Dubois Gold - DAU ERC20 Token

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **September 11th, 2023 - September 22nd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/21/2023
0.2	Document Updates	09/22/2023
0.3	Final Draft	09/22/2023
0.4	Draft Review	09/25/2023
0.5	Draft Review	09/25/2023
1.0	Remediation Plan	10/04/2023
1.1	Remediation Plan Review	10/04/2023
1.2	Remediation Plan Review	10/04/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

DAU ERC20 is a centralized ERC20 token that represents physical ownership of gold bars.

Dubois Gold engaged Halborn to conduct a security assessment on their smart contracts beginning on September 11th, 2023 and ending on September 22nd, 2023. The security assessment was scoped to the smart contracts provided in the DuboisGold/DAU-ERC20-Contract GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 12 days for the engagement and assigned a team of 1 full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were partially addressed by Dubois Gold. The main one was the following:
- Validating that the destination chains are whitelisted.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs (MythX).
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Project Name

- Repository: DuboisGold/DAU-ERC20-Contract
- Commit ID: cf285ee
- Remediations Commit ID: 8751846
- Smart contracts in scope:
    1. DAU.sol (/contracts/DAU.sol)

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	4	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING RECIPIENT CHAIN VALIDATION CAN LEAD TO LOSS OF TOKENS	Medium (6.3)	SOLVED - 09/28/2023
(HAL-02) MISSING STORAGE GAPS	Medium (5.2)	RISK ACCEPTED
(HAL-03) FEE ON TRANSFER ROUNDING DOWN ERROR	Low (4.2)	RISK ACCEPTED
(HAL-04) CENTRALIZATION RISK	Low (2.5)	RISK ACCEPTED
(HAL-05) ETHER CAN BE LOCKED IN THE CONTRACT	Low (2.1)	RISK ACCEPTED
(HAL-06) ACCESS CONTROL DOES NOT FOLLOW SECURITY BEST PRACTICES	Low (2.0)	RISK ACCEPTED
(HAL-07) FLOATING PRAGMA	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISSING RECIPIENT CHAIN VALIDATION CAN LEAD TO LOSS OF TOKENS - MEDIUM (6.3)

Description:

The CATERC20Upgradeable contract inherited by the DAU contract allows the DAU Token to be bridged across chains through Wormhole messages.

The bridgeOut() function burns the tokens in the source chain and sends the message, so it can be minted in the destination chain. However, the function does not whitelist the recipientChain parameter. Therefore, if a user mistakenly sends the tokens to a recipient chain where the DAU contract is not deployed, the tokens are burnt and lost forever.

Code Location:

Listing 1: src/ERC20/CATERC20Upgradeable.sol (Line 60)

```
60 /**
61  * @dev To bridge tokens to other chains.
62  */
63 function bridgeOut(
64     uint256 amount,
65     uint16 recipientChain,
66     bytes32 recipient,
67     uint32 nonce
68 ) external payable returns (uint64 sequence) {
69     require(isInitialized() == true, "Not Initialized");
70     require(evmChainId() == block.chainid, "unsupported fork");
71
72     uint256 fee = wormhole().messageFee();
73     require(msg.value >= fee, "Not enough fee provided to publish
↳ message");
74     uint16 tokenChain = wormhole().chainId();
75     bytes32 tokenAddress = bytes32(uint256(uint160(address(this)))
↳ );
76
77     _burn(_msgSender(), amount);
```

```
78
79      CATERC20Structs.CrossChainPayload memory transfer =
↳ CATERC20Structs
80          .CrossChainPayload({
81              amount: amount,
82              tokenAddress: tokenAddress,
83              tokenChain: tokenChain,
84              toAddress: recipient,
85              toChain: recipientChain,
86              tokenDecimals: getDecimals()
87          });
88
89      sequence = wormhole().publishMessage{value: msg.value}(
90          nonce,
91          encodeTransfer(transfer),
92          finality()
93      );
94
95      emit bridgeOutEvent(
96          amount,
97          tokenChain,
98          recipientChain,
99          addressToBytes(_msgSender()),
100         recipient
101     );
102 } // end of function
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:H/Y:N/R:N/S:C (6.3)

Recommendation:

Add whitelist-based verification of the requested recipient chains.

Remediation:

SOLVED: The Dubois Gold team solved the issue by whitelisting the destination chains in commit 8751846.

4.2 (HAL-02) MISSING STORAGE GAPS - MEDIUM (5.2)

Description:

The contract CATERC20Storage inherited by CATERC20Upgradeable does not contain a __gap variable, even though it is upgradeable.

This means that in the event of adding new state variables when upgrading the contracts to the CATERC20Storage.State struct in the _state variable can lead to storage slot collisions.

BVSS:

AO:A/AC:L/AX:H/C:N/I:C/A:C/D:N/Y:N/R:N/S:C (5.2)

Recommendation:

Consider adding a __gap variable in the CATERC20Storage contract.

Remediation:

RISK ACCEPTED: The Dubois Gold team accepted the risk of this finding.

4.3 (HAL-03) FEE ON TRANSFER ROUNDING DOWN ERROR - LOW (4.2)

Description:

The DAU ERC20 contract has a fee-on-transfer functionality where a fee is taken from the amount every time a transfer is performed.

The getFeeFor() function calculates the fee to be taken from the transfer based on a percentage. However, because Solidity cannot handle decimals, if the fee is below 1, it is possible to perform a transfer for free.

Therefore, it is possible for a user to split a transfer into smaller amounts, so a transfer could be free, however the likelihood of this being exploited is significantly lower because of the gas cost of sending multiple small-amount transfers instead of one for the full amount.

Code Location:

Listing 2: src/DAU.sol (Line 264)

```
264 function getFeeFor(uint256 value) public view returns (uint256) {
265     if (feeRate == 0) {
266         return 0;
267     }
268     return (value * feeRate) / feeParts;
269 }
```

BVSS:

AO:A/AC:M/AX:L/C:N/I:N/A:N/D:N/Y:M/R:N/S:C (4.2)

Recommendation:

Set a minimum fee of 1, so users cannot perform free transfers.

Remediation:

RISK ACCEPTED: The Dubois Gold team accepted the risk of this finding.

4.4 (HAL-04) CENTRALIZATION RISK - LOW (2.5)

Description:

The DAU contract is meant to be an ERC20 token that is centrally minted and burned, representing the physical ownership of gold bars.

This can pose a centralization risk in case the private keys for any of the privileged roles are compromised.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:C (2.5)

Recommendation:

Use multi-signatures to mitigate the centralization risk.

Remediation:

RISK ACCEPTED: The Dubois Gold team accepted the risk of this finding.

4.5 (HAL-05) ETHER CAN BE LOCKED IN THE CONTRACT - LOW (2.1)

Description:

The CATERC20Upgradeable contract inherited by the DAU contract allows the DAU Token to be bridged across chains through Wormhole messages.

Sending messages across the Wormhole bridge requires paying a fee in ETH. Therefore, the bridgeOut() function is payable, so users can send the ether to pay the fee.

Furthermore, there is the require statement that validates the amount of ether sent to pay the fee. However, it allows users to send more ether than the required for the fee, which can lead to the remaining amount becoming locked in the contract.

Code Location:

Listing 3: src/ERC20/CATERC20Upgradeable.sol (Line 72)

```
73 uint256 fee = wormhole().messageFee();
74 require(msg.value >= fee, "Not enough fee provided to publish
↳ message");
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:N/Y:M/R:P/S:C (2.1)

Recommendation:

Change the require statement to force a strict equality, so the user can only pay the exact fee amount.

Remediation:

RISK ACCEPTED: The Dubois Gold team accepted the risk of this finding.

4.6 (HAL-06) ACCESS CONTROL DOES NOT FOLLOW SECURITY BEST PRACTICES - LOW (2.0)

Description:

The DAU contract inherits from the OpenZeppelin's AccessControl library. However, this library does not follow some security best practices, for example, the DEFAULT_ADMIN_ROLE is also its own admin, meaning it has permissions to grant and revoke this role.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

Consider following security best practices and OpenZeppelin's recommendations, and use the AccessControlDefaultAdminRules extension to enforce additional security measures over this role.

Remediation:

RISK ACCEPTED: The Dubois Gold team accepted the risk of this finding.

4.7 (HAL-07) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. 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.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Set the pragma to a fixed version.

Remediation:

ACKNOWLEDGED: The Dubois Gold team acknowledged the finding.

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

Slither results for DAU	
Finding	**Impact**
CATERC20Getters.normalizeAmount(uint256,uint8,uint8) (src/ERC20/Getters.sol#63-75) performs a multiplication on the result of a division: - amount /= 10 ** (foreignDecimals - localDecimals) (src/ERC20/Getters.sol#69) - amount *= 10 ** (localDecimals - foreignDecimals) (src/ERC20/Getters.sol#72)	Medium
DAU.initialize(string,string,uint8,uint16,address,uint8).wormhole (src/DAU.sol#83) shadows: - CATERC20Getters.wormhole() (src/ERC20/Getters.sol#19-21) (function)	Low
CATERC20Governance.updateFinality(uint8).finality (src/ERC20/Governance.sol#36) shadows: - CATERC20Getters.finality() (src/ERC20/Getters.sol#35-37) (function)	Low

Finding	Impact
CATERC20Governance.registerChain(uint16,bytes32).chainId (src/ERC20/Governance.sol#18) shadows: - CATERC20Getters.chainId() (src/ERC20/Getters.sol#23-25) (function)	Low
DAU.registerChains(uint16[],bytes32[]).chainId (src/DAU.sol#303) shadows: - CATERC20Getters.chainId() (src/ERC20/Getters.sol#23-25) (function)	Low
DAU.updateFinality(uint8).finality (src/DAU.sol#310) shadows: - CATERC20Getters.finality() (src/ERC20/Getters.sol#35-37) (function)	Low
DAU.registerChain(uint16,bytes32).chainId (src/DAU.sol#296) shadows: - CATERC20Getters.chainId() (src/ERC20/Getters.sol#23-25) (function)	Low
DAU.initialize(string,string,uint8,uint16,address,uint8).finality (src/DAU.sol#84) shadows: - CATERC20Getters.finality() (src/ERC20/Getters.sol#35-37) (function)	Low
DAU.initialize(string,string,uint8,uint16,address,uint8).chainId (src/DAU.sol#82) shadows: - CATERC20Getters.chainId() (src/ERC20/Getters.sol#23-25) (function)	Low
CATERC20Governance.registerChains(uint16[],bytes32[]).chainId (src/ERC20/Governance.sol#25) shadows: - CATERC20Getters.chainId() (src/ERC20/Getters.sol#23-25) (function)	Low
Reentrancy in CATERC20Upgradeable.bridgeOut(uint256,uint16,bytes32,uint32) (src/ERC20/CATERC20Upgradeable.sol#63-102): External calls: - sequence = wormhole().publishMessage{value: msg.value}(nonce,encodeTransfer(transfer),finality()) (src/ERC20/CATERC20Upgradeable.sol#89-93) Event emitted after the call(s): - bridgeOutEvent(amount,tokenChain,recipientChain,addressToBytes(_msgSender()),recipient) (src/ERC20/CATERC20Upgradeable.sol#95-101)	Low
End of table for DAU	

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives, and the confirmed ones are included in the report.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

contracts/DAU.sol

```
Found 1 job(s). Submit? [y/N]: y
  [####################################]  100%
Report for src/DAU.sol
https://dashboard.mythx.io/#/console/analyses/1d2f4a1d-86be-41c2-8cda-4d2c5a10e88a
```

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

The findings obtained as a result of the MythX scan were examined, and they were not included in the report because they were determined false positives.

THANK YOU FOR CHOOSING

// HALBORN