

Security Audit

of ALPROCKZ's Smart Contracts

November 30, 2018













Produced for



by



Table Of Content

Foreword	1
Executive Summary	1
Audit Overview	2
1. Scope of the Audit	2
2. Depth of Audit	2
3. Terminology	2
Limitations	4
System Overview	5
1. Token Sale Overview	5
2. Token Overview	5
3. Extra Features	5
Best Practices in ALPROCKZ's project	6
1. Hard Requirements	6
2. Soft Requirements	6
Security Issues	7
1. Dependence on <code>b!ock.timestamp</code>  	7
2. Locked Tokens  	7
Trust Issues	8
1. Owner has full minting control  	8
2. Delayed minting of vested tokens  	8
3. Finalization and transferability  	8
Design Issues	9
1. Outdated compiler version  	9

2.	Struct Locking can be optimized			9
3.	mintVested visibility can be restricted			10
4.	Block gas limit exhaustion			10
5.	onlyOwner modifier is not used			10
6.	Inconsistency between function implementations			10
	Recommendations / Suggestions			11
	Disclaimer			12

Foreword

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

– ChainSecurity

Executive Summary

The ALPROCKZ smart contracts have been analyzed under different aspects, with a variety of tools for automated security analysis of Ethereum smart contracts. Overall, we found that ALPROCKZ has clean, well-documented code.

We have no security concerns about the smart contracts and ALPROCKZ successfully implemented all requirements, suggestions and addressed most issues raised in a professional manner. However CHAINSECURITY notes that the full potential of smart contracts is not used by ALPROCKZ, as unnecessary manual interaction required by the smart contract introduces a more traditional trust model.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. All of these source code files were received on September 24, 2018 and an updated version on November 27, 2018:

File	SHA-256 checksum
AlprockzToken.sol	42d25f9bd90e7d7c2feccdd1843a6d2ee241c065a58d2feb88321d1d1e1d572
LockedToken.sol	7095931127f15ffb1c8dd2740ed1bea5172c7d1242982e13cc2bf399ad7d720b
VestingPrivateSale.sol	8f83af49c23153243c31c37400e95d67ceee0bacdc959e357a02b2c3d92612dd
VestingTreasury.sol	444f319e77c230aabb4484f988137d53bf530fc4944cb9c69c6a58657c4c443d

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 methodology¹).

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:

-  Low: can be considered as less important
-  Medium: should be fixed
-  High: we strongly suggest to fix it before release
-  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.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

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

During the audit concerns might arise or tools might flag certain security issues. If our careful inspection reveals no security impact, we label it as **✓ No Issue**. If during the course of the audit process, an issue has been addressed technically, we label it as **✓ Fixed**, while if it has been addressed otherwise by improving documentation or further specification, we label it as **✓ Addressed**. Finally, if an issue is meant to be fixed in the future without immediate changes to the code, we label it as **✓ Acknowledged**.

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	ALPROCKZ TOKEN, APZ
Minting Hard Cap	175,000,000 APZ tokens
Decimals	18 decimals
Phases	Off-chain
Minimum contribution	Off-chain
Token Type	ERC 20
Token Generation	Capped, Mintable
Vesting	Selective

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

In the following we describe the ALPROCKZ TOKEN (APZ) and its corresponding Token Sale. The table above gives the general overview.

Token Sale Overview

The token sale happens externally through the Swisscom blockchain ICO kit which is not in the scope of this audit. However, the owner of the ALPROCKZ TOKEN contract needs to mint the tokens for investors. At the end of the ICO the owner of the contract must finalize the minting and unlock the tokens to enable transfers.

Token Overview

ALPROCKZ TOKENS are ERC20 tokens issued by ALPROCKZ during the ICO. These tokens are intended to be utility tokens that allow users to access the ROCKZ platform and its services.

Extra Features

Extended Vesting Founders and certain investors may be subject to a vesting period to mitigate the risk of an immediate sell-off of a large amount of tokens shortly after the ICO.

Vested tokens will be managed by the `VestingPrivateSale` smart contract. These tokens cannot be transferred until the vesting period expires.

The administrator will mint these tokens using the special `mintPrivateSale` function of the ALPROCKZ TOKEN contract at the end of the ICO. This is the only way tokens can be locked up in the vesting contract.

- 25% of the tokens can be transferred immediately
- 25% are released after 6 months
- 25% are released after 12 months
- 25% are released after 18 months

To release these tokens either the account holding these tokens or the admin needs to call the release function of the vesting contract before these tokens can be transferred.

Treasury vesting Part of the ALPROCKZ TOKEN supply will be distributed to a treasury escrow with a vesting mechanism over a minimum of 36 months. This mechanism consists of a 6 month lock period and a 30 month release calculated as a linear function over time.

Best Practices in ALPROCKZ's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when ALPROCKZ's project fitted the criterion when the audit started.

Hard Requirements

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

- ☒ The code is provided as a Git repository to allow the review of future code changes.
- ☒ Code duplication is minimal, or justified and documented.
- ☒ Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with ALPROCKZ's project. No library file is mixed with ALPROCKZ's own files.
- ☒ The code compiles with the latest Solidity compiler version. If ALPROCKZ uses an older version, the reasons are documented.
- ☒ There are no compiler warnings, or warnings are documented.

Soft Requirements

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

- ☒ There are migration scripts.
- ☒ There are tests.
- ☒ The tests are related to the migration scripts and a clear separation is made between the two.
- ☒ The tests are easy to run for CHAINSECURITY, using the documentation provided by ALPROCKZ.
- ☒ The test coverage is available or can be obtained easily.
- ☒ The test coverage is high.
- ☒ 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 not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- ☒ There is no dead code.
- ☒ The code is well documented.
- ☒ The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- ☒ Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- ☒ There are no getter functions for public variables, or the reason why these getters are in the code is given.
- ☒ Function are grouped together according either to the Solidity guidelines², or to their functionality.

²<https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions>

Security Issues

In the following, we discuss our investigation into security issues. Therefore, we highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

Dependence on `block.timestamp`

ALPROCKZ's code makes use of the field `block.timestamp`. CHAINSECURITY wants to raise awareness that a malicious miner is able to move forward block timestamps by up to 900 seconds (15min) compared to the actual time. We recommend ALPROCKZ and its users should be aware of this and adhere to the 30 seconds rule³

Likelihood: Low

Impact: Low

Acknowledged: ALPROCKZ acknowledges the possibility and claims that there are no critical time issues since the vesting mechanism is in months and a drift of up to 15 minutes would be insignificant.

Locked Tokens

Tokens mistakenly sent to the `AlprockzToken` contract will be locked forever as the contract has no functionality to handle them. Numerous historic cases⁴ have shown that accidental ERC20 transfers to token contracts occur frequently. Currently, such ERC20 tokens would be locked.

CHAINSECURITY draws attention to the fact that the vesting contract must not be given a generic token recovery function or else locked up tokens may be transferred.

Likelihood: Medium

Impact: Low

Fixed: ALPROCKZ fixed the issue by implementing a check on the token recipient in the `transfer` and `transferFrom` functions, which prevents APZ tokens from being sent to the `AlprockzToken` smart contract.

³ <https://consensys.github.io/smart-contract-best-practices/recommendations/#30-second-rule>

⁴ <https://coincentral.com/erc223-proposed-erc20-upgrade/>

Trust Issues

The issues described in this section are not security issues but describe functionality which is not fixed inside the smart contract and hence requires additional trust into ALPROCKZ, including in ALPROCKZ's ability to deal with such powers appropriately.

Owner has full minting control

Although this is intended by the specification, such a setup does not adhere to the trust model the community is used to from other ICOs and counteracts the potential use of decentralization. While CHAINSECURITY understands the potential compliance requirements, the current design introduces a significantly centralized trust model.

Users must place high trust in the contract owner to behave honestly:

- The owner allocates the correct amount of tokens after the investor has paid (or refunds money)
- A malicious owner may simply allocate tokens to an address under his control and as the process is not transparent, this is not verifiable by other investors or users. This contradicts a fundamental advantage of a decentralized ledger.

If ALPROCKZ wants to proceed with this setup, it is necessary to ensure that the minted tokens are only delivered to legitimate addresses and ALPROCKZ must be aware of the risk that a rogue employee may mint tokens to his own addresses.

Ideally, the smart contract would implement a fully transparent process, e.g. the user sends some ETH to the contract and tokens are minted and allocated according to a certain exchange rate automatically.

Addressed: ALPROCKZ addresses the issue by implementing off-chain KYC and AML procedures before investing in the ICO.

Delayed minting of vested tokens

Investors must trust the owner of the ALPROCKZ TOKEN contract to only mint the vested tokens for the founders, advisors and core team as promised in the specifications at the very end of the ICO and with the correct amount.

The design might be changed such that this happens before the ICO starts and that the `startDate` for the vesting of the tokens is updated to the current timestamp once the minting finalizes.

Addressed: ALPROCKZ added a corresponding section "DELAYED MINTING OF VESTED TOKENS" to its whitepaper to explain its reasoning behind this design choice, phrased as follows: "The reason for this is that vesting applies from the minting and it would therefore be wrong and unfair to mint team and advisor tokens before the end of the ICO".

Finalization and transferability

Investors must trust the owner of the ALPROCKZ TOKEN contract to finalize the minting and enable the token transfer once the ICO is over.

Note that both are done independently, finalizing is done by the owner calling `finalizeMinting` and unlocking the token transfers happens when the owner calls `UnlockTransfer`. A malicious owner may do one but not the other. CHAINSECURITY also wants to note that a malicious owner may activate any of the two functions at any point throughout the ICO.

Addressed: ALPROCKZ acknowledges the issue but states that ALPROCKZ Ltd does not have any interest in acting in a malicious manner given that it would discredit and destroy all value for users of the ROCKZ platform, thus destroying the company's value. A corresponding section "FINALIZATION AND TRANSFERABILITY" was added to the whitepaper.

Design Issues

The points listed here are general recommendations about the design and style of ALPROCKZ's project. They highlight possible ways for ALPROCKZ to further improve the code.

Outdated compiler version

Without a documented reason, the newest compiler version should be used homogeneously.

The code uses `pragma solidity ^0.4.24` and CHAINSECURITY wants to make ALPROCKZ aware that this compiler version suffers from the following critical bug⁵:

Higher order bits in the exponent are not properly cleaned before the EXP opcode `is` applied `if` the type of the exponent expression `is` smaller than 256 bits and not smaller than the type of the base. In that case, the result might be larger than expected `if` the exponent `is` assumed to lie within the value range of the type. Literal numbers `as` exponents are unaffected `as` are exponents or bases of type `uint256`.

ALPROCKZ needs to be aware that the AlprockzToken contract may be affected in Line 20:

```
(10 ** uint256(decimals));
```

If ALPROCKZ still chooses to stay with the outdated compiler version, the bytecode generated by the compiler should be crosschecked that it's not affected by this issue.

Fixed: ALPROCKZ successfully updated the `pragma` to use the latest compiler version, which is 0.4.25.

Struct Locking can be optimized

The following struct is declared in the Vesting contract:

```
struct Locking {
    uint256 bucket1;
    uint256 bucket2;
    uint256 bucket3;
    uint256 startDate;
}
```

The bucketX fields store the amount of tokens, startDate the date the tokens were added to the vesting contract. Using `uint256` for all field is unnecessary, e.g. the timestamp is a 32-bit unsigned integer only. For the bucket fields, consider the following corner case:

There exist 175 million ALPROCKZ TOKENS, with 18 decimals. Thus the maximal amount of tokens times decimals is $175 \times 10^6 \times 10^{18}$. Even if all tokens are locked up in one bucket, a `uint88` would be sufficient to store it, as $2^{88} - 1 > 175 \times 10^6 \times 10^{18}$.

The storage of Ethereum is organized in slots of 256 bits. ALPROCKZ's current implementation uses four storage slots to store one `struct Locking`. By using the suggested datatypes this can be reduced to $3 \times 88 \text{ bit} + 32 \text{ bit} = 296 \text{ bit}$, thus only two storage slots of 256 bits are needed to store the corresponding struct, cutting the storage requirements by a half.

By using a different design this might be even optimized further, especially as the amounts of all buckets are always equal in the current version of the code.

Acknowledged: ALPROCKZ acknowledges the issue but avoids making optimizations since doing so would increase the risk of bugs.

⁵<https://etherscan.io/solcbuginfo?a=ExpExponentCleanup>

mintVested visibility can be restricted

The `mintVested` function in the `AlprockzToken` contract is a function which enables the administrator to bulk-mint tokens in order to save gas. This function is only called externally by the administrators and arrays are passed to the function as arguments.

In Solidity `public` functions copy their arguments to memory when they are called which is costly in terms of gas, especially for large arrays. External calls however can be read directly from the field `CALLDATA` of the transaction which is much cheaper.

By changing this function from `public` to `external` the gas usage can be reduced.

Fixed: ALPROCKZ fixed the issue by marking the relevant functions as `external`.

Block gas limit exhaustion

The two functions `mintVested` and `mint` are used by the owner to distribute tokens to multiple addresses in one transaction. ALPROCKZ should be aware that these functions contain loops that can quickly exceed the maximum amount of gas currently available in a block which is around eight million.

The loop in `mintVested` will exceed this limit after processing only around 60 addresses, while the iteration inside the `mint` function will hit the block gas limit after processing around 250 addresses.

In issue 2. we discuss how the storage of the `Lockup` struct can be improved which would roughly double the amount of possible iterations for `mintVested`.

The limitations applying to `mint()` might become an acute problem that will be encountered during token distribution. ALPROCKZ should note that token distribution may be only possible in small batches. Even if the transaction to mint these tokens would use less than the current block gas limit, the transaction might not get included in a block. It has been observed in the past that transactions that consume a high percentage of the block's gas limit are only likely to be included in a block if a rather high gas fee was paid.

Fixed: ALPROCKZ fixed the issue by limiting the number of recipients in the bulk functions `mintPrivateSale`, `mintTreasury` and `mintArray` to respectively 10, 10 and 40.

onlyOwner modifier is not used

The `AlprockzToken` and `LockedToken` contracts inherit from `Ownable` but use an explicit `require` check instead of the already provided function modifier - `onlyOwner`. This is still acceptable and does not compromise security, but using the modifier will result in a more concise contract which makes use of the information it inherits.

Fixed: ALPROCKZ fixed the issue and is using the inherited modifier instead of an explicit `require` check.

Inconsistency between function implementations

The two functions `releaseBuckets()` and `releaseBuckets(address _tokenHolder)` both serve to initiate a release of locked buckets but have inconsistencies in the `require` statements used.

The first function has a guard in place to ensure that this address has a `startDate` greater than zero as a `startDate` equalling zero would imply that there are no tokens to be released, while the second function which can only be used by the owner is missing this check.

This leads to a inconsistency where one function fires a `transfer` event for the parameter being zero and the other one does not.

Fixed: ALPROCKZ fixed the issue by implementing the same requirements for both functions.

Recommendations / Suggestions

- ✓ The current error message provided in the Alprockz contract would give no information on what went wrong. CHAINSECURITY advises to either provide a more descriptive one or remove the error message since that would reduce gas costs.

```
function splitToFour(uint256 _amount)
    private pure returns (uint256 first, uint256 second, uint256 third
        , uint256 fourth)
{
    require(_amount >= 4, "Otherwise_it_makes_just_no_sense");
```

- ✓ CHAINSECURITY's investigations showed that ALPROCKZ's test suite reaches a test coverage of roughly 75%. However, some functions in the Vesting contract have never been tested which also applies to the releaseBuckets function.
- ✓ CHAINSECURITY recommends to follow the Solidity documentation and use the compiler's optimization flag before deploying the contract⁶, which allows to save gas upon deployment and during the contract's lifetime.
- ✓ Given the specific trust setup of ALPROCKZ, CHAINSECURITY recommends to consider emitting additional events at critical points such as the minting finalization and vesting release which will help to simplify tracking the ICO process for investors and users.
- ✓ ALPROCKZ has 78 test cases passing and 10 failing. CHAINSECURITY strongly recommends ensuring all tests are passing before deployment.
- ✓ ALPROCKZ has introduced some minor spelling mistakes which can be easily fixed. Examples include error messages such as "faild" and comments in the VestingPrivateSale and VestingTreasury contracts.
- ✓ ALPROCKZ has done a good job providing documentation for the functionality implemented in their smart contracts. However, the file treasury-vesting.xlsx provides a diagram which wrongfully shows that there is a 12 month lock and 24 month release. CHAINSECURITY suggests that ALPROCKZ updates the documentation.

Post-audit comment: ALPROCKZ has fixed all of the above issues. Therefore, ALPROCKZ has to perform no more code changes with regards to these recommendations.

⁶<https://solidity.readthedocs.io/en/latest/using-the-compiler.html#using-the-commandline-compiler>

Disclaimer

UPON REQUEST BY ALPROCKZ, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..