SECURITY AUDIT OF

WORLD OF METASEA SMART CONTRACTS

Public Report

Dec 17, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Dec 17, 2021. We would like to thank the World Of MetaSea for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the World Of MetaSea Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified one vulnerable issue in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About World Of MetaSea Smart Contracts

MetaSea - The Amazing Adventure, a story that revolves around an exciting journey, with unexpected unimaginable as well as a series of thorns and challenges of Professor McLean, a scientist specializing in the study of marine species, with the loyal Servant Briant, a guy who is passionate about the ocean...

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the token vesting and staking part of the World Of MetaSea Smart Contracts.

It was conducted on commit 8cafc4d7be83bbc3f72e8be69c5676687b89de63 from git repository *https://github.com/worldmetasea/metasea-contracts*.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The World Of MetaSea Smart Contracts was written in Solidity language, with the required version to be ^0.8.8.

2.1.1. MetaSea Token contract

This table lists some properties of the audited World Of MetaSea Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Decimals	6
Max Supply	100,000,000,000 (x10^6) Note: the number of decimals is 6, so the total representation token will be 100,000,000,000 or 100 billion.

Table 2. The World Of MetaSea Smart Contracts properties

MetaSea Token contract extends ERC20Burnable, ERC20Pausable, ERC20Capped and AccessControlEnumerable contracts. With AccessControlEnumerable, by default, contract deployer has ADMIN, MINTER and PAUSER roles. ADMIN can set any role for anyone.

PAUSER can pause/unpause contract using ERC20Pausable contract, users can only transfer unlocked tokens and only when contract is not paused. MINTER can mint tokens but the amount of minted tokens can not pass max supply due to ERC20Capped.

2.1.2. TokensSale contract

This is the token sale contract in the World Of MetaSea Smart Contracts, which extends AccessControlEnumerable contract. With AccessControlEnumerable, by default, contract deployer has ADMIN and MAINTAINER roles. ADMIN can set any role for anyone.

This contract is used for selling MetaSea tokens. MAINTAINER can add/update BatchSale, add addresses to BatchSale whitelist, allow only whitelisted addresses or all addresses to buy a BatchSale and update vesting plan for it.

2.1.3. TokensVesting contract

This is the token vesting contract in the World Of MetaSea Smart Contracts, which extends AccessControlEnumerable contract. With AccessControlEnumerable, by default, contract deployer has ADMIN and MAINTAINER roles. ADMIN can set any role for anyone.

This contract is used for vesting tokens after buying tokens with TokensSale. When vesting schedule time comes, either users can claim their tokens or MAINTAINER can call contract to send tokens to them. MAINTAINER can also revoke permission to vest tokens of any address. If an address is revoked, it can not claim tokens anymore.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. TokenSale.sol - Wrong checking in total percentage condition LOW

The total percentage checking statement in the updateVestingPlan function of the TokenSale.sol contract is incorrect. The expression 100**percentageDecimals_ should be 10**percentageDecimals_ instead.

```
function updateVestingPlan(
    uint256 batchNumber_,
    uint256 percentageDecimals_,
    uint256 tgePercentage_,
    uint256 finalPercentage_,
    uint256 basis_,
    uint256 cliff_,
    uint256 duration_,
    uint8 participant_
)
    external
    onlyRole(MAINTAINER_ROLE)
    batchExisted(batchNumber_)
    onlySupportedParticipant(participant_)
{
    require(
        tgePercentage_ + finalPercentage_ <= 100 * 100**percentageDecimal…
  s_,
        "TokensSale: bad args"
    );
    // ...
```

RECOMMENDATION

Update the updateVestingPlan function as below.

```
function updateVestingPlan(
    uint256 batchNumber_,
    uint256 percentageDecimals_,
    uint256 tgePercentage_,
    uint256 finalPercentage_,
    uint256 basis_,
    uint256 cliff_,
    uint256 duration_,
    uint8 participant_
)
    external
    onlyRole(MAINTAINER_ROLE)
    batchExisted(batchNumber_)
    onlySupportedParticipant(participant_)
{
    require(
        tgePercentage_ + finalPercentage_ <= 100 * 10**percentageDecimals…
  _,
        "TokensSale: bad args"
    );
    // ...
```

UPDATES

- *Dec 17, 2021*: This recommendation has been acknowledged by the World Of MetaSea team.

2.2.2. TokenSale.sol - Hardcoded values in the constructor function INFORMATIVE

In the constructor function of the contract TokenSale, the Participant enums should be used instead of hardcoding the integer value as below.

```
constructor(address tokenVestingAddress_) {
    _updateTokensVesting(tokenVestingAddress_);
    _supportedParticipants.add(1); // seeding
    _supportedParticipants.add(2); // private sale
    _supportedParticipants.add(3); // public sale

    _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
    _setupRole(MAINTAINER_ROLE, _msgSender());
}
enum Participant {
    Unknown,
    Seeding,
    PrivateSale,
    PublicSale,
    Team,
    DeFi,
    Marketing,
    Reserve,
    GameIncentives,
    Liquidity,
    Development,
    OutOfRange
}
```

UPDATES

- *Dec 17, 2021*: This recommendation has been acknowledged by the World Of MetaSea team.

2.2.3. TokenSale.sol - Duplicated batch number checking INFORMATIVE

In the updateBatchSaleInfo function, the batchNumber_ checking statement should be replaced by the batchExisted modifier.

```
function updateBatchSaleInfo(
    uint256 batchNumber_,
    address recipient_,
    address paymentToken_,
    uint256 price_,
    uint256 softCap_,
    uint256 hardCap_,
    uint256 start_,
    uint256 end_,
    uint256 releaseTimestamp_,
    uint256 tgeCliff_
) external onlyRole(MAINTAINER_ROLE) {
    require(batchNumber_ > 0, "TokensSale: batchNumber_ is 0");
    require(
        recipient_ != address(0),
        "TokensSale: recipient_ is zero address"
    );
    require(
        paymentToken_ != address(0),
        "TokensSale: paymentToken_ is zero address"
    );
    require(price_ > 0, "TokensSale: price_ is 0");
    require(
        _batches.contains(batchNumber_),
        "TokensSale: batchNumber_ does not exist"
    );
    // ...
}
```

RECOMMENDATION

We can update the code as below.

```
function updateBatchSaleInfo(
    uint256 batchNumber_,
    address recipient_,
    address paymentToken_,
    uint256 price_,
    uint256 softCap_,
    uint256 hardCap_,
    uint256 start_,
    uint256 end_,
    uint256 releaseTimestamp_,
    uint256 tgeCliff_
) external onlyRole(MAINTAINER_ROLE) batchExisted(batchNumber_) {
    require(batchNumber_ > 0, "TokensSale: batchNumber_ is 0");
    require(
        recipient_ != address(0),
        "TokensSale: recipient_ is zero address"
    );
    require(
        paymentToken_ != address(0),
        "TokensSale: paymentToken_ is zero address"
    );
    require(price_ > 0, "TokensSale: price_ is 0");
    // ...
}
```

UPDATES

- *Dec 17, 2021*: This recommendation has been acknowledged by the World Of MetaSea team.

APPENDIX

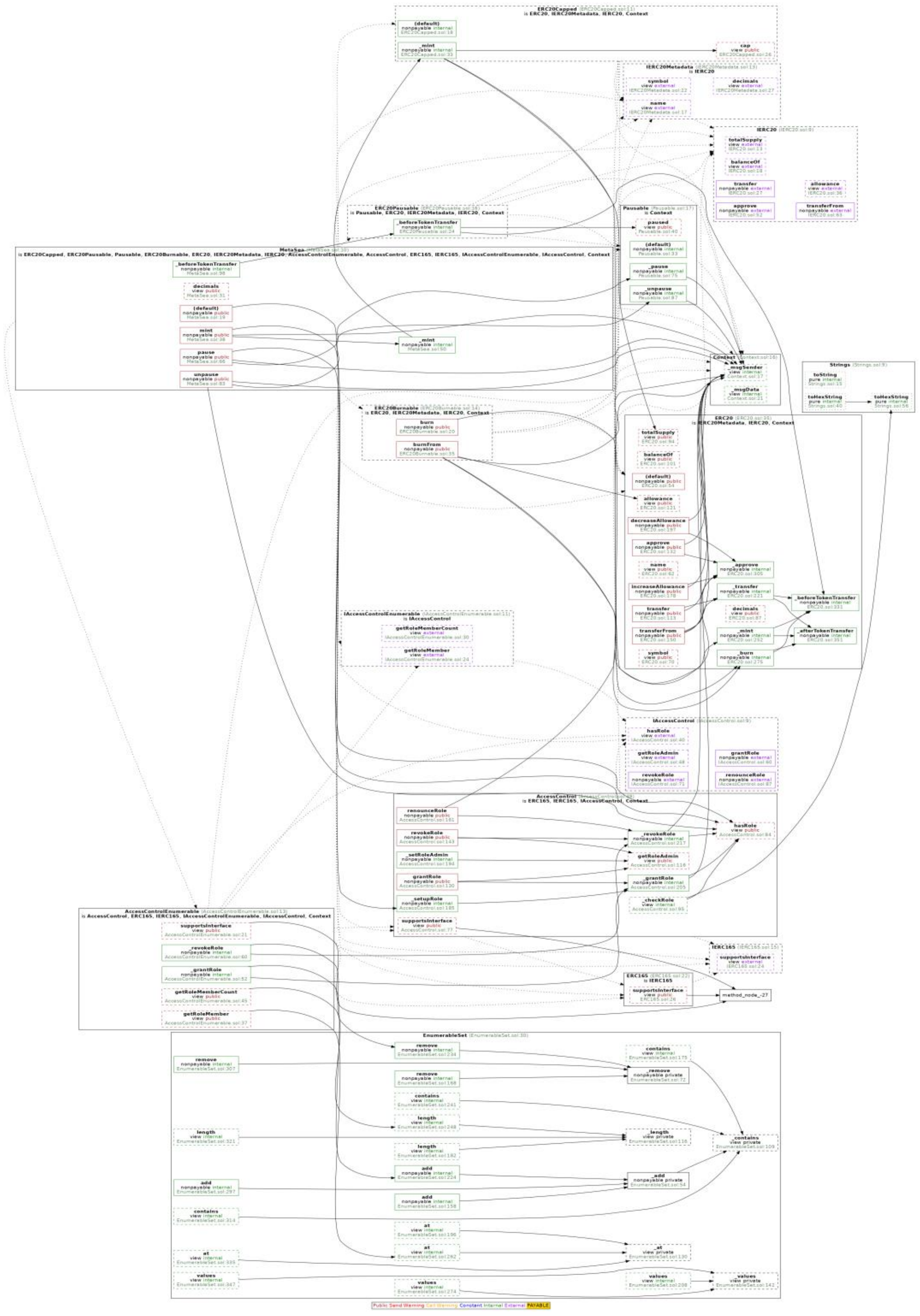
Image 1. ROFI Token call graph

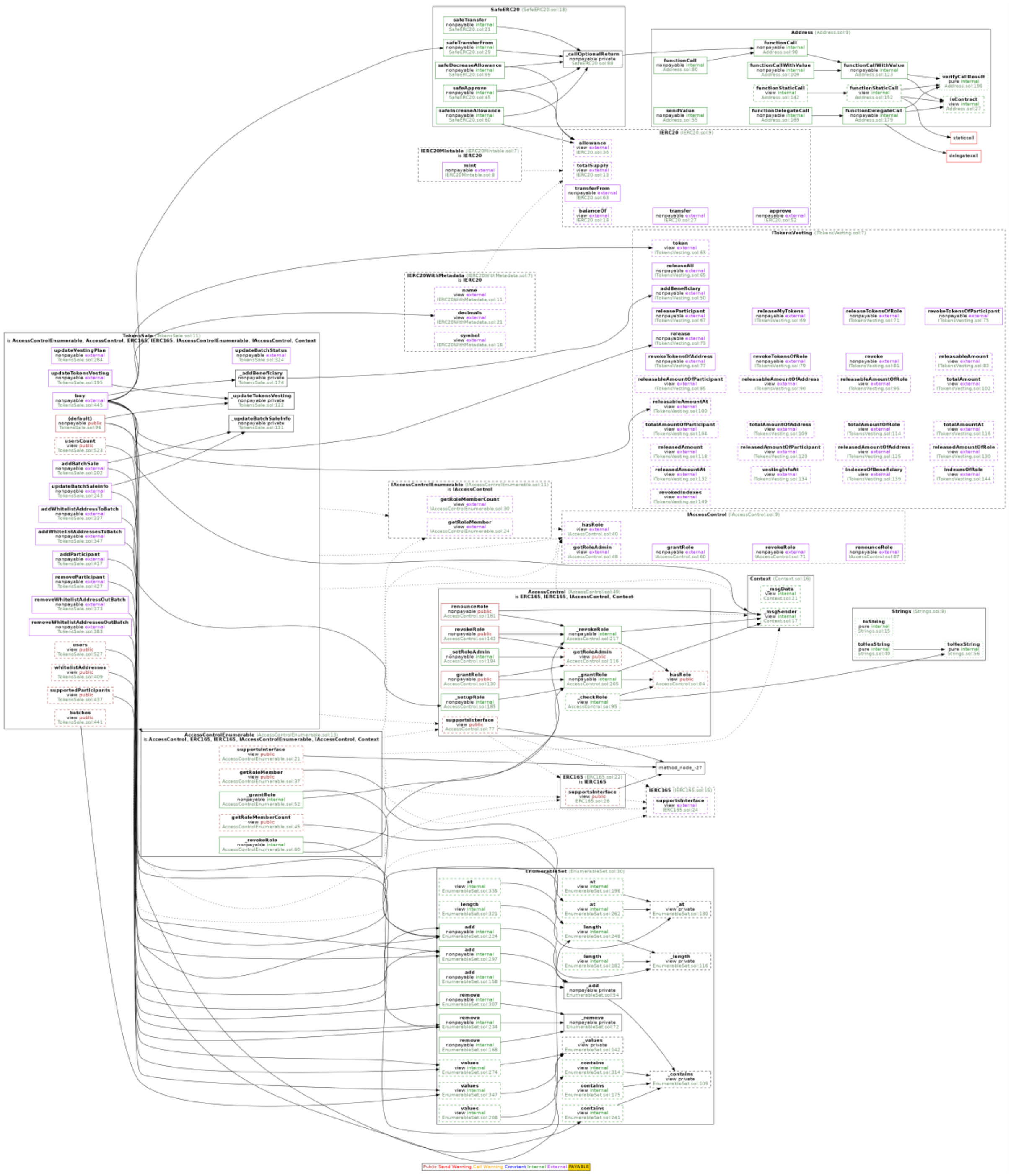

Image 2. Token Sale call graph

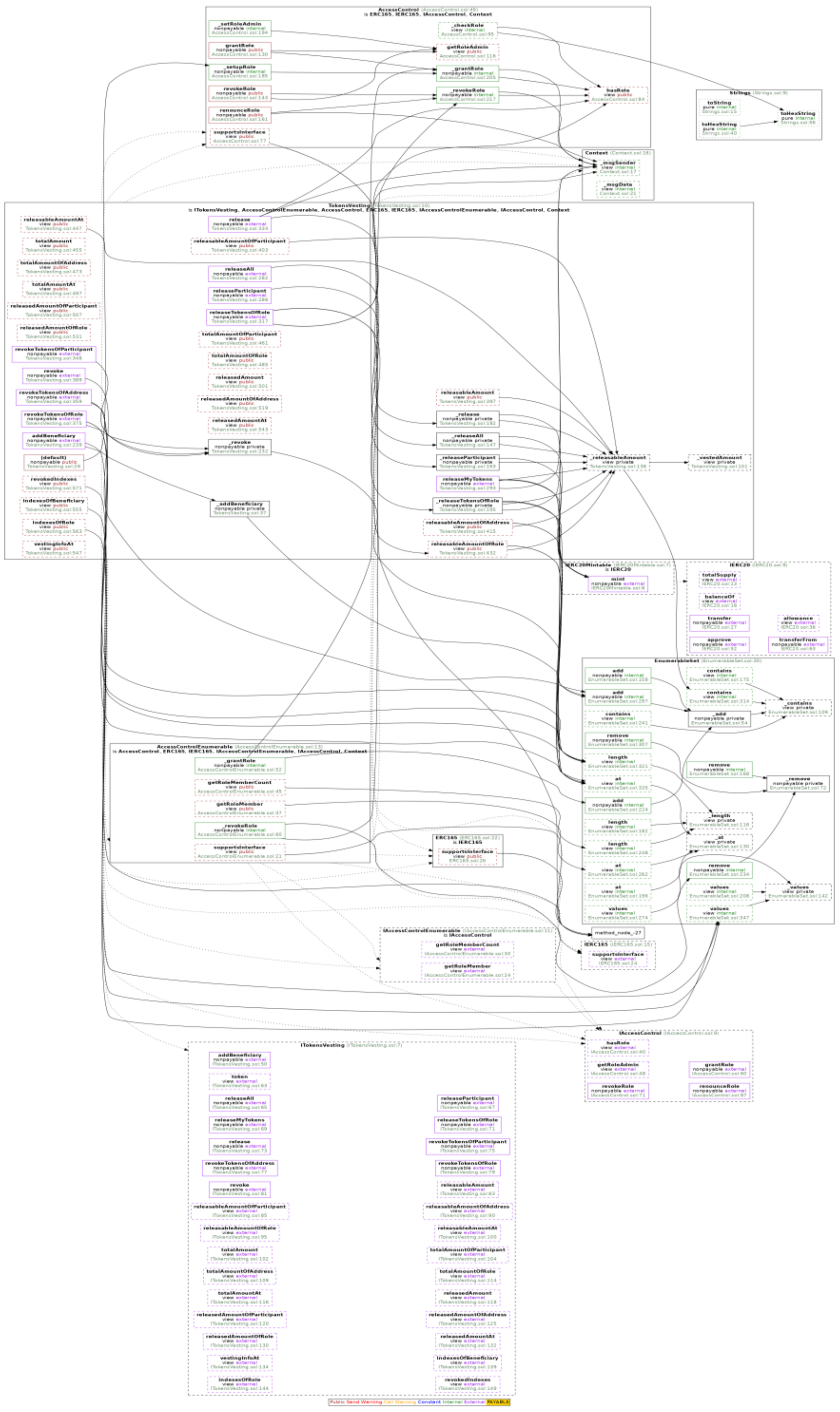

Image 3. Token Vesting call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 17, 2021*	Public Report	Verichains Lab

Table 3. Report versions history