PUBLIC

Security Audit

of SLIC Smart Contracts

June 7, 2019

Produced for

by

Table Of Content

Foreword

We first and foremost thank SLIC INTERNATIONAL for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

– ChainSecurity

Executive Summary

SLIC INTERNATIONAL engaged CHAINSECURITY to perform a security audit of SLIC, an Ethereum-based smart contract system.

The SLIC INTERNATIONAL token is a standard ERC-20 token raising capital for state-of-the-art modularly constructed portable data center facilities. Token holders may benefit from dividends being payed out.

CHAINSECURITY audited the smart contracts which are going to be deployed on the public Ethereum chain. Audits of CHAINSECURITY use state-of-the-art tools for detection of generic vulnerabilities and checks of custom functional requirements. Additionally, a thorough manual code review by leading experts helps to ensure the highest security standards.

During the audit, CHAINSECURITY was able to help SLIC INTERNATIONAL in addressing several security, trust and design issues of high, medium and low severity.

All reported issues have been addressed by SLIC INTERNATIONAL. In particular, all security and design issues have been eliminated with appropriate code fixes.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. They have been reviewed based on SOLC compiler, version 0.5.0 and EVM version PETERSBURG. All of these source code files were received on April 24, 2019 as part of the git commit `4dde3651afd7e6e1ca98cffc47f16d9430a1c0e0`. The latest update has been received on June 6, 2019 with the commit `3e274c700e754d097fee6e28bd785f9607a110dc`.

File	SHA-256 checksum
./contracts/SlicToken.sol	`d38484610440764402cc8848e0289d72b5c92c4fa8139216975a31c39630f7e7`

For these files the following categories of issues were considered:

In Scope	Issue Category	Description
☑	Security Issues	Code vulnerabilities exploitable by malicious transactions
☑	Trust Issues	Potential issues due to actors with excessive rights to critical functions
☑	Design Issues	Implementation and design choices that do not conform to best practices

Depth of Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

Terminology

For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology[1]).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.

Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 4 distinct categories, depending on their severities:

- L Low: can be considered as less important
- M Medium: should be fixed
- H High: we strongly suggest to fix it before release
- C Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

[1] `https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology`

	IMPACT		
LIKELIHOOD	**High**	**Medium**	**Low**
High	C	H	M
Medium	H	M	L
Low	M	L	L

During the audit concerns might arise or tools might flag certain security issues. After careful inspection of the potential security impact, we assign the following labels:

- ✓ No Issue: no security impact
- ✓ Fixed: the issue is addressed technically, for example by changing the source code
- ✓ Addressed: the issue is mitigated non-technically, for example by improving the user documentation and specification
- ✓ Acknowledged: the issue is acknowledged and it is decided to be ignored, for example due to conflicting requirements ot other trade-offs in the system

Findings that are labelled as either ✓ Fixed or ✓ Addressed are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

System Overview

Token Name & Symbol	SLIC TOKEN, SLIC
Decimals	18 decimals
Tokens issued	507835888
Token Type	ERC 20
Token Generation	Pre-Minted

Table 1: Facts about the SLIC token and the Token Sale.

In the following we describe the SLIC TOKEN (SLIC) and its corresponding token distribution. Table 1 gives the general overview.

Token Deployment Overview

The SLIC token is distributed in 60 stages, the stages can be overlapping and are invoked by the `Admin`. These tokens are called SLIC deployment token 1 to 60. Only the `Admin` can distribute these deployment tokens to addresses. This can be done without restriction as long as there are tokens available. For each of the 60 deployment tokens, the `Admin` needs to call `startLockUpCountdown()` individually which starts a 6 months counter. After this counter has been expired, holders of a SLIC deployment token can redeem them for actual SLIC tokens.

Token Overview

The SLIC token is a standard ERC-20 token raising capital for state-of-the-art modularly constructed portable data center facilities. Token holders may benefit from dividends being payed out.

Extra Token Features

Tracing Token holder tracing

The SLIC token features a functionality to keep track of which addresses currently hold tokens. Upon any transfer the tracing functionality is invoked and updates the array with the sender and receiving account accordingly. A `getHolders()` function returns an array with all addresses currently holding tokens.

Freezing Freezing Addresses

The `Admin` of the SLIC token can freeze addresses. The token balance of a frozen account cannot change, this account cannot send or receive any tokens. `burn()` and `mint()` are also inhibited for frozen accounts.

Redeeming Redeeming SLIC deployment tokens

Upon deployment SLIC deployment tokens get distributed. After their lock up period of 6 months expired, users can call `redeem()` to exchange their tokens to actual SLIC tokens. The `Admin` can forcefully redeem an account's token for the user.

Best Practices in SLIC INTERNATIONAL's project

CHAINSECURITY is determined to deliver the best results to ensure the security of a project. To enable us to do so, we are listing Hard Requirements which must be fulfilled to allow us to start the audit. Furthermore we are providing a list of proven best practices. Following them will make audits more meaningful by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Hard Requirements

These requirements ensure that the SLIC INTERNATIONAL's project can be audited by CHAINSECURITY.

- ☑ **All files and software for the audit have been provided to CHAINSECURITY**
  The project needs to be complete. Code must be frozen and the relevant commit or files must have been sent to CHAINSECURITY. All third party code (like libraries) and third-party software (like the solidity compiler) must be exactly specified or made available. Third party code can be located in a folder separated from client code (and the separation needs to be clear) or included as dependencies. If dependencies are used, the version(s) need to be fixed.
- ☑ The code must compile and the required compiler version must be specified. When using outdated versions with known issues, clear reasons for using these versions are being provided.
- ☑ There are migration/deployment scripts executable by CHAINSECURITY and their use is documented.
- ☑ The code is provided as a Git repository to allow reviewing of future code changes.

Best Practices

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable.

- ☑ There are no compiler warnings, or warnings are documented.
- ☑ Code duplication is minimal, or justified and documented.
- ☑ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ☑ The project only contains audit-related files, or, if this is not possible, a meaningful distinction is made between modules that have to be audited and modules that CHAINSECURITY should assume are correct and out-of-scope.
- ☑ There is no dead code.
- ☑ The code is well-documented.
- ☑ The high-level specification is thorough and enables a quick understanding of the project without any need to look at the code.
- ☑ Both the code documentation and the high-level specification are up-to-date with respect to the code version CHAINSECURITY audits.
- ☑ Functions are grouped together according to either the Solidity guidelines[2], or to their functionality.

[2] https://solidity.readthedocs.io/en/v0.4.24/style-guide.html#order-of-functions

Security Issues

This section relates our investigation into security issues. It is meant to highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

Ineffective blacklisting M ✓ Fixed

The SLIC token implements a blacklist. Blacklisted addresses however are simply blacklisted from receiving tokens, it's important to note that they can still transfer their current balance onward to any other non-blacklisted address. Thus the effect of blacklisting address is very limited and most likely does not suite SLIC INTERNATIONAL's need. The current blacklisting for `_burn` is useless, as a blacklisted account can first perform a `transfer` and then a `_burn`.

Likelihood: Medium
Impact: Medium

Fixed: SLIC INTERNATIONAL fixed the issue by replacing the blacklisting with a feature to freeze transfers, minting and burning.

Trust Issues

This section mentions functionality which is not fixed inside the smart contract and hence requires additional trust into SLIC INTERNATIONAL, including in SLIC INTERNATIONAL's ability to deal with such powers appropriately.

Admin controlled token distribution M ✓ Addressed

The deployer of the `SlicToken` bears the `Admin` role and has full control over the distribution of SLIC deployment tokens. Only this `Admin` address can successfully call `distribute(address to, uint256 amount, uint8 deploymentId)` and thereby distribute tokens of the passed deploymentId to any address, with the only condition that sufficient tokens for this deploymentId are available.

Users need to fully trust SLIC INTERNATIONAL to behave honestly. Normally the benefit of smart contracts is to minimize the amount of trust needed by participating actors.

Furthermore note that the `Admin` keeps control over all undistributed deployment tokens even after the official distribution is over and may redeem those at any time for actual SLIC tokens after the `unlockTime` of this deployment token has been reached.

Addressed: SLIC INTERNATIONAL is aware of this issue and addressed the problem by introducing a separate role called `IcoManager`. SLIC INTERNATIONAL states that the system needs to be centralized due to business requirements.

Admin controlled start of `LockUpCountdown` allows blocking of deployments H ✓ Addressed

The distribution of the SLIC token happens in 60 stages. For every single stage the `admin` needs to call `startLockUpCountdown()` to enable the 6 months countdown before holders of the deployment token can redeem their token for the actual SLIC token. If the admin role never calls `startLockUpCountdown()` for a given deployment SLIC token, holders of this deployment tokens will never be able to redeem their tokens for the actual SLIC token rendering these useless.

Additionally the `Admin` fully controls which address gets deployment tokens of which deployment id, which elevates this trust issue.

Addressed: SLIC INTERNATIONAL is aware of this issue and addressed the problem by introducing a separate role called `IcoManager`. SLIC INTERNATIONAL states that the system needs to be centralized due to business requirements.

`recoverERC20Tokens()` may be used to transfer locked SLIC tokens H ✓ Addressed

`createDepolymentToken()` enables the `Admin` to create the next deployment token in sequence. This deploys a new `SlicDeploymentToken` and mints the amount of actual SLIC tokens to this contracts for later redemption. Normally the `Admin` would then distribute these deployment tokens to users and start the `LockUpCountdown`. Only after the unlock time has expired, token holders can redeem their deployment SLIC tokens for actual SLIC tokens. The amount of distributed deployment SLIC tokens is limited to the amount of minted SLIC tokens to this contract, ensuring that there are enough tokens available for distribution after the lock up time has expired after 6 months.

However the `SlicDeploymentToken` also features a generic `recoverERC20Tokens` function which enables the `Admin` to recover ERC20 tokens owned by this address. This function may be misused by the `Admin` to withdraw SLIC tokens. This can be done at any time, including during the lock up countdown. If this is done, the remaining balance of this `SlicDeploymentToken` contract may be insufficient to fullfill all `redeem()` calls by legitimate users after the lock up period has ended and users are stuck with their sub SLIC tokens. This creates a trust issue as users cannot trust that their deployment tokens can eventually be redeemed for SLIC tokens.

Addressed: SLIC INTERNATIONAL states that the system needs to be centralized due to business requirements and this includes the assumption that the admin role must be trusted.

Design Issues

This section lists general recommendations about the design and style of SLIC INTERNATIONAL's project. They highlight possible ways for SLIC INTERNATIONAL to further improve the code.

Broken token tracing C ✓ Fixed

All addresses holding tokens are stored in a dynamic array `holdersSet` and a mapping `holdersIndices` stores the position of an address inside the array `holdersSet` or 0 if the address is not in the `holdersSet`.

If an account transfers all remaining tokens to another account, the sending account will be deleted from the record. However this is not implemented correctly. SLIC INTERNATIONAL forgot to update the new `holdersIndices` of the `lastHolder`. Therefore, the scheme is broken.

Meaningful tests would have helped SLIC INTERNATIONAL to uncover and avoid issues like this.

Fixed: SLIC INTERNATIONAL fixed the issue by updating the index correctly.

Inefficient token distribution M ✓ Addressed

The SLIC INTERNATIONAL code deploys 60 individual contracts apart from the main token contract to track the different stages of the token sale. This creates a high overhead both in gas costs as in administration. CHAINSECURITY notes that there are different ways to keep track of token sale stages and later the correct token distribution. What is currently achieved through separate contracts can be done via mappings or libraries. As there no is no documentation justifying the current design, it appears overly complicated.

Addressed: SLIC INTERNATIONAL explained CHAINSECURITY why this setup is intended. SLIC INTERNATIONAL wants to easily enable token holders to manage and exchange tokens of a specific stage.

Transfers of 0 tokens adds recipient to tracked holders M ✓ Fixed

A transfer of 0 tokens to a receiver not yet tracked in the `holdersSet` adds the address of the receiver to the `holdersSet`. Contrary transferring away all tokens from an account removes the senders address from the `holdersSet`. Consequently the behavior is inconsistent.

Note that transfers of 0 tokens are possible and must be allowed according to the ERC-20 token standard:

```
Note Transfers of 0 values MUST be treated as normal transfers and fire the
   Transfer event.
```

See ERC-20 specifications [3].

Fixed: SLIC INTERNATIONAL added a check to avoid tracking `0` transfers.

Incomplete token recovery in MultiSig contract M ✓ Fixed

The `MultiSigAdmin` bears the `Admin` role in the deployed `SlicToken` contract. The `SlicToken` features a generic ERC-20 token recovery functionality to recover any ERC-20 token features by the token which is only callable by the `Admin`. To enable this, the `MultiSigAdmin` contract features a `recoverER20Tokens` function which calls the `recoverERC20Tokens` function of the Token contract.

This function currently only works if the token to be recovered is the SLIC token as the incorrect variable is used to call to invoke the final transfer of the tokens. Initiating the token recovery for any other ERC-20 token currently results in these tokens being locked in the `MultiSigAdmin` contract forever.

Omitting the `MultiSig` functionality for simplicity, this function enables the token recovery in 2 steps: Note that the function takes parameter `address _address`, the address of the ERC-20 token to be recovered.

[3] https://eips.ethereum.org/EIPS/eip-20

First `token.recoverERC20Tokens(_address)` is called which transfers the tokens to be recovered to this `MultiSigAdmin` contract. Next these recovered tokens should be transferred onward to their final destination, `msg.sender`. This has to be done by calling `(address of the token).transfer()`.

The implementation however call `token.transfer()` instead of `_address.transfer()`. `token` stores the address of the SLIC token contract, however `_address` contains the address of the current ERC-20 token to be recovered. Thus currently the functionality only works correctly to recover SLIC tokens, as in this case `token` is equal to `_address`.

Fixed: SLIC INTERNATIONAL fixed the issue by calling the correct contracts.

Inconsistent enforcement of freezing accounts M ✓ Fixed

By freezing SLIC INTERNATIONAL does not allow an account to receive funds via minting and does not allow an account to burn or send funds. Still, it is possible for a freezed account to receive funds from a non-freezed accounts. Allowing to receive funds via a transfer but not via a mint, does not seem to make sense (nor is it consistent). If there is no good reason to keep this behavior, SLIC INTERNATIONAL might also disallow transfers to accounts which are freezed.

Fixed: SLIC INTERNATIONAL fixed the problem by disallowing sending and receiving any tokens for an account if freezed.

Redundant check L ✓ Fixed

The `recoverERC20Tokens` function of the `SlicToken` contract enables the `Admin` to recover any ERC-20 token owned by this contract. Before transferring the respective token balance to `msg.sender`, there is a call to `erc20Token.balanceOf()` to query the contract's current balance and a redundant check if the balance is `> 0`. When calling `erc20Token.transfer()` the amount to be transferred is queried again by a second call to `erc20Token.balanceOf()`.

Calls to other smart contracts are rather expensive in solidity at a minimum cost of `700` gas each. Temporarily storing the balance would be significantly cheaper. Furthermore ensuring that the balance is `> 0` is unnecessary and doesn't bring any advantage for SLIC INTERNATIONAL, simplifying this function would enable some gas savings.

Fixed: SLIC INTERNATIONAL removed the redundant check.

Unused `WhitelistToken` contract L ✓ Fixed

The file `Slictoken.sol` includes the contract `WhitelistToken`. This contract is not used or inherited, thus is "dead code". Note that the `SlicToken` has no whitelisting functionality. If this contract is not used, CHAINSECURITY suggests to remove it.

Fixed: SLIC INTERNATIONAL solved the issue by removing the `WhitelistToken` contract.

Dividend payout L ✓ Addressed

The limited specification available claims:

```
Dividends will be paid out to the token holders who have redeemed their sub
    tokens for the master tokens; the dividends are paid manually in a
    centralized manner
```

Note that there is no designated token reserve / address where tokens are stored to be used for dividend payout, rather it is assumed that the admin account has not distributed all tokens and has enough tokens available for dividend payout. It is unclear what incentive the admin role has to actually proceed with the payout of these dividends or how user can rely on eventually getting dividends payed out. It is also unclear how long into the future dividends will be paid out and which amount is therefore required.

Addressed: SLIC INTERNATIONAL explained the setup to CHAINSECURITY. The dividends will be payed in ETH or a suitable ERC20 token (like USDC).

Inefficient repeated storage access L ✓ Fixed

Function `_traceSender` of `ERC20Traceable` accesses `holdersIndices[from]` twice in sequence:

```
if(holdersIndices[from] != 0) {
     uint256 senderIndex = holdersIndices[from];
```

Currently the Solidity compiler is unable to optimize these 2x accesses to storage, resulting in two rather expensive SLOAD operations to be executed. CHAINSECURITY recommends to optimize this by locally caching the variable, executing `uint256 senderIndex = holdersIndices[from]` first and later accessing the value with the local variable `senderIndex`.

Note that there is a proposal, EIP-1884 [4], to increase the gas costs for the SLOAD operation considerably in the future. The proposal is backed by the head of security of the Ethereum Foundation and is likely to be implemented in the future.

Fixed: SLIC INTERNATIONAL stores the value in a variable and therefore, avoids the second access.

Admins of `MultiSigAdmin` not enforced to be different L ✓ Fixed

The `MultiSigAdmin` contract implements a 2 out of 3 multi signature scheme. While in order to perform an action the address of the proposer has to be different from the address of the second `Admin` actually executing the process, the contract does not enforce that all admin address differ.

Consequently, if all addresses are set to the same address the `MultiSigAdmin` contract becomes blocked with no way to restore it's functionality.

Fixed: SLIC INTERNATIONAL fixed the issue by adding an appropriate `require` statement.

[4] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1884.md

Recommendations / Suggestions

☑ CHAINSECURITY doubts that SLIC INTERNATIONAL's approach to track the token holders is efficient. It considerably increases the gas costs of token transfers. The limited documentation available states:

```
a special smart contract function lets the centralized system extract the
    token holders set without the need to replay the blockchain to track
    each transfer event; the SC maintains internally a set of the current
    token holders.
```

As CHAINSECURITY understands the list of token holders is not needed inside the smart contract but only off chain for analysis and dividend calculation. Note that each transfer of an ERC-20 token emits a `Transfer` event. Filtering logs would be a simple and efficient way to generate a list of all token holders at a certain block and only require a full node (or a service provider like INFURA [5]) without the need to replay the blockchain.

☑ The `require()` function allows the developer to specify error messages. Error messages do increase gas costs upon deployment of the smart contracts and when triggered, but are worth the slightly increased gas costs as it provides users with information about why their call failed. CHAINSECURITY recommends SLIC INTERNATIONAL to consider adding error messages to the require statements.

☑ The function `setBlacklisted()` which can only successfully be called by an `Admin` sets the `Boolean` for the passed address regardless of it's previous state. Note that Solidity does not optimize this, consequently every call results in a write to storage at a minimum cost of 5000 gas even if the value remains unchanged. If SLIC INTERNATIONAL expects this may happens occasionally, CHAINSECURITY recommends to consider introducing a simple check if the passed value is different from the already stored one and skip the write to storage if these are equal. This introduces an additional overhead of slightly above 200 gas (costs for an `SLOAD` and a comparison operation). Furthermore SLIC INTERNATIONAL may consider to validate the input argument and return if the zero address is passed.

☑ Contracts should be deployed with the same compiler version they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another compiler version that might introduce bugs that affect the contract system negatively.

CHAINSECURITY recommends locking the pragma version[6].

☑ The function `freeze()` also un-freezes accounts. Therefore, SLIC INTERNATIONAL could consider renaming the function to `toggleFreeze()` or similar to make sure, that the name is meaningful for what the function actually does.

[5] https://infura.io
[6] https://smartcontractsecurity.github.io/SWC-registry/docs/SWC-103

Disclaimer