
# Smart Contract - Audit Report Example

*Scope details:* Example SC Audit of an ERC-20 Token

*Audited project:*
Redlight Node District ($Playmates)
*Deployed address:*
0x490bf3ABcAb1fB5c88533d850F2a8d6D38298465
*Main contact:*
https://twitter.com/redlight
*Blockchain*:
Avalanche Mainnet
*Website*:
https://redlight.finance/

*Audit date:* August 14th 2022

# Disclaimer

FnR Audits does not guarantee that the Smart Contract Audit will identify all instances of security vulnerabilities or other related issues. This audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model or the bug-free status of the contracts. This report could include false positives, false negatives, and other unpredictable results.

Code recommendations are not intended to be comprehensive or applicable in all situations and FnR Audits disclaims liability over any consequences that may occur as a result from any and all code recommendations. This audit is not an endorsement of the Redlight protocol. This audit should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

FnR Audits assumes no responsibility for errors, omissions, or damages resulting from the use of the provided information. We recommend that the Redlight protocol put in place a bug bounty program to encourage further analysis of the smart contract by other third parties. By reading this report and consuming the information within this Audit, you agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

*Andeh*

FnR Audits

# Audit Context

FnR audits was commissioned by _____ Protocol to perform an audit of the following:

- ERC-20 Token Contract

Both Parties agreed on the following audit objectives:

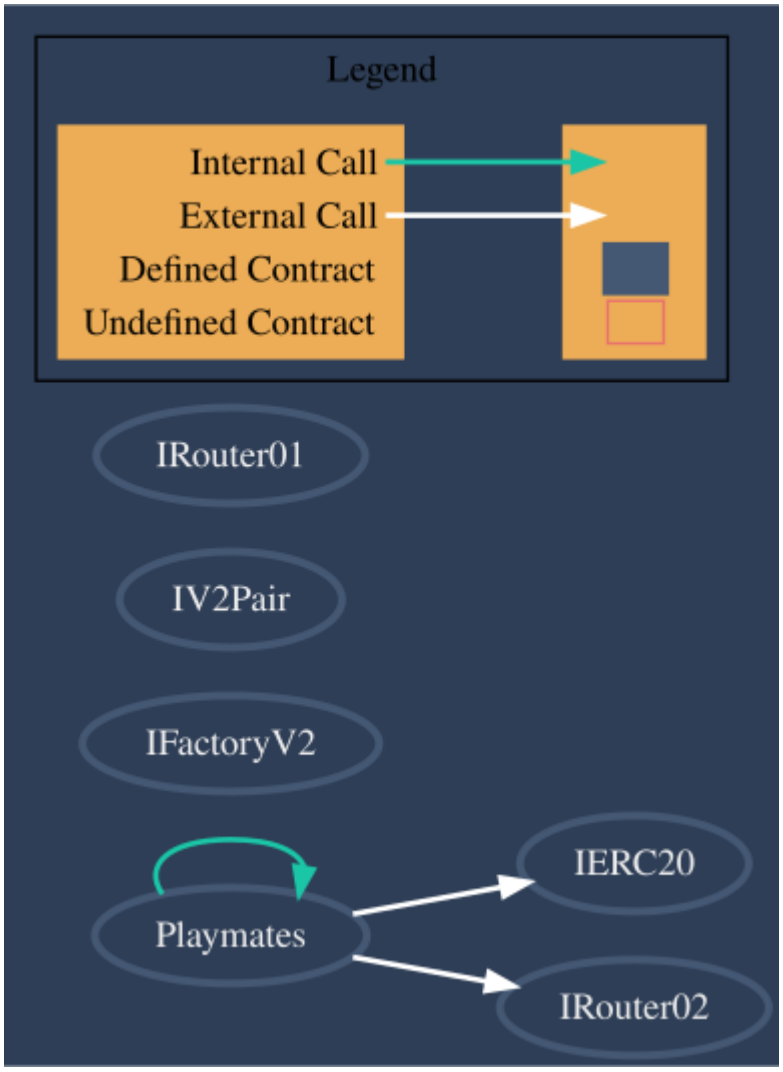
- Ensure the smart contract functions as intended
- Identify potential security issues with the smart contract

The content of this report should be used to understand the risk exposure of the smart contract, and as a guide to better secure them by remediating the issues that were identified.

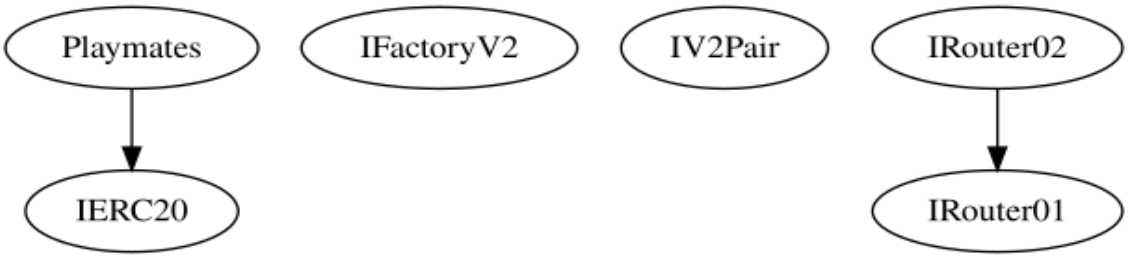
By reading the contents of this audit, you agree to having read and agreed with the contents in the Disclaimer is printed above on Page 2.

# Contract Broad-level Details

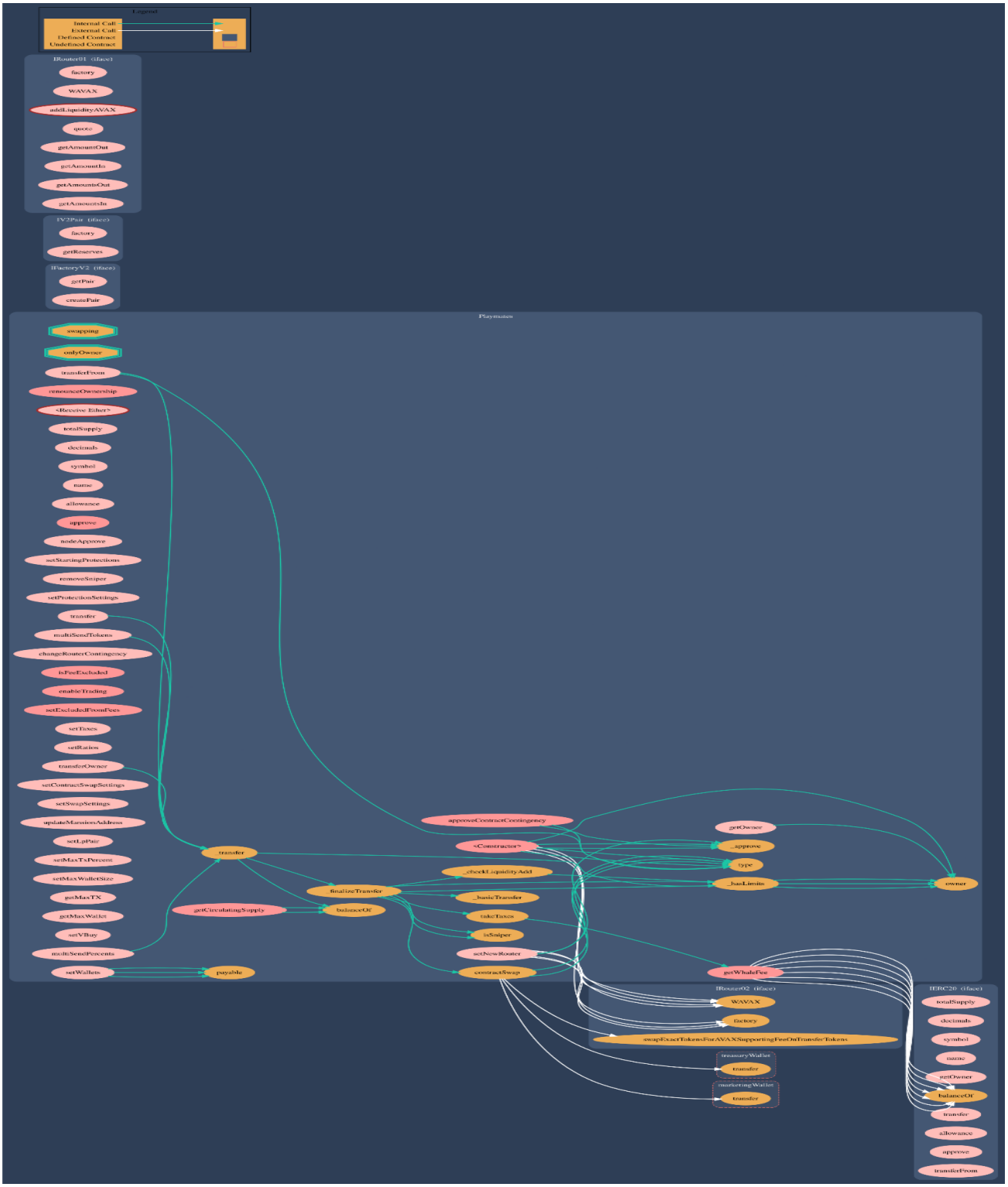
## Visual Depiction of Contract Interaction




## Contract Inheritance Graph



# Visual Depiction of Contract Function Calls



# Contract Issues Checking Status

**PASS** : no issue was found

**ISSUE** : at least one issue was found

**INFO/OPTIM** : informative findings or gas optimization opportunities

**N/A** : that control was not applicable due to the nature of the code

| # | Issue description | Checking status | Issues IDs |
|---|---|---|---|
| **1** | Compiler: version, warnings | **ISSUE** | L01, L02 |
| **2** | Reentrancy, vulnerability to transaction reordering | **ISSUE** | L04 |
| **3** | Third party dependencies | **ISSUE** | M01 |
| **4** | Randomness / Timestamp dependency | **PASS** | / |
| **5** | Overflow underflow | **PASS** | / |
| **6** | DOS with revert | **ISSUE** | L03 |
| **7** | DOS with block gas limit | **ISSUE** | L07 |
| **8** | Methods execution permissions and privileges | **PASS** | / |
| **9** | Economic model<br>If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc | **N/A** | |
| **10** | Private user data leaks | **N/A** | / |
| **11** | Critical action management and event logs | **ISSUE** | L05 |
| **12** | Scoping and declaration<br>Shadowed variables / functions | **INFO/OPTIM** | INF02 |
| **13** | Uninitialized storage variables | **N/A** | |
| **14** | Arithmetic accuracy | **N/A** | |

| | | | |
|---|---|---|---|
| **15** | Open Zeppelin modules | **ISSUE** | L06 |
| **16** | Fallback and receive functions | **PASS** | / |
| **17** | Input validation | **ISSUE** | L09 |
| **18** | Impact of sending native assets | **PASS** | / |
| **19** | Management of function arguments and return values | **ISSUE** | L10 |
| **20** | Usage of tx.origin and selfdestruct | **PASS** | / |
| **21** | Dead code | **INFO/OPTIM** | INF03 |
| **22** | Revert, assert, uncovered conditional paths | **ISSUE** | L08 |
| **23** | Other best practices | **INFO/OPTIM** | INF01, INF04 |

# Security Issue Summary

## High Severity Issues

No high severity issues found

## Medium Severity Issues

[M01] The contractSwap function is vulnerable to price manipulation in case of flash loans on the liquidity pair.

1. **Issue**: The contractSwap function is vulnerable to price manipulation in case of flash loans on the liquidity pair. If two different liquidity pairs exist, or only one exists but isn't protected against reentrancy attacks, it could impact the price used during a transfer.
2. **Consequences:** A user could make a flash loan on the liquidity pair and then make a transfer of Playmates. It would modify the price of the pool. Combining that action with manually transferring native tokens to the auditing contract could disrupt the contract_swap function.
3. **Recommendation:** The audited contract could add a storage attribute "lastSeenPrice" and would update it at each contractSwap successful call. Then, when interacting with the dexRouter, it should make sure that the price is not too far from lastSeenPrice.
4. **Team's comments**: [No feedback, this report is an example]

## Low Severity Issues

[L01] Compiler version with floating pragma

1. **Issue**: The compiler version used is "pragma solidity >=0.6.0 <0.9.0;"
2. **Consequence:** Making it possible to use different compiler versions for the same code can lead to undetected vulnerabilities.
3. **Recommendation:** Use an exact compiler version, if possible without known vulnerability.
4. **Team's comments**: [No feedback, this report is an example]

[L02] Deployed contract uses a compiler version with known warnings

1. **Issue:** The contract was deployed with solc v0.8.11 which has four low to very-low severity warnings.
2. **Consequence:** These vulnerabilities could be exploited in some situations.
3. **Recommendation:** Make sure that these warnings cannot lead to exploits in the current code. Also, use a compiler version that is not linked to warnings.
4. **Team's comments**: [No feedback, this report is an example]

[L03] In some cases, modifying marketingWallet, treasuryWallet or the router could block the token transfers by reverting transactions.

1. **Issue:** The function contractSwap() can be called during transfers and it sends native tokens to treasuryWallet and marketingWallet.
2. **Consequence:** If at least one of these addresses is a contract with no fallback or receive function, it could lead to a transaction reversion each time contractSwap is called during a transfer.
3. **Recommendation:** First, the owner should be cautious when he modifies these addresses. Also, using a function that returns a status when transferring native tokens would strengthen that portion of code.
4. **Team's comments**: [No feedback, this report is an example]

[L04] In some cases, modifying marketingWallet, treasuryWallet or the router could lead to reentrancy attacks.

1. **Issue**: The function _finalizeTransfer could lead to reentrancy attacks that could drain the contract's balance of native token if the owner changes at least one of these 3 addresses by a malicious one.
2. **Consequence**: A malicious contract could make other transfers until the balance of the audited contract reaches a given threshold of balance of native tokens.
3. **Recommendation**: First, the owner should be cautious when he modifies these addresses. Also, either add a modifier "ReentrancyGuard" or use the Check Effects Interactions pattern.
4. **Team's comments**: [No feedback, this report is an example]

[L05] Critical modifications of the contract's state are not easily traceable by users.

1. **Issue**: The owner of the contract can make critical modifications to the contract but these are not always logged with events.
2. **Consequence:** It makes it more difficult for users to monitor the modifications of the contract. As well, in case of unexpected modifications during an exploit, the reaction could be delayed by lack of detection tools.
3. **Recommendation**: Consider adding relevant event emission in functions where important parameters can be changed (ex: in setTaxes, setRatios, setWallets).
4. **Team's comments**: [No feedback, this report is an example]

[L06] The current contract does not leverage on existing standard and secure contracts.

1. **Issue:** The contract is using an IERC20 interface but implements it totally. It is also the case of a reimplementation of Ownable. Also, a function getOwner() is added to the IERC20 interface which is not a common practice. Another example is the fact the balances of users are stored in the variable _tOwned instead of the standardized balances.
2. **Consequence:** Manually rewriting contracts that were standardized and secured by teams of experts prevents from benefiting and could lead to undetected vulnerabilities.
3. **Recommendation**: Import an official standardized contract, for example from OpenZeppelin, extend it with your contract and override only the functions that should have a different behavior.
4. **Team's comments**: [No feedback, this report is an example]

[L07] The functions multiSendTokens and multiSendPercents could reach the block gas limit when called with big arrays.

1. **Issue**: these functions can be called with an unchecked number of addresses..
2. **Consequence**: if someone calls one of these functions with too many addresses in the same transaction, the execution will consume too much gas and may ultimately revert.
3. **Recommendation**: allow a maximum limit of addresses per call.
4. **Team's comments**: [No feedback, this report is an example]

[L08] One require statement in changeRouterContingency is defined without any revert message.

1. **Issue:** These functions can be called with an unchecked number of addresses.
2. **Consequence**: if someone calls one of these functions with too many addresses in the same transaction, the execution will consume too much gas and may ultimately revert.
3. **Recommendation**: allow a maximum limit of addresses per call.
4. **Team's comments:** [No feedback, this report is an example]

[L09] There is no threshold in the function setRatios.

1. **Issue**: The owner can update the ratios used to compute the transfer fees. However, there is no upper threshold so all value can be set.
2. **Consequence**: The owner could make a mistake, or he could set very high ratios.
3. **Recommendation**: add a check of the thresholds.
4. **Team's comments**: [No feedback, this report is an example]

[L10] The function contractSwap ignores the value returned by dexRouter.addLiquidityAVAX.

1. **Issue:** The function contractSwap ignores the value returned by dexRouter.addLiquidityAVAX.
2. **Consequence:** The function could fail silently without any reaction from the audited contract..
3. **Recommendation**: Check the returned value of external calls.
4. **Team's comments**: [No feedback, this report is an example]

# Info - Optimizations

[INF01] There are two redundant functions returning the same value.

1. **Issue**: both functions getOwner() and owner() return the same value _owner.
2. **Consequence**: the code is more complex and its size is bigger, meaning higher costs to deploy it.
3. **Recommendation**: use only one of the two functions.
4. **Team's comments:** [No feedback, this report is an example]

[INF02] Some attributes could be declared as constant, and their visibility should be explicit.

1. **Issue**: in the declaration part of the contract, some variables are not expected to be modified like zero_address or dead_address, others have a visibility that is not explicitly defined (like vBuy1 to vBuy4).
2. **Consequence**: such settings can increase the size of the code, leading to higher gas cost to deploy the contract.
3. **Recommendation**: for all attributes, add an explicit visibility and a constant keyword if its value is never modified.
4. **Team's comments**: [No feedback, this report is an example]

[INF03] The IV2Pair Interface is defined but not used.

1. **Issue**: the IV2Pair Interface is defined but not used.
2. **Consequence**: it adds code to the deployed code, which will increase the cost to deploy the contract.
3. **Recommendation**: Remove unused or dead code.
4. **Team's comments:** [No feedback, this report is an example]

[INF04] The attributes of the contract could be reordered to save gas.

1. **Issue**: the order of declaration of the attributes is not optimal.

2. **Consequence**: the compiler can put several small types of attributes inside the same memory slot to save memory space. But it does this in the order of declaration. It adds gas costs during the contract deployment that could be avoided.
3. **Recommendation**: Reorder the declaration of attributes to get the optimal order.
4. **Team's comments**: [No feedback, this report is an example]

## Owner privileges

- Owner can change setRatios() to alter transfer fees.
- Owner can make critical modifications to the contract with minimal event logs.

# Conclusion

15 issues were identified (0 with High severity, 1 with Medium severity, 10 with Low Severity and 4 are only informative or could lead to gas optimization) and the owner has several privileges.
The team fixed _ of them, acknowledged _ of them and denied _ of them.

The Playmates ERC-20 Contract does not present with critical or high-level security issues at the time of this audit. Several issues around modification and event logs may be incidentally or maliciously altered by the owner and result in vulnerability.

The Playmates team was not informed about this audit due to the nature of it being an example, however we encourage projects we work with to work closely with us and implement / address changes that are recommended during the audit process. We encourage teams to review the entire audit document and consider all recommendations including issues and potential code optimizations.

*Please check the disclaimer above on Page 2 and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the team.*