SOLIDIFIED

Audit Report for Du Bois Gold - May 7, 2021

Summary

Audit Report prepared by Solidified covering the Du Bois Gold v2 smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code in several rounds. The debrief took place on 7 May 2021.

Update: Fixes received on 31 May 2021.

Audited Files

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

https://github.com/DuboisGold/dgm-contracts-v2

Commit number: `39162824772ca7d717a54570a2cb37acc8f95ab4`

The scope of the audit was limited to the following files:

```
contracts
├── Access.sol
├── Blacklist.sol
├── Convertor.sol
├── GoldbarLogic.sol
├── INftStorage.sol
├── NftStorage.sol
└── TokenControl.sol
```

Intended Behavior

The smart contracts implement version 2 of the Du Bois gold bar non-fungible tokenization protocol. The contracts are designed to be operated by a number of different privileged roles.

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 has its 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	Medium	Although no additional documentation has been provided the purpose of the code is clear from the inline commenting.
Test Coverage	N/A	No tests were submitted to the audit, so the test coverage could not be evaluated.

Issues Found

Solidified found that the Du Bois Gold contracts contain 2 critical issues, no major issues, 4 minor issues, in addition to 8 informational notes.

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	GoldbarLogic.sol: An attacker can use function buyGoldBar() to obtain a gold bar at a fraction of its set price	Critical	Resolved
2	GoldbarLogic.sol: Anyone can buy gold bar sold for ETH without paying AND/OR drain the whole contract	Critical	Resolved
3	Contracts can become unusable if the admin roles are renounced or accidentally set to an invalid address	Minor	Resolved
4	GoldBarLogic.sol: ETH transfer can potentially fail if receiver is a smart contract	Minor	Resolved
5	GoldbarLogic.sol: Function buyGoldBar() does not refund extra ETH sent	Minor	Resolved
6	GoldbarLogic.sol: Function updateTotalFeeAll() will always fail after a large enough number of gold bars have been minted	Minor	Resolved
7	TokenStake.sol: Pragma allows for a wide range of compiler versions	Note	Acknowledged
8	Converter.sol: stringToUint256() unused	Note	Acknowledged
9	Serial number uniqueness is not enforced at contract level	Note	Non-issue
10	GoldbarLogic.sol: Consider redeclaring function mintBatchAndRefund() as external to save on	Note	Resolved

	gas		
11	GoldbarLogic.sol: Consider emitting events across the contract’s different functions	Note	Acknowledged
12	GoldbarLogic.sol: Storage Fee is not reset and liability is transferred to the buyer	Note	Acknowledged
13	GoldbarLogic.sol: Dedicated deposit method can be removed	Note	Resolved
14	NFTStorage.sol: Mint, Burn and Settle includes duplicate checks	Note	Resolved

Critical Issues

1. GoldbarLogic.sol: An attacker can use function buyGoldBar() to obtain a gold bar at a fraction of its set price

Function `buyGoldBar()` does not verify that the passed `symbol_` is the same as the gold bar's unit price in `meta.symbol`. This means that a gold bar that is priced in `ETH` can be bought with an identical amount of `USDT`, for instance.

Recommendation
Require that `symbol_` is identical to `meta.symbol`, otherwise revert.

IMPORTANT
The same vulnerability is also present in `buyGoldBarBatch()`.

2. GoldbarLogic.sol: Anyone can buy gold bar sold for ETH without paying AND/OR drain the whole contract

The contract does not validate `msg.value` when a buyer buys a gold bar using ETH. If the contract balance is equal to or more than gold bar price then the contract transfers funds to the seller and transfers it to the buyer who did not pay anything.

Furthermore, a malicious seller can price the gold bar the same as the contract balance and buy without passing any value to drain the whole contract balance.

The minimum amount validator is susceptible to overflow attacks.

Recommendation
It is recommended to validate the `msg.value` when a user buys the gold bar using ETH. Include overflow checks too.

Major Issues

No major issues have been found.

Minor Issues

3. Contracts can become unusable if the admin roles are renounced or accidentally set to an invalid address

The contracts rely on a number of privileged roles to function correctly. By default, the OpenZeppellin libraries on which the access control logic is based allow renouncing roles, in order to migrate to trustless setups. However, such a scenario does not exist in this protocol, and renouncing roles would simply break the protocol completely. This issue is further complicated to fact that the constructor sets all roles to a single address.

Recommendation
Ensure that `renounceRole()` cannot be called and add zero checks to role assignations.

4. `GoldBarLogic.sol`: ETH transfer can potentially fail if receiver is a smart contract

Function `withdraw()`, `payFee()`, `_transferAssetand()` and `_refundAsset()` call `transfer()` when sending ETH to an address, which only forwards 2300 gas. In cases where this receiver address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts from receiving transfers.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` with `call()`.

5. `GoldbarLogic.sol`: Function `buyGoldBar()` does not refund extra ETH sent

Function `buyGoldBar()` does refund any extra ETH sent by the buyer.

Recommendation
Refund any ETH that is greater than `meta.price + buyFee`.

Note
The same issue also exists in functions `buyGoldBarBatch()` and `payFee()`.

6. `GoldbarLogic.sol`: Function `updateTotalFeeAll()` will always fail after a large enough number of gold bars have been minted

Since function `updateTotalFeeAll()` iterates over all `_serials`, its gas consumption will exceed the block gas limit after a large enough number of gold bars have been minted. This will cause the function to always fail when called.

Recommendation
Create a function that can update only a subset of the `_serials` array.

Informational Notes

7. `TokenStake.sol`: Pragma allows for a wide range of compiler versions

Function `pragma` statement allows for a very large range of compiler versions, including some versions with known bugs. In addition, the language syntax has changed since the earlier versions that are allowed.

Recommendation
Consider limiting the compiler to at least a single major version number.

Team Reply
“The referred openzeppelin libraries has same range of compiler versions”

8. `Converter.sol`: stringToUint256() unused

The `stringToUint256()` function does not seem to be used anywhere in the codebase.

Recommendation
Consider removing unused code.

Team Reply
“Frontend uses that function”

9. Serial number uniqueness is not enforced at contract level

It is possible to mint gold bars with the same serial number if the minter makes a mistake.

Recommendation
Consider adding controls to enforce uniqueness.

10. `GoldbarLogic.sol`: Consider redeclaring function `mintBatchAndRefund()` as external to save on gas

Since function `mintBatchAndRefund()` can potentially be passed an array of large size, redeclaring it as `external` instead of `public` can save a significant amount of gas. When declared as `external`, the `serials_` array will be read directly from calldata (as opposed to being copied to memory), which can potentially save a lot of gas.

Note
The same issue exists with functions `enrollGoldBarBatch()`, `buyGoldBarBatch()` and `setFeeLocaBatch()`.

11. `GoldbarLogic.sol`: Consider emitting events across the contract’s different functions

Consider emitting events in order to enable a potential Dapp to be able to monitor state-changing operations such as deposits, withdrawals, minting, burning, etc.

Team Reply
“Current frontend does not need other events”

12. `GoldbarLogic.sol`: Storage Fee is not reset and liability is transferred to the buyer

The storage fee will keep on accumulating and the gold bar can be settled for the fee. This affects the final asset holder and not the previous owners.

Recommendation
It is recommended to collect the fee before the gold bar is transferred to some other account.

Team Reply
“Storage fee can be considered as a debt. It is entirely about policy.”

13. `GoldbarLogic.sol`: Dedicated deposit method can be removed

The function `deposit()` does not track coins deposited or emit any event. This functionality can be achieved by directly transferring ETH or ERC20 tokens to the contract address.

Recommendation
Consider removing the `deposit()` method and use direct transfer.

14. NFTStorage.sol: Mint, Burn and Settle includes duplicate checks

The method Mint, Burn and Settle includes validations to check for blacklist and freeze list. This can be avoided since the validation is performed by the beforeTokenTransfer method.

Recommendation

Remove duplicate validations to save some gas.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Du Bois Gold 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 platform from legal and financial liability.

Solidified Technologies Inc.