verichains

SECURITY AUDIT OF

DSTARTER SMART CONTRACTS

DStarter

Public Report

Oct 18, 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 was prepared by Verichains Lab on Oct 18, 2022. We would like to thank the DStarter 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 DStarter 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About DStarter Smart Contracts

DStarter Is a DAO-Governed Secure Escrow-Based Smart Contract & Funding Platform For #Web3 Investors & Startup Founders.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of DStarter Smart Contracts. It was conducted on commit `38c200e611eb1bd98b977f281b463822d021ba37` from git repository *https://github.com/timestarter/dao-smart-contract/tree/main/contracts*.

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

SHA256 Sum	File
d10ea247cc99a6755845301a65ba41cab7014fc463626147b6589faed1b77fa1	TSGovernor.sol
202a4c808d2d82dcf0968ef29750a5d686317e1fbf46b0fa69e63ee43e1ac44c	TSProject.sol
f65aecf92b704e22ec86aa69610f80b921d4eedf02dbc621b0c5ed5918558ae7	TSSaleFactory.sol
10bfdc0636da75afbcec2fd600174a821d46de7bdb1feae2c4c99b954f1cb06e	TSVesting.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 were 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 DStarter Smart Contracts was written in Solidity language, with the required version to be ^0.8.4.

2.1.1. TSProject contract

This is the project management contract in the DStarter Smart Contracts, which extends the Ownable contract. With Ownable, by default, the contract owner is contract deployer, but he can transfer ownership to another address at any time. This contract is the place where the user can register as startup/invertor. After register, the startup can declare new project in here.

2.1.2. TSSaleFactory contract

TSSaleFactory contract extends Ownable, Proxy and ReentrancyGuard abstract contracts. With Ownable, by default, the contract owner is the contract deployer, but he can transfer ownership to another address at any time.

The startup can create pools through the createSale function. After the pools were approved by the contract owner, users in the whitelist can invest to the contract to get weights that affect decisions in the governor contract.

The contract extends the Proxy abstract contract which allows the owner to upgrade the TSSaleFactory with the logic that is not in our audit scope.

2.1.3. TSVesting contract

TSVesting contract extends EIP712, Votes, Ownable and ReentrancyGuard abstract contracts. With Ownable, by default, the contract owner is the contract deployer, but he can transfer ownership to another address at any time. Votes is a base abstract contract that tracks voting units, which are a measure of voting power that can be transferred, and provides a system of vote delegation, where an account can delegate its voting units to a sort of "representative" that will pool delegated voting units from different accounts and can then use it to vote in decisions.

Not like the normal Vesting, the users can't deposit to this contract directly, user can only deposit through the invest flow in the TSSaleFactory contract. All the deposited tokens will be released following the linear logic of every duration set by the owner. The users' balance is used to be the factor releasing tokens. Besides, it is also used as weight in vote decisions in the governor contract.

2.1.4. TSGovernor contract

The TSGovernor extends `Governor`, `GovernorCountingSimple`, `GovernorVotes`, and `GovernorVotesQuorumFraction` abstract contracts. With these abstracts and some override functions, the DStarter team creates a contract with voting protocols where project owners and investors can propose, vote, and implement changes through administrative functions.

In the contract, the DStarter team supports project owner to create proposals. All investors, who were deposited in `TSVesting`, can vote `approve/decline` proposals through `castvote` function. After a period, if the proposal is approved, the user can trigger `TSGovernor` contract to execute the `calldata` in the proposal.

2.2. Findings

During the audit process, the audit team found some vulnerability issues in the given version of DStarter Smart Contracts.

2.2.1. TSVesting.sol - Attacker can call `claimRefund` with startUp address to lock `startUp` from calling `claimTokenRefund` function CRITICAL

Both `claimRefund` and `claimTokenRefund` functions use `isRefund` state variable to mark the user was refunded. The `claimRefund` function doesn't check the caller and user parameter, so the attacker can call this function with the user parameter as the `startup` address value to mark that this address was refunded. Therefore, the `startup` can't call `claimTokenRefund` to get refund tokens anymore.

```
function claimRefund(address user) external  nonReentrant{
        require(refund,"TSVesting: not refund");
        require(isRefund[user]==false, "TSVesting: user claimed");
        isRefund[user] = true;
        uint256 amountRefund = totalBalanceRefund * _balances[user] / total;
        SafeERC20.safeTransfer(IERC20(tokenRefund),user, amountRefund);
    }

    function claimTokenRefund() external nonReentrant{
        require(refund,"TSVesting: not refund");
        require(msg.sender==startup,"TSVesting: only startup call");
        require(isRefund[_msgSender()]==false, "TSVesting: user claimed");
        isRefund[_msgSender()] = true;
        uint256 amountRemaining = total * (BPS - rate) / BPS;
        SafeERC20.safeTransfer(IERC20(_tokenAdd), startup, amountRemaining);
    }
```

UPDATES

- *Oct 17, 2022*: This issue has been acknowledged and fixed by the DStarter team.

2.2.2. TSGovernor.sol - Using normal account to fake numberVote MEDIUM

The `castVote` function doesn't check the caller, so anyone can call this function to increase the `history[proposalId].numberVote`. If some voters have weights larger than the `PERCENT_VOTE_REFUND_SUCCESS`/`PERCENT_VOTE_DISBURMENT_SUCCESS`, they can use another accounts to trigger this value to force the proposalID to success.

```
function castVote(uint256 proposalId, uint8 support) public virtual override returns
(uint256) {
        address voter = _msgSender();
        history[proposalId].numberVote += 1;
        return _castVote(proposalId, voter, support, "");
    }

    function _quorumReached(uint256 proposalId) internal view virtual override(Governor,
GovernorCountingSimple) returns (bool) {
        TSVesting tsVesting = TSVesting(tokenVesting);
        return history[proposalId].numberVote*100 >= PERCENT_VOTE_DISBURMENT_SUCCESS *
tsVesting.totalUserInvest();
    }

    function _voteSucceeded(uint256 proposalId) internal view virtual override(Governor,
GovernorCountingSimple) returns (bool) {
        (, uint256 forVotes, ) = proposalVotes(proposalId);
        TSVesting tsVesting = TSVesting(tokenVesting);
        uint256 pencent = history[proposalId].voteRefund ? PERCENT_VOTE_REFUND_SUCCESS :
PERCENT_VOTE_DISBURMENT_SUCCESS;
        return forVotes * 100 >= pencent * tsVesting.total();
    }

    function state(uint256 proposalId) public view virtual override returns (ProposalState)
{
        ProposalCore storage proposal = _proposals[proposalId];
        ...
        uint256 deadline = proposalDeadline(proposalId);

        if (deadline >= block.number) {
            return ProposalState.Active;
        }

        if (_quorumReached(proposalId) && _voteSucceeded(proposalId)) { //attacker can
bypass `_quorumReached` contraint
            return ProposalState.Succeeded;
        } else {
            return ProposalState.Defeated;
        }
    }
```

UPDATES

- *Oct 17, 2022*: This issue has been acknowledged and fixed by the DStarter team.

2.2.3. TSSaleFatory.sol - Best practice INFORMATIVE

```
function claimToken(uint256 saleIndex, address user)
        external
        indexValid(saleIndex)
        nonReentrant
    {
        ...
        if(tsProject.userIsVc(user)){
            if (saleConfig.status == SaleStatus.SUCCESS) {
                require(amountBuys[saleIndex][user].amount > 0, "TS: balance is zero");
                amountBuys[saleIndex][user].amount = 0;
                TSVesting tsVesting = TSVesting(saleConfig.tsVesting);
                IERC20 token = IERC20(saleConfig.tokenAddress);
                token.safeApprove(saleConfig.tsVesting,
historyBuys[saleIndex][user].amount);
                tsVesting.deposit(user, historyBuys[saleIndex][user].amount);
            } else {
                require(amountBuys[saleIndex][user].amountTokenBuy>0,"TS: balance is
zero");
                amountBuys[saleIndex][user].amountTokenBuy = 0;
                IERC20 token = IERC20(saleConfig.tokenBuy);
                token.safeTransfer(user,
(historyBuys[saleIndex][user].amountTokenBuy/BPS)*(BPS-saleConfig.feeRefund));
        ...
    }
```

The result of `(historyBuys[saleIndex][user].amountTokenBuy/BPS)` may be zero if the value of `BPS` is larger than `historyBuys[saleIndex][user].amountTokenBuy`.

The mul operator should be placed before the div operator.

UPDATES

- *Oct 17, 2022*: This issue has been acknowledged and fixed by the DStarter team in commit `92d3024a46f4d4cb9727fd124eac63900376a879`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 18, 2022*	Public Report	Verichains Lab

Table 2. Report versions history