FIAT

Smart Contract Audit

coinspect

FIAT Protocol

Smart Contract Audit

V220406 Prepared for FIATDAO • January 2022

1. Executive Summary

In **January 2022, FIATDAO** engaged Coinspect to perform a source code review of **FIAT Protocol**. The objective of the project was to evaluate the security of the smart contracts.

Coinspect found the smart contracts to be properly designed, implemented and documented, besides respecting security best practices. The FIATDAO team is aware of the common programming flaws that affect smart contracts security and showed a clear understanding of the threat model.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
1	**2**	**0**
Fixed 1	Fixed 0	Fixed 0

The only high risk vulnerability reported (FIA-1 Oracle returns incorrect asset price) was introduced in a late modification to the protocol's oracles. The other findings in this report are relevant only to edge cases.

2. Assessment and Scope

The audit started on **January 3, 2022** and was conducted on the `main` branch of the git repository at fiatdao/fiat-lux, as of commit **f49a9457fbcbdac1969c35b4714722f00caa462c**.

FIAT Protocol allows users to deposit fixed income assets as collateral in exchange for the FIAT stable-value token. By means of collateral, debt and surplus auctions the protocol aims to stabilize the FIAT token price around the 1 USD target.

The scope of the audit was limited to the following Solidity source files, with their `sha256sum` hash:

```
c6b644aac61c327629f14b8e41f7d413260ccabc3b248f3c344d80a945a4be14  Codex.sol
fe0c7c42854057d59c34a747e17bc68dbcb66d4b5235dbf4c04da543ab0b312a  Collybus.sol
7bf686ca6d94690963e912527dfa8ec3aa8f132c0b449a8a55abcd353b68f191  Aer.sol
b7ac206322e66b801629f523ea2776c16edc70ce462a829488f481ee0c626147  FIAT.sol
237128e5588cc5aea56d4136c9c2da2713ddc1ee6c6c0f6ad6532d93712b83c8  Limes.sol
9770193fea05c1dd52f6ef589797fca685de89b1ad3b2d515e474d2836e9e06d  Moneta.sol
ceaf741de2f51f1ea5bc02b23922373249c33cc21d2ba5fc7152644567ade331  Publican.sol
93c671513e1b57d00dcf626b0ece1fb1a31273a95816a6310d8913bebadf882c  Tenebrae.sol
f767c977ffe2da2cd7f334cd2d2d6d1fbbf582e3f8db061c475fb2e291fbed42  auctions/CollateralAuction.sol
41ea7ef638b585b0fcb74d6b3d5baa02ee37c4b75f3072b2324d773fe770e534  auctions/DebtAuction.sol
7826fc9a8f5c1caf73001efe28e2694bcb81cbcb8fc498ef7f5dd848ba7377f7  auctions/NoLossCollateralAuction.sol
12766b6416a15545a85ad912ab791d4226cdc9d768c44a9fa5b37682be6de19a  auctions/PriceCalculator.sol
e01db7da7d2764fca54bf3582a7b9f93f1f20d5aecb58441cec0dd603dd30d44  auctions/SurplusAuction.sol
6f2c47ef1fc2301aca064e720c9fc9647f112d57efe1182a5063579fe2bc51be  utils/Guarded.sol
16e433013e7f22fc55060fd59d15f49b7707189b9d931cb9a3e4a3a68d610183  utils/Math.sol
59e4e424bb2fd1b979651842ab8904a539ac3cd8c964bc3cb706042de2fca33b  vaults/Vault.sol
```

As per client's request, Coinspect reviewed the following additional changes that were performed while the audit was being conducted:

- https://github.com/fiatdao/fiat-lux/pull/40
- https://github.com/fiatdao/fiat-lux/pull/31
- https://github.com/fiatdao/fiat-lux/pull/41
- https://github.com/fiatdao/fiat-lux/pull/42

The system oracles and the off-chain keeper agents were not in scope for this audit.

In **March 2022**, **FIATDAO** split the fiat-lux repository in three, renaming `fiat-lux` to `fiat`. Coinspect updated the report, after reviewing the changes performed to the repository at fiatdao/fiat as of commit **358490a6c48580dda63fe5728e92cbfc12834846**.

The updated Solidity source files, with their `sha256sum` hash:

```
a83ac818f22e35089e9047971d791002725273700a6ee24c1eb6a676362ceca4  Codex.sol
a645bdd60255cd3d8974111afec9e64972e3fe64ff68476fcd0d57d3edb5b9e1  Collybus.sol
1e12cb12a9282f342a47e8870d1160cac268d31799cd983c64899658aef5c3bf  Aer.sol
963f4ec64d41741e80293f55bcc9ee7b4e76e4d1b862284c8390d76c87f6d4bc  FIAT.sol
```

```
c0f482b15c0e3c98318b5b5f1193e4ff0d496dd04cbf6068ba83d2da393ffc95  Limes.sol
dc8c55e0ca75cec93160349766461d46c8f18fe3f864b29ba80cd1b267c7816a  Moneta.sol
e913d96b012afdd121ca4d14fa0c6ec11431cf700dcb6ce5800f006631dce580  Publican.sol
aada589afe806efc831d7cc410147bef1991b88b35f9c4bc060a0c2253f7b0fa  Tenebrae.sol
2032527dbbd26a8ea7219f2a655f6796e7853379979ee768b02a9fdcbb0860d2  auctions/CollateralAuction.sol
7f00119b460da58c15cab650da5c5d7016c602365d86cf87aef2fda8ba2d03c0  auctions/DebtAuction.sol
c9d198a845020ca02e8d438f7485a72139fbe22db5d35e3acdafb80d518cf820  auctions/NoLossCollateralAuction.sol
4bd80cb7e935ed4766d9cfb0bf36879578660c40e5740b78ffd12a011eb8bf6c  auctions/PriceCalculator.sol
fcb8b4ae4c433b7e33352c4f4466453b31396703776664cd5c2b626ede9e03b4  auctions/SurplusAuction.sol
e2b1fb4cb353534437061e34db6c384f6b461aed04c0f119f635c17e4a736118  utils/Guarded.sol
5e5fc4384bd62e7b87f817c7cd52435fbc482779113fd98c3ab5f66ca12c2833  utils/Math.sol
```

Besides refactoring, these updates included minor changes and two considerable ones:

1. The `DebtAuction` contract no longer has the ability to mint the tokens required to pay the auction winner, but now, instead, it depends on those tokens to be made available to the auction contract by the DAO by an off-chain triggered action.
2. The `transferCredit` function was added to the `Aer` contract. This new function is protected by the `checkCaller` modifier (responsible for all access checks in the project). It allows the caller to transfer the available credit out from the `Aer` contract. The FIATDAO team clarified it is intended to provide more flexibility with the credit accrued in `Aer` and to also be used during the contract upgrade.

Observations and recommendations

FIAT Protocol utilizes a method signature based authentication scheme. All relationships between contracts can be dynamically modified and contracts can be replaced on the fly. The protocol security is highly dependent on how the specific deployment is done.

The protocol has many critical configurable parameters that must be changed by external agents to adapt to context changes. The protocol's stability and security depend on the correct parameters being used at all times. As context changes, failure to adapt those parameters in a timely-fashion could result in different threats scenarios becoming possible.

These configuration parameters and the dynamic authentication mechanism have the power to completely affect the protocol's behavior, stored funds, and its security. They are intended to be handled by the protocol governance mechanisms, which were out of scope.

Coinspect observed some contracts in the platform implicitly trust other components. This is a design decision and does not represent a vulnerability by itself, but extreme care should be taken when replacing contracts or adding new ones that could break previous assumptions. For example: because the Codex

accounting contract trusts vaults, introducing a new malicious or badly implemented Vault would allow stealing from the system.

The platform depends on external keepers/actors to trigger necessary actions in the system. It is suggested these keeper's implementation is reviewed to guarantee it is able to properly monitor blockchain events and act promptly when required. During the engagement Coinspect auditors and the FIATDAO team discussed potential issues that could arise because of network congestion and mempool manipulation. The MakerDAO's 2020 "zero-bid" incident was used as a concrete example of how delayed transactions could affect the system as described in: Evidence of Mempool Manipulation on Black Thursday: Hammerbots, Mempool Compression, and Spontaneous Stuck Transactions. Coinspect recommends keeping the following in mind when developing, testing and maintaining the off-chain keepers infrastructure: transaction nonce awareness, gas price oracle heuristics, mempool observation, ability to refund liquidations bots during a low liquidity event, and ability to handle chain reorganization scenarios. It is also worth noting that only surplus auctions start with a zero bid in the FIAT Protocol.

Throughout the audit, the extensive MakerDAO documentation was consulted. The FIAT Protocol threat model is similar to MakerDAO's and most of the security relevant comments and edge cases described in MakerDAO's documentation apply. One specific example where the documentation was consulted was the global settlement mechanism and how it favors vault owners versus FIAT holders. Given how the shutdown procedure works, it is possible to create big positions backed with a collateral asset whose price is going down and redeem the maximum amount of credit right before shutdown starts. This would harm all holders and properly collateralized positions when the lost collateral from the undercollateralized positions is distributed and collateral from all vaults is redeemed. Coinspect observed this in the implementation but after understanding the design rationale further research was abandoned, because it is a known and accepted compromise that can only be exploited in an unlikely edge case.

Lastly, Coinspect recommends reviewing the platform tests to guarantee they prevent small mistakes from slipping into the code base, making sure the values used do not result in false possibilities and that no tests are commented out.

Minor issues and code suggestions

1. `FIAT.sol`: incorrect error message, should be `FIAT__burn_insufficientAllowance()` instead of `FIAT__transferFrom_insufficientAllowance()` in the `burn` function.

2. `Limes.sol`: incorrect documentation, `/// @dev The third argument is the address that will receive the liquidation reward, if any.`
3. `Tenebrae.sol`: unnecessary addition in line 311

```
if (normalDebt != 0) revert Tenebrae__closePosition_normalDebtNotZero();
normalDebtByTokenId[vault][tokenId] = add(normalDebtByTokenId[vault][tokenId],
        normalDebt);
```

4. `Tenebrae.sol`: `redemptionPrice()` could be pre-calculated after debt is settled to save user gas during user redemptions.

3. Summary of Findings

Id	Title	Total Risk	Fixed
FIA-1	Oracle returns incorrect asset price	High	✔
FIA-2	Attackers can block time-sensitive functions to harm users	Medium	✘
FIA-3	Vault allows minting more FIAT than collateral deposited	Medium	✘
FIA-4	FIAT token non-standard ERC20 implementation	Info	✔
FIA-5	Known ERC-20 approve() front-running	Info	✔

4. Detailed Findings

FIA-1 Oracle returns incorrect asset price

Total Risk	Impact	Location
High	High	`Collybus.sol`
Fixed	**Likelihood**	
✔	High	

Description

The price oracle contract the FIAT Protocol relies on provides incorrect prices which ignore the discount rate to be applied to each asset.

This issue was introduced in the `fix/collybus-read` branch which was reviewed during the last week of the audit, and was not present in the initially reviewed code.

The contract code was modified to support `rateIds`, however, the `read` function still indexes the `rates` mapping by the provided `tokenId` (instead of using the `rateId`), which results in no discount rate being applied when pricing assets.

```
/// @notice Returns the internal price for an asset
/// @dev
///                 redemptionPrice
/// v = ---------------------------------------
///                      (maturity - timestamp)
///     (1 + discountRate)
///
/// @param vault Address of the asset corresponding Vault
/// @param tokenId ERC1155 or ERC721 style TokenId (leave at 0 for ERC20)
/// @param maturity Maturity of the asset [unix timestamp in seconds]
/// @param net Boolean (true - with liquidation safety margin, false - without)
/// @return price Internal price [wad]
function read(
    address vault,
    address underlier,
    uint256 tokenId,
    uint256 maturity,
    bool net
) external view override returns (uint256 price) {
    VaultConfig memory vaultConfig = vaults[vault];
    // fetch applicable fixed interest rate oracle system rateId
```

```
uint rateId = rateIds[vault][tokenId];
if(rateId == uint256(0)) rateId = vaultConfig.defaultRateId;
// doesn't exists, use default rateId
// fetch discount rate
uint256 discountRate = rates[tokenId];
// apply discount rate if discountRate > 0
```

Coinspect observed the corresponding test is commented out. There is a related test that is successful, but only because the `tokenId` and `rateId` used are both 0.

Recommendation

Replace the incorrect line with:

```
uint256 discountRate = rates[tokenId];
```

Revisit tests to ensure the values used do not result in false positives.

Status

Coinspect verified this issue was correctly fixed and the latest version of the `Collybus` contract in the following commits:

- https://github.com/fiatdao/fiat-lux/commit/539ed57df01b841dea1be800f3ef8cef5eaba72d
- https://github.com/fiatdao/fiat-lux/pull/31/commits/556779bd73fd6889af33ae2da3e2647bd82017a6

FIA-2	Attackers can block time-sensitive functions to harm users	

Total Risk
Medium
	Impact
	High
		Location
		`Aer.sol`
		`Tenebrae.sol`

Fixed ✘	Likelihood Low	

Description

Coinspect found two scenarios where externally triggered actions could be temporarily prevented by attackers, impacting the overall platform stability as a consequence.

The ability to block (or delay) critical functions can be abused to affect the platform's ability to react, for example during a quick collateral price fluctuation, and impact the overall platform stability.

Even though these situations can be easily addressed, if the off-chain code and actors responsible for triggering these actions are not aware of these potential abuse mechanisms, it could take them more time than the necessary one to react, and this would further aggravate the situation and harm the protocol users.

The first scenario affects the `startDebtAuction` function in the `Aer` contract. This function reverts if the `Aer` contract has credit available:

```
if (codex.credit(address(this)) != 0) revert Aer__startDebtAuction_surplusNotZero();
```

Attackers only need to transfer 1 credit to the `Aer` contract in order to stop debt auctions from being started. As a consequence, the outstanding debt in the system will not be auctioned and this could impact the system's ability to stabilize the FIAT price. In certain unfavorable circumstances (e.g., massive liquidations following quick collateral price drop) this issue, if unknown and not properly handled, could be exploited to worsen the situation.

The second scenario is similar to the first one and affects the `fixGlobalDebt` function in the `Tenebrae` contract. The `fixGlobalDebt` function can be blocked by anybody by transfering credit to the `Aer` contract after all positions have been settled:

```
if (codex.credit(address(aer)) != 0) revert Tenebrae__fixGlobalDebt_surplusNotZero();
```

As a result, FIAT holders will be prevented from redeeming their collateral until this situation is remediated. This would harm FIAT holders if their collateral keeps depreciating quickly while the issue is understood and the required actions taken.

Recommendation

These scenarios can be avoided by either creating a contract that calls the required functions atomically (for example `settleDebtWithSurplus` and `fixGlobalDebt`) or by adding a function to each contract that calls both functions.

FIA-3	Vault allows minting more FIAT than collateral deposited

Total Risk	Impact	Location
Medium	High	`Vault.sol`

Fixed	Likelihood
✘	Low

Description

The Vault contracts could be abused to mint more FIAT tokens than the amount corresponding to the collateral actually deposited.

The `enter` functions do not verify if the collateral token `safeTransferFrom` resulted in the expected amount being transferred, instead it trusts the total amount was transferred.

The total amount is assumed to be transferred after the transfer and the user provided amount parameter is used to modify the user's balance in the Codex contract, which in turn can be used to mint FIAT tokens. This is one example of the `enter` function:

```
/// ======== Entering and Exiting Collateral ======== ///

/// @notice Enters `amount` collateral into the system and credits it to `user`
/// @dev Caller has to set allowance for this contract
/// @param user Address to whom the collateral should be credited to in Codex
/// @param amount Amount of collateral to enter [wad]
function enter(address user, uint256 amount) external virtual override {
    if (live == 0) revert Vault20__enter_notLive();
    int256 wad = toInt256(wdiv(amount, 10**dec));
    codex.modifyBalance(address(this), 0, user, wad);
    token.safeTransferFrom(msg.sender, address(this), amount);
    emit Enter(user, amount);
}
```

There are tokens that transfer less than the specified amount. For example, the USDT token's `transfer` and `transferFrom` functions (Tether: USDT Stablecoin | 0xdac17f958d2ee523a2206206994597c13d831ec7) deduct a fee for each transfer if a fee percentage is configured. While it is not configured to take fees right now, this could change in the future.

Similar scenarios are possible in the future depending on the tokens that are allowed as collateral in FIATDAO vaults. Also, some contracts could be upgraded to incorporate this behavior and introduce a vulnerability as an unintended side-effect.

Recommendation

Check the balance of the contract before and after the `transferFrom` calls are performed in order to determine the exact amount that was received to bulletproof FIATDAO vaults for future updates.

FIA-4 FIAT token non-standard ERC20 implementation

Total Risk	Impact	Location
Info	-	FIAT.sol
Fixed	**Likelihood**	
✔	-	

Description

The current FIAT token implementation does not check allowances if the sender and receiver are the same address. Although unlikely, this could lead to potential problems if external parties handling FIAT incorrectly assume it will behave as other implementations do.

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public override returns (bool) {
    if (from != msg.sender) {
        uint256 allowance_ = allowance[from][msg.sender];
        if (allowance_ != type(uint256).max) {
            if (allowance_ < amount) revert FIAT__transferFrom_insufficientAllowance();
            allowance[from][msg.sender] = sub(allowance_, amount);
        }
    }
```

EIP-20 states the `transferFrom` function SHOULD throw unless the `_from` account has deliberately authorized the sender of the message via some mechanism.
Many used implementations, such as openzeppelin-contracts/ERC20.sol at master are not implemented in this fashion.

Recommendation

Consider making the FIAT token implementation as standard as possible in order to avoid potential issues.

Status

This issue was addressed by replacing FIAT token implementation with a new one based on OpenZeppelin's ERC20.

FIA-5	Known ERC-20 approve() front-running

Total Risk	Impact	Location
Info	-	`FIAT.sol`
Fixed	**Likelihood**	
✔	-	

Description

ERC20 tokens suffer from a well-known front-running issue affecting the allowance mechanism. This is described in ERC20 API: An Attack Vector on Approve/TransferFrom Methods.

Recommendation

Consider adding the `increaseAllowance` and `decreaseAllowance` functions to the FIAT.sol contract. The following implementation can be used as reference: openzeppelin-contracts/ERC20.sol at master.

Status

This issue was addressed by replacing the FIAT token implementation with a new one based on OpenZeppelin's ERC20 as suggested.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.