SOLIDIFIED

Audit Report for Braintrust - September 13, 2022

Summary

Audit Report prepared by Solidified covering the Braintrust NFT smart contracts.

Process and Delivery

Independent Solidified experts performed an unbiased and isolated audit of the code. The debrief was on 12 August 2022.

Audited Files

The source code has been supplied in the form of a source code repository:

https://github.com/oak-security/audit-braintrust-nft

Final Commit hash: `70301189ac59f9b02d123a1c57da15ebe2c97b09`

This version of the codebase has been deployed to address:
`0xe977321ABf636620ac78107FAe3AB0EB99E303F7`

Files audited:

```
contracts
├── BNFT.sol
└── IBTRST.sol
```

Intended Behavior

The smart contracts implement the Braintrust membership NFT.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases have their limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for Braintrust - September 13, 2022

Issues Found

Solidified found that the Braintrust NFT contracts contain 0 critical issues, 1 major issues and 3 minor issues, 7 informational notes complete the report.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Anyone can set an arbitrary external id for a beneficiary	Major	Resolved
2	Only BNFT token holders can call BNFT.deposit	Minor	Resolved
3	A user can have more than one membership BNFT	Minor	Resolved
4	NFT token id is set on every deposit if the token id is 0	Minor	Resolved
5	Anyone can create zero-value locked deposits	Note	Resolved
6	Use of floating pragma	Note	Resolved
7	Deposits can be locked for zero seconds	Note	Resolved
8	Relayer could be set to the zero-address	Note	Resolved
9	setRelayer does not emit an event	Note	Resolved
10	BNFT.getTotalLockedDepositAmount and BNFT.getTotalLockedDepositByAddress could get unexecutable within a write transactions due to reaching the block gas limit	Note	Acknowledged
11	Miscellaneous Comments	Note	Resolved / Acknowledged

Critical Issues

No critical issues found.

Major Issues

1. Anyone can set an arbitrary external id for a beneficiary

Braintrust users have a unique off-chain member ID which can be set when depositing `$BTRST` ERC-20 tokens via the `BNFT.deposit` and `BNFT.lock` functions. However, as anyone can deposit funds on behalf of a beneficiary, a malicious user can set an arbitrary external id for any beneficiary address. Once an external id is set for a deposit on behalf of a beneficiary, this wrongly set external id cannot be changed again. This could lead to off-chain issues, depending on the use of the external id.

Recommendation

Consider adding a mapping `mapping(address => uint256) public externalIdByBeneficiary` in storage to store the `externalId` for a given beneficiary address within the `BNFT.mint` function. Instead of providing the `externalId` as a function parameter in `BNFT.deposit` and `BNFT.lock`, use the previously stored `externalId` value from the `externalIdByBeneficiary` mapping.

In general, the smart contracts do not make any significant use of `externalId`, other than storing it and emitting its data where needed. Since the accuracy of the `externalId` according to the address of the user is not verified on-chain, and there is one-to-one relation between `externalId` and `beneficiary` address, it might not be necessary to store the `externalId` on-chain.

Minor Issues

2. Only BNFT token holders can call BNFT.deposit

According to the NatSpec documentation of the `BNFT.deposit` function, anyone should be able to deposit `$BTRST` ERC-20 tokens on behalf of a `beneficiary`. However, due to checking the `BNFT` token balance of the `msg.sender` address, only owners of a `BNFT` token can call the function.

Recommendation

Consider removing the `balanceOf(msg.sender) <= 0` check as the subsequent if statement already ensures that the beneficiary address owns a membership NFT.

3. A user can have more than one membership BNFT

The `relayer` could potentially mint more than one NFT to the same address. This might create inconsistencies and confusion in cases where two `tokenIds` are associated with the same `externalId`.

Recommendation

Since the NFT is used to represent membership it makes sense to restrict the minting to a maximum of one per user.

4. NFT token id is set on every deposit if the token id is 0

The `nftTokenId` is set once on the initial deposit with the `BNFT.deposit` function. The current implementation assumes `nftTokenId` is uninitialized if the value is equal to `0`. However, NFT token ids start with a value of `0`, therefore deposits for the BNFT with the token id `0` will set the value on each deposit.

Recommendation

Consider minting NFTs with token ids starting with a value of `1`.

Informational Notes

5. Anyone can create zero-value locked deposits

The `BNFT.lock` function is used to create a locked deposit of `$BTRST` ERC-20 tokens. However, by providing the function argument `amount = 0`, no token transfers take place, but deposits are unnecessarily added to `lockedDeposits`.

Recommendation

Consider adding a check to the `BNFT.lock` function to ensure the `amount` is larger than `0` to prevent zero-value deposits.

6. Use of floating pragma

It is best for contracts to be deployed with the same compiler version and flags that they have been tested with. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

https://swcregistry.io/docs/SWC-103

Recommendation

We recommend locking the pragma version in the `BNFT.sol` contract and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

7. Deposits can be locked for zero seconds

By calling the `BNFT.lock` function with the parameter `availableTimeInSeconds` set to a value of `0`, the deposit is not locked and can be immediately unlocked within the same block.

Recommendation

Consider adding reasonable lower and upper bounds for `availableTimeInSeconds`.

8. Relayer could be set to the zero-address

When a new relayer is set through `setRelayer()` the address of the new relayer is not checked against a possibility of setting the `relayer` to the zero-address.

Recommendation

Consider introducing a zero-address check in the `setRelayer()` function.

9. `setRelayer` does not emit an event

In function `setRelayer()`, no event is emitted for the change of the relayer.

Recommendation

Consider emitting an event if the calling application might need to know about a potential `relayer` change.

10. `BNFT.getTotalLockedDepositAmount` and `BNFT.getTotalLockedDepositByAddress` could get unexecutable within a write transactions due to reaching the block gas limit

The `BNFT.getTotalLockedDepositAmount` and `BNFT.getTotalLockedDepositByAddress` view functions return the total locked deposits for a given address by iterating over all locked deposits `lockedDeposits[_address]`. However, due to using unbound `for` loops, these functions could become quite expensive if the data-structure grows too large. This is fine in this case, since the functions are marked as `view` and intended for read-only queries. However, if the functions were to be called from another smart contract in the context of a write transaction, they could hit the block gas limit. This could render the transaction unexecutable.

Recommendation

Ensure the functions are never used within the context of a write transaction. Alternatively, consider adding `offset` and `limit` function parameters to

`BNFT.getTotalLockedDepositAmount` and `BNFT.getTotalLockedDepositByAddress` to implement a "paginated" `for` loop.

11. Miscellaneous Comments

The following are some recommendations to improve the overall code quality and readability.

- Spelling issues: **(Status: Resolved)**
  - `BNFT.sol#L54` - `offchain` - Suggestion: `off-chain`
  - `BNFT.sol#L137` - `transfered` - Suggestion: `transferred`
  - `BNFT.sol#L172` - `offchain` - Suggestion: `off-chain`
  - `BNFT.sol#L183` - `cummulative` - Suggestion: `cumulative`
  - `BNFT.sol#L183` - `offchain` - Suggestion: `off-chain`
  - `BNFT.sol#L185` - `idealy` - Suggestion: `ideally`
  - `BNFT.sol#L190` - `offchain` - Suggestion: `off-chain`
  - `BNFT.sol#L207` - `deponsits` - Suggestion: `deposits`
  - `BNFT.sol#L225` - `cummulative` - Suggestion: `cumulative`
  - `BNFT.sol#L225` - `offchain` - Suggestion: `off-chain`
  - `BNFT.sol#L287` - `enlught` - Suggestion: `enough`
  - `BNFT.sol#L294` - `btstamount` - Suggestion: `btrstAmount`
- `BNFT.sol#L33` - Wrong NatSpec comment. Comment should mention Braintrust ERC-20 token. **(Status: Resolved)**
- `BNFT.sol#L236` - Missing function parameter documentation for parameter `availableTimeInSeconds`. **(Status: Acknowledged)**
- `BNFT.sol#L218` - As the caller of `BNFT.deposit` is not necessarily the `beneficiary`, the emitted event `Deposited` should contain an additional value for the function caller. **(Status: Acknowledged)**
- As a naming convention, the `BraintrustMembershipNFT` contract should match its filename and vice versa. **(Status: Resolved)**
- Consider reformatting `if` & `revert` statements into `modifiers`, to increase code reusability. **(Status: Acknowledged)**

Audit Report for Braintrust - September 13, 2022

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Braintrust or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified from legal and financial liability.

Oak Security GmbH