PUBLIC

# Security Audit

## of STOKR's Smart Contracts

December 11, 2019

Produced for

STOKR

by

CHAINSECURITY

# Table Of Content

# Foreword

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

– ChainSecurity

# Executive Summary

The STOKR 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 STOKR has a well written code and extensive tests with 100% code coverage. CHAINSECURITY did not find major issues. Nonetheless, CHAINSECURITY raised some minor issues and suggestions. These issues were all acknowledged or fixed in a professional manner.

CHAINSECURITY extended the audit after completion on request. The extended review now also includes the post-audit added changes listed below.

- Token destruction now emits an event
- The TokenDistribution event's information was extended
- A function to change the closing time of the crowdsale was added

This report was updated in December 2019 because of an issue that was independently reported to STOKR. STOKR made the appropriate code changes, CHAINSECURITY reviewed their correctness and believes that they effectively mitigate the raised issue.

# 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 February 22, 2019.
The corresponding Git commit was: `be93ca797a9096fb3f157fbcefe3dee05214e66c`.
An update has been received on March 21, 2019.
The corresponding Git commit is: `aa0d4dc69087d2d668b921803026644d23c8443c`.
The third update has been received on April 29, 2019.
The corresponding Git commit is: `9a967af293168dd1d2773336ada11b3534c6031f`.
The latest update has been received on December 11, 2019 as STOKR was alerted to an issue which they then reported to CHAINSECURITY. CHAINSECURITY reviewed the corresponding code updates.
The corresponding Git commit is: `88a9b9c6ba5342391b7581525358924e7ae018e8`

| File | SHA-256 checksum |
|---|---|
| ./StokrProjectManager.sol | `4158c3220f6558db3368e6e27c8370879e990c60a74f7501a9df3a44e4c0b852` |
| ./ownership/Ownable.sol | `5eeea189402b6b8faf82d36305c6f11208d00b609d480369a16eb79f9c0104a6` |
| ./token/StokrToken.sol | `b3633d94be89d6ed4e14bb4b30399fa283aeccb65e7079da6f19c77cf14738f4` |
| ./token/TokenRecoverable.sol | `1fc8e574d8e1519b40ecf11740fcbf78bce12d9c170cccd3ecf2eed6a1a1586e` |
| ./token/ERC20.sol | `285471f24fa8eda2fec6d45bbb7c4bac41788d7085c489e8e95f27c46d5b846c` |
| ./token/StokrTokenFactory.sol | `96f3dad89ead696abd768f9a55325884a5204fa868694960b71ad3153a96e04d` |
| ./token/ProfitSharing.sol | `3083ca7f62721a5dbce6e795e989484a3db7adffff649645099e327fe2a405a9` |
| ./token/MintableToken.sol | `340d7ab55fbd00072242c69f29e5fba72cb470ae66f4b4cab9408cd13b4dca60` |
| ./crowdsale/StokrCrowdsale.sol | `187a0e9dd48b5224db254bf506035d2ade29072856170d3471a649dedbe8f42c` |
| ./crowdsale/RateSourceInterface.sol | `e403f1aedbbc1158b5cb228e3ae83261647cc2cd46d50bc6c5c5644ee0360e0b` |
| ./crowdsale/MintingCrowdsale.sol | `9aba5f7987faa0aa0dc0c76f6e15fc0609cca179b85cf11fc23f76716a1a3648` |
| ./crowdsale/StokrCrowdsaleFactory.sol | `b8ba30a0c0be261782e6ddb9f933fcdfb9dd7be81db8a701c11c441cbbc42d0d` |
| ./whitelist/Whitelisted.sol | `a5a6828079fe6ea621ec38819dccfb32fad52ea25de2d21b691cdeef582f4d94` |
| ./whitelist/Whitelist.sol | `40e5fb39cb6475f02f2fffa4e1c7eaf031484f24708bc3c81ca805d138509c59` |
| ./math/SafeMath.sol | `62931a5a942f6440f080bd0afebfb73ba68d96552bc64aa1e602c391ef63ed00` |

## 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.

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

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.

| | IMPACT | | |
|---|---|---|---|
| **LIKELIHOOD** | **High** | **Medium** | **Low** |
| **High** |  |  |  |
| **Medium** |  |  |  |
| **Low** |  |  |  |

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: during the course of the audit process, the issue has been addressed technically
- ✓ Addressed: issue addressed otherwise by improving documentation or further specification
- ✓ Acknowledged: issue is meant to be fixed in the future without immediate changes to the code

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

STOKR is as crowd-investing platform based on smart contracts on the Ethereum blockchain. STOKR enables ventures to create projects and investors to invest into these projects. For this purpose STOKR implemented a system which has built-in features to support investors and ventures. Each project launched on STOKR's platform has a crowdsale contract to manage the sale of a dedicated security token with profit sharing and a global whitelist. Thus, only whitelisted investors can invest.

The profit sharing schemes distributes all deposited profits among the token holders according to their token balance at the time of deposit. A user's profit share is tracked automatically and can be withdrawn at any time using the corresponding function.

The crowdsale has multiple configurable parameters such as an individual purchase cap or start and end times. In case a crowdsale, doesn't reach its defined investment goal, then all investor can obtain a refund. In case of a successful crowdsale, investors can withdraw their tokens after the completion of the crowdsale.

# Best Practices in STOKR'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 STOKR's project fitted the criterion when the audit started.

### Hard Requirements

These requirements ensure that the STOKR'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 STOKR's project. No library file is mixed with STOKR's own files.
- ☑ The code compiles with the latest Solidity compiler version. If STOKR 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 STOKR.

- ☑ 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 STOKR.
- ☑ The test coverage is available or can be obtained easily.
- ☑ 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 guidelines[2], or to their functionality.

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

# Security Issues

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

### Batch function call may fail L ✓ Acknowledged

`distributeRefunds()` calls the untrusted address `_investor` provided in the function argument. Even though the `transfer()` function is considered safe regarding reentrancy, a malicious address could intentionally make the `transfer()` call fail. As a result, the function `distributeRefunds()` is blocked because the loop cannot be executed anymore. But `distributeRefunds()` is just the "batch" version of `claimRefund()` and therefore the impact is low as each investor can still use `claimRefund()` to get their refund. The same issue occurs in `withdrawProfitShares()`, but again the impact is low as the non-batch version could still be called individually by each investor.

STOKR could consider preventing one failing ETH transfer in the loop from reverting the entire transaction.
**Likelihood:** Low
**Impact:** Low

**Acknowledged:** STOKR acknowledged that they are aware of this issue.

### Potential overflow in `setRate()` L ✓ Fixed

STOKR does not use `SafeMath` to verify the new `rate` to be set. This could lead to an overflow in the multiplication part of the check, `newRate < 10 * rate`. If `rate` is near the maximum uint256 value, multiplying by 10 might cause an overflow. This function can only be called by `rateAdmin`, which is set by the owner.

STOKR should consider using `SafeMath.mul()` to prevent overflow.
**Likelihood:** Low
**Impact:** Low

**Fixed:** STOKR solved the problem by adding end enforcing a maximum rate of `uint(-1)/10`.

### Adjusting Allowance can lead to front-running L ✓ Fixed

Inside the ERC20 token standard, there is a well-documented[3] issue with front-running when it comes to the `approve` function. When changing a particular allowance from $X$ to $Y$, where $X \neq 0$ and $Y \neq 0$ using a single transaction, this transaction is susceptible to front-running. The spender of the allowance can send a competing `transferFrom` transaction and therefore transfer $X + Y$ tokens.

This issue can either be addressed in client applications by performing two transactions, where the first transaction resets the allowance to $0$, or `increaseAllowance` and `decreaseAllowance` functions can be introduced to allow a safe allowance adjustment in a single transaction.
**Likelihood:** Low
**Impact:** Medium

**Fixed:** STOKR solved the issue by adding the two functions `increaseAllowance` and `decreaseAllowance`. STOKR also added a `require()` to check for over- and underflows.

[3] https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

# Design Issues

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

### Functions visbility can be set to external L ✓ Fixed

There are a number of functions declared as `public` when they could also be set to `external`. A non-exhaustive list of these functions is:

- `distributeTokensViaPublicSale()`
- `distributeTokensViaPrivateSale()`
- `addToWhitelist()`
- `removeFromWhitelist()`
- `withdrawProfitShares()`
- `distributeRefunds()`
- `createNewProject()`

Functions with visibility `external` can directly read from `calldata`[4] and do not copy function arguments to memory. Thus, declaring these functions `external` might optimize the gas costs. If and how much gas is saved, depends on the compiler version and the optimization done by the compiler.

**Fixed:** STOKR fixed the issue by setting the visibility from `public` to `external`.

### Failing checks silently continue L ✓ Acknowledged

STOKR uses an `if` in multiple places to perform a check when a `require` would make more sense. For example the second condition in `TokenRecoverable.sol` on line 39.

```
function setTokenRecoverer(address _newTokenRecoverer) public onlyOwner {
  require(_newTokenRecoverer != address(0x0), "New token recoverer is zero");

  if (tokenRecoverer!=address(0x0) && _newTokenRecoverer != tokenRecoverer) {
    emit TokenRecovererChange(_newTokenRecoverer);
  }
  tokenRecoverer = _newTokenRecoverer;
}
```

CHAINSECURITY sees no reason to allow token recoverer to be updated to a new address if the new address is the same as the current address. STOKR is advised to reevaluate the usage of `if` statements vs throwing an error in such cases as described above.

**Acknowledged:** STOKR partially solved the problem by only updating the value if it it differs from the current value. However, calling this function to update the value to the same value does still not result in an error.

### Crowdsale continues if sold out L ✓ Acknowledged

In case a crowdsale is sold out, it continues until the `hasClosed` condition is fulfilled, which is that the `closingTime` has passed. Up until that time the `finalize` function is not callable. Hence, during this time tokens are not transferable and no deposits can be made even though no more tokens can be purchased.

[4] https://solidity.readthedocs.io/en/v0.5.3/types.html?highlight=external#data-location%7D

**Acknowledged:** STOKR acknowledged the issue and explained that this behavior is intended.

### Remaining tokens might not be purchasable L ✓ Acknowledged

The crowdsale enforces a `tokenPurchaseMinimum` for each token purchase. This can lead to issues towards the end of a crowdsale in case the crowdsale is close to selling out. In particular, it might be impossible to purchase remaining tokens.

As an example let's say that `tokenPurchaseMinimum = 100` and that `tokenRemainingForPublicSale = 80`. Hence, 80 tokens are still for sale. If a buyer tries to purchase 80 tokens, the purchase will fail as it is below the minimum. If a buyer tries to purchase the minimum of 100 tokens, it will fail, because there are only 80 tokens left. Obviously, the impact and likelihood of this issue depend on the choice of the parameters.

**Acknowledged:** STOKR is aware of this issue and acknowledges it.

### Whitelisting of Token Recovery Address is not Enforced L ✓ Fixed

In the documentation, STOKR states that the whitelist defines which addresses "are able to send or receive tokens". This is ensured during regular transfers. However, as part of the `recoverToken` function, tokens can also be transferred and there no whitelist check is performed.

**Fixed:** STOKR solved the problem by adding the `onlyWhitelisted` modifier to the `recoverTokens` function.

# Recommendations / Suggestions

- ☑ The function `transferOwnership()` transfers the ownership directly to the address given in the function's argument. In case of a mistake, the ownership will be transferred to some random account and is most likely "lost". Therefore, client could consider using a scheme in which the new owner needs to claim the ownership, to finally get it. This makes sure that at least the account is controlled by some user.
- ☑ In `MintingCrowdSale.sol` and `ProfitSharing.sol` STOKR makes use of the fallback function to catch ether sent to the contract and call `buyTokens()`. Since the fallback function is not only called for plain ether transfers (without data) but also when no other function matches, STOKR should check that the data is empty if the fallback function is intended to be used only for the purpose of forwarding ether to `buyTokens()`. Otherwise, callers will not notice if your contract is used incorrectly and functions that do not exist are called[5].
- ☑ The storage variable `deploymentBlockNumber` in `StokrProjectManager` is only set once in the constructor. The variable is not used anywhere else. If it is not required urgently by any other interacting contract CHAINSECURITY is not aware of, STOKR could consider removing the variable.
- ☑ STOKR might consider using the `indexed` keyword in some logged events. This might make sense to later better search through the events.
- ☑ The token contract has a special role called token recoverer. In most places of the code, it is referred to as `tokenRecoverer`. However, there are also multiple occurrences where this role is called `keyRecoverer`. This naming could be harmonized.
- ☑ There are different implementations of the ERC20 token standard, but some have started emitting a `Approval` event whenever the token allowance has been changed. In case, STOKR wants to adapt this, an additional `Approval` event could be emitted from the `transferFrom` function.
- ☑ In the code, there are two semantically identical getter functions. The two functions `mintingFinished()` and `totalSupplyIsFixed()` both return the value of the variable `totalSupplyIsFixed`. Therefore, one of the two getter functions could be omitted to achieve some gas savings during deployment.

[5] `https://consensys.github.io/smart-contract-best-practices/recommendations/#check-data-length-in-fallback-functions`

# Notes

This section highlights additional remarks about the behaviour of the smart contracts. These are not considered as issues. However, CHAINSECURITY still wants to point them out for completeness and for educational purposes.

### Rounding Errors ✓ Acknowledged

(Unsigned) integer divisions generally suffer from rounding errors. The same holds true for divisions inside the EVM. Therefore, the results of arithmetic operations can be imprecise. The effects of these errors can be reduced by ordering arithmetic operations in a numerically stable manner. However, even then minor errors (e.g. in the order of one token wei) can occur.

In the STOKR contracts, any user can call `updateProfitShare(A)` on another user `A` to trigger additional rounding errors and therefore effectively lower the token balance of `A`. Furthermore, the rounding errors during the calculation of profit shares will lead to an accumulated amount of "lost" tokens inside the `ProfitSharing` contract over time. However, CHAINSECURITY expects both of these errors to have a negligible effect due to use of 18 decimals.

**Acknowledged:** STOKR acknowledged the theoretical possibility, but also correctly pointed out that a potential attacker has no economic incentive to perform such an attack, as it: (i) only incurs a tiny damage on the victim, (ii) incurs gas costs on the attacker, and (iii) does not provide any direct benefit to the attacker.

# Disclaimer

UPON REQUEST BY STOKR, 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..