SOLIDIFIED

Audit Report for DAU ERC20 Token - April 13, 2023

Summary

Audit Report prepared by Solidified covering the DAU ERC20 token.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on March 24, 2023, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/Lexim-gold/Lexim-ERC20-Token
Commit hash: `3f7807cba6f6f7facf50013c9b3432500c215169`

```
contracts
└── DauToken.sol
```

Fixes were received on April 13, 2023.
Updated commit number: `a8d5fc173cc11b2b6256778d96ceea7e3daa9f71`

Intended Behavior

The code base implements an ERC20 token that is issued by Lexim and backed by physical gold.

Findings

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 not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	Low	No tests were provided.

Issues Found

Solidified found that the Lexim ERC20 Token contracts contain no critical issues, no major issue, 3 minor issues, and 8 informational notes.

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

Issue #	Description	Severity	Status
1	High degree of centralization	Minor	Acknowledged
2	A removed gold bar cannot be added again	Minor	Resolved
3	Change of fee rate can be set unreasonably high post-minting	Minor	Acknowledged
4	Changeable token name and symbol may be problematic for integrations	Note	Acknowledged
5	Fee-on-transfer mechanism may be problematic for integrations	Note	Acknowledged
6	notPaying functions do not emit events	Note	Acknowledged
7	Use of deprecated OpenZeppelin library	Note	Acknowledged
8	Contract size exceeds 24 Kibibytes	Note	Resolved
9	_beforeTokenTransfer missing virtual keyword	Note	Resolved
10	No check to ensure total supply of ERC20 tokens is sufficiently backed by gold bars	Note	Acknowledged
11	Miscellaneous Comments	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

No critical issues have been found.

Minor Issues

1. High degree of centralization

The contract deployer withholds all the roles upon initialization, including the `DEFAULT_ADMIN_ROLE`. The degree of centralization creates a single point of failure that introduces significant risks for the contract and its users. Loss of admin private keys or accidental revocation or renunciation of the admin role could have severe consequences for the functionality of the contract, if no other accounts have been granted roles. Concentration of power enables the contract deployer to pause the contract indefinitely, freeze accounts, or set arbitrarily high fees, something which might damage trust in the community.

Recommendation
Consider distributing role responsibilities to different accounts after deployment. Also, consider using a strong multisig wallet, requiring for example 4 out of 5 signatures. Further, consider eliminating the pausing functionality or set a maximum amount of time that the contract could be paused for.

Status
Acknowledged. Team's response: "We recognize the problem and will make sure to correctly distribute the roles between several addresses. We also agree with the fact that any account with Admin privileges is extremely powerful and correctly protecting access to it is of utmost

importance. We do plan to use MPC and/or multisignature accounts to mitigate any risks involved".

2. A removed gold bar cannot be added again

When a gold bar is removed using the function `removeGoldBar`, the value is swapped with the value at the end of `_goldBars.values` and `_goldBars.isIn` of the value that was previously the last element is updated. However, `_goldBars.isIn` of the removed element is not set to 0. This is problematic because `addGoldBar` checks that `_goldBars.isIn` of the element that should be added is 0 to ensure that no duplicates are added. However, when the element was previously removed, this check will erroneously prevent adding it again.

Recommendation

Consider setting `_goldBars.isIn` to 0 when removing a gold bar.

Status

Resolved

3. Change of fee rate can be set unreasonably high post-minting

Anyone with a granted fee role can modify the feeRate using the `setFeeRate` function. The fee rate can be modified at any time and set arbitrarily high, up to a maximum of 100% of the transaction value. Such an increase in the fee rate might result in unexpected losses for users.

Recommendation

Consider enforcing a reasonable maximum value for the `feeRate` or setting a long-enough time lock before any change in the `feeRate` so that users can act accordingly.

Status

Acknowledged. Team's response: "We acknowledge this fact. However, we strongly believe that transparency and well-defined operations apparatus are important guarantees of the value of

our token together with the smart contract. Our approach is to guarantee the responsible approach to fee rate on the side of the operations team, while leaving the current smart contract implementation as is ”.

Informational Notes

4. Changeable token name and symbol may be problematic for integrations

The name of the token can be changed by the controller using `setNameAndSymbol`. This can be problematic for the integration with some DeFi protocols (or off-chain components) that query the name and symbol in the beginning (e.g., when the token is added to the protocol) and do not expect changes, as this is usually not supported for ERC20 tokens. When the name or symbol is changed, there will be a discrepancy between the value that the DeFi protocol uses and the current value for such protocols.

Recommendation
Consider removing the functionality to change the symbol and name.

Status
Acknowledged. Team’s response: “We acknowledge the problem and view this implementation as a supporting mechanism rather than an only solution to the possible problem of token renaming. We also view this possibility as extremely unlikely at this point. As such we did not change the implementation”.

5. Fee-on-transfer mechanism may be problematic for integrations

The token charges a configurable fee on every transfer. This can be problematic for many DeFi protocols (such as AMMs or lending protocols), as they often assume that the requested amount equals the transferred amount when the transfer succeeds.

Recommendation
Consider using an alternative fee scheme such as an issuance/minting fee. This would have the additional benefit that the user would not need to pay fees when switching wallets (e.g., because his private key is compromised) and transferring all assets.

Status
Acknowledged. Team's response: "We acknowledge the problem. This approach, however, is necessary for our vision of the DAU ecosystem. We will endeavor to solve problems with integrations on a case-by-case basis".

6. notPaying functions do not emit events

The functions `addNotPaying` and `removeNotPaying` do not emit events.

Recommendation
Consider emitting events with the address added to or removed from the list of non-paying addresses.

Status
Acknowledged. Team's response: "The list of notPaying accounts list a minor feature which has been added as a last resort measure if we run out of other ways to circumvent problems with integration. As such there is a high chance it will never get used and we decided to leave the code as is".

7. Use of deprecated OpenZeppelin library

The contract inherits `ERC20PermitUpgradeable` from OpenZeppelin's `draft-ERC20PermitUpgradeable.sol` library. The library has been deprecated since EIP-2612 has been finalized.

Recommendation
Consider using `@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol`.

Status
Acknowledged. Team's response: "At the time of the development the mentioned code has not been a part of the release yet. Since there are no affecting changes in the library we did not make changes at this stage".

8. Contract size exceeds 24 Kibibytes

The contract size is 28882 bytes, exceeding the limit of 24576 bytes introduced in the Spurious Dragon hard fork, following the finalization of EIP-170: Contract code size limit. With this size, the contract will not be deployable.

Recommendation
Consider enabling the optimizer with a low "runs" value, to allow for mainnet deployment.

Status
Resolved

9. `_beforeTokenTransfer` missing virtual keyword

The overridden `_beforeTokenTransfer` hook is missing the `virtual` keyword. OpenZeppelin rules for overriding hooks stipulates that the `virtual` keyword be added to allow child contracts to add further functionality to the hook (kindly refer to https://docs.openzeppelin.com/contracts/4.x/extending-contracts#rules_of_hooks).

Recommendation

Consider adding the `virtual` keyword to the hook.

Status
Resolved

10. No check to ensure total supply of ERC20 tokens is sufficiently backed by gold bars

The README stipulates that the total supply of Lexim-issued ERC20 is backed by physical gold bars under custody. It also stipulates that the list of gold bar ids, maintained by the smart contract, is used as backing for ERC20. However, there are no checks to ensure that the total supply of ERC20 is sufficiently backed by gold bars.

Recommendation
Consider adding a check in the `mint` function to ensure that `__goldBars.values.length * 1e18 >= totalSupply` after minting.

Status
Acknowledged. Team's response: "The backing is guaranteed by the operations team and our gold bar audits. As such the changes were NOT implemented".

11. Miscellaneous Comments

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

- `ERC20Upgradeable` is inherited through `ERC20BurnableUpgradeable`. Consider removing the relevant import and inheritance. **Status:** Acknowledged.
- `Initializable` is inherited through `UUPSUpgradeable`. Consider removing the relevant import and inheritance. **Status:** Acknowledged.
- Inaccurate comments in lines `129`, `136`, referring to the ERC-721 protocol. **Status:** Resolved.

- It is recommended to place the function visibility specifier before any modifiers. Consider changing the order of the modifiers in the `initialize` function. **Status:** Resolved.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Du Bois Gold AG 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.

Oak Security GmbH