#HashEx
Blockchain Security

Gangster City

smart contracts
final audit report

February 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

2. Overview

HashEx was commissioned by the Gangster City team to perform an audit of their smart contract. The audit was conducted between 27/01/2022 and 03/02/2022.

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 the public Github repository after the 2b4811e commit.

Update: the updated code is available after the a8456f6 commit.

The audited contracts are deployed to the Avalanche C-Chain:

Cash : 0xedA770856191f16ac54d3181B2baC8AF0c485b72

GangsterCityNFT: 0x87e8AeE88AE939465dA795D6C32963854Bb5fBA0

Royalties: 0x6Ec1a31A92B9724cfEb71F135C4563D8E8F06585

GangsterCityStaking: 0xcBB55b9bfEAe51EDa8fe7f39e09ECd88C4b4bb46

2.1 Summary

Project name	Gangster City
URL	https://thegangstercity.xyz/

Platform	Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
GangsterCityStaking	`0xcBB55b9bfEAe51EDa8fe7f39e09ECd88C4b4bb46`
GangsterCityNFT	`0x87e8AeE88AE939465dA795D6C32963854Bb5fBA0`
Cash	`0xedA770856191f16ac54d3181B2baC8AF0c485b72`
Royalties	`0x6Ec1a31A92B9724cfEb71F135C4563D8E8F06585`

3. Found issues

18
Total issues

●	High	6 (33%)
●	Medium	3 (17%)
●	Low	7 (39%)
●	Info	2 (11%)

C1. GangsterCityStaking

ID	Severity	Title	Status
C1-01	● High	Not minted yet token can be staked	✓ Resolved
C1-02	● High	Unstaking by an arbitrary user	✓ Resolved
C1-03	● High	unstakeWorker() can be applied to the same token multiple times	✓ Resolved
C1-04	● High	Unstake functions depend on backend service	✓ Partially fixed
C1-05	● High	If the contract runs out of reward tokens users' NFTs may be locked	✓ Resolved
C1-06	● Medium	Rewards can be claimed for an unstaked token	✓ Resolved
C1-07	● Medium	Unbeneficial random seeds can be ignored	✓ Resolved
C1-08	● Low	Payable modifier is redundant	✓ Resolved

C1-09	● Low	Gas optimisation	✓ Resolved
C1-10	● Low	No events	✓ Resolved
C1-11	● Info	No error message in require() statement	✓ Acknowledged

C2. GangsterCityNFT

ID	Severity	Title	Status
C2-01	● High	Signed data can be replicated	✓ Resolved
C2-02	● Medium	Off-chain randomness	✓ Acknowledged
C2-03	● Low	Royalty fees may exceed 100%	✓ Resolved
C2-04	● Low	No parameters check	✓ Resolved
C2-05	● Low	No events	✓ Resolved
C2-06	● Low	Gas optimisation	✓ Acknowledged
C2-07	● Info	Payable modifier is redundant	✓ Resolved

4. Contracts

C1. GangsterCityStaking

Overview

Gamified staking contract for Gangster City's NFTs. Rewards distribution for each game class is described in the project's whitepaper [1]. To unstake an NFT a user has to pay a randomness provider fee. The fee is arbitrary, but not less than 0.007 AVAX.

Issues

C1-01 Not minted yet token can be staked ● High ✓ Resolved

Staking a not yet minted token is possible. The token won't be added to any of the class mappings, but `lastClaimedReward` and `ownerOfDeposit` will be recorded. If the added token gets worker class, `ownerOfDeposit` can freely claim the rewards, without actually staking or even being the owner of the NFT.

Recommendation

Revert the transaction instead of returning a false boolean in `stakeNft()`.

C1-02 Unstaking by an arbitrary user ● High ✓ Resolved

The Unstake functions of the Staking contract are available to an arbitrary `msg.sender`. Without authorization, anyone is able to receive the rewards and unstake NFTs of other NFT owners. Also, users can avoid the undesirable random seed by simply not sending the transaction. And valid signature parameters could be picked up from failed transactions.

Recommendation

We suggest adding a check whether a `msg.sender` is the owner of the deposit in the unstaking functions.

C1-03 unstakeWorker() can be applied to the same token multiple times ● High ✓ Resolved

If an unstaked token is passed to the `unstakeWorker()` function, the transaction won't be reverted in cases when a worker's income is split or burned. Thus, the token can bring endless reward earnings to landlords, business owners, and gangsters, even if it's unstaked.

Recommendation

The issue can be eliminated by an NFT ownership check at the beginning of the `unstakeWorker()` method. If the passed token ID doesn't belong to the staking contract, the transaction should be reverted.

C1-04 Unstake functions depend on backend service ● High Partially fixed

Tokens can be withdrawn only if the `randomness` parameter signed by `randomnessProviderV1` was passed to the unstake function.

```
string memory randomnessString = uint2str(randomness);
...
if(verifyRandomness(randomnessString, v, r, s) != randomnessProviderV1) revert
Unauthorized();
```

If the backend service providing this signed randomness breaks down, users' assets will be locked.

Recommendation

We advise providing the `emegencyWithdraw()` method that would work independently from the signing randomness service. For example, the last successful unstake timestamp can be tracked and if certain time passes since it, `emegencyWithdraw()` can simply return the token to an owner.

Update

The signature is no longer required for unstake functions, but in the updated token withdrawal flow unstake operations are processed with backend service and still depend on it.

C1-05 If the contract runs out of reward tokens users' NFTs may be locked ● High ✓ Resolved

The rewards for staking must be externally fulfilled with reward tokens. If the contract runs out of reward tokens the unstake function will work only in situations when reward token transfer is not needed.

```
if (hasLostNft == false)
     {
         IERC721(NftAddress).safeTransferFrom(
             address(this),
             ownerOfDeposit[tokenId],
             tokenId
         );
     }

     if (finalLandlordReward != 0)
     {
         lastClaimedReward[tokenId] = block.timestamp;
         tokenIdReward[tokenId] = 0;
         IERC20(ErcAddress).transfer(msg.sender, finalLandlordReward);
     }
```

Otherwise, the transaction will be reverted during the reward token `transfer()` method call due to insufficient balance.

Recommendation

We advise providing the `emegencyWithdraw()` method that won't depend on reward token transfer success.

Update

The problem was handled by `hasContractAwardedAllTokens()` function. Before each unstake function, it checks whether there are available reward tokens. If the contract awarded all tokens, staked NFTs will be transferred to their depositors during unstake.

C1-06 Rewards can be claimed for an unstaked token ● Medium ✓ Resolved

Rewards claiming methods revert a transaction if `ownerOfDeposit` is not `msg.sender`. Since the token depositor's address is set in `ownerOfDeposit` mapping while staking and not affected during the unstaking, therefore a user is able to unstake a token and then claim rewards for it.

C1-07 Unbeneficial random seeds can be ignored ● Medium ✓ Resolved

`randomness` is used in unstake functions for `chance` calculation. `chance` determines the token unstake scenario. The scenarios affect the ultimate user's benefit, so they have to be more or less attractive. Since `chance` is obtained as `(randomness % 100) +1`, it can be seen in advance and users can avoid the undesirable random seed by simply not sending the transaction. Also, valid signature parameters could be picked up from failed transactions.

C1-08 Payable modifier is redundant ● Low ✓ Resolved

Rewards claiming and unstake methods should not be payable, as they do not support any logic for `msg.value` and accidentally sent AVAX will permanently stay on the contract.

C1-09 Gas optimisation ● Low ✓ Resolved

a. `dayTimeInSeconds` can be declared as constant. Also, the “days” keyword should be used.

C1-10 No events ● Low ✓ Resolved

No event is emitted when `ErcAddress` or `NftAddress` is changed with the `changeERCNFTAddr()` function. Also, these variables violate Solidity naming conventions, i.e. they're not in mixedCase. Changing the `isStakingEnabled` variable with `changeStakingEnabled()` function requires emitting an event as well.

C1-11 No error message in require() statement ● Info ✓ Acknowledged

If the first requirement in `transferRandomnessCost()` is not satisfied, the transaction will be reverted with an empty message. The absence of transaction failure explanation may be confusing for the user and will complicate problem investigation.

C2. GangsterCityNFT

Overview

ERC-721 token with ERC-165 and ERC-2981 interface support. A created token represents one of the game's characters: worker, landlord, business owner, or gangster.

Issues

C2-01 Signed data can be replicated ● High ✓ Resolved

The `createNft()` function checks signs of passed `tokenURI` and `randomness` parameters to ensure they were generated by `metadataProviderAddress`. However, the current validation has a security breach allowing the re-usage of these parameters in the next calls [3]. An intruder is able to calculate the desired `randomness` parameter and copy it from a mined transaction's data to get minted NFT ownership. The issue also refers to the GangsterCityStaking contract.

Recommendation

Possible remediations are listed in the SWC problem's record [3].

C2-02 Off-chain randomness ● Medium ✓ Acknowledged

A passed `randomness` parameter is received off-chain and can be fabricated, which may lead to a controlled token mint for the desired address. If tokens have a diverse price on token markets this also can be abused by the admin for secretly receiving the most valuable token.

Recommendation

A random parameter can be obtained on-chain with the use of an oracle [2]. This will reinforce users' trust and ensure the absence of the developer's malicious intent.

C2-03 Royalty fees may exceed 100% ● Low ✓ Resolved

The parameter of `setRoyaltiesFees` should be capped since it represents a share in percentage.

C2-04 No parameters check ● Low ✓ Resolved

Taken from `tokenURI` `shouldTransferToGangster` and `selectedClass` mint parameters can mistakenly drop from permissible value ranges. There should be requirements on obtained `shouldTransferToGangster` and `selectedClass` values to prevent a token from having no role and being owned by `msg.sender` instead of `randomGangsterAddress`.

C2-05 No events ● Low ✓ Resolved

No event is emitted when `ErcAddress` or `StakingAddress` is changed with `changeErcAndStakingAddr()` function. Also, these variables violate Solidity naming conventions, i.e. they're not in mixedCase.

C2-06 Gas optimisation ● Low ✓ Acknowledged

a. The `randomGangsterAddress` receive function call can be skipped for tokens with id equal or lower than 1000;

b. L115 `k` variable declaration is excessive. `len` can be used;

c. L229, 237, 245, 253, 262 multiple storage read;

d. L118, 121 `(_i / 10)` operation duplication. Also, instead of the expression `_i - (_i / 10) * 10` the modulo operator could be used.

C2-07 Payable modifier is redundant ● Info ✓ Resolved

The `address` parameter of the `withdrawERCBalance()` function should not be payable as it is not involved in the native currency transfer operation within the function.

C3. Cash

Overview

Granted for staking ERC-20 token. No issues were found.

C4. Royalties

Overview

Royalties distribution contract, splitting royalties between community and project owner. No issues were found.

5. Conclusion

6 high severity issues were found, 5 of them have been fixed with the update and 1 has been fixed partially. The reviewed 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. We recommend adding tests with coverage of at least 90% with any updates in the future.

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.
- **Info.** Issues that do not impact the contract operation. Usually, info 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

Appendix C. References

1. Gangster City - Agony&Chaos, Whitepaper.

2. Ethereum.org - Oracles.

3. Missing Protection against Signature Replay Attacks.

contact@hashex.org

@hashexbot

blog.hashex.org

linkedin

github

twitter

HashEx
Blockchain Security