DSLA Protocol

SMART CONTRACT AUDIT

28.04.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Stacktical SAS. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (15.03.2021)	Layout
0.5 (16.03.2021)	Verify Claims and Test Deployment
0.6 (17.03.2021)	Testing SWC Checks
0.8 (19.03.2021)	Automated Security Testing Manual Security Testing
0.9 (20.03.2021)	Summary and Recommendation
1.0 (22.03.2021)	Rechecking after contract update
1.1 (24.03.2021)	Final document
1.2 (28.04.2021)	Added deployed contract

2. About the Project and Company

Company address:

Stacktical SAS
3 Boulevard de Sébastopol
75001 Paris
FRANCE

Website: https://stacktical.com

Twitter: https://twitter.com/stacktical

Reddit: https://www.reddit.com/r/Stacktical

Telegram: https://t.me/stacktical

Youtube: https://www.youtube.com/channel/UCG1S3V4AbJK_YOZa9OOZykw

Facebook: https://facebook.com/stacktical

Instagram: https://www.instagram.com/stacktical

Blog: https://blog.stacktical.com

2.1 Project Overview

Reliability and speed are fundamental features of any online service, and to keep users satisfied, platforms must be consistently bug-free and reliable. Today, it is hard for service providers to know how to scale their platforms, and Stacktical uses data analytics and AI to help predict performance management focused on whether or not projects can scale effectively. In addition to helping online services predict scalability, Stacktical wants to help ensure that users are compensated when platforms fail; this is where the blockchain comes into play.

Stacktical connects performance management via data analytics to a decentralized SLA platform. They are providing smart contracts on the Ethereum blockchain to ensure users get compensated when a service doesn't perform. The smart contracts can facilitate, verify, and execute the terms of the SLAs and enable stakeholders to automate and externalize the efforts involved in establishing and enforcing SLAs, all while automating the settlement of any violations for users. This is a great use case for blockchain because today this is hard to enforce and monitor SLAs. In fact, in the event that AWS is down, AWS asks users to bring their own data to prove that the platform was not reliable. How can users do this transparently?

The team, has experience in the scalability infrastructure space, and have a working product for their predictive scalability auditing platform. The token itself is a utility model, and it allows users to get compensation for failed platforms as well as incentivize service providers through tokens when they maintain their SLAs.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

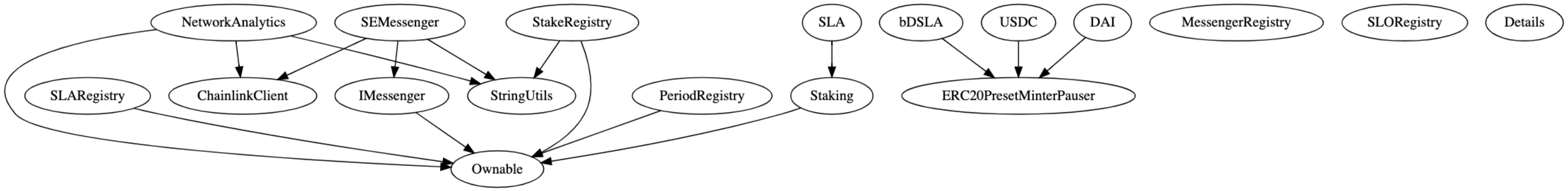
4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@chainlink/contracts/src/v0.6/ChainlinkClient.sol	https://github.com/smartcontractkit/chainlink/blob/develop/evm-contracts/src/v0.6/ChainlinkClient.sol
@chainlink/contracts/src/v0.6/PreCoordinator.sol	https://github.com/smartcontractkit/chainlink/blob/develop/evm-contracts/src/v0.6/PreCoordinator.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/access/Ownable.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/math/SafeMath.sol
@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/presets/ERC20PresetMinterPauser.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/SafeERC20.sol

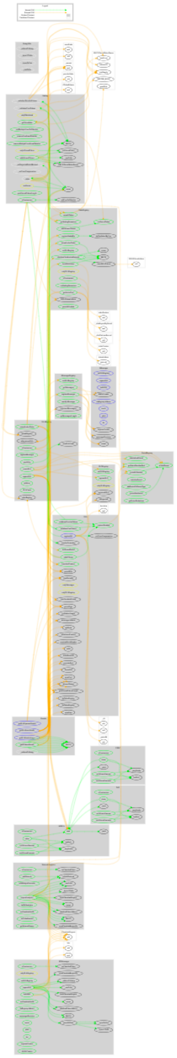
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
contracts/Staking.sol	c7dfef14531e4caeef4585e7fe74bb97
contracts/SLARegistry.sol	4bf7a14a577ff8792cfdcbbc0b423ac1
contracts/tokens/bDSLA.sol	3a27e062a0ddaff3eb1edda41580f14b
contracts/tokens/USDC.sol	a2f9cbe0b5c1c2b2706a782447195eb2
contracts/tokens/DAI.sol	6b63f0903296ff187359a0d03784c0a5
contracts/StakeRegistry.sol	0b108453ec037171321ecaaec085e78d
contracts/StringUtils.sol	e9f33aeba42cb64fbbc97011389e0bf1
contracts/messenger/IMessenger.sol	7731395981ce5f87489a66f7df8470bb
contracts/staking-efficiency/NetworkAnalytics.sol	c56bfafbbe02e9f92c745b2a1b03c29c
contracts/staking-efficiency/SEMessenger.sol	7afcb0defb4df06acc55ca35dd5fd8f9
contracts/staking-efficiency/PreCoordinator.sol	8e06aa3f7735e1072eed95d2b5765652
contracts/PeriodRegistry.sol	140da8af2ff23af2b14df8dcc3e59d8a
contracts/MessengerRegistry.sol	62c0d7a9e0c4abd8d7156ed9ccd0e308
contracts/SLORegistry.sol	e54a6b00ef3d346bac289411ad0f4835
contracts/SLA.sol	5a3bfbc750f59e562938900a674ea721
contracts/Details.sol	b74ba6fc9475b277c5e69417767705d5

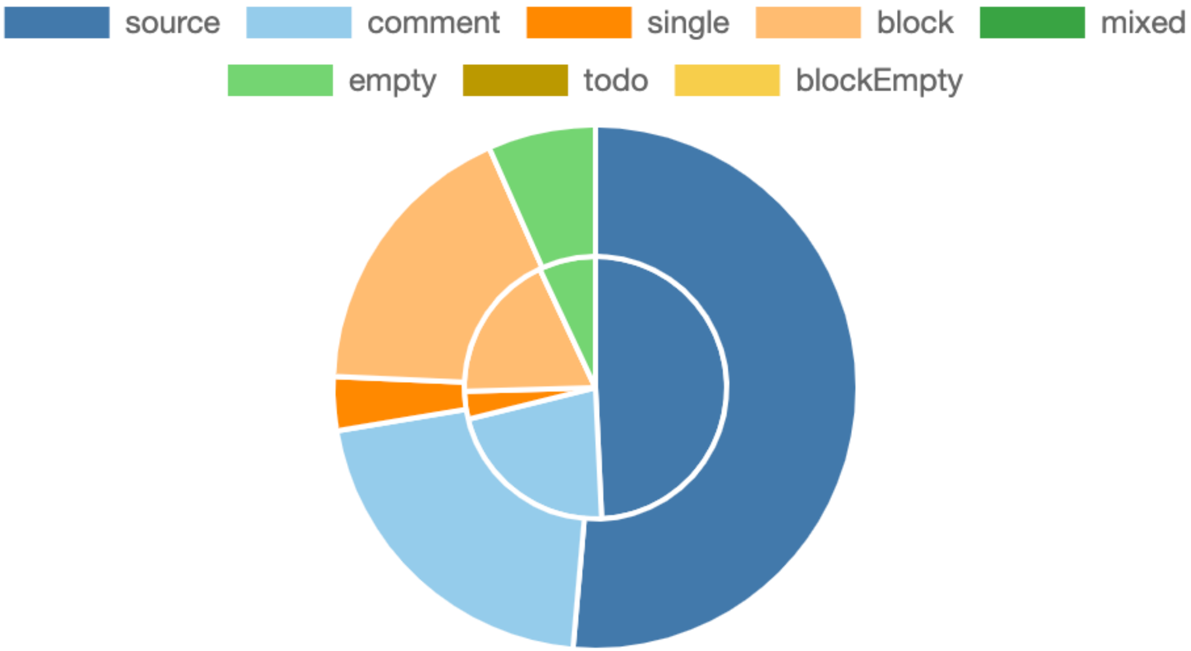


4.4 Metrics / CallGraph



Full Version: http://chainsulting.de/wp-content/uploads/2021/03/stacktical_solidity-metrics.html

4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`0.6.6`	`ABIEncoderV2`		**** (0 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
			`yes`		`yes` → `NewContract:SLA` → `NewContract:ERC20PresetMinterPauser`

🌐Public	💰Payable
109	0

External	Internal	Private	Pure	View
14	80	0	7	46

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/Staking.sol	1		419	390	269	83	181	
📝	contracts/SLARegistry.sol	1		281	264	166	78	111	🧪🌀
📝	contracts/tokens/bDSLA.sol	1		130	127	49	70	39	
📝	contracts/tokens/USDC.sol	1		131	128	49	70	39	
📝	contracts/tokens/DAI.sol	1		130	127	49	70	39	
📝	contracts/StakeRegistry.sol	1		420	377	276	71	175	🧪🧮🌀
📝	contracts/StringUtils.sol	1		95	79	54	21	66	
🎨	contracts/messenger/IMessenger.sol	1		94	33	14	48	23	
📝	contracts/staking-efficiency/NetworkAnalytics.sol	1		271	255	155	74	102	🧪
📝	contracts/staking-efficiency/SEMessenger.sol	1		290	274	168	79	131	🧪
	contracts/staking-efficiency/PreCoordinator.sol			3	3	2			

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/PeriodRegistry.sol	1		206	174	107	53	67	🧪
📝	contracts/MessengerRegistry.sol	1		160	153	119	20	54	🧪
📝	contracts/SLORegistry.sol	1		101	93	62	18	22	🧪
📝	contracts/SLA.sol	1		250	245	146	69	80	🧪
📝	contracts/Details.sol	1		198	149	125	21	154	🧪
📝 🎨	**Totals**	**15**		**3179**	**2871**	**1810**	**845**	**1283**	🧪🧮🌀

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

5. Scope of Work

The Stacktical Team provided us with the files that needs to be tested. The scope of the audit are the Stacktical Protocol contracts.

Following contracts with the direct imports has been tested:

- SLA.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- Providers stake DSLA tokens to pay periodic verifications. After a DSLA Period is finished, it can be verified. The verification fee is split between: The User doing the verification 25% and 25% goes to the SLA's Messenger owner, to cover the expenses of calling this function, and to incentivize a fast verification after the period is finished.
- The SLA contract stake can be whitelisted
- The users can stake at any period, if the contract is not finished.
- The provider can stake at any period, if the contract is not finished.
- The users can only withdraw stake after the contract is finished.
- The provider can withdraw stake at any time, as long as his pool is greater than or equal to the users pool to enforce hedge after an eventually contract breach.
- Only the SLA contract owner (i.e. the address associated to the contract creation transaction) can stake on the provider pool.
- User is able to withdraw a DSLA, USDC and DAI stake from an expired or breached DSLA contract
- Provider is able to withdraw a DSLA, USDC and DAI stake as long as the total user stake is below the provide stake, whilst the contract is not finished. If the contract is finished then the provider can withdraw all of his stake, since the last verification of the rewards for the last period was calculated, or the compensation was distributed to the users, in case that the contract was breached.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting’s experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, Chainsulting’s experts found **no Low issues** in the code of the smart contract.

INFORMATIONAL ISSUES

5.1.1 Equal operator on a boolean

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: SLA.sol

Attack / Description	Code Snippet	Result/Recommendation
It is not necessary to use equal operator on a boolean. It leads to bad code quality.	Line 212 isContractFinished == false, Line 238 Require(isContractFinished == true)	It is recommended to use only the variable name to reduce code as much as possible (i.e. isContractFinished).

5.1.2 Same require check in multiple contexts

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: SLA.sol

Attack / Description	Code Snippet	Result/Recommendation
Usage of same require check in multiple contexts.	Line 209, 223 & 235 Require (amount > 0, „amount cannot be 0“)	It is recommended to use a modifier for these checks to avoid writing same code multiple times.

5.1.3 Not used code in comments.

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: SLA.sol

Attack / Description	Code Snippet	Result/Recommendation
Existence of not used code in comments.	Line 165 – 173 Out commented code	Delete code, which is not used for advanced code quality.

5.1.4 Equal operator on a boolean

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: Staking.sol

Attack / Description	Code Snippet	Result/Recommendation
It is not necessary to use equal operator on a boolean. It leads to bad code quality.	Line 148 & 150 Require (isAllowedToken == false)	It is recommended to use only the variable name to reduce code as much as possible (i.e. !isTokenAllowed).

5.1.5 Equal operator on a boolean

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: StakeRegistry.sol

Attack / Description	Code Snippet	Result/Recommendation
It is not necessary to use equal operator on a boolean. It leads to bad code quality.	Line 151 require(isAllowedToken(_tokenAddress) == false) Line 190: slaRegistry.isRegisteredSLA(msg.sender) == true Line 193: slaWasStakedByUser(_owner, msg.sender) == false Line 209: slaRegistry.isRegisteredSLA(msg.sender) == true Line 247: lockedValue.verifiedPeriods[_periodId] == false	It is recommended to use only the variable name to reduce code as much as possible (i.e. !isAllowedToken).

5.1.6 Equal operator on a boolean
Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: SLARegistry

Attack / Description	Code Snippet	Result/Recommendation
It is not necessary to use equal operator on a boolean. It leads to bad code quality.	Line 161 breachedContract == false, Line 166 slaAllowedPeriodId == true, Line173 periodFinished == true Line 202 _sla.breachedContract() == true	It is recommended to use only the variable name to reduce code as much as possible (i.e. breachedContract).

5.1.7 Equal operator on a boolean

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: PeriodRegistry.sol

Attack / Description	Code Snippet	Result/Recommendation
It is not necessary to use equal operator on a boolean. It leads to bad code quality.	Line 49 periodDefinition.initialized == false Line 90 periodDefinition.initialized == true Line 163 & line 181: isValidPeriod(_periodType, _periodId) == true,	It is recommended to use only the variable name to reduce code as much as possible (i.e. periodDefinition.initialized).

5.1.8 Not used code in comments.

Severity: INFORMATIONAL
Status: Acknowledge
File(s) affected: Details.sol

Attack / Description	Code Snippet	Result/Recommendation
Existence of not used code in comments.	Line 171 - 180 Out commented code	Delete code, which is not used for advanced code quality.

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	☑
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	☑
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	☑
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	☑
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	☑
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	☑
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	☑
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	☑
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	☑
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	☑

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✅
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6. Test Deployment

6.1 Deployment of Contracts

Tx: https://kovan.etherscan.io/tx/0x033add98dfe66be2b80c3f1925549317ecdb02375d9c4bef9ca2332c7d5c25b1
Contract: https://kovan.etherscan.io/address/0x5e42e130efa20531a58a2daf55544be91900ff47

6.2 Deployment bDSLA

Tx: https://kovan.etherscan.io/tx/0x072be471ff25589af6b11060fab85c03bfb6ab8d8cf36ddfb8db40b55f3ee274
Contract: https://kovan.etherscan.io/address/0xe94db60a8870dc6a98dd9dbc54e7ff45d270ded6

6.3.Deployment PeriodRegistry

Tx: https://kovan.etherscan.io/tx/0x5ccbf2ac0e9c86197ef294d58fee51f1cd020b58efe7c3280208196833b3a6b0
Contract: https://kovan.etherscan.io/address/0xe7e0513835585a9cbf785b8cf49dab21977b0271

6.4.Deployment SLORegistry

Tx: https://kovan.etherscan.io/tx/0x2efad17c87adc348850b6ad50d8721753bfe9177e05eabdcf5b8635b13e88c8b
Contract: https://kovan.etherscan.io/address/0x50dbff93ddd7f0fcc1deb286563b3a2476341de5

6.5 Deployment MessengerRegistry

Tx: https://kovan.etherscan.io/tx/0x46d80d790317797c852b116ac7e0d831297b6b47d276efef8d5f7babadf727b0
Contract: https://kovan.etherscan.io/address/0x1af098514d6db2d7be2d528fbfe786fbb8cf2894

6.6 Deploy StakeRegistry

Tx: https://kovan.etherscan.io/tx/0x1b6bfa689e9bd9a8ff0d65c6fe67b032515d54a53b3a587c0f688dc633946af3
Contract: https://kovan.etherscan.io/address/0xb8d4d6e8e73a2bbe46f39ca2d75286edf28c0d72

6.7 Deploy SLARegistry

Tx: https://kovan.etherscan.io/tx/0x825ea4c29804fa50b30858bbf869249eaed29974e68dde6f0eb817a9fad6fbea
Contract: https://kovan.etherscan.io/address/0x96fb3fe8af16c566515da25bf1c3c5ffbef01c7f

6.8 Deployment PreCoordinator

Tx: https://kovan.etherscan.io/tx/0x8e53803067bce3e441582c2d343d491e4dd55e49bcbd514bb83230e5d4fcf952
Contract: https://kovan.etherscan.io/address/0x2a074a5e32f84fa552a0d4a9a389cc482a809071

6.9 Deployment NetworkAnalytics

Tx: https://kovan.etherscan.io/tx/0x11e8b3431b98fac5751b0f27d03d5fbdb6ad3e65ce70c7313d451e30cb1768b0
Contract: https://kovan.etherscan.io/address/0x8c161818017806388007bd76828d00855bca3a363

6.10 Deployment SEManager

Tx: https://kovan.etherscan.io/tx/0x0bfd8af427aa10fd7c41e4eb1993174bb2aae1c04257ef541f02bf7ce387680f
Contract: https://kovan.etherscan.io/address/0x7f31ef4e85a0f1593e1e2054fff879d44af02390

6.11 Deployment Token

DAI
Tx: https://kovan.etherscan.io/tx/0xffd87b9b9ba6d74d827269bc5bf84e0b4e6d6fa8692c7ace4db26e95573d41a2
Contract: https://kovan.etherscan.io/address/0xad23a45e6737bc7e0a71c6293a87bfa9232313a2

USDC
Tx: https://kovan.etherscan.io/tx/0xdd962bf7bb7e9f65f3c9a22c91b4d6fc08612fa7fef7c1d10c955f9a854e3e30
Contract: https://kovan.etherscan.io/address/0x98269d85b2b1b8d472e5e06c940c8159c58a38cc

6.12 Minting bDSLA tokens

Everyone can mint bDSLA tokens.
Owner minting tokens:
Owner mints 10 000 000 bDSLA tokens
Tx: https://kovan.etherscan.io/tx/0xb7485da3e539c8fc9b76a91a011649db55a19240dbb700a5bbf24dae9710d2c5

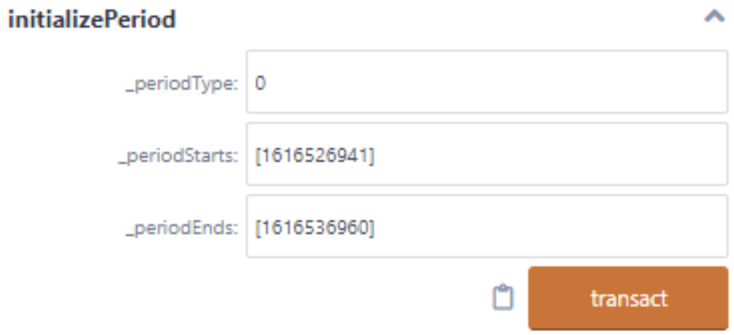
User minting tokens
Not owner mints 10 000 000 bDSLA tokens
Tx: https://kovan.etherscan.io/tx/0xbe0a37dd04a4bf15ea5aa1589c7a6f9b3949aba4d371b49b9dcb4c36ee87162f

6.13 Creating SLA with whitelist

Initialize new period in PeriodRegistry

Only the owner can initialize a new time period.

Tx: https://kovan.etherscan.io/tx/0xaa253443d9c56515a6823a4c5d865e5f20b9ce6c4a161755ef348f7751009388



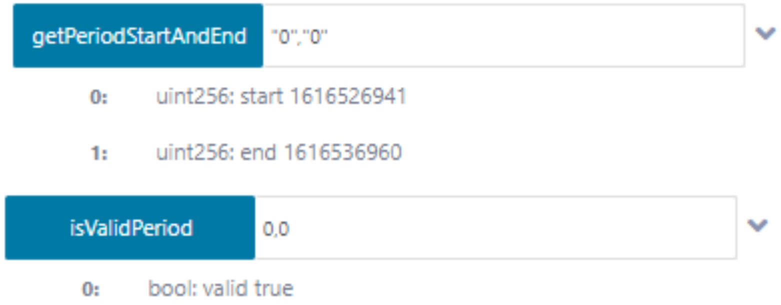
initializePeriod

_periodType: 0

_periodStarts: [1616526941]

_periodEnds: [1616536960]

Valid time period created.



getPeriodStartAndEnd "0","0"

0: uint256: start 1616526941

1: uint256: end 1616536960

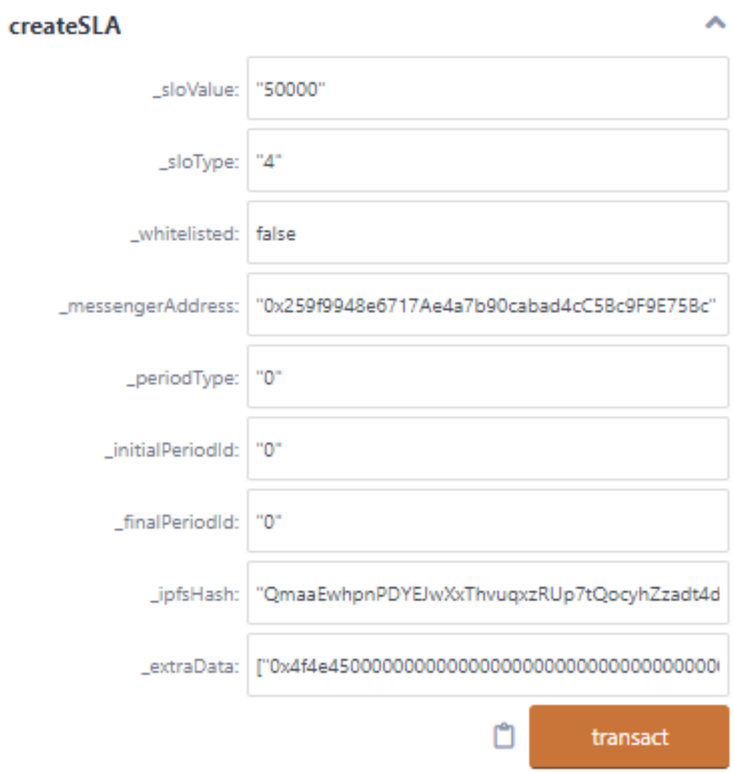
isValidPeriod 0,0

0: bool: valid true

6.14 Create SLA in SLARegistry

Sends initial stake of 20 000 bDSLA tokens on provider stake.

Tx: https://kovan.etherscan.io/tx/0xe22b4d198faf8182ac30221aa67fe7770d800cb57138cbf01bafd72605a2461c



createSLA

_sloValue:	"50000"
_sloType:	"4"
_whitelisted:	false
_messengerAddress:	"0x259f9948e6717Ae4a7b90cabad4cC58c9F9E758c"
_periodType:	"0"
_initialPeriodId:	"0"
_finalPeriodId:	"0"
_ipfsHash:	"QmaaEwhpnPDYEJwXxThvuqxzRUp7tQocyhZzadt4d
_extraData:	["0x4f4e450000000000000000000000000000000000

6.15 Provider stakes tokens

Owner stakes bDSLA and receives the same amount of DSLA-PROVIDER-bDSLA-0. The provider can stake at any period, if the contract is not finished. Only the provider (contract owner) can stake on provider pool.

6.16 User stakes bDSLA and receives the same amount of DSLA-USER-bDSLA-0.

Users can stake at any period, if the contract is not finished. Users can only stake on users pool.
Cannot stake more than SLA provider stake
The user cannot stake more tokens than the SLA provider staked
Tx: https://kovan.etherscan.io/tx/0xb64db68d794d0eb8b2a9661861f03110e18503e2f8154d87377735208535b138

6.17 Try to stake on finished contract

Transaction gets reverted. Error: Can only stake on not finished contracts.
Tx: https://kovan.etherscan.io/tx/0xe838e4051dd55d13b32505cffd75f579f62cc7b7818a5ffb62abf92e8338032b

6.18 Stake with not whitelisted user

Transactions from not whitelisted users gets reverted. Error: User is not whitelisted.
Tx: https://kovan.etherscan.io/tx/0x41572d6d68d1a9511bf0dce79ecfdf6a29212d399d15295131a2f2e795bb6d92

6.19 Stake with whitelisted provider

Provider stakes 20 000 bDSLA and receives 20 000 DSLA-PROVIDER-bDSLA-1 token. The user can stake at any time of each period.
Tx: https://kovan.etherscan.io/tx/0xb2162c292e74978da4715192290bda84345778bb3fe7031a63ede445d3ff2842

6.20 Stake with whitelisted user

Provider stakes 1 000 bDSLA and receives 1 000 DSLA-USER-bDSLA-1 token. The user can stake at any time of each period.
Tx: https://kovan.etherscan.io/tx/0x46390d1b37b4df49d4ac3d9f8f51773a90ef4baada34d14571d103c17f1ad2e7

6.21 Withdraw users stake

The user can only withdraw its stake after the contract is finished.
Tx: https://kovan.etherscan.io/tx/0xa80529d7cddcc2b56b0a4ff0d93d3de6ac6dcf4270fa6dcf962d3f17227f3ead

6.22 User cannot withdraw stakes, if the contract is not finished.

Tx: https://kovan.etherscan.io/tx/0x6c2cd14b7abaf1f1cf181b4b65e3bf18f950fc31d383bb8e8e87e906bbc5640e

6.23 Partial withdraw of providers stake

The provider can withdraw partial stake as long as the user stake is smaler than provider stake. After the contract is finished or breached, the provider can withdraw his hole stake.
Tx: https://kovan.etherscan.io/tx/0x6c8fd55f650000f3345be6ea87654da5d82fa589e470291339409ccd10ab38c7

7. Verify claims

7.1 Providers stake DSLA tokens to pay periodic verifications. After a DSLA Period is finished, it can be verified. The verification fee is split between: The User doing the verification 25% and 25% goes to the SLA's Messenger owner, to cover the expenses of calling this function, and to incentivise a fast verification after the period is finished.
Status: tested and verified ✅

7.2 The SLA contract stake can be whitelisted
Status: tested and verified ✅

7.3 The users can stake at any period, if the contract is not finished.
Status: tested and verified ✅

7.4 The provider can stake at any period, if the contract is not finished.
Status: tested and verified ✅

7.5 The users can only withdraw stake after the contract is finished.
Status: tested and verified ✅

7.6 The provider can withdraw stake at any time, as long as his pool is greater than or equal to the users pool to enforce hedge after an eventually contract breach.
Status: tested and verified ✅

7.7 Only the SLA contract owner (i.e. the address associated to the contract creation transaction) can stake on the provider pool.
Status: tested and verified ✅

7.8 User is able to withdraw a DSLA, USDC and DAI stake from an expired or breached DSLA contract
Status: tested and verified ✅

7.9 Provider is able to withdraw a DSLA, USDC and DAI stake as long as the total user stake is below the provide stake, whilst the contract is not finished. If the contract is finished then the provider can withdraw all of his stake, since the last verification of the

rewards for the last period was calculated, or the compensation was distributed to the users, in case that the contract was breached.
Status: tested and verified ✅

8. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The overall code quality of the project is very good. It implemented widely-used and reviewed contracts from OpenZeppelin and Chainlink.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found, after the manual and automated security testing. Only informational issues were found, to increase the code quality. Overall, everything was well documented and worked as it was supposed to be.

9. Deployed Smart Contract

VERIFIED

Smart Contract is deployed here:

DSLAToken: https://etherscan.io/address/0x3affcca64c2a6f4e3b6bd9c64cd2c969efd1ecbe#code
SLORegistry: https://etherscan.io/address/0x1bE60A36Ba9De2eCeFe8be8d2720B67f932EC487#code
SLARegistry: https://etherscan.io/address/0xB63a13825e129fBa2f2205847158461bec5f265A#code
MessengerRegistry: https://etherscan.io/address/0x766C0b52fADC43Bc3EEAe8BC64536404981951bE#code
PeriodRegistry: https://etherscan.io/address/0x5Da279bE9D6CeB11e7D71179150750669093 57bc#code
StakeRegistry: https://etherscan.io/address/0x4b48AdDd838A11061cE285106f4a30cc5636735C#code
SEMessenger: https://etherscan.io/address/0xFB29aFC3F4B78755f07faD5B86448595D2EEC86C#code
NetworkAnalytics: https://etherscan.io/address/0xC33492F8D76918A9527165A9fD71089980656357#code
Details: https://etherscan.io/address/0x38b0cd8BB4C4608E32EE75b25A8846459cEAd513#code