



verichains

SECURITY AUDIT OF
**METASTRIKE TOKEN SMART
CONTRACTS**



Public Report

Dec 03, 2021

Verichains Lab

info@verichains.io

<https://www.verichains.io>

Driving Technology > Forward

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



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 xRP20 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.

Report for MetaStrike**Security Audit – MetaStrike Token Smart Contracts**

Version: 1.0 – Public Report

Date: Dec 03, 2021



EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Dec 03, 2021. 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.

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



TABLE OF CONTENTS

1. MANAGEMENT SUMMARY	5
1.1. About MetaStrike Token Smart Contracts	5
1.2. Audit scope	5
1.3. Audit methodology.....	5
1.4. Disclaimer.....	6
2. AUDIT RESULT	7
2.1. Overview.....	7
2.2. Contract codes.....	7
2.2.1. MetaStrikeMTS contract.....	8
2.2.2. MetaStrikeMTT contract.....	8
2.3. Findings	8
2.4. Additional notes and recommendations.....	8
2.4.1. Missing check length of array in batchBlackList function INFORMATIVE	8
3. VERSION HISTORY	12

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



1. MANAGEMENT SUMMARY

1.1. About MetaStrike Token Smart Contracts

MetaStrike 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 MetaStrike players use in the game.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Kingdom Raids Token. It was conducted on commit [0424e4c5053e91045ccd3eb91c12797da44ce61e](#) from git repository <https://github.com/MetastrikeHQ/smартcontracts/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

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



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.

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



2. AUDIT RESULT

2.1. Overview

The initial review was conducted in Dec 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of MetaStrike Token Smart Contracts

There are 2 token contract in the MetaStrike Token Smart Contracts. They are MetaStrikeMTS and MetaStrikeMTT contracts.

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

PROPERTY	VALUE
Name	Metastrike
Symbol	MTS
Decimals	18

Table 2. The MetaStrikeMTS contract properties

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

2.2. Contract codes

The MetaStrike Token Smart Contracts was written in [Solidity](#) language, with the required version to be [0.8.2](#).

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



2.2.1. MetaStrikeMTS contract

MetaStrikeMTS contract is an ERC20 token contract. Almost contracts was inherited from OpenZeppelin contract that include [ERC20](#), [ERC20Burnable](#), [Pauseable](#) and [Context](#) contracts.

The contract implements [TwoPhaseOwnable](#) abstract contract to allow [owner](#) [transferOwnership](#) with two phase which ensure the receiver accepts the tranference.

In addition, the contract implements [mint](#) public function which allow [owner](#) contract can [mint](#) unlimited token. Therefore, the [totalSupply](#) value can be changed by this function.

2.2.2. MetaStrikeMTT contract

MetaStrikeMTT contract is an ERC20 token contract. Almost contracts was inherited from OpenZeppelin contract that include [ERC20](#), [ERC20Burnable](#), [Pauseable](#) and [Context](#) contracts.

In addition, the contract implements [mint](#) public function which allow [owner](#) contract can [mint](#) unlimited token. Therefore, the [totalSupply](#) value can be changed by this function.

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 2 contract of MetaStrike Token Smart Contracts uses 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) external 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) external onlyOwner {
```

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



```
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

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 - Public Report

Date: Dec 03, 2021



APPENDIX

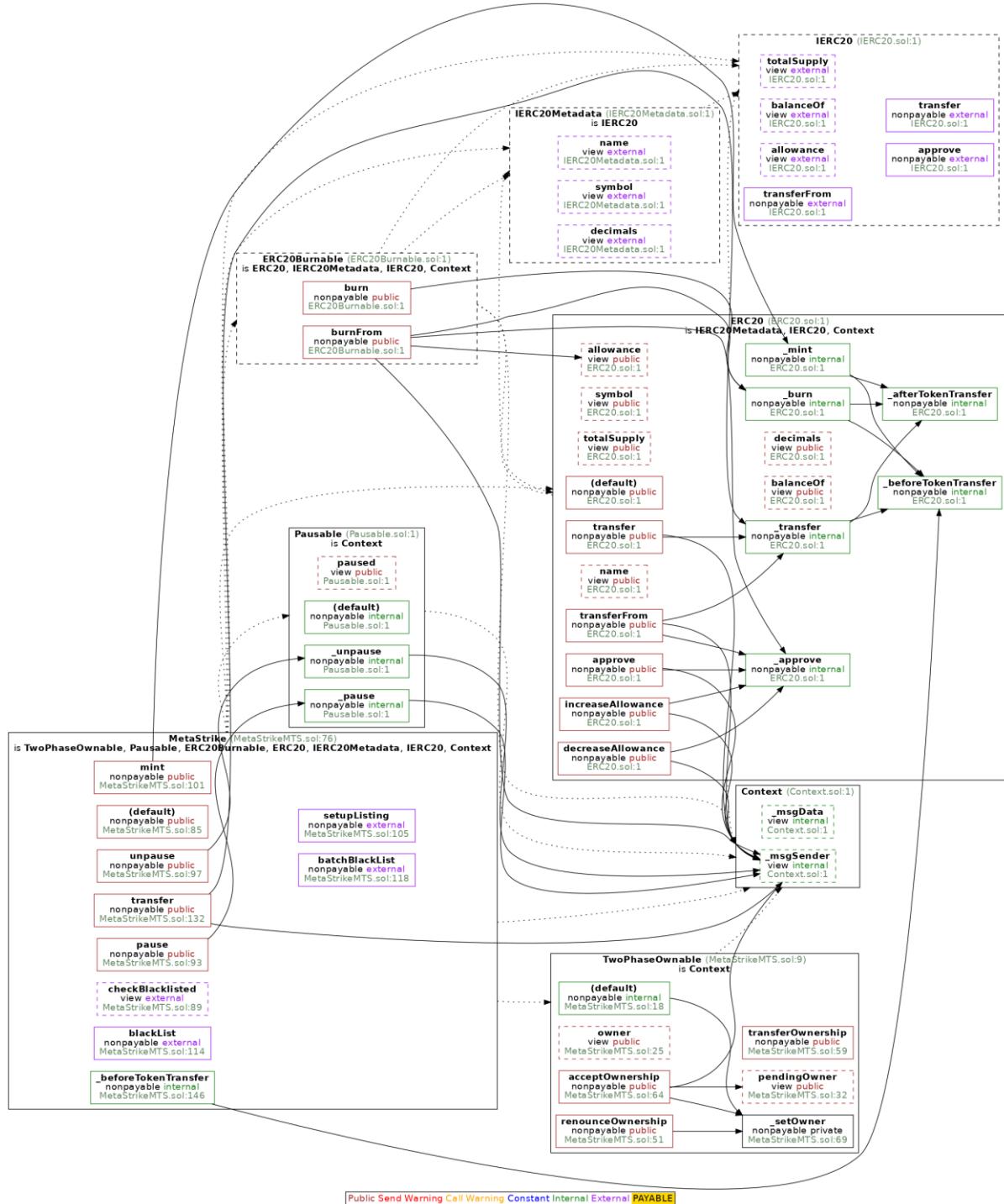


Image 1. MetaStrikeMTS smart contract call graph

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 - Public Report

Date: Dec 03, 2021

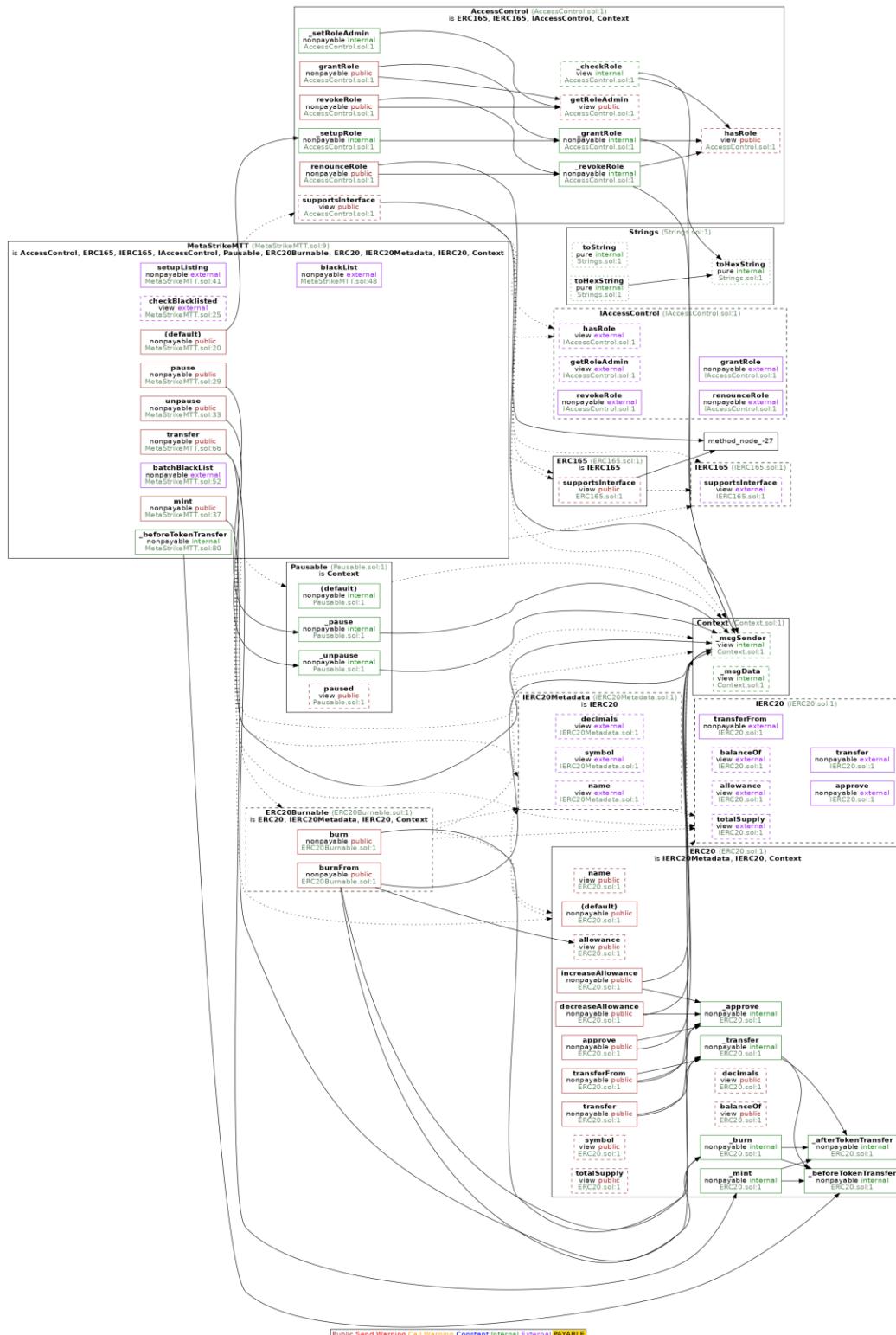


Image 2. MetaStrikeMTT smart contract call graph

Report for MetaStrike

Security Audit – MetaStrike Token Smart Contracts

Version: 1.0 – Public Report

Date: Dec 03, 2021



3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	<i>2021-12-03</i>	Public Report	Verichains Lab

Table 4. Report versions history