#HashEx
BLOCKCHAIN SECURITY

Anito

smart contracts final audit report

June 2022

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 Anito team to perform an audit of their smart contract. The audit was conducted between 8/06/2022 and 14/06/2022.

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 in the Binance testnet:

Laro 0xC2B980821010dC7DCB6E00D02cAD6933e41bb2AC,

Ginto 0xebCd9C071792ab6fd2B349246DBc3ed6F335b818,

Anito 0x786FF6AA212B10E3B1f24D27623dB8d95F893b6d,

Stones 0x3F494a389251a07559c02329e69011748a2296dD,

Marketplace 0x1B89457d8cf48125632c3Bd112D8092E0B294590,

Summoner 0xceb4Aa3e864F844EEFa2C681A3262Cc443357eb5,

MasterChef 0x50E0208E60482CddEbC30F00d94c7DDdD4DFFd24,

Reward Manager 0xFd09FcdaCf75a144B13053b6eDdAfFE1fF03Bf7a.

Update: the Anito team has responded to this report. The updated code can be found at:

Laro 0x7bCfE5752B2CC8fe035806F21f61bF7880eE9374,

Ginto 0x4Dd0FFcbF2F542894e5DacAE7e18A78D2dC1c62e,

Anito 0xd6e8F7aEDA13d2dA86d3a9245801E2daE7c7dcF3,

Stones 0x49Ac17e3e1C2076d28BbceEB646CD71562198DaC,

Marketplace 0x8699661653B8360014E9688f9aB14b862fA793Fb,

Summoner 0xbE19009582AD1799B6fde595ad10Be86E3A43577,

MasterChef 0x3640Eae2551600D63B2049669d6dbF55cef8B81b,

Reward Manager 0xAF535e7e0232cb53179dBCb052daEFF4815D0E29.

2.1 Summary

Project name	Anito
URL	https://anitolegends.com
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
BEP20Detailed	
BEP20	
Laro	
BEPPausable	

Anito

Stones

MasterChef

RewardManager

Summoner

Marketplace

Ginto

3. Found issues

Critical	1 (4%)
High	4 (14%)
Medium	6 (21%)
Low	13 (46%)
Info	4 (15%)

C1. BEP20Detailed

ID	Severity	Title	Status
C1-01	Low	Gas optimization	Acknowledged

C2. BEP20

ID	Severity	Title	Status
C2-01	High	Excessive owner's rights	Resolved

C5. Anito

ID	Severity	Title	Status
C5-01	High	Excessive owner's rights	Resolved
C5-02	High	Excessive operator's rights	Partially fixed
C5-03	Medium	Max summon count is hardcoded	Resolved

C5-04	Low	Gas optimization	Acknowledged
C5-05	Low	No events	Acknowledged
C5-06	Info	Code style	Acknowledged
C5-07	Info	baseURI public visibility	Acknowledged

C6. Stones

ID	Severity	Title	Status
C6-01	High	Excessive operator's rights	Partially fixed
C6-02	Low	Gas optimization	Acknowledged
C6-03	Low	No events	Acknowledged
C6-04	Info	Code style	Acknowledged

C7. MasterChef

ID	Severity	Title	Status
C7-01	Medium	No guarantees the reward contract has enough tokens	Acknowledged
C7-02	Low	massUpdate flag is optional	Acknowledged
C7-03	Low	No support of tokens with commissions	Acknowledged
C7-04	Low	Gas optimization	Acknowledged
C7-05	Info	harvestAll() can fail if there are too many pools	Acknowledged

C8. RewardManager

ID	Severity	Title	Status
C8-01	Medium	Excessive owner's rights	Acknowledged
C8-02	Low	No events	Acknowledged
C8-03	Low	Gas optimization	Acknowledged

C9. Summoner

ID	Severity	Title	Status
C9-01	Medium	Stones constants point to similar integer	Resolved
C9-02	Medium	Optional booleans in summon()	Resolved
C9-03	Low	Gas optimization	Partially fixed

C10. Marketplace

ID	Severity	Title	Status
C10-01	Critical	createMarketSale() doesn't require market item isn't sold	Resolved
C10-02	Medium	setAllowAsset() may lead to unsupplied MarketItems	Partially fixed
C10-03	Low	Gas optimization	Partially fixed

C11. Ginto

ID	Severity	Title	Status
C11-01	Low	Gas optimization	Acknowledged

4. Contracts

C1. BEP20Detailed

Overview

Part of the Laro inheritance scheme.

Issues

C1-01 Gas optimization

Low | Acknowledged

`_name`, `_symbol`, `_decimals` shoud be marked immutable.

C2. BEP20

Overview

ERC20 interface implementation. Part of the Laro inheritance scheme.

Issues

C2-01 Excessive owner's rights

High | Resolved

Critical token transfer methods have a `whenNotPaused` modifier. If the owner pauses the contract, token transfers will become unavailable.

Recommendation

Remove pause logic or renounce ownership.

C3. Laro

Overview

An ERC20 interface implementation. Like BUSD, it is used as a payment method for market items in the Marketplace contract. Reward medium for the MasterChef contract. One of two payment mediums for stones in the Stones contract. Token transfers can be paused by the owner. No issues were found.

C4. BEPPausable

Overview

The base contract which allows descendants to implement an emergency stop mechanism. Part of the Laro inheritance scheme. No issues were found.

C5. Anito

Overview

An ERC721 interface implementation

The NFTs are minted by operators either directly by mint methods or via summoning the apprentice: passing 2 "masters" NFTs owned by a user, another Anito NFT (apprentice) is minted for the user.

Issues

C5-01 Excessive owner's rights

High Resolved

The owner can pause all NFT transfers. Hasty contract pause will arouse inevitable tokens' price collapse and a loss of users' assets.

Recommendation

Remove pause logic or renounce ownership.

C5-02 Excessive operator's rights

High Partially fixed

Operators' list should include only verified and reliable addresses, as they have access to uncontrolled mint and summoning. Also, one operator can revoke another operator's role including himself.

Recommendation

Grant the operator role only to the `Summoner` contract passing its address as an argument in the constructor or manually delete all other operators except `Summoner`.

Update

Now only the owner can include/exclude operators, but they still have access to uncontrolled mint and summoning.

C5-03 Max summon count is hardcoded

Medium Resolved

Max summon count is a hardcoded value in `executeSummoning()`:

```
require(anitoInfo[master1].summonCount < 7 && anitoInfo[master2].summonCount < 7,"Maximum
apprentice reached");
```

Therefore the `maxSummonCount` variable change will not have an effect and the max possible

summon count is always constant;

Recommendation

Replace the problematic line with the snippet below:

```
require(anitoInfo[master1].summonCount < maxSummonCount && anitoInfo[master2].summonCount
< maxSummonCount,"Maximum apprentice reached");
```

C5-04 Gas optimization

Low | Acknowledged

a. `baseExtension`, `LAUNCH_MAX_SUPPLY`, should be const.

b. `LAUNCHPAD` should be marked as immutable.

c. L:50 requirement should be checked in the constructor during token creation.

d. `getMaxLaunchpadSupply()`, `getLaunchpadSupply()` are redundant as `LAUNCH_MAX_SUPPLY` and `LAUNCH_SUPPLY` have public visibility.

e. Direct boolean comparison in L109.

f. `notRevealedUri` is redundant. In L110 a return `“”` could be used instead.

g. The `ReentrancyGuard` inheritance is not used.

h. Instead of `LAUNCH_SUPPLY` incrementation in the `mintTo()` function just add `size` after the for loop.

i. `getMaxLaunchpadSupply()`, `getLaunchpadSupply()`, `pause()`, `unpause()`, `addOperationsAddress()`, `removeOperationsAddress()`, `setMaxSummonCount()`, `getAnitoOfAddress()` should have external visibility.

C5-05 No events

Low Acknowledged

No events are emitted in `reveal()`, `addOperationsAddress()`, `removeOperationsAddress()`, `setMaxSummonCount()`.

C5-06 Code style

Info Acknowledged

a. The `ERC721` import is unnecessary since it's already included in `ERC721Enumerable`.

b. `maxSummonCount`, `operationsAddresses` have default visibility.

c. Better to use multi-line comment notation `/**/` in L200-203.

d. No space after requirement in the first argument in L170-171.

e. Missed spaces in for loop L:161.

f. The `Zero` address is more preferable than the `empty` address in the L205 error message.

g. A redundant dev comment in L62.

h. No error messages in requiremnts L133, 138.

C5-07 baseURI public visibility

Info Acknowledged

`baseURI` can be seen even if the reveal flag is false since `baseURI` has public visibility. However, if `baseURI` stores confidential information which premature disclosure can lead to certain losses, it should not be stored in the blockchain at all if its sudden publicity causes any undesirable consequences.

C6. Stones

Overview

ERC1155Supply interface implementation.

Initially, the owner of the contract sets prices for the "stones" tokens in Laro and Ginto.

Then tokens are bought by users, sending the Laro to treasuryAddress and Ginto to `0xdead` (arbitrary non-existent address, implicit burn).

There are 4 stone types used to pay for summoning: `ST_SUMMONING_STONE` is obligatory, and `ST_RARITY_STONE`, `ST_CLASS_STONE`, `ST_STAT_STONE` are optional.

Issues

C6-01 Excessive operator's rights

High | Partially fixed

Operators have uncontrolled access to mint. Also, one operator can revoke another operator's role including his own.

Recommendation

Operators must be `Timelock` contracts with a minimum delay of at least 24 hours. This won't stop the operators from possible rights abuse but it will help users to be informed about upcoming changes.

Update

Now only the owner can include/exclude operators, but they still have access to uncontrolled mint.

C6-02 Gas optimization

Low | Acknowledged

a. `Pausable` inheritance is not used.

b. Constant variables are not used in L17-L20.

c. Direct boolean comparison in L34.

d. `setTreasuryAddress()`, `setPrices()`, `setTokenName()`, `addOperationsAddress()`, `removeOperationsAddress()`, `setURI()`, `mint()`, `mintBatch()`, `buyStone()` should have external visibility.

C6-03 No events

Low | Acknowledged

No events are emitted in `setTreasuryAddress()`, `setPrices()`, `setTokenName()`, `addOperationsAddress()`, `removeOperationsAddress()`, `setURI()`.

C6-04 Code style

Info | Acknowledged

a. Ginto token can be burnt instead of being sent to the `0xdead` address in the `buyStone()` function. This will help keeping `totalSupply()` more precise;

b. `operationsAddresses` mapping has default visibility.

C7. MasterChef

Overview

Staking contract with Laro reward token.

Issues

C7-01 No guarantees the reward contract has enough tokens

Medium | Acknowledged

Reward tokens are stored in the `RewardManager` address and are transferred to the user during `withdraw()`, `deposit()`, and `harvestAll()` functions calls. Since it's not checked that `RewardManager` has sufficient balance and allowance during pool updates, reward payments are not ensured to a user. Furthermore, the owner can set an arbitrary `rewardManager` address with `setRewardManager()`.

Recommendation

Redesign the architecture that MasterChef mints tokens by itself or keeps distributed amount of tokens.

C7-02 massUpdate flag is optional

Low | Acknowledged

`add()` and `set()` functions have an optional `_withUpdate` flag which calls `massUpdatePools()` if set `false` and may cause unfair rewards in case of rarely updated pools, see more info here.

C7-03 No support of tokens with commissions

Low | Acknowledged

The deposit function doesn't have a check on the actual transferred amount of deposited tokens. The owner must not add pools with tokens with fees on transfers. In general, any rebasing token is not supported.

C7-04 Gas optimization

Low | Acknowledged

a. `devAddress` is not used anywhere except `setDevAddress()` function.

b. `laro` can be sent directly to the ultimate receiver in `safeLaroTransfer()`.

c. `add()`, `set()`, `pendingReward()`, `deposit()`, `withdraw()`, `harvestAll()`, `emergencyWithdraw()`,

`setDevAddress()`, `setTreasuryAddress()`, `updatestartTime()` should have external visibility.

d. `DEPOSIT_FEE_CAP`, `ALLOC_POINT_CAP` should be const, also no visibility specified for `DEPOSIT_FEE_CAP`.

e. `laro` should be marked immutable.

f. `poolInfo.length` in `massUpdatePools()` should be stored in the local variable.

C7-05 harvestAll() can fail if there are too many pools ● Info ⊘ Acknowledged

The `MasterChef` interface has only the `harvestAll()` function to collect rewards without amount withdrawal. Under the hood, it iterates over the pools and sums up all pending rewards. If the number of pools is great enough, iteration may exceed the block gas limit and all `harvestAll()` calls will be reverted. Users will have to collect their rewards with `withdraw()` and `deposit()` functions.

C8. RewardManager

Overview

Masterchef's reward tokens' holder.

It approves (non-automatically) the MAX_INT allowance of the specified ERC20 token for the specified Masterchef contract.

A multisig account set by the owner or operator is able to withdraw ERC20 tokens from the RewardManager contract.

Issues

C8-01 Excessive owner's rights

Medium | Acknowledged

`Multisig` wallet can withdraw all MasterChef reward tokens with `withdrawERC20()`.

Recommendation

Add a requirement `_token` is not equal to the `Laro` address.

C8-02 No events

Low | Acknowledged

No events are emitted in `setOperator()`, `setMasterchef()`, `setMultisig()`.

C8-03 Gas optimization

Low | Acknowledged

a. The `SafeERC20` library is not used.

b. `setOperator()`, `setMasterchef()`, `setApproval()`, `setMultisig()`, `withdrawERC20()` should have external visibility.

c. `onlyOperator()` modifier is never used.

C9. Summoner

Overview

An operator contract for the `Anito` token, calling `executeSummoning` method on it.

It decreases the user's Stones balance of type `ST_SUMMONING_STONE` by 2, and arbitrary (depending on user's input) decreasing by 1 Stat, Class, or Rarity stones.

Issues

C9-01 Stones constants point to similar integer

Medium Resolved

`ST_RARITY_STONE`, `ST_CLASS_STONE`, `ST_STAT_STONE` point to the same integer value 1, providing transfers of the same type in `summon()`.

Recommendation

Assign different values to stone types.

C9-02 Optional booleans in summon()

Medium Resolved

a. A user can set all optional booleans to false and execute `summon()` only for 2 `ST_SUMMONING_STONE` tokens.

b. A user can accidentally set multiple booleans to true and pay multiple times for summoning.

Recommendation

a. Require one of the booleans is set to true.

b. Use `else if` statement instead of `if` in L57, 62.

C9-03 Gas optimization

Low Partially fixed

a. `anitoNFT`, `stones` should be marked as immutable.

b. Direct boolean comparison in L27.

c. The `maintenanceMode` variable is not used.

d. `ERC20`, `ERC721` imports are not used.

e. L45-46 requirements are double-checked in anito `executeSummoning()` function.

f. The `operationsAddresses` mapping and `onlyOperators` modifier are not used.

g. Consider using just `IERC1155` interface instead of whole contract import.

h. Inherited functionality of the `Ownable` contract is unused.

i. `pause()`, `unpause()`, `summon()` should have external visibility.

C10. Marketplace

Overview

ERC721Holder and ERC1155Holder implementation contract.

Marketplace where ERC721 and ERC1155 can be changed to Laro or BUSD.

Issues

C10-01 createMarketSale() doesn't require market item isn't sold

Critical Resolved

The `createMarketSale()` function doesn't make a vital check that the passed `itemId` argument hadn't been already sold. This oversight leads to serious security breaches in the NFT marketplace and endangers the safety of users' NFTs. A malefactor can utilize this vulnerability in one of the following ways. ERC721 can be bought with the lowest historical sale price on market place, i.e. if an NFT was sold with one price and after some period of time it was placed on market place again with price gains it can be bought with itemId of the previous sale and consequently, it's price. Moreover, even this partial payment will go to the `seller` of the previous `itemId`. ERC1155 can be exploited quite similarly, but in this case, a hacker may take into account multi-token standard and buy small amounts of some type, place them for peanuts on the marketplace and then exhaust all tokens of this type by multiple purchase of his own `itemId`.

Recommendation

Add constraint on `itemId` that it isn't sold:

```
require(!idToMarketItem[itemId].sold, "ItemId is already sold");
```

C10-02 setAllowAsset() may lead to unsupplied MarketItems

Medium | Partially fixed

As the `setAllowAsset()` method doesn't provide value for `assetTypes`, adding new contracts may lead to `MarketItem` unsupplied with ERC721 or ERC1155. If the user mistakenly buys the `MarketItem`, he transfers `Laro` or `BUSD` to a seller and gets nothing in exchange. Consider providing value for `assetTypes` in `setAllowAsset()` or make users check whether they buy proper `MarketItem`.

Recommendation

Require `_assetAddress` value in `assetTypes` mapping isn't null.

Update

Although `assetTypes[_assetAddress]` can be modified in the new version, the parameter provided by user may contain unsuppoted by market asset type.

Consider adding the requirement below in `setAllowAsset()`:

```
require(_assetType == ASSET_ERC721 || _assetType == ASSET_ERC1155, "Unexpected asset
type");
```

C10-03 Gas optimization

Low | Partially fixed

a. All public functions in the contract should have external visibility.

b. Castings of `msg.sender` to payable type are unnecessary L150,151,202, 247.

C11. Ginto

Overview

An ERC20 token implementation.

Minted either by the owner or by another user with previously signed messages by the signer once in 2 weeks.

Blacklisting both for sender and receiver is implemented.

Used as one of the payment mediums for stones in the Stones contract.

Issues

C11-01 Gas optimization

Low Acknowledged

a. Ineriantace of `Pausable` and `ReentrancyGuard` is unnecessary;

b. `BURNER_ROLE` is not used.

5. Conclusion

1 critical, 4 high, and 6 medium severity issues were found. 1 critical, 2 high, and 2 medium severity issues have been resolved with the update, while 2 high and 1 medium severity ones have bee resolved 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 code improvement and the prevention of 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.
- **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