Horizon by Aave Labs Report

- Prepared for: Aave Labs
- Code produced by: Aave Labs
- Report prepared by: Emanuele Ricci (StErMi), Independent Security Researcher

A time-boxed security review of the **Horizon by Aave Labs** protocol was done by **StErMi**, with a focus on the security aspects of the application's smart contracts implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Horizon by Aave Labs

The Horizon by Aave Labs instance is a specialized, permissioned fork of the Aave Protocol v3.3 designed to support Real-World Asset (RWA) integration in a compliant manner for institutional use. It enables the borrowing of non-RWA tokens, such as stablecoins, using Real-World Assets (RWAs) as collateral, while meeting regulatory compliance requirements. RWA tokens are permissioned and enforce access restrictions at the asset level. In contrast, the rest of the assets—including non-RWA tokens such as stablecoins—behave as usual and can be integrated into the Horizon Pool in a normal, permissionless manner.

Key technical features include:

- aToken Extension for RWA Support: The standard aToken logic is extended to support RWA tokens, where certain actions—such as borrowing or transferring aTokens—are restricted to meet regulatory requirements.
- Authorized Transfer Functionality: A new mechanism allows authorized entities to forcibly transfer aTokens, within collateralization limits, between users, providing operational robustness and security in edge cases (e.g. regulatory or legal interventions).

References:

- ARFC Horizon’s RWA Instance
- Aave V3 Horizon GitHub Repository

About StErMi

StErMi, is an independent smart contract security researcher. He serves as a Lead Security Researcher at Spearbit and has identified multiple bugs in the wild on Immunefi and on protocol's bounty programs like the Aave Bug Bounty.

Do you want to connect with him?

- stermi.xyz website
- @StErMi on Twitter

Summary & Scope

- ***review commit hash*** **-** `04419e25d3e87327487517bf0846ffa65aac35a2`
- ***FIX REVIEW*** **-** **PR 12**
- ***FIX REVIEW*** **-** **PR 15**
- ***FIX REVIEW*** **-** **PR 17**
- ***Final Fix Review commit hash*** **-** `417a4768051126e14e492dd088c5b64add4a5b24`

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack
Likelihood - the chance that a particular vulnerability gets discovered and exploited
Severity - the overall criticality of the risk

Findings Summary

ID	Title	Severity	Status
[I-01]	Missing Reward Controller configuration documentation and possible legal edge cases	Info	Ack
[I-02]	Missing Oracle configuration for RWA assets	Info	Ack
[I-03]	Enhance the documentation relative to the configuration of RWA, Non-RWA Tokens and Pool	Info	Fixed
[I-04]	Consider refactoring the `RwaATokenManager` to improve the configuration requirements	Info	Ack
[I-05]	Consider restricting how many `aRWA` tokens a single `AUTHORIZED_TRANSFER_ROLE` user can manage via the `RwaATokenManager`	Info	Ack
[I-06]	The `RwaAToken.authorizedTransfer` execution should emit more events	Info	Fixed
[I-07]	Consider preventing the `ATOKEN_ADMIN_ROLE` from "stealing" tokens from the configured treasury	Info	Fixed
[I-08]	Consider refactoring the `RwaAToken.authorizedTransfer` auth logic	Info	Ack
[I-09]	`RwaAToken.authorizedTransfer` does not validate if `from` or `to` is a blacklisted `RWA` user	Info	Fixed
[I-10]	Minor natspec/documentation issue/typos/improvements	Info	Fixed
[I-11]	"coordinated liquidation" could create bad debt and deficit	Info	Fixed
[I-12]	Improve the description and provide practical example for the "Edge Cases" section of the "Horizon overview"	Info	Fixed
[I-13]	Non-RWA tokens, such as stablecoins, could also be borrowed by non-whitelisted RWA holders	Info	Fixed

[I-01] Missing Reward Controller configuration documentation and possible legal edge cases

Context

- Horizon-overview.md

Description

Documenting how the Reward system will be configured

The current specification document for the Horizon Protocol is not disclosing or documenting how the Reward System will be configured for both the `RWA` suppliers and the `non-RWA` token suppliers/borrowers.

1. Will `RWA` (as an underlying) be a `reward` for suppliers/borrowers of a `non-RWA` reserve?
2. Will `RWA` (as an underlying) be a `reward` for suppliers of a `RWA` reserve?
3. Will `aRWA` be a `reward` for suppliers/borrowers of a `non-RWA` (stablecoin) reserve?
4. Will `aRWA` be a `reward` for suppliers of a `non-RWA` (stablecoin) reserve?
5. Will `aRWA` be an `asset` that can receive rewards?

Sanctioned user, rewards accrual and claim

Let's assume that `ALICE` has supplied `1000 RWA_1` tokens and has borrowed `500 DAI`. Let's assume that the reward system has been configured to incentivize the `RWA_1` suppliers with `USDC` rewards.

The `RWA_1` issuer sanctioned `ALICE` and blacklisted her.

- the user is blacklisted at the `RWA` level and can't `transfer`, `transferFrom` or receive any `RWA` tokens
- the RWA issuer will try to "force transfer" the user's `aRWA` balance as much as possible (up to `HF >= 1`)

1. should ALICE be able to claim and withdraw the rewards that have been accrued from a sanctioned `aRWA` balance?
2. assuming that ALICE has a borrow position and that the RWA issuer **cannot** fully "force transfer" all the `aRWA` tokens (it can't bring the `aRWA.balanceOf(ALICE)` to `0`). Should ALICE still keep accruing rewards that come from a sanctioned balance?

Claiming `RWA` generated rewards on behalf of the owner

The `RewardsController` contract allows the claimer to claim rewards on behalf of the owner of the reward (who has enabled the caller to act "on behalf" of him, see the `onlyAuthorizedClaimers` logic).

Let's assume that the claimer is claiming rewards accrued by the owner that has supplied `RWA` assets to the the Horizon Pool.

1. Should the Horizon Protocol allow a 3rd party to claim rewards on behalf of the owner of the `RWA` ? `RwaAToken` does not allow the caller to supply `RWA` on behalf of someone else.

Claiming `RWA` generated rewards to an arbitrary `receiver`

The `RewardsController` contract allows the reward owner or the claimer (who claims on behalf of the reward's owner) to claim the rewards and send them to an arbitrary `receiver`.

Should this be allowed from a legal perspective?

Recommendations

Aave Labs should provide the Reward Controller configuration documentation for the Horizon Protocol Instance and consider documenting the above scenarios to be fully compliant with the legal requirements relative to the RWA tokens.

Aave Labs: Acknowledged. Exact reward controller configuration is in progress and documentation will be updated when this is resolved. For now a note has been left in the documentation to clarify this.

StErMi: Aave Labs has stated in the PR 17 that currently "liquidity mining rewards are to be determined"

[I-02] Missing Oracle configuration for RWA assets

Context

- Horizon-overview.md

Description

The current Horizon Protocol document does not provide any information relative to how the `Aave Oracle` contract will be configured to price the `RWA` assets provided as collateral.

1. Which oracle providers will be used?
2. Will the Aave CAPO system be used for the RWA assets?

Recommendations

Aave Labs should document how the Aave Oracle will be configured for RWA assets and if they are planning to use Aave CAPO.

Aave Labs: Acknowledged. Oracle configuration is in progress and out of scope at the moment.

[I-03] Enhance the documentation relative to the configuration of RWA, Non-RWA Tokens and Pool

Context

- Horizon-overview.md

Description

The current Horizon documentation relative to the RWA and non-RWA Tokens, such as stablecoins, could be enhanced and expanded by providing the following information:

1. For each `Pool` Operation (`deposit`, `withdraw`, `borrow`, `repay`, `liquidate`, `set-as-collateral`, `set-e-mode`, ...) and `AToken` Operation (`transfer`, `transferFrom`) provide an extended and detailed documentation, documenting all the edge case for both the `RWA` and `non-RWA` token reserves. Be as detailed as possible and for each operation provide all the edge cases.
2. For both `RWA` and `non-RWA` token, provide the reserve configuration/deployment parameters with which they will initialize with. Specify which values are "in common" across all the deployments and which instead will be custom for each reserve.
3. Detail the configuration that will be used to initialize the pool with: `_flashLoanPremiumTotal` and `_flashLoanPremiumToProtocol`
4. Document if the Horizon Protocol will be configured with custom liquid e-modes and which configurations will be used for each e-mode
5. Disclose any specific configurations. For example, during the review, Aave Labs has disclosed that `non-RWA` tokens, such as stable coins, may be borrowable in isolation mode. With which debt ceiling limit will the RWA reserves be configured with?

Recommendations

Aave Labs should enhance the documentation relative to the configuration of RWA, non-RWA tokens and Pool.

Given the Pool configuration, the RWA reserves configurations, the non-RWA tokens configurations and the e-modes, Aave Labs should also check if those configurations will create unexpected behaviors or issues given the legal requirements relative to the RWAs or the operations that should be allowed/disallowed at the Pool or Token level.

StErMi: The recommendations have been **partially** implemented in the PR 17.

Note: some important details and configuration still needs to be documented and disclosed, as Aave Labs has officially stated in the "Further Configuration" of the Horizon Overview specification file.

> Exact configuration details for eMode, isolated mode, flashloan fees, and liquidity mining rewards are to be determined.

[I-04] Consider refactoring the `RwaATokenManager` to improve the configuration requirements

Context

- RwaATokenManager.sol

Description

The current implementation of the `RwaATokenManager` does not perform any sanity checks on the `address aTokenAddress` input parameter of the `grantAuthorizedTransferRole`, `revokeAuthorizedTransferRole` or `transferRwaAToken` functions.

This enables the caller to grant/revoke roles for a non-existing `aTokenAddress` contract. While it's not possible with the current code to know if an `AToken` is an `RwaAToken` token or a "normal" `AToken` (without implementing custom behavior on the `RwaAToken` contract itself), it's still possible to enhance the security of those functions.

Aave Labs could:

1. add the `PoolAddressesProvider` as an input parameter of the `RwaATokenManager` `constructor`
2. add an immutable variable `pool` that is initialized during the `constructor` execution with the value returned by `poolAddressProvider.getPool()`
3. change the signature of all the functions to replace `aTokenAddress` with `reserveAddress`. The `aTokenAddress` can be fetched via `pool.getReserveAToken(reserveAddress)`. The function can now revert if the address returned is `address(0)` (the reserve does not exist)

Recommendations

Aave Labs should consider applying the above refactoring changes to enhance the sanity checks of the `RwaATokenManager` functions

Aave Labs: Acknowledged. However, under the current implementation, an admin who is granted an authorized transfer role to a non-existent `aToken` still would not be able to pose risks to the protocol. Therefore, we view this additional layer of sanity check as extraneous and prefer the simplicity of the current implementation.

[I-05] Consider restricting how many `aRWA` tokens a single `AUTHORIZED_TRANSFER_ROLE` user can manage via the `RwaATokenManager`

Context

- RwaATokenManager.sol

Description

The current implementation of the `RwaATokenManager` allows an `account` to manage multiple `aRWA` tokens. Given that this role, for a specific `RWA` token should be granted to `RWA` issuer, it could be expected that an `account` should be allowed to manage only one single `RWA` and not multiple ones.

Recommendations

Aave Labs should consider refactoring the `RwaATokenManager` implementation to limit how many `RWA` a single `account` should be able to manage.

Aave Labs: Acknowledged. This will be considered from a legal compliance perspective, but if needed will be enforced through configuration rather than within the smart contract.

[I-06] The `RwaAToken.authorizedTransfer` execution should emit more events

Context

- RwaATokenManager.sol#L43
- RwaAToken.sol#L117

Description

The execution of `RwaAToken.authorizedTransfer` will internally trigger multiple events, but **none** of them tracks the following information:

1. who is the `msg.sender`
2. who is the "root caller"

Aave Labs should apply the following changes:

1. `RwaAToken.authorizedTransfer` should emit a custom event like `ForcedBalanceTransfer(msg.sender, from, to, amount)` where in this case `msg.sender` is the account with the `ATOKEN_ADMIN_ROLE` role.
2. `RwaATokenManager` should emit `TransferRwaAToken(msg.sender, aTokenAddress, from, to, amount)` where `msg.sender` in this case is the account that has been granted the permission to "force transfer" tokens for the `aRWA` token `aTokenAddress`

Recommendations

Aave Labs should implement the emission of the recommended events during the execution of the `RwaAToken.authorizedTransfer` flow.

StErMi: The recommendations have been implemented in the PR 15

[I-07] Consider preventing the `ATOKEN_ADMIN_ROLE` from "stealing" tokens from the configured treasury

Context

- RwaAToken.sol#L106-L119

Description

The `ATOKEN_ADMIN_ROLE` role of the `RwaAToken` can transfer an arbitrary `amount` of the `aRWA` balance from any arbitrary `from` user, as long as the user is still healthy after the transfer. The current logic allows such admin user to also move funds from the `aRWA` treasury itself, which in theory, should be seen as a "protected" and "untouchable" account.

With the current implementation and configuration of the Horizon Pool, no `aRWA` token shares will ever be accounted to the `reserve.accruedToTreasury`, so the treasury will never receive any `aRWA` shares. This makes the current behavior (of being able to move treasury shares) "safe".

Recommendations

Aave Labs should consider implementing the above security mechanism to prevent the `ATOKEN_ADMIN_ROLE` role from moving shares from the treasury account for future scenarios, or at least document the acknowledged and accepted behavior.

StErMi The recommendations have been documented in the PR 17

[I-08] Consider refactoring the `RwaAToken.authorizedTransfer` auth logic

Context

- RwaAToken.sol#L112-L115

Description

With the current implementation, the `RwaAToken.authorizedTransfer` function can be executed by **anyone** who has the `ATOKEN_ADMIN_ROLE` in the `ACL Manager`.

This permission is in common between **all** the `RwaAToken` contracts deployed, meaning that any users who have such a role, can execute the `authorizedTransfer` for multiple different `RWA` that could be associated to different Issuers.

Recommendations

Aave Labs should consider refactoring the existing auth logic used by `RwaAToken` and perform the following changes:

1. Only the `RwaATokenManager` should be able to execute `RwaAToken.authorizedTransfer`. The `RwaATokenManager` allows the owner to specify auth access in a more secure and granular way
2. Remove the usage of the `ACLManager` from `RwaAToken` and use the `PoolAddressesProvider`. The `RwaATokenManager` can be configured as a "vetted" address in the `PoolAddressesProvider` (and swapped if needed). The `RwaAToken.authorizedTransfer` function should revert if `msg.sender != PoolAddressesProvider.getAddress('RWA_A_TOKEN_MANAGER')`

The above changes also lift the Horizon Protocol from the legal responsibility and compliance that comes by transferring `RWA`, which the `aRWA` is a receipt of.

Aave Labs: Acknowledged. While this would indeed provide more validation for `authorizedTransfer` logic, we prefer the flexibility here. The DAO can decide to grant Aave Governance, `RwaATokenManager` or any other address with this role.

[I-09] `RwaAToken.authorizedTransfer` does not validate if `from` or `to` is a blacklisted `RWA` user

Context

- RwaAToken.sol#L106-L119

Description

The `RwaAToken.authorizedTransfer` function should be called to allow the `RWA` Issuer (or an authorized user) to forcefully move funds from the `from` user to a `to` user. This function makes two assumptions that are not validated on chain

- the `from` user is a sanctioned user
- the `to` user is **not** a sanctioned user

Recommendations

Aave Labs should comment and disclose the above two assumptions and specify that the "root caller" (`msg.sender` could be the `RwaATokenManager`) is the sole responsible party for the execution and validation of input parameters.

StErMi: the recommendations have been implemented in the PR 17

[I-10] Minor natspec/documentation issue/typos/improvements

Description

- Horizon-overview.md?plain=1#L25: specify that `ATOKEN_ADMIN` is a **role** which will be given to the `RwaTokenManager` (as far as I get). The `RwaTokenManager` contract will be able to execute `RwaAToken.authorizedTransfer` on **every** `aRWA` instances. The real auth of the caller will be done at the `RwaTokenManager` level, where the Horizon Protocol specifies which user can manage the "force transfer" for which `aRWA` token.

- Horizon-overview.md?plain=1#L57 + Horizon-overview.md?plain=1#L72: rewrite the documentation part relative to the "authorized flashborrow". Currently, it seems to imply that only some "specific users" with a specific role will be able to execute a flashloan/flashborrow. The flashloan operation can be performed by anyone as long as he can repay/sustain the borrow position. In this case, the "flash borrower" is just a user with a specific role that will grant the permission to execute a **complex** flashloan without paying any premium.
- Horizon-overview.md?plain=1#L67: consider re-writing this paragraph to be more explicit and clear. User that own the `RWA` are strictly bound to the issuer's blacklist/transfer logic. Once the `RWA` has been supplied to the Horizon Pool and the `aRWA` is minted, the user still owns the underlying `RWA` (the `aRWA` can be seen as a "receipt of ownership of X amount") but your ownership is much more restricted because you won't be able to perform the same action (transfers) that you would be able to do with the `RWA` token itself. The `aRWA` token is fully "locked" and cannot be transferred.

Recommendations

Aave Labs should implement the suggestions listed in the above section.

StErMi: The recommendations have been implemented in the PR 17

[I-11] "coordinated liquidation" could create bad debt and deficit

Context

- Horizon-overview.md?plain=1#L61-L63

Description

In the "Edge Cases of Note" section, it's mentioned the edge case where a user with a borrow position is sanctioned (blacklisted) by the `RWA` issuer. The procedure expects that the following actions will happen:

1. The `RWA` Issuer will forcefully transfer the sanctioned user collateral as much as possible, bringing the `HF` user to `1`
2. The `RWA` Issuer will coordinate off-chain with "authorized" RWA holders to liquidate the user's debt and fully "forcefully transfer" the remaining `aRWA` balance to bring it to zero.

The sanctioned user debt will keep accruing interest, and it's not clear how much time it will take for the Issuer and the "authorized" liquidators to coordinate the liquidation process. It's possible that the accrued interest could lead to bad debt and deficit accounting if the remaining collateral is not enough to cover the liquidation operation.

Recommendations

Aave Labs should document and disclose this possibility

StErMi: The recommendations have been implemented in the [PR 17]

[I-12] Improve the description and provide practical example for the "Edge Cases" section of the "Horizon overview"

Context

- [Horizon-overview.md?plain=1#L54-L63]

Description

The current documentation relative to the possible edge cases to be handled is not detailed enough and lacks a practical example that would help to understand it, removing any possible doubts and misunderstandings.

For each edge case, Aave Labs should:

1. Provide a practical example with step by step execution and real values
2. Provide the full list of off-chain and on-chain operations that must be executed step-by-step, specifying which is the actor who is executing it. If the operation involves an interaction with an the Horizon Protocol Component (`Pool` , `AToken` , ...) it should be explicitly stated.
3. Disclose and document any assumption or off-chain agreement that is required for the edge case
4. Document any possible issue/limitations relative to the edge case

Edge case "user loses wallet key"

Here's a practical example that could be used for the edge case "user with borrow position loses the wallet's private key".

Assumptions:

- `RWA_1_ISSUER` has the role "flash borrower". The account will not pay a premium on the "complex" flashloan amount loaned.

- `RWA_1_ISSUER` has an agreement with `RWA` suppliers to "rescue" their Aave positions

1. `ALICE` supplies `100 RWA_1`
2. `ALICE` borrows `50 DAI`
3. `ALICE` loses the wallet key
4. `RWA_1_ISSUER` executes a "complex" flashloan for `50 DAI`. Inside the flashloan callback, the `RWA_1_ISSUER` will
   1. Generate a multisig wallet `NEW_ALICE_WALLET` to be later on "transferred" to `ALICE`
   2. Use the `50 DAI` flashloaned amount to repay `ALICE` debt
   3. Execute `aRWA_1.authorizedTransfer` to move the `100 RWA_1` collateral to `NEW_ALICE_WALLET`
   4. Borrow `50 DAI` on behalf of the new `NEW_ALICE_WALLET` account
   5. Repay the flashloan with the `50 DAI` borrowed
   6. Transfer the `NEW_ALICE_WALLET` ownership to `ALICE`

`ALICE` could need to pay a fee or repay the gas spent by the `RWA_1_ISSUER` to execute the operation

Limitations:

- The Horizon Pool could not have enough liquidity to execute the flashloan operation
- add more possible limitations

Edge case: borrower becomes sanctioned

Like for the other edge scenario, this one lacks a practical and detailed example and required assumptions that would solve many doubts and confusion.

1. What prevents a whitelisted RWA that can liquidate the sanctioned user to execute the liquidation that can't be "blocked" by the Horizon Protocol logic directly?
2. What should happen after that the `RWA` Issuer has "forcefully transferred" the sanctioned user `aRWA` balance, bringing the user to `HF = 1`?
3. What is the expected "next step" after that the Issuer has "forcefully transferred" the sanctioned user collateral balance?

Recommendations

Aave Labs should improve, detail and extend the "Edge Cases of Note" section of the documentation, providing, for each case, all the required assumptions and practical example needed.

StErMi: The recommendations have been implemented in the PR 17

[I-13] Non-RWA tokens, such as stablecoins, could also be borrowed by non-whitelisted RWA holders

Context

- Horizon-overview.md?plain=1#L67

Description

The current Horizon Overview document states

> Borrowing will be implicitly permissioned because only users that have supplied RWA assets can borrow stablecoins

While this would be usually true, there's an edge case scenario that should be considered:

The owner of the `ATOKEN_ADMIN_ROLE` role in a `RwaAToken` contract could "forcefully transfer" `aRWA` tokens (that should be considered a receipt of an `RWA` token) to a not-whitelisted `RWA` holder.

The Horizon Protocol does not perform any "whitelisted" check on the user that interact with the pool, relying on the fact that if you can supply `RWA` it means that you are indeed whitelisted by the `RWA` and you can transfer your `RWA` tokens to be deposited into the protocol.

The receiver of the `aRWA` tokens could be not-whitelisted but still leverage the new collateral to borrow `non-RWA` tokens, such as stablecoins, and open a debt position on the Horizon Protocol.

Recommendations

Aave Labs should consider updating the documentation disclosing this and similar edge cases that could allow non-whitelisted `RWA` users to borrow on the Horizon Protocol.

StErMi: The recommendations have been implemented in the PR 17