project has a complete system of liquidity pools that evolve according to its progress. Users can, depending on the progress of the project, add their BNB to different LPs to obtain rewards in Samurai and Warrior (WARRIOR) according to the project's process.
Platform	Binance Smart Chain
Language	Solidity
Codebase	https://docs.samuraifinance.io/protocol/contracts-and-wallets (the three smart contracts concerned by the audit are WarriorRewardPool, Samurai and Warrior)

FINDINGS SUMMARY

Vulnerability	Total	Resolved
● Critical	0	0
● Major	0	0
● Medium	5	1
● Minor	2	5
● Informational	23	2

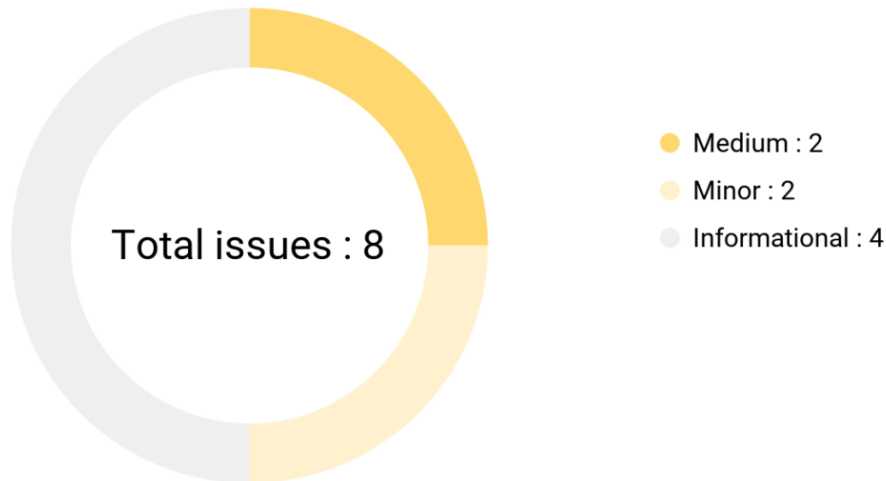
EXECUTIVE SUMMARY

The [Samurai Finance](#) project is an entire ecosystem (forked from Tomb Finance) operating around an algorithmic token attached to the BNB. The project has a complete system of liquidity pools that evolve according to its progress. Users can, depending on the progress of the project, add their BNB to different LPs to obtain rewards in tokens created for the project (samurai, warrior and rhonin).

There have been no major or critical issues related to the codebase and all findings listed here are minor, medium or informational. The main problem is the centralization of privileges which would require a more secure approach.

The codebase would need a more secure approach, especially in error messages, to ensure maintainability and code health in the future.

AUDIT FINDINGS



Code	Title	Severity
CENT-1	Centralization of major privileges	● Medium
THRE-2	Missing threshold in major func	● Medium
THRE-1	Missing threshold in minor func	● Minor
TIME-1	Global use of block.timestamp	● Minor
VAR-1	Using wrong uint	● Informational
MSG-1	Unclear error message	● Informational
MSG-2	Missing error message	● Informational
MSG-3	Error message too long	● Informational

MSG-1 | Wrong error message

Description

Some of the error messages in the smart contract ([samurai.sol](#)) are unclear or badly formulated. In order to optimize error handling and project maintenance, we recommend using the clearest possible error messages.

3 issues of this type have been found in [samurai.sol](#).

5 issues of this type have been found in [warrior.sol](#).

4 issues of this type have been found in [WarriorRewardPool.sol](#).

Recommendation

We recommend replacing these ambiguous messages with clearer ones :

```
//Edited code with clearer error messages for samurai.sol  
//line 170  
require(!autoCalculateTax, "auto tax setting cannot be  
enabled");  
//line 176  
require(!excludedAddresses[_address], "address already  
excluded");  
//l182  
require(excludedAddresses[_address], "address already  
included");  
//Edited code containing clearer error messages for  
warrior.sol
```

//L48 : three unclear messages corrected in the constructor

```
require(_teamFund != address(0), "Teamfund address cannot be 0");
```

```
require(_devFund != address(0), "Devfund address cannot be 0");
```

```
require(_communityFund != address(0), "Communityfund address cannot be 0");
```

//L83

```
require(_devFund != address(0), "Dev fund cannot be zero address");
```

//L134

```
require(_farmingIncentiveFund != address(0), "Cannot distribute to address 0");
```

//Edited code containing clearer error messages for WarriorRewardPool.sol

//L63

```
require(block.timestamp < _poolStartTime, "Pool already started : too late");
```

//L220

```
require(user.amount >= _amount, "Cannot withdraw that much");
```

```
[...]
```

//L265 : two error messages modified in this function

```
require(_token != warrior, "Transfer token is not warrior");
```

```
require(_token != pool.token, "Cannot transfer to the token's pool");
```

MSG-2 | Missing error message

Description

Some require statements of the smart contract ([samurai.sol](#)) guarantee validity of conditions but do not throw any error message if these conditions are not met. We recommend documenting errors as best as possible in order to improve their handling and the user interface.

2 issues of this type have been found in [samurai.sol](#)

Recommendation

We recommend adding an error message to the following require statements :

```
//Edited code including missing error messages  
//line 105  
require(_value > taxTiersTwaps[_index - 1], "Taxtierstwap in  
ascending order");  
  
//l108  
require(_value < taxTiersTwaps[_index + 1], "Taxtierstwap in  
ascending order")
```

MSG-3 | Too long error messages

Description

The smart contract (samurai.sol) has some error messages that are too long. The industry standards specify error messages must have a maximal length of 32 bytes. We recommend having the shortest possible error messages to optimize gas costs (see github.com/ethereum/solidity/issues/4588) and improve error handling. 6 issues of this type have been found in [samurai.sol](#).

Recommendation

We recommend shortening these error messages :

```
//Edited code with shorted messages  
//L67  
require(isOperator() || taxOffice == msg.sender, "Caller  
must be operator/taxoffice");  
//L78  
require(_taxCollectorAddress != address(0), "tax  
collector cannot be address 0");  
//L103  
require(_index < getTaxTiersTwapsCount(), "Index must be  
under tax tier count");  
//L116  
require(_index < getTaxTiersRatesCount(), "Index must be  
under tax tier count");  
  
//L158
```

```
require(_taxOffice != address(0), "tax office address != 0  
address");  
//l164  
require(_taxCollectorAddress != address(0), "tax collector  
address != 0 address");
```

VAR-1 | Using wrong uint type

Description

Some variables in the smart contract ([samurai.sol](#)) are of too high a uint type. For example, some variables are of type uint256 while their logical maximum is lower than the maximum uint16. 4 issues of this type have been found in [samurai.sol](#).

Recommendation

We recommend using as low uint types as possible to optimize memory allocation :

```
//Edited code including uint type modifications
//L55
uint16[] public taxTiersRates = [2000, 1900, 1800, 1700,
1600, 1500, 1500, 1500, 1500, 1400, 900, 400, 200, 100];
//L44
uint256 public taxRate;
//L89
function getTaxTiersTwapsCount() public view returns
(uint8 count) {
    return taxTiersTwaps.length;
}
//L93
function getTaxTiersRatesCount() public view returns
(uint8 count) {
    return taxTiersRates.length;
}
```

THRE-1 | Missing threshold in tax tiers setting function

Description

The tax tier rate setting function (setTaxTiersRate from [samurai.sol](#)) does not implement a tax threshold. This means that tax office can potentially set a tax tier rate of 10,000 (which means 100%, this should not be possible). However, this is not a critical problem as a check that the tax is below 10,000 is implemented in updateTaxRate function but it can block the automatic update of taxes. 1 error of this type has been found in [samurai.sol](#).

Recommendation

We recommend adding a threshold to this function :

```
//Edited code featuring a threshold  
//L114  
function setTaxTiersRate(uint8 _index, uint256 _value)  
public onlyTaxOffice returns (bool) {  
    require(_index >= 0, "Index has to be higher than  
0");  
    require(_index < getTaxTiersRatesCount(), "Index  
has to lower than count of tax tiers");  
    require(_value < 10000, "Tax rate cannot be  
100%");  
    taxTiersRates[_index] = _value;  
    return true;  
}
```

THRE-2a | Missing threshold in minting function

Description

Minting function (mint from [samurai.sol](#)) does not implement any threshold. The operator can then mint a potentially infinite amount of tokens.

1 error of this type has been found in [samurai.sol](#).

Recommendation

We recommend adding a threshold (as example : set a maximum minting threshold of 10 tokens) to the function to avoid this problem or set the function as internal since it is used elsewhere in the contract (but you have to assume that the operator does not have the right to mint as much as it wants to whom it wants) :

```
function mint(address recipient_, uint256 amount_)  
  
public onlyOperator returns (bool) {  
  
    uint256 balanceBefore = balanceOf(recipient_);  
    require(amount < 10, "cannot mint that much");  
    _mint(recipient_, amount_);  
    uint256 balanceAfter = balanceOf(recipient_);  
  
    return balanceAfter > balanceBefore;  
}
```

THRE-2b | Missing threshold in burning function

Description

Operator's burning function (burnFrom from samurai.sol) does not feature any threshold or cooldown. The operator can burn as many tokens as he wants from any account without limit.* *Even if a burnThreshold variable is defined and used in transfer functions, the operator still can directly burn money by calling the burnFrom function. 1 error of this type has been found in [samurai.sol](#).

Recommendation

We recommend adding a threshold (as example : set a maximum burning threshold of 10 tokens) to the function to avoid this problem or set the function as internal since it is used elsewhere in the contract (but you have to assume that the operator does not have the right to burn as much as he wants from any accounts) :

```
//Edited code including threshold  
//L205  
function burnFrom(address account, uint256 amount)  
  
public override onlyOperator {  
    require(amount < 10, "Cannot burn more than 10  
token");  
    super.burnFrom(account, amount);  
}
```

CENT-1 | Centralization of major privileges

Description

The operator of the smart contract ([samurai.sol](#)) has major privileges over it (he can burn any amount from any account, mint any amount to any account, change the tax office address and then modify tax collector address)*. The devfund address of the smart contract ([warrior.sol](#)) has major privileges over it (he can set a new treasury fund and team fund address and potentially rug money from these addresses).

This can be a problem, in the case of a hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

1 error of this type has been found in [samurai.sol](#).

1 error of this type has been found in [warrior.sol](#).

*This list is not exhaustive but presents the most sensitive points

Recommendation

We recommend at least to use a multi-sig wallet as owner address, and at best to establish a community governance protocol to avoid such centralisation.

For more information see :

<https://solidity-by-example.org/app/multi-sig-wallet/>

GLOBAL SECURITY WARNINGS

These are safety issues for the whole project. They are not necessarily critical problems but they are inherent in the structure of the project itself. Potential attack vectors for these security problems should be monitored.

CENT-1 | Global SPOF (Single Point Of Failure)

Description

The project's smart contracts often have a problem of centralized privileges. The [operator](#) and [taxoffice](#) system in particular can be subject to attack.

Recommendation

To address this security issue we recommend using a multi-sig wallet, establishing secure project administration protocols and strengthening the security of project administrators.

TIME-1 | Global use of block.timestamp

Description

We fully understand that the project works with specific deadlines and requires the use of `block.timestamp`. Nevertheless, you should be aware of the risks of using `block.timestamp` : the miner can partially manipulate this variable, and influence the logic of the contract. In particular, it can slightly manipulate the amounts claimable by the different funds. For more information, see :

<https://cryptomarketpool.com/block-timestamp-manipulation-attack/>

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without [Safetin'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 considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts [Safetin](#) 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

Safetin security assessment 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. Safetin's position is that each company and individual are responsible for their own due diligence and continuous security. Safetin'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 fun.