



SharkTeam

No. 20240201033C

Smart Contract Audit Report

DioneProtocol (Locker-SCs)



Feb. 1st, 2024

Copyright

All rights reserved by SharkTeam. Unless otherwise specified, the copyright or other related rights of any text description, document format, illustration, photo, method, process, etc. in this document belong to SharkTeam. Without the written consent of SharkTeam, no one is allowed to copy, extract, back up, modify, disseminate, translate into other languages or use all or part of this manual for commercial purposes in any way or form.

1. Overview

SharkTeam recently received the requirements for Locker-SCs smart contracts audit.

In this audit, the SharkTeam security experts communicate with Locker-SCs team to conduct smart contract security audit under controllable operation, so as to avoid any risk in the audit process as far as possible.

Project Overview :

Project Name	DioneProtocol(Locker-SCs)
Description	ERC20, Locker for odyssey chain (Dione mainnet)
Language	Solidity
Codebase	Private Repo
MD5	5f8a511f5e2104a898e3008a186c0b92

Audit methods :

Firstly, through static analysis, dynamic analysis and other analysis technologies, the smart contracts in the project were automatically scanned and manually reviewed; after that, the SharkTeam security experts conducted a detailed manual audit of the smart contracts line-by-line. From the four dimensions of high-level language, virtual machine, blockchain, and business logic, nearly 200 audit items of smart contracts have been comprehensively audited.

Core Audit Types:

For smart contract, SharkTeam's audit items cover 4 layers: language, virtual machine, blockchain, and business, and 6 severity levels of critical (abbreviated as

CR) risk, high risk, medium (abbreviated as MED) risk, low risk, informational (abbreviated as INFO) risk and optimizable (abbreviated as OPT) risk. Below are some common (but not all) vulnerabilities, most of which are high-risk or above.

- Function Visibility
- Reentrancy Vulnerability
- Contract and Storage Initialization
- Token Minting and Burning
- Authorization vulnerability and Access Control
- Centralization of Power
- Function Logic Vulnerability
- Flashloan and Price Manipulation
- DAO Proposal Attack
- Contract Upgrade Issues
- Randomness and Revert Attack
- Insufficient Randomness
- Integer Overflow/Underflow
- Divide Before Multiply and Integer Precision
- Unchecked/Unused Return Values
- Blockchain Timestamp Dependency
- Transaction Order Dependency and Front Running
- Denial of Service (DoS)

Audit Scope :

Contract Files	MD5
MultiSigWallet.sol	f3fc8a1653cad030f5b6adfa9ac08b08
MyTestToken.sol	15565683fdd82fbd8ad62116c221f01e
NativeVesting.sol	bf0b6cd81d214a6a058f16d40daed021
TokenVesting.sol	e1d3b461d9832f6e06d11b0607c14d7e
Vesting.sol	6afc0c5361243c714f4c4c0400332703
VestingFactory.sol	c99e5c5187991342ab5e08c4917cfed3

Audit Details**1. MultiSigWallet.sol**

The contract defines a multi-signature wallet that can handle four types of transactions, and contains 3 external business functions that can modify the state:

- `submitTransaction (address _vesting, uint8 _transType, address[] memory _recipients)`

Everyone can submit a new transaction to the chain by calling this function. The type of the transaction must be one of Funds, ReleaseToOwner, ReleaseToRecipients and revoke. After submission, the transaction is waiting for confirmation of owners.

- `confirmTransaction (uint _transactionId)`

This function can only be called by owners and then the transaction will be confirmed by the caller.

- `executeTransaction (uint _transactionId)`

This function can only be called by owners to execute the transaction while it is

confirmed by at least numConfirm owners if the transaction is never executed. In the function, the contract can call different functions of Vesting according the type of the transaction.

2. VestingFactory.sol

The contract defines the creation function of the Vesting contract and contains 3 external functions that can modify state variables.

- create (string memory _name, address _token, uint256 _amount, uint256 _start, uint256 _end, uint256 _rate, uint256 _period, bool _revokable, address[] memory _recipients)

This function can called by everyone and creates a vesting contract that vests its balance of _token to the owner, gradually in a linear fashion until start + duration. By then all of the balance will have vested.

- createWithMultisigWallet (string memory _name, address _token, uint256 _amount, uint256 _start, uint256 _end, uint256 _rate, uint256 _period, bool _revokable, address[] memory _recipients, address[] memory _owners, uint8 _numConfirmationRequired)

This function can called by everyone and also create contracts similar to those created by the create function. at the same time, create a new multi-signature contract and set the state variable multiSignContract of the new Vesting contract to the new multi-signature contract address.

- setFee (address _treasury, uint256 _fee)

This function just sets the values of the state variables treasury and fee and can only

be called by the owner.

3. MyTestToken.sol

This contract defines a standard ERC20 contract.

4. Vesting.sol, NativeVesting.sol and TokenVesting.sol

Vesting Contract is an abstract contract and contains 5 external functions that can modify state variables:

- `setMultisigContract (address _multiSignContract)`

This function can be called by everyone and set the state variable `multiSignContract` to the new value `_multiSignContract`. This function will be called in the `VestingFactory` contract and save the value to the backend. Then the state variable `multiSignContract` will no longer be read.

- `setOwner (address _newOwner)`

This function can only be called by the owner and set the state variable to the new value `_newOwner`.

- `release ()`

This function can release the token which has already vested but hasn't been released yet to the owner.

- `release (address[] memory _recipients)`

This function can release the token which has already vested but hasn't been released yet to `_recipients`.

- `claim ()`

This function can release the token which has already vested but hasn't been

released yet to the caller which must be a recipient of this contract.

Audit results :

Locker-SCs smart contracts audit results: **Pass**.

SharkTeam

2. Findings

2.1 Summary

Vulnerability list :

ID	Item	Severity	Category	Status
1	For-Loop-Logic-Error	■ HIGH	Language	Fixed
2	Missing-Parameter-Check	■ LOW	Language	Fixed
3	Missing-Token-Balance-Check	■ LOW	Language	Fixed
4	Missing-Else-Logic	■ LOW	Language	Fixed
5	Use-Interfaces-Instead-of-Contracts	■ OPT	Language	Fixed
6	Centralization-Risk	■ INFO	Business	⚠ Unfixed

2.2 Detailed Results

2.2.1 Centralization-Risk [Info]

Description:

The owners (deploy addresses) are centralized high-privilege addresses, these addresses can update some key state variables. Once the private keys are leaked or stolen, the project will suffer a devastating blow.

Recommendation:

Properly keep the private keys of the high-privilege addresses secure. And try to avoid the loss, leakage, theft, etc. of any private key as much as possible.

Appendix A: Smart Contract Audit Items

ID	Item	Category	Severity
TVE-001	Different-Pragma-Directives-Are-Used	Language	INFO
TVE-002	Incorrect-Versions-of-Solidity	Language	OPT
TVE-003	Solidity-Version-Is-Outdated	Language	INFO
TVE-004	Reentrancy-Eth-Vulnerabilities	VM	CR
TVE-005	Reentrancy-No-Eth-Vulnerabilities	VM	MED
TVE-006	Reentrancy-Benign-Vulnerabilities	VM	LOW
TVE-007	Reentrancy-Events-Vulnerabilities	VM	LOW
TVE-008	Reentrancy-Unlimited-Gas-Vulnerabilities	VM	OPT
TVE-009	Erc777-Callbacks-And-Reentrancy	Language	CR
TVE-010	State-Variable-Shadowing	Language	HIGH
TVE-011	State-Variable-Shadowing-From-Abstract-Contracts	Language	MED
TVE-012	Builtin-Symbol-Shadowing	Language	LOW
TVE-013	Local-Variable-Shadowing	Language	LOW
TVE-014	Uninitialized-Local-Variables	Language	MED
TVE-015	Uninitialized-Storage-Variables	Language	HIGH
TVE-016	Uninitialized-State-Variables	Language	HIGH
TVE-017	Dos-Attack-Call-Failed	Language	MED
TVE-018	DoS-With-Block-Gas-Limit	VM	MED
TVE-019	Unused-State-Variable	Language	OPT

TVE-020	Variable-Names-Too-Similar	Language	INFO
TVE-021	State-Variables-Could-Be-Declared-Constant	Business	OPT
TVE-022	Local-Variables-Are-Not-Used	Language	OPT
TVE-023	Unrestricted-State-Variable-Writing	Language	HIGH
TVE-024	Arbitrary-Jump-Function-Type-Variable	Language	MED
TVE-025	State-Variable-Access-Permissions-Default	Language	INFO
TVE-026	Variable-Classification-And-Sorting	Business	HIGH
TVE-027	Dangerous-State-Variable-Shadowing	Language	HIGH
TVE-028	Modifier-To-Modify-State-Variables	Language	HIGH
TVE-029	There-Are-External-Calls-In-The-Modifier	Language	HIGH
TVE-030	Incorrect-Modifier	Language	LOW
TVE-031	Multiple-Constructor-Schemes	Language	HIGH
TVE-032	Reused-Base-Constructors	Language	MED
TVE-033	Void-Constructor	Language	LOW
TVE-034	Incorrect-Constructor-Name	Language	LOW
TVE-035	Suicidal	Language	HIGH
TVE-036	Fallback-And-Receive()	Language	HIGH
TVE-037	Function-Initializing-State	Language	INFO
TVE-038	Unimplemented-Functions	Language	OPT
TVE-039	Public-Function-Could-Be-Declared-External	Business	OPT
TVE-040	Function-Default-Permissions	Language	INFO

TVE-041	Unprotected-Withdraw-Function	Language	CR
TVE-042	Unchecked-Send	Language	MED
TVE-043	Unchecked-Transfer	Language	HIGH
TVE-044	Missing-Events-Access-Control	Language	LOW
TVE-045	Missing-Events-Arithmetic	Language	LOW
TVE-046	Unindexed-Erc20-Event-Oarameters	Language	INFO
TVE-047	Incorrect-Erc20-Interface	Business	MED
TVE-048	Incorrect-Erc721-Interface	Business	MED
TVE-049	Erc20-Approve()-Race-Condition	Language	CR
TVE-050	Costly-Operations-Inside-A-Loop	Language	OPT
TVE-051	Calls-Inside-A-Loop	Language	LOW
TVE-052	Unchecked-Low-Level-Calls	Language	MED
TVE-053	Low-Level-Calls	Language	INFO
TVE-054	Controlled-Delegatecall	VM	HIGH
TVE-055	Message-Call-With-Hard-Coded-Gas-Number	VM	LOW
TVE-056	Public-Mappings-With-Nested-Variables	Language	HIGH
TVE-057	Deletion-On-Mapping-Containing-A-Structure	Language	MED
TVE-058	Functions-Send-Ether-To-Arbitrary-Destinations	Language	CR
TVE-059	Missing-Zero-Address-Validation	Language	LOW
TVE-060	Critical-Address-Change	Language	INFO
TVE-061	Signature-Replay	VM	CR

TVE-062	Lack-of-Protection-From-Signature-Replay-Attacks	VM	CR
TVE-063	Redundant-Statements	Language	OPT
TVE-064	Unreached-Code	Language	OPT
TVE-065	Code-Not-Achieve-The-Desired-Effect	Language	LOW
TVE-066	Weak-Prng	Blockchain	HIGH
TVE-067	Block-Timestamp	Blockchain	LOW
TVE-068	Block-Values-As-Time-Proxies	Blockchain	HIGH
TVE-069	Missing-Inheritance	Language	OPT
TVE-070	Incorrect-Order-of-Inheritance	Language	LOW
TVE-071	Whether-To-Inherit	Business	LOW
TVE-072	Boolean-Equality	Language	INFO
TVE-073	Misuse-of-A-Boolean-Constant	Language	MED
TVE-074	Tautology-Or-Contradiction	Language	MED
TVE-075	Dangerous-Strict-Equalities	Language	MED
TVE-076	Dangerous-Unary-Expressions	Language	LOW
TVE-077	Misuse-of-Assert	Language	OPT
TVE-078	Dangerous-Usage-of-tx.origin	Language	MED
TVE-079	Unexpected-Ether-And-this.balance	Language	MED
TVE-080	Integer-Overflow	Language	CR
TVE-081	Divide-Before-Multiply	Language	MED
TVE-082	Too-Many-Digits	Language	INFO

TVE-083	Dirty-High-Order-Bits	Language	LOW
TVE-084	Modifying-Storage-Array-By-Value	Language	HIGH
TVE-085	Array-Length-Assignment	Language	HIGH
TVE-086	Incorrect-Shift-In-Assembly	Language	HIGH
TVE-087	Name-Reused	Language	HIGH
TVE-088	Right-To-Left-Override-Character	Language	HIGH
TVE-089	Unprotected-Upgradeable-Contract	Language	HIGH
TVE-090	Contracts-That-Lock-Ether	Business	HIGH
TVE-091	Unused-Return	Business	MED
TVE-092	Assembly-Usage	Language	INFO
TVE-093	Deprecated-Standards	Language	OPT
TVE-094	Conformance-To-Solidity-Naming-Conventions	Language	OPT
TVE-095	Hash-Collision-With-Multiple-Variable-Length-Parameters	VM	HIGH
TVE-096	Lack-of-Proper-Signature-Verification	VM	HIGH
TVE-097	Insufficient-Gas	VM	LOW
TVE-098	Private-On-Chain-Data	Business	LOW
TVE-099	Condition-Violation	Language	LOW
TVE-100	Write-Repeatedly	Language	MED
TVE-101	Incorrect-Access-Control	Language	HIGH
TVE-102	Transaction-Order-Dependence	Blockchain	HIGH
TVE-103	Contract-Check	Language	LOW

TVE-104	Deprecated-Keywords	Language	OPT
TVE-105	Unprotected-Initializer-In-Upgradeable-Contract	Business	CR
TVE-106	Initialize-State-Variables-In-Upgradeable-Contract	Business	HIGH
TVE-107	Import-Upgradeable-Contracts	Business	HIGH
TVE-108	Avoid-Using-Selfdestruct-Or-Delegatecall-In-Upgradeable-Contracts	Business	HIGH
TVE-109	State-Variables-In-Upgradable-Contracts	Business	HIGH
TVE-110	Function-Id-Collision-Between-Agents-And-Upgradable-Contracts	Business	HIGH
TVE-111	Functions-Shadowing	Business	HIGH
TVE-112	Initialization-Function-Is-Called-Multiple-Times	Business	HIGH
TVE-113	Initialization-Function-of-The-Proxy-Contract-Is-Not-Called	Business	LOW
TVE-114	Check-the-Content-That-Must-Be-Initialized-During-Deployment-Combined-With-Business	Business	LOW
TVE-115	Check-For-Required-Initialization	Business	LOW
TVE-116	Check-Whether-The-Initialization-Function-Conforms-To-Openzeppelin	Business	LOW
TVE-117	Variables-That-Should-Not-Be-Constant	Language	LOW
TVE-118	Initialize-Functions-Are-Not-Called	Language	LOW
TVE-119	Initializer()-Is-Not-Called	Language	LOW
TVE-120	Incorrect-Variables-With-The-V2	Language	LOW
TVE-121	Incorrect-Variables-With-The-Proxy	Language	LOW
TVE-122	State-Variable-Initialized	Language	LOW
TVE-123	Variables-That-Should-Be-Constant	Language	LOW
TVE-124	Extra-Variables-In-The-Proxy	Language	LOW

TVE-125	Missing-Variables	Language	LOW
TVE-126	Extra-Variables-In-The-V2	Language	LOW
TVE-127	Initializable-Is-Not-Inherited	Language	LOW
TVE-128	Initializable-Is-Missing	Language	LOW
TVE-129	Initialize-Function	Language	LOW
TVE-130	Initializer()-Is-Missing	Language	HIGH
TVE-131	Abiencoderv2-Array	Language	HIGH
TVE-132	Storage-Type-Signed-Integer-Array-Error	Language	HIGH
TVE-133	Enumeration-Conversion	Language	MED
TVE-134	Constant-Function-Using-Assembly-Code	Language	MED
TVE-135	Constant-Function-To-Modify-State-Variables	Language	MED
TVE-136	Uninitialized-Function-Pointer-In-The-Constructor	Language	LOW
TVE-137	Pre-Declared-Usage-of-Local-Variables	Language	LOW
TVE-138	Implicit-Constructor-Callvalue-Check	Language	MED
TVE-139	Incorrect-Event-Signature-In-Library	VM-layer	LOW
TVE-140	Call-An-Uninitialized-Function-Pointer-In-The-Constructor	Language	LOW
TVE-141	Dynamic-Constructor-Parameters-Are-Abiencoderv2	Language	LOW
TVE-142	Storage-Array-of-Multi-Slot-Elements-With-Abiencoderv2	Language	LOW
TVE-143	Use-Abiencoderv2-To-Read-Calldata-Structure-Containing-Static-Size-And-Dynamic-Encoding-Members	Language	LOW
TVE-144	Package-Storage-Using-abiencoderv2	Language	LOW
TVE-145	Incorrect-Loading-Using-Yul-Optimizer-And-Abiencoderv2	Language	LOW

TVE-146	Use-Abiencoderv2-To-Dynamically-Encode-Base-Type-Array-Slices	Language	LOW
TVE-147	When-Using-Abiencoderv2-The-Formatting-Process-Lacks-Escaping	Language	LOW
TVE-148	Double-Shift-Overflow	Language	HIGH
TVE-149	Incorrect-Byte-Instruction-Optimization	Language	LOW
TVE-150	Use-Yul-Optimizer-To-Remove-Necessary-Assignments	Language	LOW
TVE-151	Private-Method-Is-Overridden	Language	LOW
TVE-152	Multi-Stack-Slot-Component-of-Tuple-Assignment	Language	LOW
TVE-153	Dynamic-Array-Cleanup	Language	LOW
TVE-154	Empty-Byte-Array-Copy	Language	LOW
TVE-155	Memory-Array-Creation-Overflow	Language	LOW
TVE-156	Calldata-Using-For	Language	LOW
TVE-157	Free-Function-Redefinition	Language	LOW
TVE-158	Token-Standard	Business	LOW
TVE-159	Asset-Lock	Business	CR
TVE-160	High-Authority-Address-Check	Business	CR
TVE-161	DAO-Proposal-Attack	Business	CR
TVE-162	Flash-Loan-Attack	Business	CR
TVE-163	Manipulating-Price-Oracles	Business	CR
TVE-164	Minting-And-Burning-Vulnerability	Business	HIGH
TVE-165	Exchange-Business-Vulnerability	Business	HIGH
TVE-166	Insufficient-Liquidity	Business	HIGH

TVE-167	Lending-Business-Vulnerability	Business	HIGH
TVE-168	Aggregate-Revenue-Vulnerability	Business	HIGH
TVE-169	Single-Currency-Pledge-Vulnerability	Business	HIGH
TVE-170	Referral-Reward-Vulnerability	Business	HIGH
TVE-171	Cross-Platform-Trading-Vulnerability	Business	HIGH
TVE-172	Standard-Library-Functions-Vulnerability	Business	HIGH

SharkTeam

Appendix B: Vulnerability Severity Classification

- ■ Critical Risk, abbreviated as CR,
- ■ High Risk, abbreviated as HIGH,
- ■ Medium Risk, abbreviated as MED,
- ■ Low Risk, abbreviated as LOW,
- ■ Informational Risk, abbreviated as INFO,
- ■ Optimizable Risk, abbreviated as OPT.

Rating is mainly based on the degree of harm and difficulty of utilization of the risk, supplemented by a comprehensive judgment of other factors.

The degree of harm is mainly defined based on the impact on the three dimensions of asset ownership, business integrity impact and contract availability, and is divided into five harm levels: severe harm, high harm, medium harm, low harm and potential harm;

Exploitation difficulty is mainly defined based on the three dimensions of attack vector, attack complexity, and authentication, and is divided into four difficulty levels: low difficulty, medium difficulty, high difficulty, and extremely high difficulty.

	Critical	High	Medium	Low	Potential
Low	CR	HIGH	MED	LOW	INFO/OPT
Medium	HIGH	MED	LOW	LOW	INFO/OPT
High	MED	LOW	LOW	INFO/OPT	INFO/OPT
Ex-high	LOW	LOW	INFO/OPT	INFO/OPT	INFO/OPT

Disclaimer

This document is based on currently available information and its contents are subject to change without prior notice.

SharkTeam has tried the best to ensure the accuracy and reliability of the content when writing this report, but SharkTeam will not be responsible for the loss and damage caused by the omission, inaccuracy or error in this report. The safety audit analysis and other contents of this report are based on the materials provided by the project team. This audit only focuses on the audit items provided in this report, and other unknown security vulnerabilities are not within the scope of this audit. SharkTeam cannot determine the security status of facts that appear or exist after the report. SharkTeam is not responsible for the background or other circumstances of the project.

The content, acquisition method, use and any services or resources involved in this report cannot be used as a basis for any form of investment, tax, legal, regulatory and advice, and will not create relevant responsibilities.

About SharkTeam

SharkTeam's vision is to protect the security of the Web3 world. The team is composed of experienced security professionals and senior researchers from all over the world. SharkTeam is proficient in the underlying theory of blockchain and smart contracts, and provides services including on-chain analysis, risk detection and alerts, smart contract auditing, crypto asset recovery, etc.

SharkTeam also developed the AI-driven risk detection and alerting platform ChainAegis, supporting unlimited levels of in-depth graph analysis. This capability effectively combats the risk of Advanced Persistent Threat (APT) in the Web3 world. SharkTeam has established long-term cooperative relationships with key players in various fields of the Web3 world, such as Polkadot, Moonbeam, Polygon, Sui, OKX, imToken, Collab.Land, etc.

We implement almost 200 auditing contents that cover four aspects: high-level language layer, virtual machine layer, blockchain layer, and business logic layer, ensuring that smart contracts are completely guaranteed and Safe.



SharkTeam

In Math, We Trust!



<https://sharkteam.org>



<https://twitter.com/sharkteamorg>



<https://discord.com/invite/jGH9xXCjDZ>



<https://t.me/sharkteamorg>