



**DEFY**  
**FINANCE**

 @DEFIAUDITS

# SMART CONTRACT SECURITY AUDIT OF

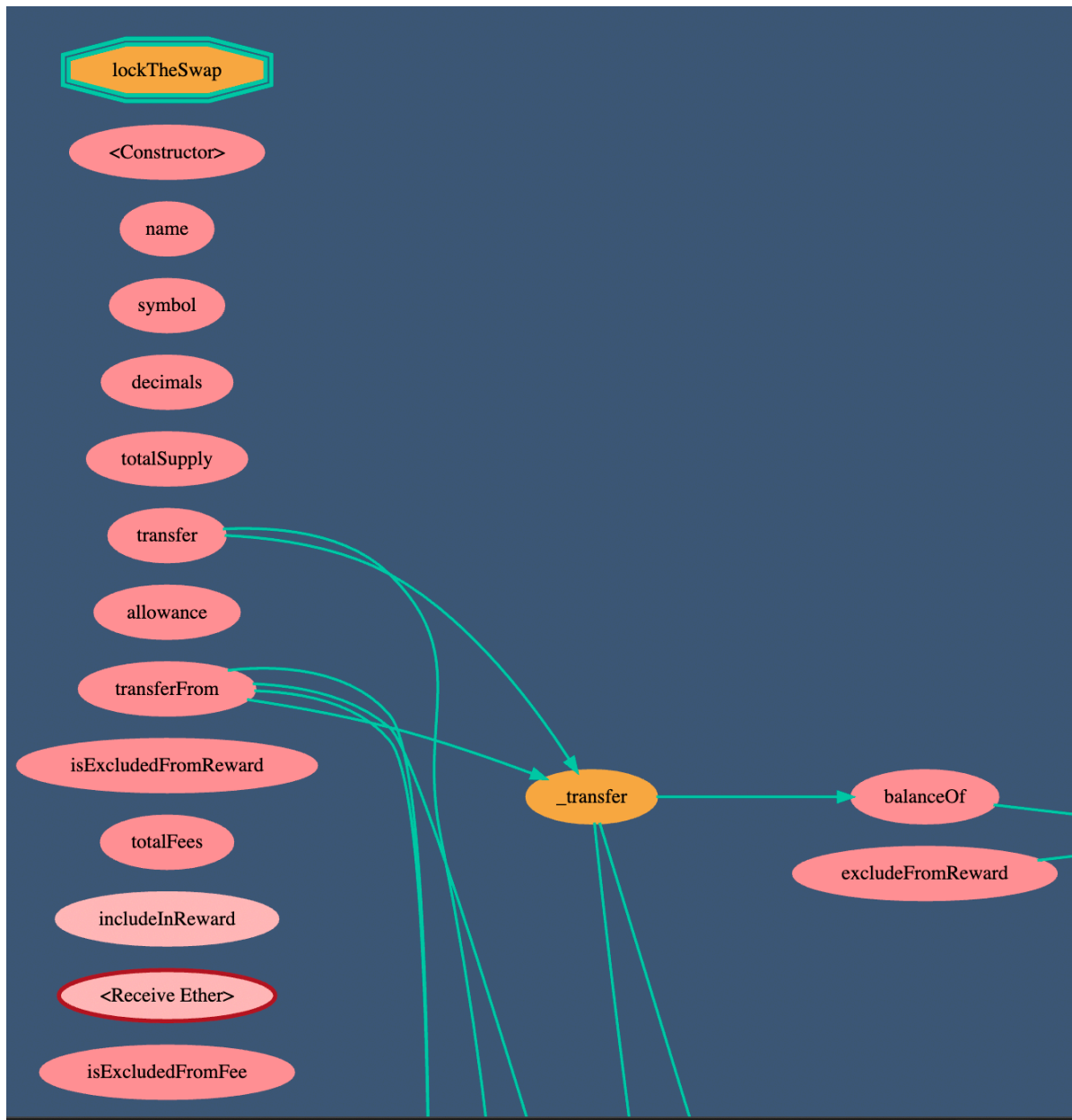


METAFINTEC

SMART CONTRACT AUDIT | TEAM KYC | PROJECT EVALUATION

MAKING DEFI SAFER | MADE IN CANADA 

FIGURE 1. VISUALIZATION OF METAFINTEC CONTRACT



About Defy Audits:

4

Project Details

5

Address

5

KYC Summary:

5

KYC Goals: Presale , then ICO , then Public sale

5

Audit Summary

6

Audit Methodology:

6

Part I: Risk Assessment

7

Summary

7

Project Overview

7

Vulnerability Assessment:

8

Centralization Risk Assessment

13

Part II: Concluding Remarks and Recommendations

13

Were there security vulnerabilities found?

14

Where the goals of the project met by the contract?

14

Disclaimer:

14

# About Defy Audits:

We are a growing community of Solidity auditors, solidity and blockchain protocol developers who provide chain agnostic auditing services for smart contracts and DeFi applications. We perform both manual and automated testing of smart contracts. A number of open source and proprietary tools to do audits. An incomplete list of tools used in audits include the following:


- ➡ Mythx
- ➡ Slither
- ➡ Echinda Fuzzer to aid Manual testing
- ➡ Proprietary Greybox Fuzzer to aid Manual testing
- ➡ Hardhat / Python
- ➡ Rinkerby, Goril for on chain testing
- ➡ Human source code review

## Project Details

---

Platform	Binance Smart Chain
Version	0.6.12 locked
Language	Solidity
Team.	Metafintec
# Lines:	1225

## Address

 0x55531463333C3A8B503E3839AD6a03A29A958D97

## KYC Summary:

Interview: Metafintec is building a New age Neo Banking solution and connecting it with blockchain technology to create a confluence of Banking facilities with Crypto currency and Forex services. The team plans to launch Crypto Payment gateway services for merchants, an NFT marketplace and an Auction marketplace at a later date.

---

KYC Goals: Presale , then ICO , then Public sale

## Audit Summary

Defy Finance team has performed an objective line-by-line analysis of the solidity smart contract including on-chain tests, manual analysis aided by fuzzing and static analysis. Based on evidence gathered at the time of audit completion, we can make the following risk assessment:

### According to the smart contract audit:

- ❖ **Metafintic** smart contract source code has **LOW THREAT LEVEL**.
- ❖ **Metafintic** has **PASSED** the smart contract audit.
- ❖ **Metafintic** has **PASSED** KYC verification.

## Audit Methodology:

1. **Common vulnerabilities:** We performed a static analysis, vulnerability assessment for common vulnerabilities including *overflow, underflow, callstack depth attack, parity multis bug, timestamp dependency and re-entrancy*.

2. **Manual Review and On-chain testing:** We performed manual source code review aided by grey box fuzzing. We performed on-chain testing against common attack vectors for specific functions that have a higher attack probability based on the latest statistics we have gathered.

3. **Overall Risk Assessment:** We interviewed the team and made an overall risk assessment based on similar contracts and the intended purpose and goals of the contract.


We report analytics on critical, major and minor security vulnerabilities found and provide a holistic assessment of the overall risk of the contract. We provided a holistic risk assessment in the conclusion of this document.

# Part I: Risk Assessment









## Summary

Auditing Firm	<b>Defy Finance</b>
Auditing Methodology	<b>Defy Finance Standard 1.0</b>
Platform	<b>Solidity</b>
Audit Check (Mandatory)	<b>Static, Manual</b>
Request Date	<b>Jan 23, 2022</b>
Final Report Date	<b>Jan 26, 2022</b>

## Project Overview

Language	<b>Solidity</b>
Contract	 0x55531463333C3A8B503E3839AD6a03A29A958D97
Source Code	<a href="https://github.com/yysmohit/metafintec/blob/mainmetafintec.sol">https://github.com/yysmohit/metafintec/blob/mainmetafintec.sol</a>

## Vulnerability Assessment:

- SWC-1 HoneyPot: Pass 
- SWC-2 Integer Underflow: Pass 
- SWC-3 Integer Overflow: Pass 
- SWC-4 Parity Multisig Bug: Pass 
- SWC-5 Callstack Depth Attack: Pass 
- SWC-6 Transaction-Ordering Dependency: Pass 
- SWC-7 Timestamp Dependency: Pass 
- SWC-8 Re-Entrancy: Pass 

### Severity Risk Levels:

**Critical:** Issues marked critical need to be addressed immediately as they pose an immediate threat to contract security

**Major:** Need to be acknowledged as they could pose a medium threat to security

**Minor:** A theoretical or long-term risk, may not pose practical security risk

**Informational:** Negligible risk to security but may be

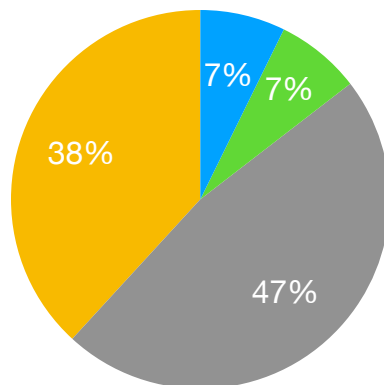


## Smart Contract – Similar Contracts

Address	Similarity Score
0x19cc7ddcbee48843e5e27d0ae5ef3b72c93ab21b	0.9
0xab29308c4937e629141f5d8a227b6127e6dcd049	0.9
0x29632c10d76bd1aed6b1524bf5bde4c992610670	0.9

## Smart Contract – Static Analysis

Minor Informational Major  
Critical







Function	Description	Line(s) affected	Severity
<b>Lack of zero check(s)</b>	Function fails to check for zero value	Metafintec.setTeamWallet(address).newWallet (contracts/MetaFint.sol#1196)  Metafintec.setCharityWallet(address).newWallet (contracts/MetaFint.sol#1199)  Metafintec.setPartnershipWallet(address).newWallet (contracts/MetaFint.sol#1202)  Metafintec.setExchangeWallet(address).newWallet (contracts/MetaFint.sol#1205)	Minor
<b>Emitting event(s)</b>	Events emitted	Event emitted after the call(s): - Transfer(sender,recipient,tTransferAmount) (contracts/MetaFint.sol#1137) - _tokenTransfer(from,to,amount) (contracts/MetaFint.sol#1026) - Transfer(sender,recipient,tTransferAmount) (contracts/MetaFint.sol#1147) - _tokenTransfer(from,to,amount) (contracts/MetaFint.sol#1026) - Transfer(sender,recipient,tTransferAmount) (contracts/MetaFint.sol#1157) - _tokenTransfer(from,to,amount) (contracts/MetaFint.sol#1026) - Transfer(sender,recipient,tTransferAmount) (contracts/MetaFint.sol#897) - _tokenTransfer(from,to,amount) (contracts/MetaFint.sol#1026)	Informational
<b>External Calls</b>	External calls need to be investigated for re-entrancy	uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (contracts/MetaFint.sol#1061-1067) External calls sending eth: - swapAndLiquify(contractTokenBalance) (contracts/MetaFint.sol#1022) - uniswapV2Router.addLiquidityETH({value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (contracts/MetaFint.sol#1075-1082)	Informational
<b>Redundant Code</b>	Duplicate expressions	Redundant expression "this" (contracts/MetaFint.sol#246)" inContext (contracts/MetaFint.sol#240-249) Reference: <a href="https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements">https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements</a>	Minor

Function	Description	Line(s) affected	Severity
<b>Dead Code</b>	It is recommended to remove unused or “dead” code paths.	SafeMath.mod(uint256,uint256) (contracts/ MetaFint.sol#218-220) is never used and should be removed Metafintec.allowance(address, address).owner (contracts/ MetaFint.sol#807) shadows: - Ownable.owner() (contracts/ MetaFint.sol#421-423) (function) Metafintec._approve(address, address,uint256).owner (contracts/MetaFint.sol#990) shadows: - Ownable.owner() (contracts/ MetaFint.sol#421-423) (function) Reference: <a href="https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing">https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing</a>	Minor
<b>Test Coverage</b>	Are there adequate test cases to cover business logic?	We used a generic test suite but recommend more specific test coverage.	Informational
<b>Shadow Variables</b>	Shadow variables are variables that can cause name conflict and unintended side effects	Metafintec.allowance(address, address).owner (contracts/ MetaFint.sol#807) shadows: - Ownable.owner() (contracts/ MetaFint.sol#421-423) (function)	Minor
<b>Gas Optimization</b>	Are there infinite loops that exceed block limits? Are there gas limit optimizations? Could the code be made smaller to reduce gas fees on deployment?	Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off revert strings, or using libraries.	Informational

## Smart Contract – Manual Analysis

We performed a manual code review of the following functions using fuzzing and on-chain tests to aid in vulnerability discovery. We also reviewed for gas optimizations, infinite loops that exceed block gas limits, external functions that should be internal and that each function is in line with . A pass indicates the function has passed manual inspection and is in line with intended goal(s) of the smart contract.

Function	Description	Functions tested	Pass/Fail
<b>Total Supply</b>	Supply of account balance of owners account	tokenSupply	Pass 
<b>Transfer</b>	used to transfer the tokens from the owner of the token to some other address	Transfer	Pass 
<b>Approve</b>	used by an address to approve the spending of a particular amount tokens by a particular address	approve	Pass 

Function	Description	Functions tested	Pass/Fail
<code>_tokenTransfer</code>	Used to send transfers to burn and marketing wallets.	<code>treasuryAmt</code> <code>marketingAmt</code> <code>partnershipAmt</code> <code>exchangeAmt</code>	Pass 

## Centralization Risk Assessment

The following functions are marked privileged (by owner) and owner can modify them including a Team, charity, partnership wallet and tax and fee structure of the smart contract. A list of these functions are below and based on additional tests pose no additional risk to security.

```

excludeFromFee(address account)
includeInFee(address account)
blacklistAddress(address account, bool value)
setLiquidityFee(uint256 newFee)
setTaxFee(uint256 newFee)
setTeamFee(uint256 newFee)
setCharityFee(uint256 newFee)setPartnershipFee(uint256 newFee)
setExchangeFee(uint256 newFee)
setTeamWallet(address newWallet)
setCharityWallet(address newWallet)
setPartnershipWallet(address newWallet)
setExchangeWallet(address newWallet)setMaxTxPercent(uint256
maxTxPercent)setSwapAndLiquifyEnabled

```

## Part II: Concluding Remarks and Recommendations

### **Were there security vulnerabilities found?**

No critical or major issues were reported during the course of the audit. This indicates the contract meets a high standard of security and low threat risk. We only recommend some minor optimizations such as removing redundant or dead code.

### **Where the goals of the project met by the contract?**

Additionally, there were no discrepancies found between the intended use of the contract and the implementation. We interviewed the Metafintec team, investigated similar contracts and searched social media for additional information as part of an internal KYC process and reported no additional risk factors.

### **Disclaimer:**

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, financial advice or legal compliance. This document should not be used in any way to make decisions around investment or involvement with any particular project.