HashEx

# BoatPilotNAVI

# Smart Contract Security Audit

Prepared for: BoatPilotNAVI

January 17, 2018

By:

HashEx
https://hashex.org

# Contents

**Disclaimer** **2**

**Background** **3**

**Critical Severity Issues** **3**

vault/FundsVault.sol 3

**High Severity Issues** **3**

AllocatedCappedCrowdsale.sol 3

**Low Severity Issues** **4**

vault/FundsVault.sol 4

token/CrowdsaleToken.sol 4

token/BurnableToken.sol 4

**General Recommendations** **4**

vault/FundsVault.sol 4

math/SafeMathUtil.sol 5

token/BurnableToken.sol 5

token/CrowdsaleToken.sol 5

token/FractionalERC20.sol 5

validation/ValidationUtil.sol 5

Haltable.sol 6

AllocatedCappedCrowdsale.sol 6

RefundableAllocatedCappedCrowdsale.sol 7

**Conclusion** **8**

# Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

# Background

HashEx was commissioned by BoatPilotNAVI to perform an audit of TGE smart contracts. The audit was conducted between December 25, 2017 and 30 December 30, 2017. Second audit was conducted between January 9, 2018 and January 10, 2018, and on January 16, 2018.

The audited code is located in the BoatPilotNAVI/TGE repository. The version used for the first audit report is commit fae694fc12f7da315c872159cbe34036f18c7b76. The version used for the second audit report is commit 29e6e47f18a8b0f13d06c07a4d596844b800f22e.

The purpose of the audit was to achieve the following:

- Ensure that the smart contracts function as intended.
- Identify potential security issues with the smart contracts.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improve the security posture of the smart contracts by remediating the issues that were identified.

# Critical Severity Issues

## vault/FundsVault.sol

When *function refund()* is called with *weiAmount* that is less than amount of *deposited[investor]*, only *weiAmount* value is returned whereas *deposited[investor]* is overwritten with zero. As there is no alternative way to withdraw funds from the contract that is in *Refunding* state, and it is not possible to update the state, the remaining funds cannot be withdrawn from the contract. This can be resolved by adding a check for the *weiAmount* value or by adding functionality of manual funds withdrawal from the contract. Line 68.

Taking into account the inheritance specifics, this issue is leveled by contracts logic. However, if this code will be used in other contracts, we strongly recommend fixing it.

# High Severity Issues

## AllocatedCappedCrowdsale.sol

If *function internalInvest()* is called with *weiAmount* value with which the amount of tokens being purchased is greater than amount of tokens available for purchase, only available number of tokens will be purchased while remaining funds will stay on the contract. Thus, tokens will be purchased at a price greater than their nominal value. This can be resolved by

adding a check for the *weiAmount* value or by adding functionality of returning excess funds to investor. Line 272.

# Low Severity Issues

## vault/FundsVault.sol

We recommend to replace *function checkAddress()* call with *require(address != 0x0).* This will facilitate reading the contract code and understanding which checks are made for the *address* parameter. Also this will allow to drop the inheritance from *ValidationUtil* contract and, thus, save gas. Line 31.

**Update**. *function checkAddress()* was renamed to *function requireNotEmptyAddress().*

## token/CrowdsaleToken.sol

1. *canMint* variable doesn't have a setter. Therefore *onlyWhenCanMint* modifier will always return a positive result.

   **Update.** Modifer and variable were removed from the contract.

2. *function mintToAddress()* does not increase the number of tokens on the balance but overwrites it. If address has 100 tokens and *function mintToAddress()* is called with parameter equal to 50, upon it execution the address balance will become 50, not 150. Line 55.

## token/BurnableToken.sol

*invalidOwnerBurner* modifier allows setting *tokenOwnerBurner* address into *function setOwnerBurner()* only once. Line 18.

# General Recommendations

## vault/FundsVault.sol

1. A typo in the 'также' world, redundant comma in the comments. Line 9.
2. Code readability can be improved by replacing *Closed()* and *Refunded()* events by *EventChanged()* event with a state parameter. Lines 20, 22.
3. After *function transfer()* is executed, no changes are made in the state of blockchain, so this function is not vulnerable to the Reentrancy attack. However, we recommend using the Pull Payments pattern instead of Push payments. It should be noted that when using the Pull Payments pattern, the transaction costs are paid by the investor, not the contract owner. Line 71.

Taking into account the inheritance specifics, this recommendation is leveled by contracts logic. However, if this code will be used in other contracts, we recommend fixing it.

## math/SafeMathUtil.sol

The use of algebraic operations in a contract with checking the results of calculations is a good practice. Contract code already uses a similar library from OpenZeppelin. We recommend using it here too, instead of math / SafeMathUtil.sol, as it has passed many audits and tests.

**Update.** SafeMath and Math libraries from OpenZeppelin are now used.

## token/BurnableToken.sol

1. *function checkAddress(tokenOwnerBurner)* should be called in the beginning of *function setOwnerBurner()*. This will help to troubleshoot why transaction is not completed. If transaction is run without this check, all operations preceding the check will use gas. Line 22.

   **Update.** Fixed.

2. Modifiers i*nvalidOwnerBurner* and *validOwnerBurner* are redundant. It is sufficient to check the *tokenOwnerBurner* value during installation. This will allow to use less gas as there will be no checks for *tokenOwnerBurner* validity when executing *function burnOwnerTokens()*, *function burnTokens()*, *function burnAllOwnerTokens()*.

We recommend that you consider using BurnableToken from OpenZeppelin.

## token/CrowdsaleToken.sol

For token creation we recommend using inheritance model from OpenZeppelin's MintableToken.

## token/FractionalERC20.sol

The contract is not utilized. It should be removed from repository.

**Update.** Fixed.

## validation/ValidationUtil.sol

The use of this contract hinders the code readability as its only purpose is to check that *_address != 0.* We recommend deleting this contract from repository and replace inheritance from *ValidationUtil* and *function checkAddress(_address)* calls in other contracts to *require(_address != address(0))*.

**Update.** *function checkAddress()* was renamed to *function requireNotEmptyAddress().*

## Haltable.sol

*function unhalt()* can be called only when *halted = true,* while *function halt()* can be called when *halted = true* or *halted = false.* The check for *halted* value should be added prior to *function halt()* call.

We recommend using the Pausable contract from OpenZeppelin.

## AllocatedCappedCrowdsale.sol

1. *State Unknown*, *PreFunding* are not utilized in the contract. Line 133.

   **Update.** Fixed.

2. Set *indexed* attribute to *address* variable in *event Invested()* to enable filtering in event queries. Line 136.

   **Update**. Fixed.

3. Move all *require()* calls to the beginning of constructor. Line 156.

   **Update.** Fixed.

4. Remove *owner = msg.sender* expression as redundant. Contract is inherited from *Haltable* and *Haltable* is inherited from *Ownable* from OpenZeppelin library. Owner will be automatically set by *Ownable* contract. Line 157.

   **Update.** Fixed.

5. Use *require()* instead of *if(isFirstStageTokensMinted) return.* Lines 200, 214.

   **Update.** Fixed.

6. *function getOneTokenInWei()* is not utilized in the code. Line 228.

   **Update**. It is used by external functionality. Attribute was changed from *public* to *external*.

7. *function assignTokens()* is redundant. It is used only once while its logic and name is identical to *function internalAssignTokens()* from which it is called. Line 242.
8. Typo in the name of *function updateStat().* Probably this function is to be named *updateStatistics* or *updateStats*. Line 434.

9. Replace *if (!token.transfer(receiver, tokenAmount)) revert();* with *assert(token.transfer(receiver, tokenAmount));*. We recommend using SafeERC20 contract from OpenZeppelin. Line 489.
10. *canEnableRefunds* modifier is redundant as the check should be moved in the function itself. This will improve the code readibility and simplicity. Lines 496, 799.
11. *function externalBuy()* must have *external* modifier instead of *public*. Line 515.

    **Update.** Fixed.

12. *isHardCapGoalReached()* is redundant as it is duplicating the *isSoftCapGoalReached()* check. Line 736.

    **Update.** Fixed.

13. Code can be rewritten with a significantly simpler logic that uses only the following states: *FirstStage, SecondStage, Success, Failure* with *isActive* modifier to check if the current stage is active. Current stage should be stored in a variable. This will allow to drop the following variables: *isFirstStageFinalized, isSecondStageFinalized, isSuccessOver, isRefundingEnabled* and statuses: *Unknown, PreFunding, FirstStageEnd, SecondStageEnd, Refunding*.
14. Token generation on the first and the second stages can be automated and triggered by a state change. This will allow to drop *isFirstStageTokensMinted* and *secondStageTokensMinted* variables.
15. *address destinationMultisigWallet* parameter in *function internalDeposit()* is not utilized. Line 376.

## RefundableAllocatedCappedCrowdsale.sol

*address receiver* parameter in *function internalDeposit(address receiver, uint weiAmount)* is not utilized. To avoid mistakes during refactoring we recommend calling *AllocatedCappedCrowdsale function internalDeposit()* with *receiver* and *weiAmout* parameters, and replace *msg.value* to *receiver* in *RefundableAllocatedCappedCrowdsale function internalDeposit().* Line 51.

# Conclusion

One critical severity and one high severity issues were identified. Taking into account the inheritance specifics, the critical severity issue is leveled by contracts logic. However, if this code will be used in other contracts, we strongly recommend fixing it.

Recommendations were provided for code improvement and protection from potential attacks.

Contracts require refactoring that can significantly simplify the code and thus lower the possibility of mistakes. Also many contracts have a functionality similar to that of contracts from OpenZeppelin library.

All contracts in their current implementation passed our security audit.