PUBLIC REPORT

SECURITY AUDIT REPORT

of Smart Contracts

December 27, 2017

Produced by

CHAINSECURITY

for

zilliqa

Table of Contents

Foreword

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

- ChainSecurity

Token name	Zilliqa (ZIL)
Decimals	12
Smallest Unit (Atom)	10^{-12} ZIL
Maximum token supply	21,000,000,000
Percentage of tokens for sale	30%
Minimum personal cap	2 ETH
Maximum personal cap	5 ETH
Max cap (# ZILs)	6,300,000,000
Max cap w/o bonus (ETH)	48,889 ETH
Max cap (USD)[1]	22,000,000 USD
Token Distribution	Past-ICO
Rewards	Yes (see TGE overview)
KYC	Yes (whitelist)

Table 1. Facts about the ZIL token and the token sale

Introduction

ZILLIQA is a new blockchain platform that offers high transaction rates and scalability. The key idea is to leverage sharding, which allows one to divide large computations and to process them in parallel among the network nodes. This would enable throughput that matches the average transaction rate of existing VISA and MasterCard systems while keeping the advantage of blockchain platforms, which offer much lower fees for merchants.

In the following we describe the Zilliqa token (ZIL) and its corresponding token sale. Table 1 gives the general overview.

TGE overview

The TGE happens as follows:

- **Early contribution:** In this phase early supporters can participate and receive a bonus of 10-15%. The cap for this phase is 44,444 ETH.
- **Determining price for community contribution:** Based on the bonuses awarded during early contributions the price for community contributions is calculated so that a complete sellout raises 48,889 ETH.

[1] fixed exchange rate 450 USD = 1 ETH

- **Community contribution:** During this phase community contributions are accepted. The minimum and maximum contribution amounts are 2 ETH and, respectively, 5 ETH.

- **Token distribution:** After all contributions have been received, Zilliqa will issue transactions to assign the corresponding amount of tokens to the contributors. Possible excess tokens will be burnt. Note, that all of these calculations and operations are happening off-chain and can therefore not be audited by ChainSecurity Ltd.

- **Tokens become transferable:** Tokens become transferable after token distribution is complete and Zilliqa has issued a corresponding transaction.

Token distribution

The ZIL tokens will be distributed as follows:

- Up to 6.3 billion (≤ 30%) tokens will be sold during the TGE.

- 8.4 billion (≥ 40%) tokens will be given to miners as “mining rewards”.

- 6.3 billion (≥ 30%) tokens will be given to the Zilliqa team, research, development and advisors. Most of these tokens are vested over three years.

Extra features

- **Pausable:** Zilliqa has the power to pause and unpause the transfer of tokens. While paused, no token transfers and approvals can be made. This pausing happens separately for the contract owner and administrator, and all other token holders. Therefore, there is a state where only the owner and the administrator can transfer tokens, e.g. to distribute tokens.

- **Burnable:** The tokens are burnable, i.e. they can be permanently destroyed using the burn function.

- **Burn From:** The smart contract contains a burnFrom function that combines transferFrom and burn to save gas.

Audit overview

Depth

The scope of the security audit conducted by ChainSecurity Ltd. was restricted to:

- Scanning the contracts listed above for generic security issues using automated systems and manually inspecting the results.
- Manual auditing of the contracts listed above for security issues.

Terminology and labels

For the purpose of this audit, ChainSecurity adopts the following terminology. To summarise security vulnerabilities, the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology[1]) are specified.

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 determined previously, and is summarised in Table 2.

The severity of each finding is categorised into four ratings, depending on their criticality:

- *L* signifies a low-risk issue that can be considered as less important.
- *M* signifies a medium-risk issue that should be fixed.
- *H* signifies a high-risk issue that should be fixed very soon.
- *C* signifies a critical issue that needs to fixed immediately.

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

Table 2. Severity of an issue based on its impact and likelihood.

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

If no security impact is found for an issue, we label it as ✔ (“no issue”).

If a security impact has been found for an issue, and it has been addressed technically during the auditing process, we also label it as ✔ (“fixed”).

If the security impact for an issue has been addressed in another manner, we label it as (“addressed”).

Findings that are labelled as either ✔ fixed or addressed are resolved and therefore pose no security threat. Their severity is still listed simply to give the reader a quick overview what kind of issues were found during the audit.

System risk classification

Points are allocated to the severity category (Table 3) of each issue found in the audit in order to determine the overall risk level (Table 4) of the smart contract system.

Severity category of finding	Points
C	30 per finding
H	10 per finding
M	4 per finding
L	1 per finding

Table 3. Allocation of points for each severity category

System risk classification		Cumulative points
	Critical risk	30 points or more
	High risk	17 – 29 points
	Medium risk	7 – 16 points
	Low risk	6 points or less

Table 4. Risk classification for systems based on the sum of all point

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 enables the discovery of vulnerabilities that were overlooked during development and areas where additional 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. ChainSecurity therefore carries out a source code review to determine all the locations that need to be fixed. ChainSecurity performs extensive auditing in order to discover as many vulnerabilities as possible.

Findings

The following issues were investigated during the audit of Zilliqa's smart contracts.

Issue 1: Conversion issues due to different decimals

Likelihood: High **Impact:** Low

It seems that ZILs from Ethereum will be eventually transferred into the mainnet. However, according to the technical whitepaper the ZIL token will have 12 decimals, while the Ethereum-based ZILs have 18 decimals:

```
uint public constant decimals = 18;
```

Therefore, during the conversion partial ZILs would be lost as they could not be expressed in the mainnet.

Fix: This issue has been fixed by Zilliqa by adjusting the number of decimals of the Ethereum-based token to 12. Therefore, the decimals now match.

Issue 2: Missing validation in the token constructor

Likelihood: Low **Impact:** Medium

The Zilliqa token is constructed using the following function:

```
function ZilliqaToken(address _admin, uint _totalTokenAmount)
{
   // assign the admin account
   admin = _admin;

   // assign the total tokens to zilliqa
   totalSupply = _totalTokenAmount;
   balances [msg.sender] = _totalTokenAmount ;
   Transfer(address(0x0), msg.sender, _totalTokenAmount);
}
```

The argument `_admin` is not validated. This argument should be validated as there is no way to change `_admin` after contract creation (there is no function that can be used to change the admin after constructing the token contract).

Fix: Zilliqa added a setAdmin function to the contract. Therefore, in case of a mishap during contract creation, this can now be fixed later.

Reentrancy analysis 

The only call to external code is made within the `emergencyERC20Drain` function, where it is safe as the contract's state is consistent and the call is controlled by the owner.

Ether transfers 

The contract receives and sends no ether and therefore contains no vulnerabilities in the regard.

Callbacks 

As the only external call is safe and has no dependencies and as no ether transfers are made no callstack-related bugs exist in the smart contract.

Safe math 

Zilliqa uses the SafeMath library to protect from over- and underflows for all important numeric variables.

Summary of system risk

Prior to audit

Severity category of finding	Number of issues found	Subtotal
C	0	0
H	0	0
M	1	4
L	1	1

Total points: 5

System risk classification:

Our audit has revealed that the Zilliqa's smart contracts are at a **low** risk level prior to our audit and our subsequent amendments.

After audit

Severity category of finding	Number of issues found	Subtotal
C	0	0
H	0	0
M	0	0
L	0	0

Total points: 0

System risk classification: -

Zilliqa's smart contracts have no known issues after the code fixes.

Recommendations

- Zilliqa's implementation of `Pausable` allows separate control for owner and admin, and all other users. The implementation looks as follows:

```
modifier whenNotPaused() {
   if (msg.sender == owner || msg.sender == admin) {
      require(!pausedOwnerAdmin);
   } else {
      require(!pausedPublic && !pausedOwnerAdmin);
   }
   _;
}
```

  By this implementation pausing owner/admin transfers implies pausing public transfers. We assume that this is intended by Zilliqa. A slightly better design would be to check this implication when `pausedPublic` and `pausedOwnerAdmin` are set, namely in the `pause` function. Then, this check would only be performed once every time the `paused` values change and not every time a transfer occurs. Consequently, some gas could be saved.

- Optionally, in order to increase the trustworthiness of Zilliqa, the vested tokens could directly be assigned to a smart contract, which administers the vesting. Alternatively, the vesting process could be defined more precisely (e.g. is it linear vesting, who controls it).

- The burn function is also contained in OpenZeppelin's BurnableToken. The two implementations differ in that Zilliqa's implementation emits a Burn and a Transfer event, while OpenZeppelin's implementation only emits a Burn event. Due to the imprecise semantics of ERC20 tokens neither implementation appears to be wrong.

- The `burnFrom(_from, _value)` function in the ZilliqaToken contract first transfers tokens from address `_from` to **msg**`.sender` and then **msg**`.sender` burns the transferred tokens.

```
function burnFrom (address _from, uint256 _value ) returns (bool) {
    assert(transferFrom (_from , msg.sender, _value));
    return burn(_value);
}
```

  The call to `transferFrom(_from, msg.sender, _value)` may fail if `_from` has not approved **msg**`.sender` to spend the specified amount of tokens. A successful execution of `burnFrom` requires that `transferFrom` must execute successfully. Posing this requirement as an assertion is misleading because a call to `transferFrom` may fail under normal conditions (due to lack of approval).

- It is currently not possible to change the token's admin, which may be desirable in certain scenarios. One option is to include a function that updates the token's admin, similar to `transferOwnership` which is used to change a contract's owner.

Conclusion

ChainSecurity Ltd. has analyzed Zilliqa's smart contract and has found no major technical vulnerabilities or shortcomings. The ZIL smart contract contains the necessary token functionality. Zilliqa has additionally addressed the minor shortcomings that were uncovered during the report and has even implemented some of the recommendations. Due to the design, ChainSecurity Ltd. cannot audit the correct calculation and allocation of token amounts to TGE contributors.

CHAINSECURITY

www.chainsecurity.com

contact@chainsecurity.com

@chain_security