MUNDUS
SECURITY

Security
Smart Contract Audit
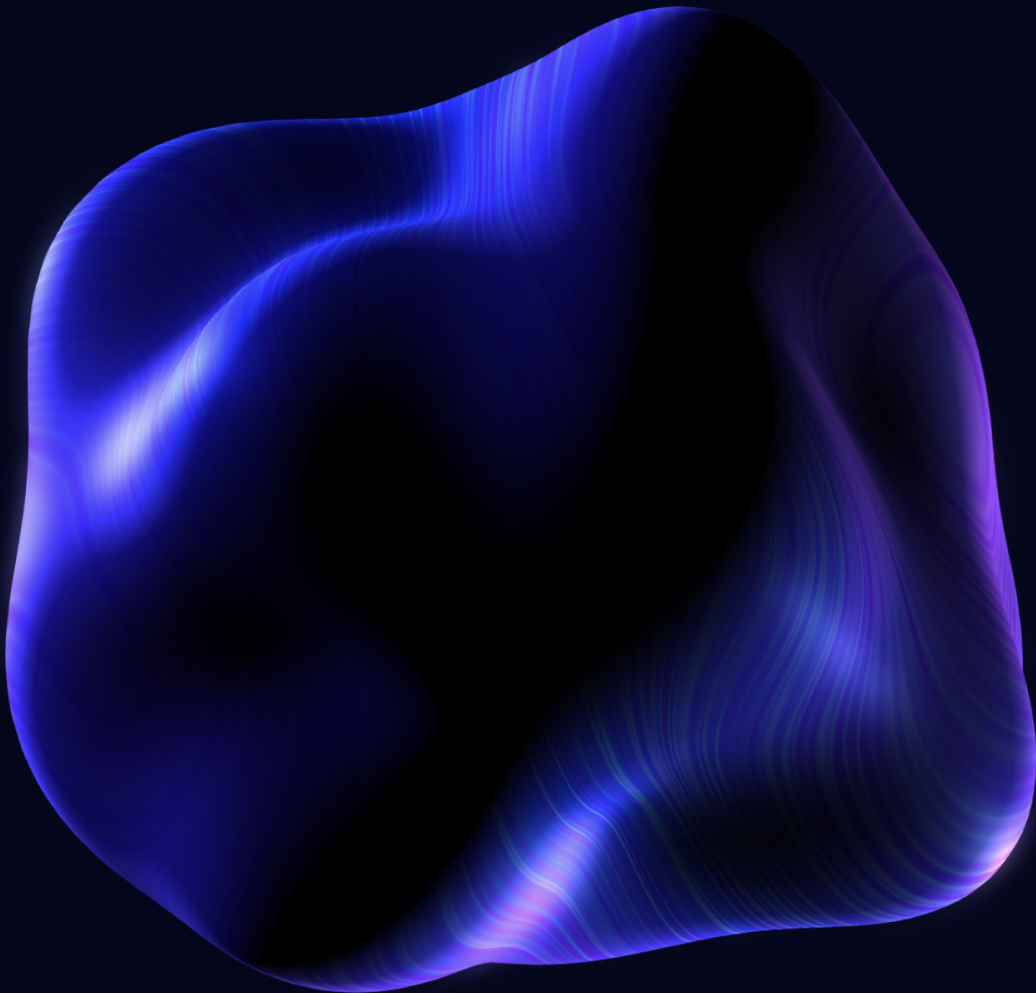
MUNDUS SECURITY

MAHADao ARTH security audit

Reference information

Name	MahaDAO ARTH
Website	https://mahadao.com/
Language	Solidity
Chain	Ethereum mainnet
Reference repositories	https://github.com/MahaDAO/arth-core https://github.com/MahaDAO/arth-strategies https://github.com/MahaDAO/gmu-oracle-contracts https://github.com/MahaDAO/chainlink-keepers https://github.com/MahaDAO/token https://github.com/MahaDAO/flashloans-arth

Scope of work: ARTH Core

Contract	Address	Repository
ActivePool	0xa443129308556ab06e69a98e1c39c81080e01530	arth-core
BorrowerOperations	0x4c50063f8238dea92c738f23221733a9a6c6888b	arth-core
CollSurplusPool	0xbb719b2d7207e8b8b13ca4dc9c8b6201d79cf7e5	arth-core
CommunityIssuance	0x61274cd1f801b097be7e5197b158999307893d2e	arth-core
DefaultPool	0x47f747fd93eef25cc1e0b6d7a239289c7cfec212	arth-core
Governance	0x91eb23b66beb3467998402ba50aa1c1a98811eb1	arth-core
SortedTroves	0xd60d7a2a8344d4f635bf9ea9f8cd015a614c3659	arth-core
StabilityPool	0x910f16455e5eb4605fe639e2846579c228eed3b5	arth-core
TroveManager	0x8b1da95724b1e376ae49fdb67afe33fe41093af5	arth-core
ARTHValuecoin	0x8cc0f052fff7ead7f2edcccac895502e884a8a71	token

Scope of work: ARTH Periphery

Contract	Address	Repository
ETHTroveStrategy *Proxy*	0xf3f261f54d8397806132598dc2b6b5c00d6eb3ea 0xa9735e594624339f8fbc8a99c57c13c7b4e8bcac	arth-strategies
USDCCurveStrategy *Proxy*	0x9ff6629d08fddaec63b0d855b9c29acdf4dc14e4 0x5480e8beedb3eba5747a4a3aef0850a3759df9b4	arth-strategies
StabilityPoolKeeper	0x5e98d3f8b5074b6389477fd88856f5209748caa7	chainlink-keepers
ARTHFlashMinter	0xc4bbefdc3066b919cd1a6b5901241e11282e625d	flashloans-arth
ETHGMUOracle	0xc31adc9ae073a1f6a9ce5c41b32c18790ea667fe	gmu-oracle-contracts
GMUOracle	0x066a917fa2e1739ccfc306dc73ff78eeca8b6f29	gmu-oracle-contracts

MUNDUS
SECURITY

Table of contents

Findings summary

Finding Severity breakdown

All vulnerabilities discovered during the source code audit are classified based on their potential severity and have the following classification:

Severity	Description
High	Bugs that can trigger a contract failure or theft of assets. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Low	Bugs that do not pose significant danger to the project or its users but are recommended to be fixed nonetheless.
Informational	All other non-essential recommendations.
Gas	Gas optimization recommendations.

Findings statistics

Severity	Number
High	2
Medium	3
Low	2
Informational	16
Gas	3
Total	26

ARTH Core findings

ID	Severity	Description
01	**Medium**	Wrong depositorETHGain receiver when invoking provideToSPFor method in StabilityPool.sol
02	Low	toggleBorrowerOperations contains dangerous centralized logic in ARTHValuecoin.sol
03	Informational	Dead code: _requireValidRecipient method in ARTHValueCoin.sol
04	Informational	openTroveFor method contains centralized logic in BorrowerOperations.sol
05	Informational	Dead code: _getUSDValue and _requireCallerIsBorrower methods in BorrowerOperations.sol
06	Informational	sendFeeToEcosystemFund should emit an event in BorrowerOperations.sol
07	Informational	receive should emit an event in CollSurplusPool.sol
08	Informational	Misleading comments in Governance.sol
09	Informational	BORROWING_FEE_FLOOR = MAX_BORROWING_FEE = 0 in Governance.sol
10	Informational	ARTH_GAS_COMPENSATION and MIN_NET_DEBT should be constant in Governance.sol
11	Informational	_getCollGasCompensation should be pure in LiquityBase.sol
12	Gas	MAX_INT and PERCENT_DIVISOR should be constant in LiquityBase.sol
13	Gas	IGovernance governance should be immutable in CommunityIssuance.sol

ARTH Periphery findings

ID	Severity	Description
14	**High**	Insufficient access control for notifyRewardAmount in StakingRewardsChild.sol -> USDCCurveStrategy.sol
15	**High**	Wrong fee mechanism when invoking flashLoan in ARTHFlashMinter.sol
16	**Medium**	minDepositForPermit is never initialized in USDCCurveStrategy.sol
17	**Medium**	Wrong fee value set in ARTHFlashMinter.sol
18	Low	Sanity check required in constructor in GMUOracle.sol
19	Informational	increase, deposit and withdraw should emit an event for totalmArthSupplied in ETHTroveStrategy.sol
20	Informational	No debt rebalancing logic in case of trove liquidation in ETHTroveStrategy.sol
21	Informational	closeTrove method contains dangerous centralized logic in ETHTroveStrategy.sol
22	Informational	_deposit and _withdraw should emit events for totalArthBorrowed and totalUsdcSupplied change in USDCCurveStrategy.sol
23	Informational	getCurrentEpoch duplicates logic of _getCurrentEpoch in Epoch.sol -> StabilityPoolKeeper.sol
24	Informational	updateMahaReward should emit an event in StabilityPoolKeeper.sol
25	Informational	Excessive if-clause inside _scalePriceByDigits in ETHGMUOracle.sol
26	Gas	maha and arthCommunityIssuance should be immutable in StabilityPoolKeeper.sol

Source code audit

For source code audit purposes we splitted SoW into two sets of contracts. The first set (we will call it ARTH Core) consists of these contracts:

- ActivePool
- ARTHValuecoin
- BorrowerOperations
- CollSurplusPool
- CommunityIssuance
- DefaultPool
- Governance
- SortedTroves
- StabilityPool
- TroveManager

These contracts are part of arth-core repository (with exception of ARTHValuecoin. It belongs to separate token repository). And all of these contracts are derived from Liquity project. For these contracts SoW was settled as audit of changes made after last audit of Liquity project in 2021.

The second set (we will call it ARTH Periphery) consists of these contracts:

- ARTHFlashMinter
- ARTHUSDCCurveStrategy with proxy
- ETHGMUOracle
- ETHTroveStrategy with proxy
- GMUOracle
- StabilityPoolKeeper

For these contracts we performed source code audit as usual.

Source code audit: ARTH Core

ID-01. **Medium**: Wrong depositorETHGain receiver when invoking provideToSPFor method in StabilityPool.sol

Description

The provideToSPFor method of StabilityPool.sol is a restricted function that allows contract admin to execute the provideToSP method on behalf of another account. In that function the depositorETHGain is sent to depositor via _sendETHGainToDepositor method. However, the _sendETHGainToDepositor function deals only with msg.sender which means that in the case of using provideToSPFor the depositorETHGain is sent to msg.sender, rather than _who.

Recommendation

Modify the _sendETHGainToDepositor(uint256 amount) function as well as all its use cases to contain the recipient address, i.e. _sendETHGainToDepositor(uint256 amount, _who).

ID-02. Low: toggleBorrowerOperations contains dangerous centralized logic in ARTHValuecoin.sol

Description

The toggleBorrowerOperations method of ARTHValuecoin.sol is an access controlled function that can grant access to mint and burn functions at the will of the contract admin.

ID-03. Informational: Dead code: _requireValidRecipient method in ARTHValueCoin.sol

Description

The _requireValidRecipient method of ARTHValueCoin.sol is never used and should be removed.

ID-04. Informational: openTroveFor method contains centralized logic in BorrowerOperations.sol

Description

The openTroveFor method of BorrowerOperations.sol is an access controlled function that contains logic able to open troves on behalf of any address.

ID-05. Informational: Dead code: _getUSDValue and _requireCallerIsBorrower methods in BorrowerOperations.sol

Description

The _getUSDValue and _requireCallerIsBorrower methods of BorrowerOperations.sol are never used and should be removed.

ID-06. Informational: sendFeeToEcosystemFund should emit an event in BorrowerOperations.sol

Description

The _sendFeeToEcosystemFund method of BorrowerOperations.sol should emit an event when invoked.

ID-07. Informational: `receive` should emit an event in `CollSurplusPool.sol`

Description

The `receive` method of `CollSurplusPool.sol` shoukd emit an event when invoked.

ID-08. Informational: Misleading comments in `Governance.sol`

Description

The source code of `Governance.sol` contains misleading comments. The calculated amounts of `BORROWING_FEE_FLOOR` and `MAX_BORROWING_FEE` in comments do not math the actual values.

```
uint256 private BORROWING_FEE_FLOOR = (DECIMAL_PRECISION / 1000) * 0; //
0.5%
uint256 private REDEMPTION_FEE_FLOOR = (DECIMAL_PRECISION / 1000) * 5; //
0.5%
uint256 private MAX_BORROWING_FEE = (DECIMAL_PRECISION / 100) * 0; // 5%
```

ID-09. Informational: `BORROWING_FEE_FLOOR = MAX_BORROWING_FEE = 0` in `Governance.sol`

Description

The `BORROWING_FEE_FLOOR` and `MAX_BORROWING_FEE` storage variables of `Governance.sol` are both equal to `0`, which contradicts the Liquity's protocol setup.

ID-10. Informational: `ARTH_GAS_COMPENSATION` and `MIN_NET_DEBT` should be constant in `Governance.sol`

Description

The `ARTH_GAS_COMPENSATION` and `MIN_NET_DEBT` variables of `Governance.sol` should be constant.

ID-11. Informational: _getCollGasCompensation should be pure in LiquityBase.sol

Description

The `_getCollGasCompensation` method of `LiquityBase.sol` should be `pure` instead of `view`, provided that `Issue 11` is resolved.

ID-12. Gas: MAX_INT and PERCENT_DIVISOR should be constant in LiquityBase.sol

Description

The `MAX_INT` and `PERCENT_DIVISOR` variables of `LiquityBase.sol` should be constant.

ID-13. Gas: IGovernance governance should be immutable in CommunityIssuance.sol

Description

The `IGovernance governance` variable of `CommunityIssuance.sol` should be immutable.

Source code audit: ARTH Periphery

ID-14. High: Insufficient access control for notifyRewardAmount in StakingRewardsChild.sol -> USDCCurveStrategy.sol

Description

The notifyRewardAmount method of StakingRewardsChild.sol in the **USDCCurveStrategy** contract lacks access control. Thus, any account is able to modify the rewardRate, lastUpdateTime and periodFinish state variables of the **USDCCurveStrategy** contract.

Recommendation

Add onlyOwner modifier to the notifyRewardAmount method of StakingRewardsChild.sol.

ID-15. High: Wrong fee mechanism when invoking flashLoan in ARTHFlashMinter.sol

Description

The flashLoan method of ARTHFlashMinter.sol is implemented with the wrong fee mechanism. According to the code below, after a successfull callback, the amount of receiver's tokens is burned and the _fee is transferred to the ecosystemFund at the expense of the **ARTHFlashMinter** contract (see @audit).

```
require(
  receiver.onFlashLoan(msg.sender, amount, _fee, data) ==
    CALLBACK_SUCCESS,
  "ARTHFlashMinter: Callback failed"
);

token.burn(address(receiver), amount);
token.transfer(ecosystemFund, _fee); // @audit
```

Recommendation

Modify the `flashLoan` method of `ARTHFlashMinter.sol` in either of the two following ways

```
// OPTION 1
token.burn(address(receiver), amount);
token.transferFrom(address(receiver), ecosystemFund, fee);

// OR

// OPTION 2
token.burn(address(receiver), amount + _fee);
token.mint(ecosystemFund, _fee);
```

ID-16. **Medium**: minDepositForPermit is never initialized in USDCCurveStrategy.sol

Description

The `minDepositForPermit` state variable of `USDCCurveStrategy.sol` is never initialized.

NOTE: At the time of initial discovery of this issue the **USDCCurveStrategy** implementation address was `0x122f4530c2c8ed9a7dc4846a155579ede0e23ecb`. Since then the MAHADao team has resolved this issue by deploying a new **USDCCurveStrategy** implementation (`0x9ff6629d08fddaec63b0d855b9c29acdf4dc14e4`) with proper `minDepositForPermit` initialization.

ID-17. **Medium**: Wrong fee value set in ARTHFlashMinter.sol

Description

The flashloan fee of the **ARTHFlashMinter** contract is 100 larger than the value stated in the comments to the source code (see @audit).

```
uint256 public fee = 1000; //  1000 == 0.1 %. @audit

...

function _flashFee(uint256 amount) internal view returns (uint256) {
  return (amount * fee) / 10000; @audit
}
```

ID-18. Low: Sanity check required in constructor in GMUOracle.sol

Description

The constructor of GMUOracle.sol relies heavily on _priceHistory30d == 30, but lacks any checks of this assumption.

Recommendation

Modify the constructor of GMUOracle.sol in either of the two following ways (see @audit)

```
// OPTION 1
constructor(
  uint256 _startingPrice18,
  address _oracle,
  uint256[30] memory _priceHistory30d // @audit
) Epoch(86400, block.timestamp, 0)

// OR

// OPTION 2
constructor(
  uint256 _startingPrice18,
  address _oracle,
  uint256[] memory _priceHistory30d
) Epoch(86400, block.timestamp, 0) {
  require(_priceHistory30d.length == 30);
  ...
}
```

ID-19. Informational: `increase`, `deposit` and `withdraw` should emit an event for `totalmArthSupplied` in `ETHTroveStrategy.sol`

Description

The `increase`, `deposit` and `withdraw` methods of `ETHTroveStrategy.sol` should emit events for `totalmArthSupplied` change.

ID-20. Informational: No debt rebalancing logic in case of trove liquidation in `ETHTroveStrategy.sol`

Description

The `ETHTroveStrategy.sol` contains no logic to address the undesirable case of liqudation of the trove attached to the **ETHTroveStrategy** contract.

ID-21. Informational: `closeTrove` method contains dangerous centralized logic in `ETHTroveStrategy.sol`

Description

The `closeTrove` method of `ETHTroveStrategy.sol` is an access controlled function that contains logic able to halt the availability of the entire contract.

ID-22. Informational: `_deposit` and `_withdraw` should emit events for `totalArthBorrowed` and `totalUsdcSupplied` change in `USDCCurveStrategy.sol`

Description

The `_deposit` and `_withdraw` methods of `USDCCurveStrategy.sol` should emit events for `totalArthBorrowed` and `totalUsdcSupplied` change.

ID-23. Informational: getCurrentEpoch duplicates logic of _getCurrentEpoch in Epoch.sol -> StabilityPoolKeeper.sol

Description

The getCurrentEpoch() external method of Epoch.sol in the **StabilityPoolKeeper** contract duplicates the logic of _getCurrentEpoch() internal.

ID-24. Informational: updateMahaReward should emit an event in StabilityPoolKeeper.sol

Description

The updateMahaReward method of StabilityPoolKeeper.sol should emit an event any time the mahaRate is updated.

ID-25. Informational: Excessive if-clause inside _scalePriceByDigits in ETHGMUOracle.sol

Description

The _scalePriceByDigits method of ETHGMUOracle.sol contains excessive condition checks (see @audit)

```
if (_answerDigits >= TARGET_DIGITS) {
    ...
}
else if (_answerDigits < TARGET_DIGITS) { // @audit
    ...
}
```

ID-26. Gas: `maha` and `arthCommunityIssuance` should be immutable in `StabilityPoolKeeper.sol`

Description

The `maha` and `arthCommunityIssuance` of `StabilityPoolKeeper.sol` shoul be immutable.

Disclaimers

Mundus disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical disclaimers

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.