SECURITY AUDIT OF

ACESTARTER SMART CONTRACTS

Public Report

Mar 11, 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 Mar 11, 2022. We would like to thank the AceStarter 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 AceStarter 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 AceStarter Smart Contracts

AceStarter is the next-generation launchpad that curates and launches world-class crypto projects on a global scale. Based on a community-centric model, demonstrated by the fair launch of its very own token, AceStarter aims to fairly distribute investment opportunities to investors with even the smallest budgets by aligning their incentives and contribution with projects they back.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the AceStarter Smart Contracts.

It was conducted on commit 9254e86b6f89fde2f47be3ebf4a15465ebd0adcc from git repository *https://github.com/AceStarterOfficial/ASTAR-contracts*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
29585c175a073b888e2b970a8a7c968226b288990f5c8fb148a64336de5f1d55	**ASTAR.sol**
06087a23bf50b78f491df623a37a14eb900ffa515b5f21e06a9bc608a6015f5c	**ASTARVesting.sol**

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

AceStarter Smart Contracts contains 2 main contracts: ASTARToken and ASTARVesting.

2.1.1. ASTARToken contract

This is the ERC20 token contract in the AceStarter Smart Contracts, which extends AccessControl, ERC20, ERC20Snapshot and ERC20Pausable contracts. AccessControl allows the contract to implement role-based access control mechanisms which adds token owner (contract deployer) OWNER_ROLE role, he then can set any roles for anyone at anytime. ERC20Snapshot helps OWNER_ROLE take a snapshot of the balances and total supply at a time for later access. OWNER_ROLE can pause/unpause contract using ERC20Pausable contract, user can only transfer tokens when contract is not paused. The contract also supports bot protection from another BP contract which is NOT in our audit scope.

The contract pre-minted 120 million tokens: 50 million for Seed Round, 50 million for IDO and 20 million for Liquidity when deployed. The rest (880 million tokens) are used for Reserve Minting which will be minted based on differences between Demand and Supply of previous day on Decentralized Exchange transactions.

This table lists some properties of the audited ASTARToken (as of the report writing time).

PROPERTY	VALUE
Name	AceStarter
Symbol	ASTAR
Decimals	18
Max Supply	1,000,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

2.1.2. ASTARVesting contract

This is the vesting contract in the AceStarter Smart Contracts, which extends AccessControl contract. AccessControl allows the contract to implement role-based access control mechanisms which adds contract owner (contract deployer) OWNER_ROLE role, he then can set any roles for anyone at anytime. This contract implements a vesting mechanism to lock and release tokens according to the schedule. The owner can withdraw tokens in vesting contract at any time in case of emergency situation.

2.2. Findings

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

2.3. Additional notes and recommendations

2.3.1. ASTAR.sol - Redundant whenNotPaused INFORMATIVE

The contract extends ERC20Pausable and the logic to pause transfer token has already been implemented, so it's no need to use the whenNotPaused modifier.

```
function _beforeTokenTransfer(address from, address to, uint256 amount) i…
  nternal  whenNotPaused  override(ERC20, ERC20Pausable, ERC20Snapshot) {…

    if (isInPreventBotMode) {
        BP.protect(from, to, amount);
    }
    if (from == pairASTARBUSD) {
        amountOut += amount;
    }
    if (to == pairASTARBUSD) {
        amountIn += amount;
    }
    super._beforeTokenTransfer(from, to, amount);
}

// ERC20Pausable
function _beforeTokenTransfer(address from, address to, uint256 amount) i…
  nternal virtual override {
    super._beforeTokenTransfer(from, to, amount);

    require(!paused(), "ERC20Pausable: token transfer while paused");
}
```

RECOMMENDATION

Consider removing the modifier.

UPDATES

- *Mar 11, 2022*: This issue has been acknowledged and fixed by the AceStarter team in commit 645680b7315b517863adfd35c8f3e2860f093e91.

2.3.2. ASTARVesting.sol - Redundant _vestable variable INFORMATIVE

The variable vestable is not used anywhere, so we can remove it and return the result instead.

```
function vestable(address beneficiary) public view returns (uint) {
    require(beneficiaries[beneficiary] > 0, "The beneficiary address is i…
  nvalid");
    require(tge > 0, "TGE is not config");
    uint amount = 0;

    if (block.timestamp < tge + cliffDurations[beneficiary]) {
        amount = tgeReleases[beneficiary];
    } else if (block.timestamp >= (tge + cliffDurations[beneficiary] + ve…
  stingDurations[beneficiary])) {
        amount = beneficiaries[beneficiary];
    } else {
        uint vestingAmount = beneficiaries[beneficiary] - tgeReleases[ben…
  eficiary];
        uint currentVestingTime = block.timestamp - (tge + cliffDurations…
  [beneficiary]);

        amount = (((vestingAmount * currentVestingTime) / vestingDuration…
  s[beneficiary])) + tgeReleases[beneficiary];
    }
    uint _vestable = (amount * getTotalToken() / reserveAmount) - release…
  d[beneficiary];
    return _vestable;

}
```

RECOMMENDATION

Consider removing _vestable variable.

For example:

```
function vestable(address beneficiary) public view returns (uint) {
    require(beneficiaries[beneficiary] > 0, "The beneficiary address is i…
  nvalid");
    require(tge > 0, "TGE is not config");
    uint amount = 0;
    if (block.timestamp < tge + cliffDurations[beneficiary]) {
        amount = tgeReleases[beneficiary];
    } else if (block.timestamp >= (tge + cliffDurations[beneficiary] + ve…
  stingDurations[beneficiary])) {
        amount = beneficiaries[beneficiary];
    } else {
        uint vestingAmount = beneficiaries[beneficiary] - tgeReleases[ben…
  eficiary];
        uint currentVestingTime = block.timestamp - (tge + cliffDurations…
  [beneficiary]);

        amount = (((vestingAmount * currentVestingTime) / vestingDuration…
  s[beneficiary])) + tgeReleases[beneficiary];
    }
    return (amount * getTotalToken() / reserveAmount) - released[benefici…
  ary];
}
```

UPDATES

- *Mar 11, 2022*: This issue has been acknowledged by the AceStarter team.

2.3.3. ASTARVesting.sol - Duplicate code INFORMATIVE

The logic require(beneficiaries[beneficiary] > 0, "The beneficiary address is invalid"); is used everywhere in the contract.

RECOMMENDATION

Consider using a modifier for code and logic consistency.

For example:

```
// NFTUpgradeable.sol
modifier validBeneficiary(address beneficiary) {
    require(beneficiaries[beneficiary] > 0, "The beneficiary address is i…
  nvalid");
    _;
}

function getAllocation(address beneficiary) public view validBeneficiary(…
  beneficiary) returns (uint) {
    return beneficiaries[beneficiary];
}
...
```

UPDATES

- *Mar 11, 2022*: This issue has been acknowledged by the AceStarter team.

2.3.4. ASTARVesting.sol - Typo INFORMATIVE

There are some typos in the contract.

require(vestableAmount > 0, "The is nothing to vest"); should be require(vestableAmount > 0, "There is nothing to vest");

require(address(token) == address(0), "The token is setted"); should be require(address(token) == address(0), "The token is set");

RECOMMENDATION

Fixing the typo.

APPENDIX

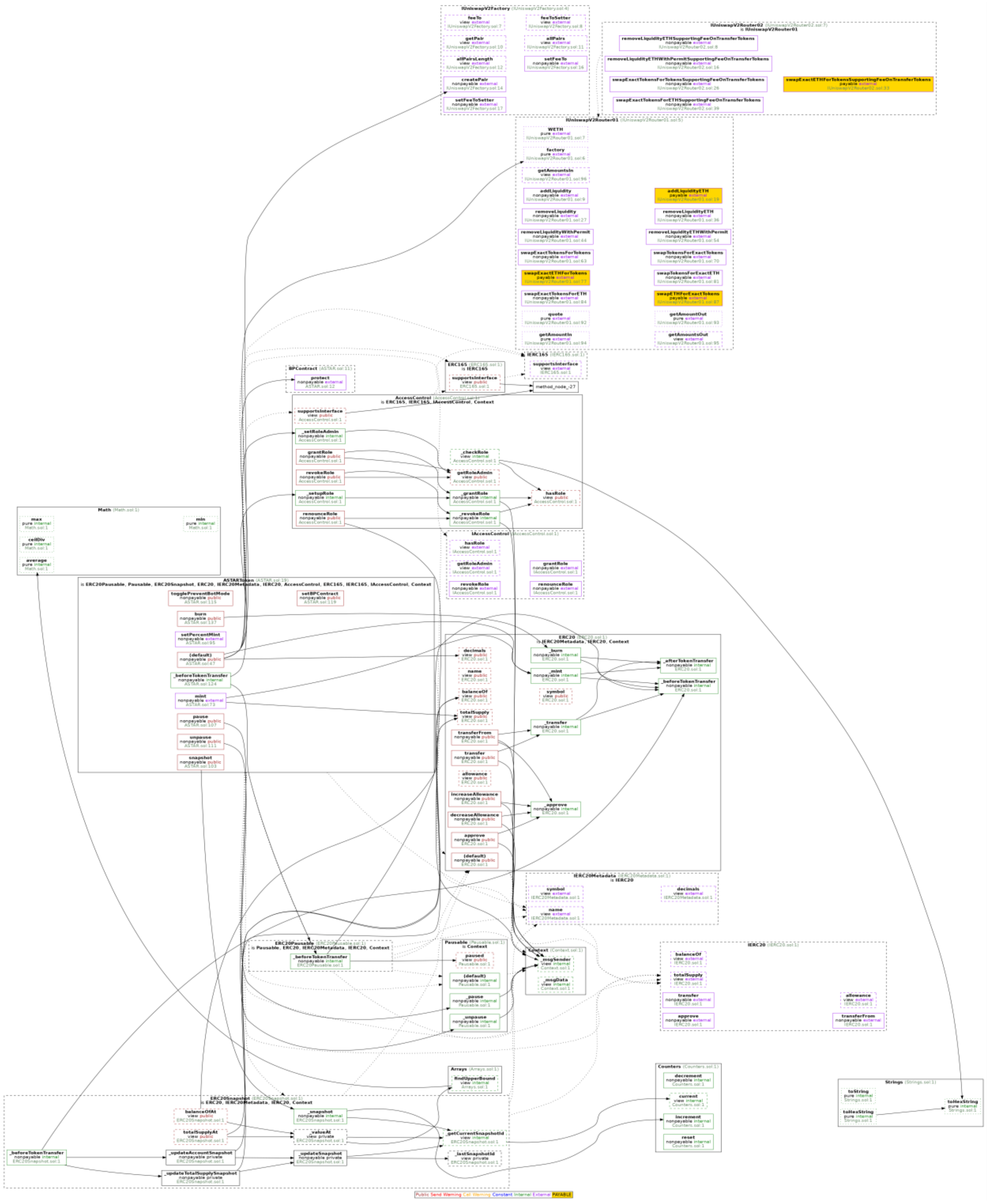

Image 1. ASTARToken call graph

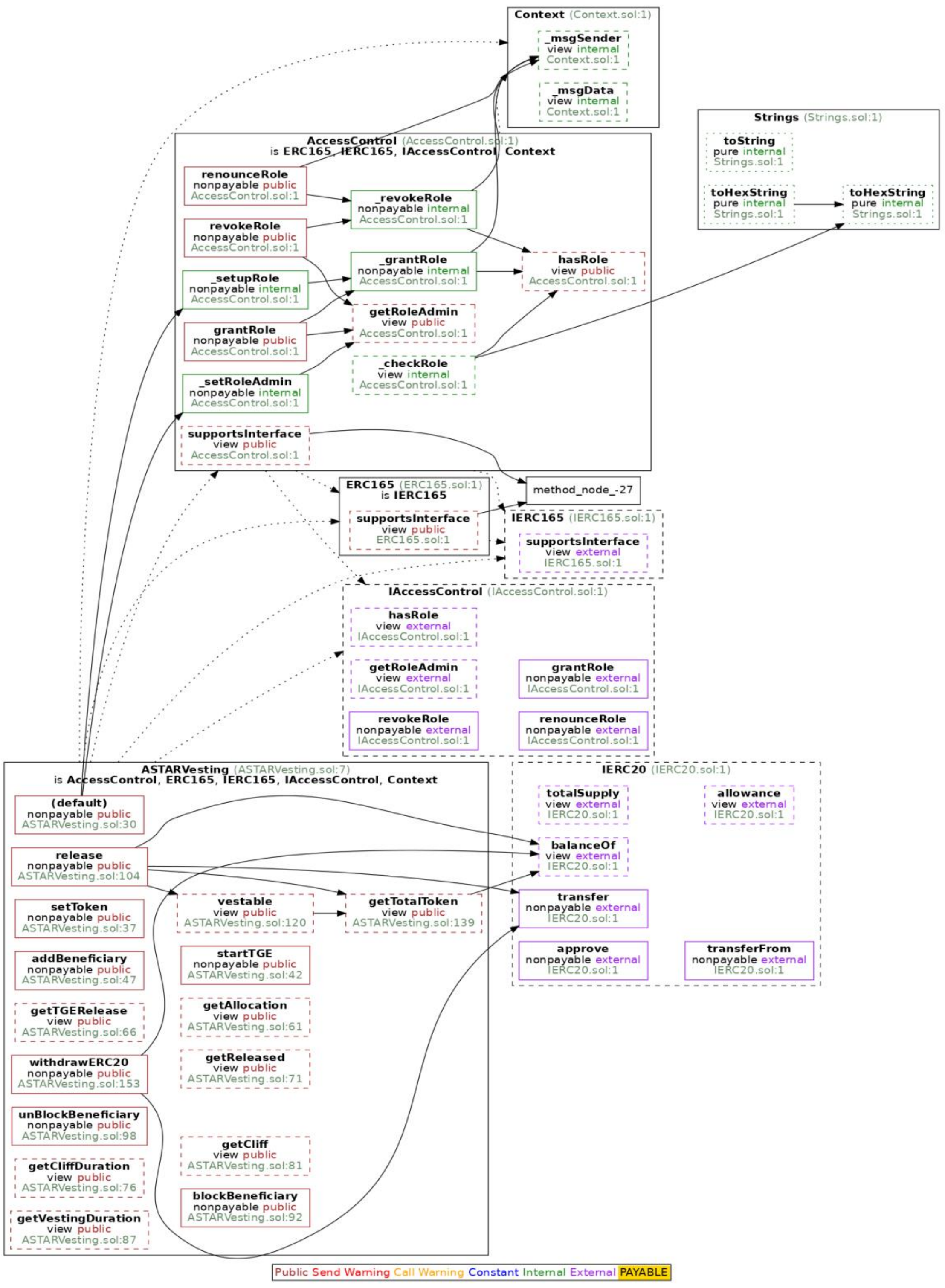
Context (Context.sol:1)
_msgSender view internal Context.sol:1
_msgData view internal Context.sol:1
Strings (Strings.sol:1)
toString pure internal Strings.sol:1
toHexString pure internal Strings.sol:1
toHexString pure internal Strings.sol:1
AccessControl (AccessControl.sol:1) is ERC165, IERC165, IAccessControl, Context
renounceRole nonpayable public AccessControl.sol:1
_revokeRole nonpayable internal AccessControl.sol:1
revokeRole nonpayable public AccessControl.sol:1
_grantRole nonpayable internal AccessControl.sol:1
hasRole view public AccessControl.sol:1
_setupRole nonpayable internal AccessControl.sol:1
getRoleAdmin view public AccessControl.sol:1
grantRole nonpayable public AccessControl.sol:1
_checkRole view internal AccessControl.sol:1
_setRoleAdmin nonpayable internal AccessControl.sol:1
supportsInterface view public AccessControl.sol:1
method_node_-27
ERC165 (ERC165.sol:1) is IERC165
supportsInterface view public ERC165.sol:1
IERC165 (IERC165.sol:1)
supportsInterface view external IERC165.sol:1
IAccessControl (IAccessControl.sol:1)
hasRole view external IAccessControl.sol:1
getRoleAdmin view external IAccessControl.sol:1
grantRole nonpayable external IAccessControl.sol:1
revokeRole nonpayable external IAccessControl.sol:1
renounceRole nonpayable external IAccessControl.sol:1
ASTARVesting (ASTARVesting.sol:7) is AccessControl, ERC165, IERC165, IAccessControl, Context
(default) nonpayable public ASTARVesting.sol:30
release nonpayable public ASTARVesting.sol:104
vestable view public ASTARVesting.sol:120
getTotalToken view public ASTARVesting.sol:139
setToken nonpayable public ASTARVesting.sol:37
startTGE nonpayable public ASTARVesting.sol:42
addBeneficiary nonpayable public ASTARVesting.sol:47
getAllocation view public ASTARVesting.sol:61
getTGERelease view public ASTARVesting.sol:66
getReleased view public ASTARVesting.sol:71
withdrawERC20 nonpayable public ASTARVesting.sol:153
unBlockBeneficiary nonpayable public ASTARVesting.sol:98
getCliff view public ASTARVesting.sol:81
getCliffDuration view public ASTARVesting.sol:76
blockBeneficiary nonpayable public ASTARVesting.sol:92
getVestingDuration view public ASTARVesting.sol:87
IERC20 (IERC20.sol:1)
totalSupply view external IERC20.sol:1
allowance view external IERC20.sol:1
balanceOf view external IERC20.sol:1
transfer nonpayable external IERC20.sol:1
approve nonpayable external IERC20.sol:1
transferFrom nonpayable external IERC20.sol:1
Public Send Warning Call Warning Constant Internal External PAYABLE

Image 2. ASTARVesting call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 10, 2022*	Public Report	Verichains Lab
1.1	*Mar 11, 2022*	Public Report	Verichains Lab

Table 2. Report versions history