

# TaleCraft

## smart contracts final audit report

---

December 2021



[hashex.org](https://hashex.org)



[contact@hashex.org](mailto:contact@hashex.org)

# Contents

1. Disclaimer	3
2. Overview	4
3. Found issues	6
4. Contracts	10
5. Conclusion	21
Appendix A. Issues' severity classification	22
Appendix B. List of examined issue types	23

# 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](https://hashex.org)).

## 2. Overview

HashEx was commissioned by the TaleCraft team to perform an audit of their smart contract. The audit was conducted between 15.11.2021 and 22.11.2021.

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 audited code is deployed to the testnet of Avalanche C-Chain:

[0x75A8BA4e0Ec3f481d2A2c77007ADa4af80b36EE4](#) PHI,

[0x6eeB8DFB57456aAfE8F8ABF43315bBCe7ae80621](#) Resource,

[0x85a24DA79b04E012bE81394Ee87432B255371feF](#) ChestSale,

[0x7452ef5e1eCA5eA5A0f898536738BF2eAEDc22dB](#) Game,

[0x2B4Ad59ff6F8FC2191189aE8e1Ce87068Eb0bf57](#) TokenVestingFactory,

[0xc63562BC48aC87EaF116fd6E3dc38b38859e28a4](#) Marketplace.

The updated contracts are deployed to the mainnet of Avalanche C-Chain:

[0xcc367e92c1b2BB0eB503F67654F3581c086eD2fc](#) Resource,

[0x337F2aB0E1A857A03B93d072656D3d52AA4A586A](#) ChestSale,

[0xFcC7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1](#) Game,

[0x71Ea4a973Be28128a299362015fF4A06b084C70a](#) MarketplaceNew.

## 2.1 Summary

Project name	TaleCraft
URL	<a href="https://talecraft.io">https://talecraft.io</a>
Platform	Avalanche Network
Language	Solidity

## 2.2 Contracts

Name	Address
Game	0xFcC7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1
ChestSale	0x337F2aB0E1A857A03B93d072656D3d52AA4A586A
PHI	0x8aE8be25C23833e0A01Aa200403e826F611f9CD2
MarketplaceNew	0x71Ea4a973Be28128a299362015fF4A06b084C70a
TokenVesting	0x2DAB3390adf79237aF1331bb7Eb4295defE6DA30
TokenVestingFactory	0x2DAB3390adf79237aF1331bb7Eb4295defE6DA30
Resource	0xcc367e92c1b2BB0eB503F67654F3581c086eD2fc

### 3. Found issues

















High	1 (3%)
Medium	4 (13%)
Low	21 (68%)
Info	5 (16%)



#### C1. Game

ID	Severity	Title	Status
C1-01	High	Adding new games	Resolved
C1-02	Medium	Low entropy randomness source	Open
C1-03	Medium	Withdraw problems	Resolved
C1-04	Medium	clearWhitelist() can't be used to clear the whole whitelist	Open
C1-05	Low	Gas optimization	Resolved
C1-06	Low	Function receive() can be removed	Resolved
C1-07	Low	Usage of transfer() function to send AVAX	Resolved
C1-08	Low	Input parameters are not filtered	Open







## C2. ChestSale

ID	Severity	Title	Status
C2-01	 Medium	Chest balances of token ids not updated	 Resolved
C2-02	 Low	Function receive() can be removed	 Resolved
C2-03	 Low	Gas optimizations	 Resolved
C2-04	 Low	Possible rounding errors if parameters are changed	 Acknowledged
C2-05	 Low	Usage of transfer() function to send AVAX	 Resolved
C2-06	 Low	Usage of blockhash(block.number)	 Resolved
C2-07	 Info	Lack of documentation	 Resolved

## C3. PHI

ID	Severity	Title	Status
C3-01	 Low	Function burn() can be declared external	 Resolved

## C4. MarketplaceNew

ID	Severity	Title	Status
C4-01	 Low	No default visibility	 Open
C4-02	 Low	Usage of transfer() function to send AVAX	 Resolved
C4-03	 Low	Gas optimization	 Resolved

C4-04	● Low	Owner is able to change fee after item listing	🔍 Open
C4-05	● Info	Contracts are allowed to sale	✅ Acknowledged

## C5. TokenVesting

ID	Severity	Title	Status
C5-01	● Low	Gas optimizations	✅ Acknowledged
C5-02	● Low	Not enough checks on input data	✅ Acknowledged
C5-03	● Info	Non-linear token release	✅ Acknowledged
C5-04	● Info	No support for rebasing tokens	✅ Acknowledged

## C6. TokenVestingFactory





ID	Severity	Title	Status
C6-01	● Low	Gas optimizations	✅ Acknowledged

## C7. Resource

ID	Severity	Title	Status
C7-01	● Low	Call on unbounded array	✅ Resolved
C7-02	● Low	Gas optimization	✅ Resolved
C7-03	● Low	The function receive() can be removed	✅ Resolved



---

C7-04	 Low	Function resourceCount() returns a wrong value	 Resolved
C7-05	 Info	Lack of documentation	 Resolved

---

## 4. Contracts


### C1. Game

#### Overview

This is the contract where users can play with their table cards and get rewards. Only two users can participate in one game. They alternately place 3 cards, and the one who has a higher total weight of cards wins.

#### Issues

##### C1-01 Adding new games

 High Resolved

In the function `startGames()` anyone can rewrite information about `poolSlot` and this may lead to fund leaks.

For example, Alice and Bob are participating in `poolSlot` number three. Some user passes an array with a length of more than 3 and after that `poolSlot` with the number three leads to a new game, not the game in which Alice and Bob were participating. After that Alice and Bob will both lose their funds.

Also, it is not clear what the argument is used for in this function. An array is used for argument, but only its length is used, not the values inside this array.

#### Recommendation

This function should check the validity of `poolSlots` that are passed to the `_createNewGame()` function.

##### C1-02 Low entropy randomness source

 Medium Open

Updated contract 0xFCc7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1:

**boost()** function uses on-chain random based on timestamp and blockhash of the previous block.

### C1-03 Withdraw problems

Medium

Resolved

For the users who are already in the game there is no withdraw function.

For example, Alice and Bob are playing with each other. Alice and Bob have already placed some cards and Bob loses access to his account or for some other reason isn't be able to end the game. In this case, Alice loses her funds.

#### Recommendation

There should be an emergencyWithdraw function for the users who are already in the game.

### C1-04 clearWhitelist() can't be used to clear the whole whitelist

Medium

Open

Updated contract 0xFCc7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1:

**clearWhitelist()** function would cause an indexation error in case the owner tries to remove more than half of the whitelist elements. Calling **\_whitelisted.remove(player)** inside the loop shortens the whitelist length without altering the loop boundary.

#### Recommendation

We recommend refactoring the **clearWhitelist()** function using the descending loop in order to save gas inside the **EnumerableSet.remove()** function.

### C1-05 Gas optimization

Low

Resolved

The variable **\_resource** can be marked as immutable. By doing this, gas consumption can be reduced because the reading of this variable will be gas-free.

## C1-06 Function receive() can be removed

● Low

✔ Resolved

The function `receive()` always reverts and can be removed from the contract.

### Update

The function has been made callable in order to replenish the reward fund.

## C1-07 Usage of transfer() function to send AVAX

● Low

✔ Resolved

In the function `burn()` the `transfer()` function is used to send AVAX. If the receiver is a contract and performs a certain operation when receiving AVAX, the call may fail as the `transfer()` function forwards only 2300 gas. It is better to use the `call()` function as it forwards all gas.

### Update

The updated contract uses `call()` to send AVAX. Native currency transfers via `call()` are usually secured with a reentrancy guard.

## C1-08 Input parameters are not filtered

● Low

❓ Open

Updated contract 0xFCc7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1:

functions `updateMinWeight()`, `updateMinCardsCount()`, `updateAbortTimeout()` should have restrictions for the input data, so the owner's error wouldn't be able to break the contract's workflow with impossible parameters.


## C2. ChestSale

## Overview

The contract that sells to users default table cards. With one transaction a user can buy one of the first 4 table cards (with ids 1-4).

## Issues

### C2-01 Chest balances of token ids not updated

 Medium Resolved

The `balances[]` array is not updated in the `openChest()` function. There are checks `balances[tokenId] == 0`, but after initialization in function `_startWeek()` its values never updated.

```
function _startWeek() private {  
    ...  
    balances[0] = balances[1] = balances[2] = balances[3] = WEEK_BALANCE;  
}
```

This can lead to a situation when more than `WEEK_BALANCE` chests of a specific token id is sold.

### C2-02 Function `receive()` can be removed

 Low Resolved

The function `receive()` always reverts and can be removed from the contract.

### C2-03 Gas optimizations

 Low Resolved

Variables `resource` and `phi` can be marked as immutable.


Functions that are not called from the contract may be declared as external.

In the function `_startWeek()` there are multiple reads of the `weeksLeft` global variable.

In the function `openChest()` there are multiple reads of these global variables:

1. `chestsLeft`
2. `chestPriceEth`
3. `chestPricePhi`
4. `phi`

## C2-04 Possible rounding errors if parameters are changed

 Low Acknowledged

The sum of the balances may be less than `CHESTS_PER_WEEK` due to the rounding errors. It is not the case with the parameters that are set in the smart contract now, but if they are changed, this invariant could break.

### Recommendation

Set last balance as a remainder of subtraction of other balances from `CHESTS_PER_WEEK` variable.

## C2-05 Usage of `transfer()` function to send AVAX

 Low Resolved

In the function `withdrawFees()` the `transfer()` function is used to send AVAX. If the receiver is a contract and performs a certain operation when receiving AVAX, the call may fail as the `transfer()` function forwards only 2300 gas. It is better to use `call()` function as it forwards all gas.

## C2-06 Usage of `blockhash(block.number)`

 Low Resolved

Function `openChest()` uses `blockhash(block.number)` as a source of pseudorandomness.

```
function openChest(uint256 count) public payable {
    ...
    for (uint256 i=0; i<count; i++) {
        uint256 tokenId = uint256(keccak256(abi.encodePacked(500 - i, block.timestamp,
            blockhash(block.number), i))) % 4;
        ...
    }
}
```

```
    }  
    ...  
}
```

The function `blockhash()` only works for 256 most recent, excluding current.

## Recommendation

Use hash of the previous block as a source of randomness:

```
blockhash(block.number - 1)
```

### C2-07 Lack of documentation

● Info

✓ Resolved

We recommend adding NatSpec documentation at least to the public functions of the contract.

## C3. PHI

### Overview

ERC20 token with burn mechanism where a user can burn their tokens. Users of the system will pay commissions in this token.

### Issues

#### C3-01 Function `burn()` can be declared external

● Low

✓ Resolved

Function `burn()` can be declared external instead of public. This will save gas on calling it.

## C4. MarketplaceNew

### Overview

This is a contract where users can sell and buy table cards.

### Issues

#### C4-01 No default visibility

● Low

ⓘ Open

The `feePercentage` variable is declared without specifying visibility. Users should find the last `FeeUpdated()` event to check the current `feePercentage` value.

#### C4-02 Usage of `transfer()` function to send AVAX

● Low

✅ Resolved

In the function `buyListing()` the `transfer()` function is used to send AVAX. If the receiver is a contract and performs a certain operation when receiving AVAX, the call may fail as the `transfer()` function forwards only 2300 gas. It is better to use the `call()` function as it forwards all gas.

#### C4-03 Gas optimization

● Low

✅ Resolved

The variable `_resource` can be marked as immutable. By doing this, gas consumption can be reduced because the reading of this variable will be gas-free.

#### C4-04 Owner is able to change fee after item listing

● Low

ⓘ Open

`updateFee()` onlyOwner function allows the owner to change the `feePercentage` variable at any moment. `putOnSale()` function should freeze the current value of the `feePercentage` variable in the Listing struct.



---

## C4-05 Contracts are allowed to sale

● Info

☑ Acknowledged

---

`putOnSale()` function doesn't check if the caller is a contract or EOA. Thus the seller is able to filter the buyers by reverting the payment transfers in `buyListing()` function.

## C5. TokenVesting

### Overview

This is a contract that is used for token vesting. The user sends tokens to the contract and it releases them gradually over time.

### Issues

---

#### C5-01 Gas optimizations

● Low

☑ Acknowledged

---

Public functions that are not used inside the contract can be declared as external. Functions such as `revoke()` and `release()`.

SafeMath library is redundant. Since 0.8.0 version of Solidity, there are built-in checks of mathematical operations.

---

#### C5-02 Not enough checks on input data

● Low

☑ Acknowledged

---

In the constructor, there should be a check for the start time to not be set in the past.

---

#### C5-03 Non-linear token release

● Info

☑ Acknowledged

---

The release of tokens is not linear but has a cliff period.

For example, Bob is vesting tokens for 3 months, with a cliff period for 2 months. In this case, he can't withdraw tokens until 2 months have passed, and right after that, he could claim 2/3

of all tokens.

## C5-04 No support for rebasing tokens

● Info

☑ Acknowledged

The math of the contract can crash if the tokens with rebasing such as the AMPL token are used.

The math will crash because the balance of the accounts can decrease in time and this leads to an underflow in math operations in the contract in such functions as `releasableAmount()`, `revoke()`, and `release()`.

# C6. TokenVestingFactory

## Overview

This is a contract that is used for deploying new TokenVesting contracts for users.

## Issues

### C6-01 Gas optimizations

● Low

☑ Acknowledged

The variable `defaultOwner` can be marked as immutable. By doing this, gas consumption can be reduced because the reading of this variable will be gas-free.

Public functions that are not used inside the contract can be declared as external. Functions such as:

1. `vestingContracts()`
2. `timelockContracts()`
3. `deployMultipleTimelockContractAndDepositTokens()`
4. `deployTimelockContractAndDepositTokens()`
5. `deployMultipleVestingContractAndDepositTokens()`
6. `deployVestingContractAndDepositTokens()`

## 7. `deployDefaultVestingContractAndDepositTokens()`

# C7. Resource

## Overview

This contract is an ERC1155 token. It contains all information about table cards. Also has the functionality for users to craft new table cards.

## Issues

### C7-01 Call on unbounded array

 Low Resolved

The enumerable set players only grow with time. The function `getPlayers()` may fail because of the large number of users. The required amount of resources may exceed the limit.

### Update

A new function `getPlayersPaginated()` was added in case the `getPlayers()` function isn't working.

### C7-02 Gas optimization

 Low Resolved

The variable `_phi` can be marked as immutable.

### C7-03 The function `receive()` can be removed

 Low Resolved

The function `receive()` can be removed from the contract code as it always reverts.

### C7-04 Function `resourceCount()` returns a wrong value

 Low Resolved

The function `resourceCount()` returns

```
_tokenIds.current() - 1
```

The actual resource count is equal to `_tokenIds.current()`.

## C7-05 Lack of documentation

● Info

✓ Resolved

We recommend adding NatSpec documentation at least to the public functions of the contract.

## 5. Conclusion

One high severity issue has been found and fixed in the update. We recommend adding tests with coverage of at least 90%, before the deployment to the mainnet.

The 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 the code improving and preventing potential attacks.

The updated contracts are deployed to the mainnet of Avalanche C-Chain:

[0x8aE8be25C23833e0A01Aa200403e826F611f9CD2](#) PHI,

[0xcc367e92c1b2BB0eB503F67654F3581c086eD2fc](#) Resource,

[0x337F2aB0E1A857A03B93d072656D3d52AA4A586A](#) ChestSale,

[0xFcC7E0eCDfF8b1DE9222d1cc4Aae74c24f121cA1](#) Game,

[0x2DAB3390adf79237aF1331bb7Eb4295defE6DA30](#) TokenVestingFactory,

[0x71Ea4a973Be28128a299362015fF4A06b084C70a](#) MarketplaceNew.

## 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](mailto:contact@hashex.org)

 [@hashexbot](https://t.me/hashexbot)

 [blog.hashex.org](https://blog.hashex.org)

 [linkedin](#)

 [github](#)

 [twitter](#)

**#HashEx**  
BLOCKCHAIN SECURITY