WorkX Security Analysis

by Pessimistic

This report is public

January 17, 2024

Abstract

In this report, we consider the security of smart contracts of WorkX project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of WorkX smart contracts. We described the audit process in the section below.

During the three stages of the audit (they are described in the audit process), we found critical, medium and low severity issues. In the current version, they have the following statuses:

- The critical issue with `addressed` status: Possible lock of unstake/destroyNft;
- The medium severity issues:
  - `fixed` status: Dubious casts, Misleading documentation, No signature deadline;
  - `commented` status: Project roles;
  - `not fixed` status Insufficient documentation.

Almost all issues of low severity and notes were fixed or commented on by the developers. During the current recheck, we discovered several new issues of low severity. All the tests passed.

General recommendations

We recommend fixing the mentioned issues and improving the documentation. We also recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with WorkX project on a public GitHub repository, commit ed78872035f962abb375d65193e32f3589038ae8.

The scope of the audit included the following contracts:

- contracts/nft/GenesisNft.sol;
- contracts/nft/GenesisNftAttributes.sol;
- contracts/nft/GenesisNftData.sol;
- contracts/sale/TokenDistribution.sol.

The documentation for the project included the following whitepaper.

All 383 tests pass successfully. The code coverage is 82.66%.

The total LOC of audited sources is 1395.

Audit process

This audit consisted of three stages: the initial audit, the first recheck, and the second recheck. This is the final version of the report after the second recheck. These stages have the following work dates and commits:

- The initial audit - `f528b541dda660d07e4ae76de1e412306705eaef`, November 29 - December 4, 2023;
- The first recheck - `34f3460a937c33ac35b53fabbb6669d034ead66d`, December 7 - 8, 2023;
- The second recheck - `ed78872035f962abb375d65193e32f3589038ae8`, January 9 - 17, 2024.

Besides, the **TokenDistribution** contract was checked separately.

We provided interim results for the developers in the form of two public hackmd: one, two.

This report displays all issues with current status that were found during the audit. The lines in the issue descriptions may not correspond to the current commit, as they could be fixed in the previous recheck.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether signatures are used securely;
- Whether the logic corresponds to the developers' description;
- Whether calculations with short types are correct;
- How much impact an owner can have on a project;
- Standard Solidity issues.

We scanned the project with the following tools and manually verified the output:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules for Slither;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in the report all the verified issues we found during the manual audit or discovered by automated tools.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Possible lock of unstake/destroyNft (addressed)

The `GenesisNft.mintNft` function lacks checks on the conclusion of the minting process, and the signatures used for minting do not use any deadline mechanism. Additionally, the usage of `monthlyTotal[0]` to update fields such as `totalShares`, `totalStaked`, and `minimumStaked` might result in the following scenario in `mintNft` function of **GenesisNft** contract:

- An exploiter receives his signature (e.g. `TYPE_INV`) and does not mint NFT with it;
- Other accounts mint some amount of NFTs;
- On any month (except `0`), holder of NFT calls some function which uses `_updateMonthly` inside of it. This call will find the most recent month with information about total stakes and copy this information for the current month;
- After that, the exploiter uses their signature to mint NFT and increase the parameters of `_monthlyTotal[0]` at lines 226-229;
- This action leads to an incorrect value of `_monthlyTotal[getCurrentMonth()]` as exploiter has some shares, but they are not accounted at the `_monthlyTotal[getCurrentMonth()]`;
- This issue will lead to `revert` during `unstake`/`destroyNft` calls for some users when the substractions at line 344 of the `_updateShares` function or at lines 442 and 446 of the `_updateMonthly` function will cause an underflow error as the fields of `_monthlyTotal[getCurrentMonth()]` contains values less than the actual stakes or shares.

> *The contract has been deployed and all tokens have already been minted.*

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Dubious casts (fixed)

There are several places in the **GenesisNft** contract where dubious casts occur, and some of them can lead to unexpected behavior:

- Throughout the code the `uint256` value of the month is casted unsafely to `uint8`. If a user stakes additional tokens during the valid period, there is a potential situation where `getCurrentMonth()` could return a value of `256`. This would lead to a value of `0` when casted to `uint8`.

  Consequently, if an NFT is destroyed at that particular moment, it might result in a loss of funds (as the staked amount at `nft.monthly[0]` is less than the real value;
- At lines 410 and 418 `amount` is cast unsafely to `uint128`;
- At line 146 `_startTime` is downcasted to `uint128`. It allows passing the checks at lines 144-145 and providing any value lower than `block.timestamp`. After that action, the owner can destroy their NFT and use obtained tokens in the same transaction.

The issues have been fixed and are not present in the latest version of the code.

M02. Insufficient documentation

The codebase of the project is well-commented. However, the comments provide only a brief description of the functionality of the contracts and do not explain the overall project structure. Due to the lack of completeness of documentation, we discussed our doubts with the developers and resolved points of contention. But there are several unclear points:

- **(fixed)** The formula of the `GenesisNftData.calculateShares` function. It seems that the parameters in the formula are chosen empirically. Therefore, we cannot fully verify the correctness of its implementation without a description;

  This function has been removed.
- We cannot fully verify the correctness of the `GenesisNftData.shares` implementation without a description.

M03. Misleading documentation (fixed)

The `mintNft` function of the **GenesisNft** contract does not match the comment at line 177 (`Only the caller of the mintNft function can be the receiving _account, because evolveTier is checking if the msg.sender is the owner of the nft.`). Since it does not check that `msg.sender` is equal to `_account` from the voucher.

The issue has been fixed and is not present in the latest version of the code.

M04. No signature deadline (fixed)

The absence of a deadline mechanism for signatures in the `mintNft` function of the **GenesisNft** contract leads to a critical issue Possible lock of unstake/destroyNft. Implementing a time limit for the consumption of signatures is a recommended practice to mitigate such problems.

The issue has been fixed and is not present in the latest version of the code.

M05. Project roles (commented)

The `owner` has the following options:

- The `setInitCompleted` function can be called before all attributes of all nfts are set;
- Incorrect data can be passed through the `setIpfsFolder` function;

> *Comment from the developers: The `setIpfsFolder` function governed by the owner behind a multisig wallet (and later the DAO) is required because the IPFS folder should be adjustable to ensure the lifespan of the NFT would be not be limited by the validity of the IPFS hashing algorithm. This is why most major NFTs also have implemented governance on this and it considered to be best practise.*

- **(addressed)** If the `owner` changes a signer of signatures with the help of the `setVoucherSigner` function, no one can mint NFT;

> *Comment from the developers: Since the owner is the same team as the voucher signer and we want the mint to succeed this temporary possibility is not in our interest, the function is there in order to take action in case the voucher signer residing in a backend server gets compromised. Otherwise, no action can be taken, again owner is in a governed multisig. Also since the mint has now completed this specific owner function has no effect on the contract anymore.*

- The `setStartTime` function allows a shift in the ability to call `destroyNft`/`unstake` by time;

> *Comment from the developers: Should be solved as above.*

- The attribute encoding is not checked in the `setNftAttributes` function. It can lead to incorrect decoding;

> *Comment from the developers: It does not give more or less power than if the metadata would have been uploaded as json to IPFS... and after the init has been completed it can never be changed again, this owner function will stop to have any effect. It is in fact more secure and future proof than it would have been on IPFS.*

- **(addressed)** The `GenesisNft.mintRemainingToTreasury` function can be called during minting NFT for users.

> *The contract has already been deployed and all users NFTs are minted.*

In the current implementation, the system depends heavily on the owner. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig in **GenesisNft** contract.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. CEI (fixed)

The CEI (checks-effects-interactions) pattern is violated in the `GenesisNft.mintNft` function at lines 207 and 211.

The issues have been fixed and are not present in the latest version of the code.

L02. Duplicate balance check (fixed)

The `GenesisNft._refundTokens` function checks that there is enough token for transfer at line 388. However, this check is performed inside the `transfer` function, and there is no need to perform it just before the transfer of tokens in `_refundTokens` function.

The function has been removed from the contract.

L03. Duplicate code (fixed)

The `GenesisNft.stakeAndEvolve` function duplicates the body of the `GenesisNft.stake` function at lines 269-275.

The issue has been fixed and is not present in the latest version of the code.

L04. External call on public function (fixed)

There is an external call to the `splitBytes` function. However, this function is public. Consider calling it as `internal` method at line 227 in `decodeAttributes` function of **GenesisNftData** contract.

The issue has been fixed and is not present in the latest version of the code.

L05. Gas Consumption (fixed)

The `nftIdCounter` variable in the `GenesisNft.mintNFT` function is read multiple times from the storage. Consider reading it from the storage to the local variable once to decrease gas consumption in the project.

The issue has been fixed and is not present in the latest version of the code.

L06. Implicit internal visibility (fixed)

At lines 33-35 of the **GenesisNft** contract, multiple storage variables are not explicitly marked with `internal` visibility. Consider specifying their visibility explicitly for improved code clarity and readability in **GenesisNft** contract.

The issue has been fixed and is not present in the latest version of the code.

L07. No event in setters (fixed)

The setters of the **GenesisNft** contract do not emit any events. Consider adding appropriate events to make offchain interaction with the contract easier.

The issues have been fixed and are not present in the latest version of the code.

L08. Overcomplicated encoding and decoding (fixed)

`GenesisNftData.splitBytes` function decodes the input bytes to obtain the correct bytes of attributes. However, it could be more efficient and safe if these attributes were supplied as raw bytes. In the current implementation, bytes of the encoded string of bytes are decoded. Such encoding, for example, encodes single byte `\x04` as two symbols, resulting in two separate bytes `\x30\x34` for decoding, overcomplicating the `splitBytes` function in `splitBytes` function of **GenesisNftData** contract.

The issue has been fixed and is not present in the latest version of the code.

L09. Unchecked transfer (fixed)

At lines 377 and 389 of the **GenesisNft** contract there are calls of `transfer` and `transferFrom` functions, the results of which are not validated. As these functions are used with `WORK` token, it will not cause any problems, but it is a good practice to always validate the result of the transfer.

The issues have been fixed and are not present in the latest version of the code.

L10. Unused import (fixed)

`import "hardhat/console.sol";` is imported but not used at line 14 in **GenesisNft** contract.

The issue has been fixed and is not present in the latest version of the code.

L11. Using small ints as values of a mapping (fixed)

At line 37 of **GenesisNft** contract, there is `uint16` value stored inside of the mapping. Consider using `uint256` for mapping values as they use a single slot for storing but are cheaper to read in terms of gas, compared with short types like `uint16`.

The issues have been fixed and are not present in the latest version of the code.

L12. Public vs external (fixed)

There are several places in the code where `external` can be used instead of `public` as these functions are called only externally in the project:

- `GenesisNft.stake` function;
- `GenesisNftData.tokenUriTraits` function.

The issues have been fixed and are not present in the latest version of the code.

L13. Implicit interface implementation

The **GenesisNft** contract imports an **ITokenDistribution** interface. This way the contract can interact with the **TokenDistribution** contract that implements the interface implicitly.

We recommend adding all implemented interfaces to the inheritance list. It guarantees that the interfaces utilized for integration are correct.

L14. Unstake fails on underflow (fixed)

The `GenesisNft.unstake` function does not explicitly validate whether the `amount` is less than or equal to the `stakedAmount`. This situation could potentially lead to an underflow error occurring at line 297. It would be better to include a failure condition with an informative message to verify that there is enough `stakedAmount` to withdraw just before the subtraction operation in `stakeAndEvolve` function of **GenesisNft** contract.

The issue has been fixed and is not present in the latest version of the code.

L15. Problems with withdrawTokens (new)

There are several issues with `GenesisNFT.withdrawTokens` function:

- Results of `transfer` calls are not validated. Moreover, `IERC20.transfer` expects that `bool` value will be returned. However, there are tokens that do not return any value e.g. USDT. In this case even successful transfers of such tokens will revert because compiled code will validate the size of the return data. Consider using SafeERC20.safeTransfer to handle all the cases mentioned earlier.
- There is no functionality to withdraw native tokens using this function. At the same time, `payable` keyword allows to send ether to the contract without the functionality to withdraw it. Consider removing `payable` keyword and adding code for native withdrawals.

L16. Misleading comment (new)

According to the `After the minting period has ended, the remaining NFT will be minted to the treasury account` comment, the `GenesisNft.mintRemainingToTreasury` function should transfer all the remaining NFTs to the treasury instead of owner.

Notes

N01. Falling of mint with valid signature (commented)

Users with types `TYPE_GUAR` and `TYPE_FCFS` can miss their mint with a valid signature if the transaction has pending status in mempool due to unforeseeable circumstances e.g. spikes in gas price at lines 197–205 in the `mintNft` function of the **GenesisNft** contract.

> *Comment from the developers: Guaranteed voucher generation lasts for 4 hours, FCFS voucher generation for only 2 hours. Our NFT partner expects all participants do it in the first minutes of the timeslot. At the end of the TYPE_INV slot (3 days, the foundation will mint the remaining NFT's to a treasury account).*

N02. Potential reentrancy (fixed)

The `_safeMint` function has the external call to any contract inside the `_checkOnERC721Received` function. The `mintNft` function is protected with `nonReentrant` modifier, but it does not protect from read-only reentrancy or other cross-contracts reentrancies. In this case, it does not cause any problems, but it is still potentially vulnerable at line 211 in `mintNft` function of **GenesisNft** contract.

The issue has been fixed. The developers have replaced `_safeMint` function with `_mint` function.

N03. Transfer with staked tokens excesses (commented)

The transfer of the NFT does not trigger the unstaking of any excess tokens for the previous owner, even if the owner could call the `unstake` function. The desired behavior might involve unstaking all available excess tokens during the transfer process in the **GenesisNft** contract.

> *Comment from the developers: As the NFT is a staking vehicle, it can be sold with all tokens inside giving intrinsic value to the NFT. The desired behaviour is that control over all tokens inside it transfer to the new owner when the NFT is sold.*

N04. The storage variable can be empty (commented)

In scenarios where there are no `stakes` or `unstakes` during certain months, it is possible that specific values within `_monthlyTotal` of the **GenesisNft** contract might remain uninitialized.

Consequently, the accurate values for such months will be stored in the preceding valid month's data. It is important to consider this behavior, especially when implementing a reward mechanism that relies on these values at line 52 in **GenesisNft** contract.

> *Comment from the developers: The intended use, provided by the getter functions, is to loop back to the most recent month that specifies values.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Senior Security Engineer
Oleg Bobrov, Junior Security Engineer
Irina Vikhareva, Project Manager
Konstantin Zherebtsov, Business Development Lead

January 17, 2024