SECURITY AUDIT OF

METASTRIKE TOKEN SMART CONTRACTS

Public Report

Jan 12, 2022

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 Jan 12, 2022. We would like to thank the MetaStrike 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 MetaStrike Token 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 no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About MetaStrike Token Smart Contracts

MetaSrike is a blockchain-based role-play shooting game with a collection of weapons for player to equip, upgrade level to complete mission and earn NFT & tokens.

The MetaStrike Token Smart Contracts include 2 ERC20 token contracts which MetaSrike players use in the game.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of MetaStrike Token Smart Contracts. It was conducted on commit 2289ddea5d6a992dd875e8ff3e5d5ecdafe7814a from git repository *https://github.com/MetastrikeHQ/smartcontracts/blob/main/contracts/Token/*.

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 initial review was conducted in Dec 2021 and a total effort of 4 working days was dedicated to identifying and documenting security issues in the code base of MetaStrike Token Smart Contracts

There are two token contracts in the MetaStrike Token Smart Contracts. They are MetaStrikeMTS and MetaStrikeMTT contracts.

2.2. Contract codes

The MetaStrike Token Smart Contracts was written in Solidity language, with the required version to be 0.8.2.

2.2.1. MetaStrikeMTS contract

MetaStrikeMTS contract is an ERC20 token contract. Almost all abstract contracts, contracts were inherited from the OpenZeppelin contract that include ERC20, ERC20Burnable, Pauseable and Context contracts.

The contract implements TwoPhaseOwnable abstract contract to allow owner transferOwnership with two phases which ensure the receiver accepts the transference.

Table 2 lists some properties of the audited MetaStrikeMTS contract (as of the report writing time).

PROPERTY	VALUE
Name	Metastrike
Symbol	MTS
Decimals	18
Total Supply	556,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 556,000,000 or 556 million.

Table 2. The MetaStrikeMTS contract properties

2.2.2. MetaStrikeMTT contract

MetaStrikeMTT contract is an ERC20 token contract. Almost all abstract contracts, contracts were inherited from OpenZeppelin contract that include ERC20, ERC20Burnable, Pauseable, AccessControl and Context contracts.

In addition, the contract implements the mint public function which allows owner contract can mint unlimited token. Therefore, the totalSupply value can be changed by this function.

Table 3 lists some properties of the audited MetaStrikeMTT contract (as of the report writing time).

PROPERTY	VALUE
Name	Metastrike
Symbol	MTT
Decimals	18

Table 3. The MetaStrikeMTT contract properties

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of the MetaStrike Token Smart Contracts.

2.4. Additional notes and recommendations

2.4.1. Missing check length of array in batchBlackList function INFORMATIVE

Both two contracts of MetaStrike Token Smart Contracts use the batchBlackList function. The function uses _black parameter but noncheck the length of this array variable. Therefore, the function can access an index which not inbound of the array causes an error in the function.

```
function batchBlackList(address[] memory _evil, bool[] memory _black) ext…
  ernal onlyOwner {
        for (uint256 i = 0; i < _evil.length; i++) {
            blacklisted[_evil[i]] = _black[i];
        }
    }
```

Snippet 1. Missing check array length in `batchBlackList` function

RECOMMENDATION

We suggest changing the function like the below code:

```
function batchBlackList(address[] memory _evil, bool[] memory _black) ext…
  ernal onlyOwner {
        uint256 length = _evil.length;
        require(length > 0 && length == _black.length, "batchBlackList: a…
  rray length is invalid");
        for (uint256 i = 0; i < length; i++) {
            blacklisted[_evil[i]] = _black[i];
        }
    }
```

Snippet 2. Recommend fixing in `batchBlackList` function

UPDATES

- *Dec 14,2021*: This issue has been acknowledged and fixed by the MetaStrike team in commit cedfacd8ec4c570e6e4dd8beed73e847fa079c5d.

2.4.2. BPContract function INFORMATIVE

Both two contracts of MetaStrike Token Smart Contracts use the BPContract contract. Since we do not control the logic of the BPContract, there is no guarantee that BPContract will not contain any security related issues. With the current context, in case the BPContract is compromised, there is not yet a way to exploit the MetaStrike Token Smart Contracts, but we still note that here as a warning for avoiding any related issue in the future.

By the way, if having any issue, the BPContract function can be easily disabled anytime by the contract owner using the setBpEnabled function. In addition, BPContract is only used in a short time in token public sale IDO then the contract owner will disable it forever by the setBotProtectionDisableForever function.

APPENDIX

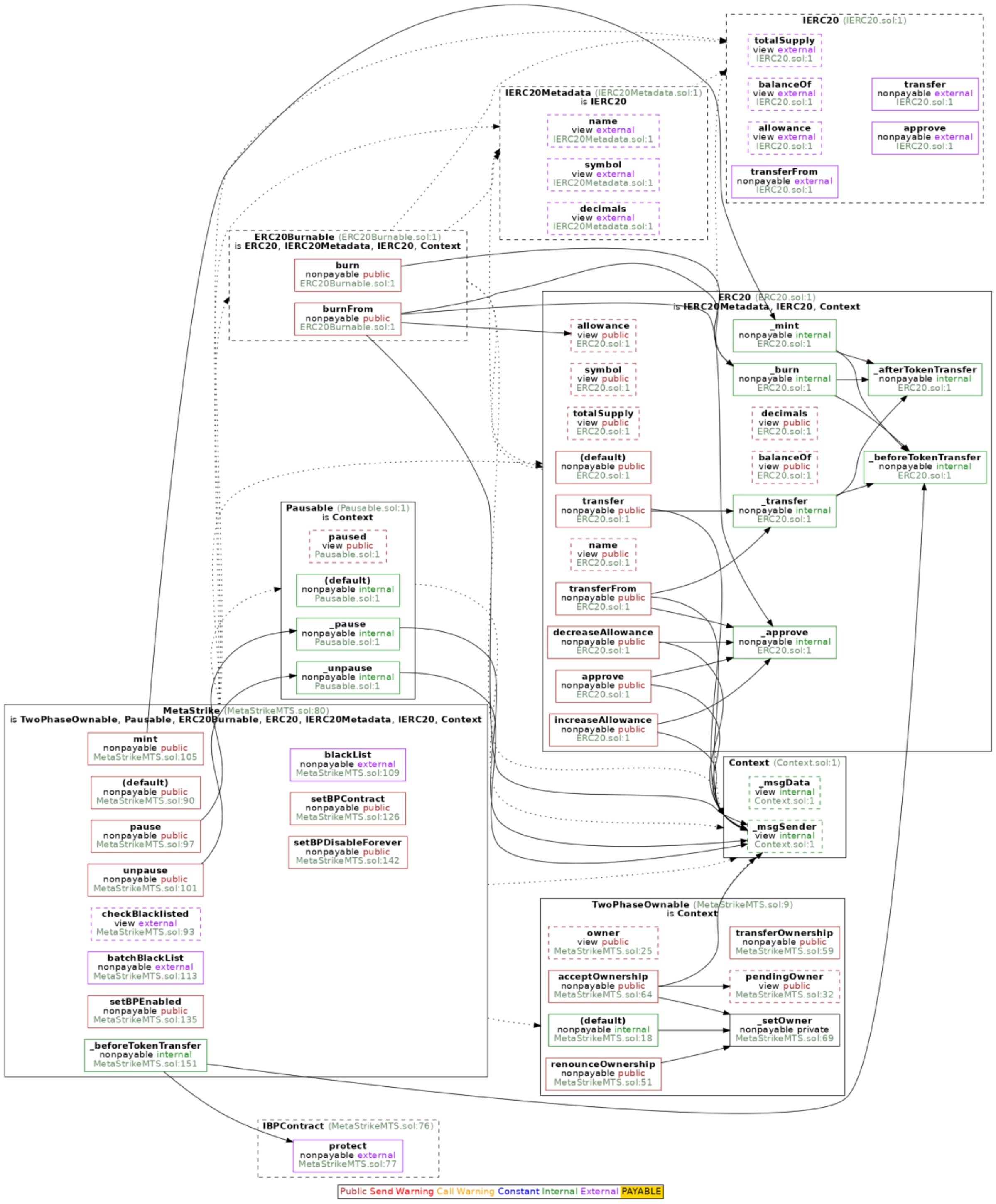
IERC20 (IERC20.sol:1)
totalSupply view external IERC20.sol:1
balanceOf view external IERC20.sol:1
transfer nonpayable external IERC20.sol:1
allowance view external IERC20.sol:1
approve nonpayable external IERC20.sol:1
transferFrom nonpayable external IERC20.sol:1
IERC20Metadata (IERC20Metadata.sol:1) is IERC20
name view external IERC20Metadata.sol:1
symbol view external IERC20Metadata.sol:1
decimals view external IERC20Metadata.sol:1
ERC20Burnable (ERC20Burnable.sol:1) is ERC20, IERC20Metadata, IERC20, Context
burn nonpayable public ERC20Burnable.sol:1
burnFrom nonpayable public ERC20Burnable.sol:1
ERC20 (ERC20.sol:1) is IERC20Metadata, IERC20, Context
allowance view public ERC20.sol:1
symbol view public ERC20.sol:1
totalSupply view public ERC20.sol:1
(default) nonpayable public ERC20.sol:1
transfer nonpayable public ERC20.sol:1
name view public ERC20.sol:1
transferFrom nonpayable public ERC20.sol:1
decreaseAllowance nonpayable public ERC20.sol:1
approve nonpayable public ERC20.sol:1
increaseAllowance nonpayable public ERC20.sol:1
_mint nonpayable internal ERC20.sol:1
_burn nonpayable internal ERC20.sol:1
decimals view public ERC20.sol:1
balanceOf view public ERC20.sol:1
_transfer nonpayable internal ERC20.sol:1
_approve nonpayable internal ERC20.sol:1
_afterTokenTransfer nonpayable internal ERC20.sol:1
_beforeTokenTransfer nonpayable internal ERC20.sol:1
Pausable (Pausable.sol:1) is Context
paused view public Pausable.sol:1
(default) nonpayable internal Pausable.sol:1
_pause nonpayable internal Pausable.sol:1
_unpause nonpayable internal Pausable.sol:1
MetaStrike (MetaStrikeMTS.sol:80) is TwoPhaseOwnable, Pausable, ERC20Burnable, ERC20, IERC20Metadata, IERC20, Context
mint nonpayable public MetaStrikeMTS.sol:105
(default) nonpayable public MetaStrikeMTS.sol:90
pause nonpayable public MetaStrikeMTS.sol:97
unpause nonpayable public MetaStrikeMTS.sol:101
checkBlacklisted view external MetaStrikeMTS.sol:93
batchBlackList nonpayable external MetaStrikeMTS.sol:113
setBPEnabled nonpayable public MetaStrikeMTS.sol:135
_beforeTokenTransfer nonpayable internal MetaStrikeMTS.sol:151
blackList nonpayable external MetaStrikeMTS.sol:109
setBPContract nonpayable public MetaStrikeMTS.sol:126
setBPDisableForever nonpayable public MetaStrikeMTS.sol:142
Context (Context.sol:1)
_msgData view internal Context.sol:1
_msgSender view internal Context.sol:1
TwoPhaseOwnable (MetaStrikeMTS.sol:9) is Context
owner view public MetaStrikeMTS.sol:25
acceptOwnership nonpayable public MetaStrikeMTS.sol:64
(default) nonpayable internal MetaStrikeMTS.sol:18
renounceOwnership nonpayable public MetaStrikeMTS.sol:51
transferOwnership nonpayable public MetaStrikeMTS.sol:59
pendingOwner view public MetaStrikeMTS.sol:32
_setOwner nonpayable private MetaStrikeMTS.sol:69
IBPContract (MetaStrikeMTS.sol:76)
protect nonpayable external MetaStrikeMTS.sol:77
Public Send Warning Call Warning Constant Internal External PAYABLE

Image 1. MetaStrikeMTS smart contract call graph

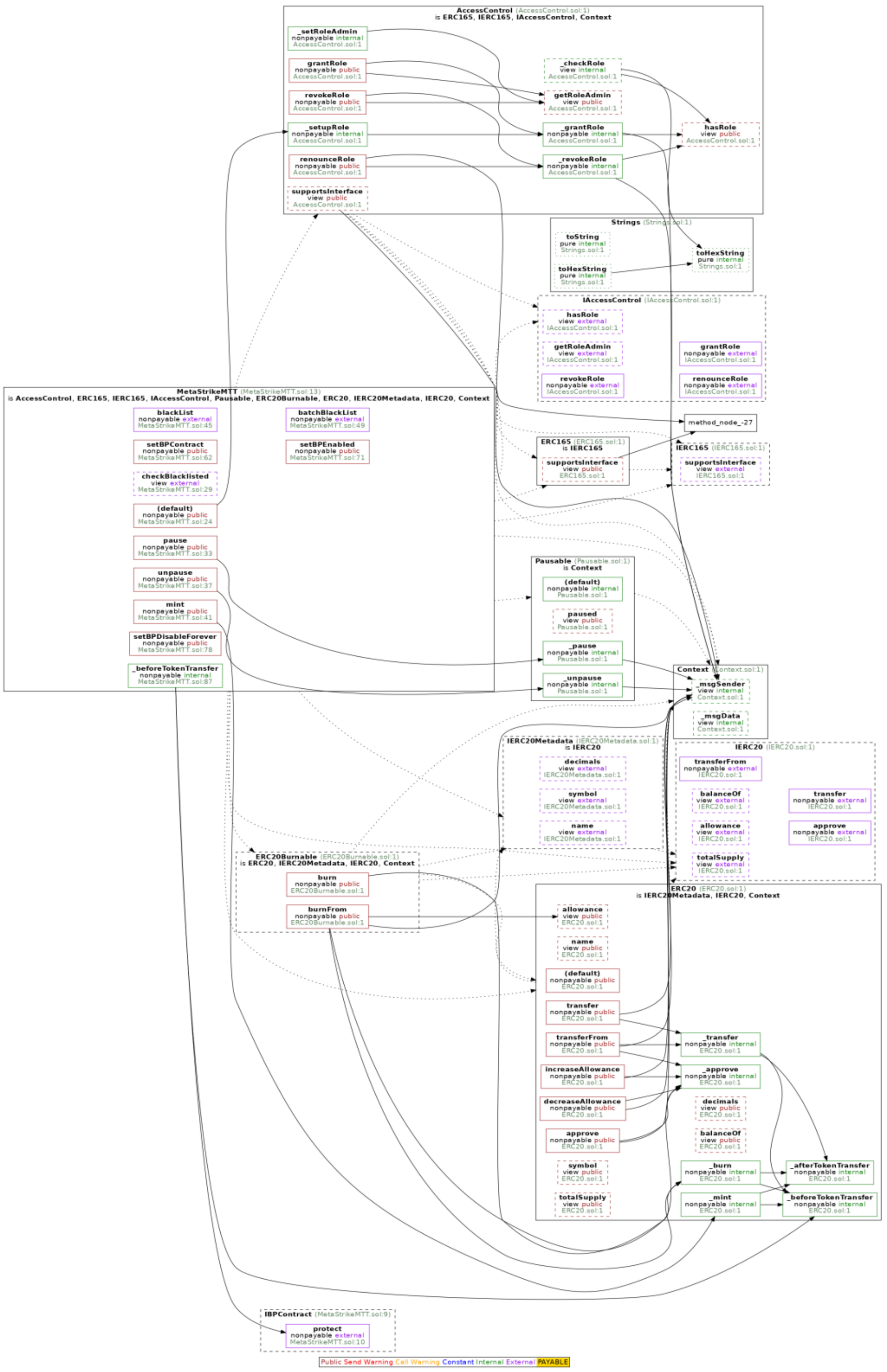

Image 2. MetaStrikeMTT smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 03, 2021*	Public Report	Verichains Lab
1.1	*Dec 14, 2021*	Public Report	Verichains Lab
1.2	*Jan 12, 2022*	Public Report	Verichains Lab

Table 4. Report versions history