#HashEx
BLOCKCHAIN SECURITY

WildLands Presale

smart contracts
final audit report

March 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice 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 disclaimer below – please make sure to read it in full.

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 the 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. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the WildLands team to perform an audit of their smart contracts. The audit was conducted between 16/02/2023 and 23/02/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @wildlandsme/smart_contracts GitHub repository after the 9f5e7e3 commit. Only Crowdsale and TokenVesting contracts were in scope of this audit.

Update. Recheck was done after the commit 062377.

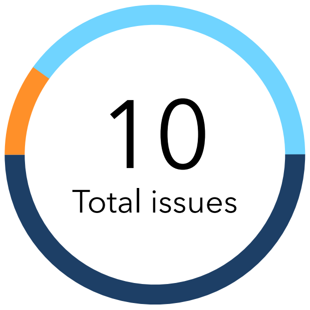
2.1 Summary

Project name	WildLands Presale
URL	https://wildlands.me
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
Crowdsale	`0x9A46cA271eA95A70b4A9C6E989Df9643B36Be6dF`
TokenVesting	`0x8b2AD260E2eB6418c29D0C4FDef7e1d927eB27D0`

3. Found issues



●	High	1 (10%)
●	Low	4 (40%)
●	Info	5 (50%)

C1. Crowdsale

ID	Severity	Title	Status
C1-01	● Low	Gas optimizations	✓ Resolved
C1-02	● Info	Typos	✓ Resolved
C1-03	● Info	Possible stalling of the sale	✓ Acknowledged
C1-04	● Info	Lack of events	✓ Resolved

C2. TokenVesting

ID	Severity	Title	Status
C2-01	● High	crowdsaleAddress can be updated	✓ Resolved
C2-02	● Low	Gas optimizations	✓ Partially fixed
C2-03	● Low	Division before multiplication	✓ Resolved
C2-04	● Low	Default visibility of variables	✓ Resolved

C2-05	● Info	Typos	✓ Resolved
C2-06	● Info	Lack of events	✓ Resolved

4. Contracts

C1. Crowdsale

Overview

A 3-round sale contract for ERC20 token with a fixed offer per round and fixed price in native EVM currency. The first round is for whitelisted users only, 100 places in the whitelist are publicly available. The second round is limited to holders of WildLands NFT member cards. The third round is a public sale. All rounds are consecutive and can't overlap. Bought ERC20 tokens are transferred to the TokenVesting contract to be claimed over time.

Issues

C1-01 Gas optimizations ● Low ✓ Resolved

1. Variables `tokenAmountRateOne`, `tokenAmountRateTwo`, `tokenAmountRateThree`, `token`, `memberCard`, `vestingContract`, and `minimumBuyAmount` should be declared as constants or immutables.
2. Multiple or unnecessary reads from the storage of `startCrowdsaleTime` variable in the `onlyAfterStart()` modifier; `whiteListSpots` variable in the `secureWhitelistSpot()` function; `tokensRaised`, `limitPhaseOne`, `limitPhaseTwo`, `limitPhaseThree`, `startSecondRoundTime`, `startThirdRoundTime` in the `buyNative()` function; `token` and `token.decimals()` in the constructor section.
3. SafeMath library could be removed since only its use case - custom errors - is not implemented in the Crowdsale contract.
4. Merkle tree could reduce gas for setting a whitelist.
5. Redundant code: the `getWhitelistSpots()` function duplicates the functionality of the `whiteListSpots()` getter.
6. Redundant code: in the `buyNative()` function the first condition in the 'else if' clauses, i.e. `tokensRaised >= limitPhaseOne` and `tokensRaised >= limitPhaseTwo`, is always true since its 'else' part ensures it.

C1-02 Typos

● Info ✓ Resolved

Typos reduce the code's readability. Typos in 'safetey', 'amont', 'crowedsale'.

C1-03 Possible stalling of the sale

● Info ✓ Acknowledged

Public sale (3rd round) or 2nd round (discount for NFT holders) could be stalled if there would be not enough participants in previous rounds.

```
function buyNative() ... {
  if (tokensRaised < limitPhaseOne) {
    ...
  } else if (... && isIcoFirstRoundCompleted) {
    ...
  } else if (... && isIcoSecondRoundCompleted) {
    ...
  }
  ...
  if (tokensRaised >= limitPhaseOne)
     isIcoFirstRoundCompleted = true;
  if (tokensRaised >= limitPhaseTwo)
     isIcoSecondRoundCompleted = true;
}
```

C1-04 Lack of events

● Info ✓ Resolved

Governance functions `setTreasury()`, `setWhitelist()`, `togglePauseCrowdsale()` don't emit events, which complicates off-chain tracking of changes in important parameters.

C2. TokenVesting

Overview

A stepped vesting contract for a single ERC20 token to be complemented by the Crowdsale contract. 40% of vested tokens can be claimed immediately, the other 60% is split into 3 equal parts to be claimed after clearance of 30 days of locking periods.

Issues

C2-01 crowdsaleAddress can be updated ● High ✓ Resolved

`setCrowdsaleAddress()` should have initializer properties, i.e. should be callable only once. Otherwise, the owner can vest virtual tokens without actual transfer, and then claim users' funds.

```
function setCrowdsaleAddress(address _crowdsaleAddress) public onlyOwner {
  require(
    _crowdsaleAddress != address(0),
    "TokenVesting: invalid zero address for crowdsale");
  crowdsaleAddress = _crowdsaleAddress;
}

function vest(
  address _to,
  uint256 _value,
  bool _revokable
) external onlyCrowdsale {
  ...
  vests[_to].value += _value;
  ...
}

function unlockVestedTokens() external nonReentrant {
  ...
  Vest storage vested = vests[_msgSender()];
  uint256 vestedAmount = calculateVestedTokens(vested);
```

```
    uint256 transferable = vestedAmount.sub(vested.transferred);
    ...
    token.safeTransfer(_msgSender(), transferable);
  }
```

Recommendation

Consider removing the possibility of `crowdsaleAddress` updating or rework the vesting logic to make `transferFrom()` transfers ensuring the vested amounts.

C2-02 Gas optimizations

● Low   Partially fixed

1. Variables `token`, `totalLimit`, `RELEASES`, `duration`, and `finishOfVest` should be declared as constants or immutables.
2. Double read from the storage of `vests[].value` and `totalVesting` variables in the `vest()` function, `start` and `duration` variables in the `countdown()` function, `vests[].transferred` variable in the `unlockVestedTokens()` function.
3. Unchecked math should be used for the calculation of `timePassedAfterStart` in the `calculateVestedTokens()` function.
4. Calculation of the `transferable` value should be moved inside the conditional section in the `vestedTokens()` function.
5. Revoking mechanism is not in use since the only caller, Crowdsale calls `vest()` function only with `_revokable = false` flag. Gas could be saved both by removing unnecessary checks and by removing boolean fields of the Vest struct.
6. SafeMath library could be removed since only its use case - custom errors - is not implemented in the TokenVesting contract.
7. Redundant code in the `vest()` function: checks on input data could be dropped since it's callable only by the Crowdsale contract with ensured non-zero values.

Update

Optimizations for 1, 4, 5, 6, 7 were done in the update.

C2-03 Division before multiplication ● Low ✓ Resolved

Division before multiplication can, in some narrow cases, cause calculation errors in integer math. Affected functions: `calculateVestedTokens()`.

C2-04 Default visibility of variables ● Low ✓ Resolved

Variables `duration` and `finishOfVest` have default visibility. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

C2-05 Typos ● Info ✓ Resolved

Typos reduce the code's readability. Typos in 'exeeds', 'inital'.

C2-06 Lack of events ● Info ✓ Resolved

Governance function `setCrowdsaleAddress()` doesn’t emit events, which complicates off-chain tracking of changes in important parameters.

5. Conclusion

1 high, 4 low severity issues were found during the audit. 1 high, 3 low issues were resolved in the update.

The audited contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY