PWN Adapters
Security Review Report

October 16, 2024

Version 1.0

Email	Telephone
audits@extropy.io	+44 1865338228

Data Classification	Client Confidential
Client Name	PWN Finance
Document Title	PWN Adapters Security Review Report
Document Version	1.0
Author	Extropy Audit Team

Contents

1 | Executive Summary

Extropy was contracted to conduct an initial code review and vulnerability assessment of the PWN Adapters component, which which constitutes a piece of the PWN protocol, whose core contracts were already audited by Extropy in June 2024 (report available here).

PWN Adapters are a set of contracts that serve to move funds between PWN's loan contracts and the various pools where the funds reside. In particular, there are three types of adapters: Compound adapter, Aave adapter and ERC4626 adapter.

As the name suggests, the first two serve for communicating, respectively, with Compound and Aave protocols, while the third serves to facilitate integration with all vaults compliant with the standard ERC4626. ERC4626 is an extension of ERC20 token and proposes a standard interface for token vaults and a basic implementation is provided by OpenZeppelin and available here.

The code of the component analyzed in this document is very concise and on a logical level quite straightforward. It does not present any critical issues.

Section 4 details the findings, and where possible we have given recommendations for their resolution.

2 | Audit summary

This audit, conducted from October 1 2024 to October 8 2024, took a comprehensive approach combining manual review and automated review methods. Our examination aimed to ensure the robustness and security of the PWN Adapters codebase.

- The code is taken from the pwn_contracts_periphery repository.
- The audit was performed on commit e7e46a0d4773ca6cf2aed0b8d7728cc5738ec37c

2.1 | Audit scope

The following contracts were audited:

Contract	LoC
AaveAdapter.sol	113
CompoundAdapter.sol	54
ERC4626Adapter.sol	67
Total	**234**

2.2 | Issues Summary

ID	Finding	Status
4.1	[MEDIUM] Inflation attack is possible by front running 'ERC4626Adapter.supply()' for empty vaults	Acknowledged
4.2	[LOW] Anyone calling 'supply()' can take funds transferred to Pool Adapters	Acknowledged
4.3	[LOW] Missing input validations	Resolved
4.4	[INFO] 'hub' can be set as immutable	Resolved
4.5	[INFO] Minimum health factor concerns	Acknowledged

2.3 | Methodology

2.3.1 | Risk Rating

The risk rating given for issues follows the standard approach of the OWASP Foundation. We combine two factors :

- Likelihood of exploit
- Impact of Exploit

The Categories we use are *Critical*, *High*, *Medium*, *Low* and *Informational* These categories may not align with the categories used by other companies.
The informational category is used to contain suggestions for optimisation (where this is not seen as causing significant impact), or for alternative design or best practices.

2.4 | Approach

The project was assessed mainly by code inspection, with auditors working initially independently and then together, looking for possible exploits.

Tests were written were possible to validate the issues found.

Static analysis tools were also used, and summary results are attached to this report. These tools are seen as an adjunct to code inspection.

3 | System Overview

The source code of the PWN Adapters component at a high level is quite straightforward to understand: the adapters serve as an intermediary in the transfer of funds between the loan protocols of PWN and the pools where the funds reside. At a more detailed level, however, they have characteristics that must be analysed and understood in order to avoid problems arising in the intercommunication between different contracts and protocols.

All adapters have two common functions named 'withdraw()' and 'supply()': their features are explained below.

3.1 | Withdraw process

A withdraw process take place when a loan is created in PWN, namely there is a user (Borrower) asking for funds. The movements that funds should do during this process are:

1. from the Pool to the Lender (handled by PWN Adapters)
2. from the Lender to the Borrower (handled by PWN core contracts)

Focusing only on step 1, the caller of the 'withdraw()' method on the related Adapter contract is the PWN protocol, specifically the 'PWNLoan' contract which should have the 'ACTIVE_LOAN' tag set to 'true' in 'PWNHub' ('PWNLoan' and 'PWNHub' are examples of PWN core contracts). Then the Pool Adapter is responsible for triggering the specific method on the Pool to actuate the withdraw from the Pool to the Lender. Of course, before the process starts, the Lender needs to approve the Pool Adapter to move funds on his behalf and, in case of Aave also of aTokens.

The flow is also illustrated in the following diagram:

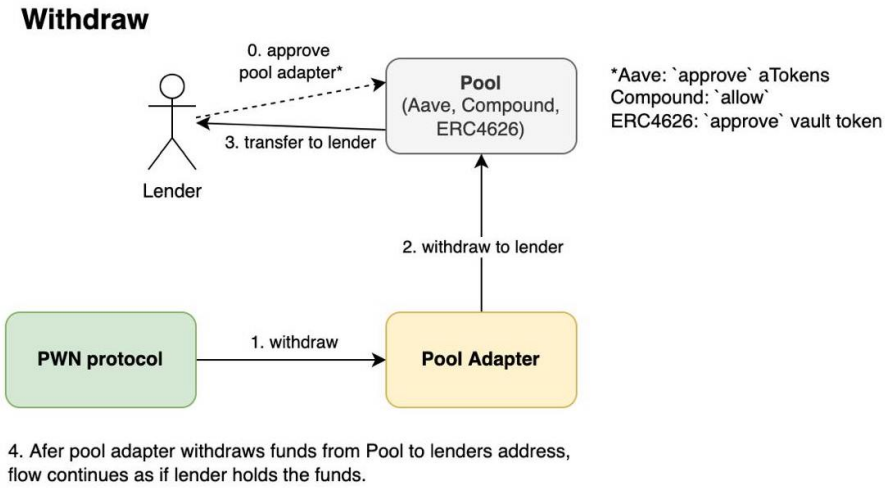

Figure 3.1: Withdraw process diagram

3.2 | Supply process

A supply process take place when a loan is repaid in PWN, namely the Borrower is giving back the funds received as a loan. The movements of the funds during this process are:

1. from the Borrower to PWN (handled by PWN core contracts)

2. from PWN contract to Pool Adapter (happens with a direct transfer)
3. from Pool Adapter to Pool (handled by PWN Adapters)

Focusing only on steps 2 and 3, the caller of the 'supply()' method on the related Adapter contract is the PWN protocol, just after having transferred supply funds to the Adapter itself. Steps 2 and 3 are usually done in a single transaction to avoid frontrunning. Then the Pool Adapter is responsible for triggering the specific method on the Pool to actuate the deposit from itself to the Pool.
The flow is also illustrated in the following diagram:

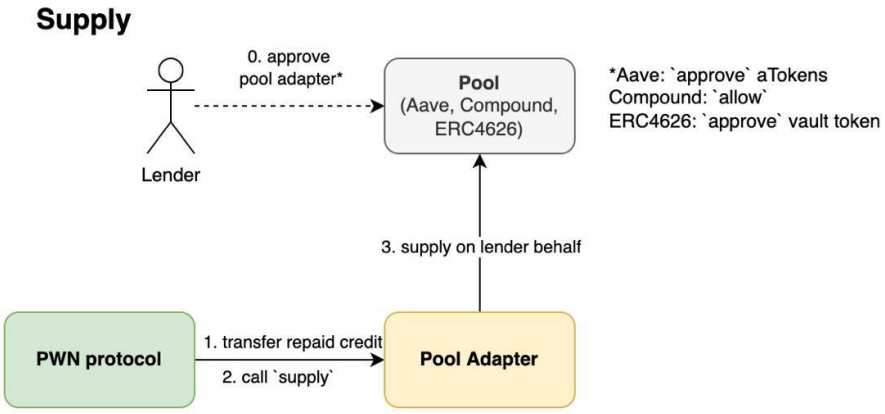

Figure 3.2: Supply process diagram

3.3 | Pool Adapters

As already mentioned, 'CompoundAdapter', 'AaveAdapter' and 'ERC4626Adapter' both come with two methods: 'withdraw()' and 'supply()'. However they behave differently since each of them call a specific external Pool Contract compliant with the related interface.
Indeed:

- '**CompoundAdapter**' invokes 'withdrawFrom()' for the withdraw process and 'supplyFrom()' for the supply process those implementations can be found in the 'Comet.sol' contract here.
- '**AaveAdapter**' invokes 'withdraw()' for the withdraw process and 'supply()' for the supply process these implementations can be found in the 'Pool.sol' here and 'SupplyLogic.sol' contracts here of Aave.
- '**ERC4626Adapter**' invokes 'withdraw()' for the withdraw process and 'deposit()' for the supply process these implementations can be found in the 'ERC4626.sol' contract here from OpenZeppelin.

The main difference that can be highlighted is that only when supplying to Aave and ERC4626 vault there is a token/share issued to the address specified during the deposit (which in both cases is referred as the 'owner'), while for Compound there is no token/share issued but just internal accounting to the 'to' address (which again in the Adapter is referred as the 'owner').

Particularly, depositing on Aave an amount of tokens X means that the 'owner' address will receive an amount X of aTokens (1:1 ratio) that can later be burned to redeem the underlying asset. For the ERC426 vault however the address referred as 'owner' will receive an amount of shares computed from the amount deposited by the function _convertToShares(). As for Aave, those shares will be burned when withdrawing.

4 | Findings

4.1 | [MEDIUM] Inflation attack is possible by front running 'ERC4626Adapter.supply()' for empty vaults

- **Location(s):** ERC4626Adapter.sol#54-65
- **Description:** Within `ERC4626Adapter.sol` the function `supply()` handle the deposit process to the ERC4626 `vault`.

```
function supply(address vault, address owner, address asset, uint256 amount) external {
    // Check the asset of the vault
    address vaultAsset = IERC4626Like(vault).asset();
    if (vaultAsset != asset) {
        revert InvalidVaultAsset(asset, vaultAsset);
    }

    // Note: Performing optimistic deposit, assuming that the vault will revert if
    //the amount exceeds the max deposit
    // Supply to the vault on behalf of the owner
    asset.ERC20(amount).approveAsset(vault);
    IERC4626Like(vault).deposit(amount, owner);
}
```

However empty ERC-4626 vault are subject to what is known as "Inflation Attack" and if a malicious user spots specific transactions in the mempool he can frontrun them and cause issues for the PWN protocol. Particularly attacker should spot one of the following:

- a transaction which both moves funds to `ERC4626Adapter` and calls `supply()` for an empty (or nearly empty) `vault`
- a transaction which directly calls "supply()" for an empty (or nearly empty) vault , after having transferred funds to `ERC4626Adapter` in a previous transaction

Front running one of them allows the attacker to manipulate the vault exchange rate and then performing the mentioned "Inflation Attack". Even if this is a problem related to how ERC-4626 computes shares, this may have consequences on the depositor.

When a deposit is made to an ERC-4626 vault, the amount of shares minted in return is computed from the current vault exchange rate and rounded down to the nearest whole number. This rounding can sometimes result in losses, especially for small deposits. For example, making a deposit which returns less than 1 share (rounded down to zero) means receiving nothing and losing the deposit, since later you will not have shares to burn for the claim. It's basically a "donation" to the vault.

In empty (or nearly empty) ERC-4626 vaults, deposits are at high risk of being stolen: attacker frontrunning the initial deposit with a "donation" to the vault can inflates the price of a share.

Let's suppose this scenario:

- an attacker makes a small deposit (e.g. 1 token) to an empty (or nearly empty) ERC-4626 vault, becoming a shareholder

- attacker "donates" a large number of assets (e.g. 1e18 tokens) to the vault shifting the exchange rate drastically. This puts the vault in a state where any deposit smaller than 1e18 would be completely lost to the vault (a donation).
- a user makes a deposit with skewed rates, receiving less shares than expected. In this context who makes the deposit is the ERC4626Adapter on behalf of the `owner`
- attacker shares worth more now and he can withdraw part of users deposits. In our example, every user deposit smaller than 1e18 tokens allows the attacker to steal all the user deposit

More on this issue here[1] and here[2].

- **Recommendation:** Consider ensuring that the ERC-4626 vaults intended to be used with the PWN protocol are not empty.
- **Status:** Acknowledged.
- **Updates:**
  - `[PWN Finance, 14/10/2024]`: The client acknowledged the issue and stated: "This is a very interesting attack vector, thank you for discovering that. Nevertheless, the attack is not unique to our case and affects any deposit, even from EOA. Some vaults implement protective measures, so instead of checking for (almost) no liquidity, we will allow lenders to turn off automatic deposit of loan repayment and notify them when we detect that it might be necessary. It will involve changes in the Loan contract, which is not part of this audit scope. "

4.2 | [LOW] Anyone calling 'supply()' can take funds transferred to Pool Adapters

- **Location(s):** CompoundAdapter.sol#48 , AaveAdapter.sol#95, ERC4626Adapter.sol#54
- **Description:** Within `CompoundAdapter`, `AaveAdapter` and `ERC4626Adapter` the function `supply()` is supposed to transfer funds from the Pool Adapter to the Pool. This should happen only after a direct transfer made by PWN to the Pool Adapter contract.

  Below there is the code snippet of `CompoundAdapter.supply()` but it works similar for the other two adapters.

```
function supply(address pool, address owner, address asset, uint256 amount) external {
    // Supply to the pool on behalf of the owner
    asset.ERC20(amount).approveAsset(pool);
    ICometLike(pool).supplyFrom(address(this), owner, asset, amount);
}
```

  If an attacker monitoring the mempool spots a transfer to a Pool Adapter, he can directly call the function `supply()` taking the deposit and specifying himself as the `owner`. Later he can redeem the underlying asset by burning the token/shares he received as the "depositor" stealing funds.

- **Recommendation:** Consider restricting the access to `supply()` function only to the address who transferred the funds to the Pool Adapter.
- **Status:** Acknowledged.
- **Updates:**

[1] https://docs.openzeppelin.com/contracts/5.x/erc4626
[2] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/8b591baef460523e5ca1c53712c464bcc1a1c467/contracts/token/ERC20/extensions/ERC4626.sol#L22

- [PWN Finance, 01/10/2024]: The client acknowledged the issue and stated: "The supply() function is intended to be used only from PWNLoan contract, called in a single transaction with the transfer itself"
- [Extropy, 16/10/2024]: We agree that within the same transaction funds are transferred from the PWN protocol to the Pool Adapter and `supply()` is invoked on the recipient (the code is visibile here[3]). Thus there is no way for an attacker to place himself in the middle of the two operations and call `supply()` before the PWN protocol.

4.3 | [LOW] Missing input validations

- **Location(s):** CompoundAdapter.sol#28 , CompoundAdapter.sol#35 , CompoundAdapter.sol#48 , AaveAdapter.sol#61 , AaveAdapter.sol#68 , AaveAdapter.sol#95, AaveAdapter.sol#105, ERC4626Adapter.sol#27, ERC4626Adapter.sol#34, ERC4626Adapter.sol#54
- **Description:** The codebase lacks of input validations. Particularlu:
  - Within the constructor of `CompoundAdapter`, `AaveAdapter` and `ERC4626Adapter` the address `hub` in input is not checked to be non zero;
  - at line 35 of `CompoundAdapter.sol` is not checked if the inputs `pool`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 48 of `CompoundAdapter.sol` is not checked if the inputs `pool`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 68 of `AaveAdapter.sol` is not checked if the inputs `pool`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 95 of `AaveAdapter.sol` is not checked if the inputs `pool`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 34 of `ERC4626Adapter.sol` is not checked if the inputs `vault`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 54 of `ERC4626Adapter.sol` is not checked if the inputs `vault`, `owner`, `asset` are not the zero address and that `amount` is greater than zero;
  - at line 105 of `AaveAdapter.sol` there is no upper bound for the input `_minHealthFactor`
- **Recommendation:** Consider adding checks on function inputs to avoid unwanted behaviors.
- **Status:** Resolved.
- **Updates:**
  - [PWN Finance, 14/10/2024]: Fixed in commit 17db9bc8a93d710c308a5e642e7f5ab5e924e733[4]

4.4 | [INFO] 'hub' can be set as immutable

- **Location(s):** CompoundAdapter.sol#26, ERC4626Adapter.sol#23, AaveAdapter.sol#54
- **Description:** Within the mentioned locations the state variable `hub` is set within the constructor and it's not supposed to change:

```
PWNHub public hub;
```

- **Recommendation:** Consider changing `hub` to immutable in order to save gas.

[3] https://github.com/PWNDAO/pwn_contracts/blob/ffb77b6867ae407b0d45208132dfb0d7647a5904/src/loan/vault/PWNVault.sol#L143-L152

[4] https://github.com/PWNDAO/pwn_contracts_periphery/commit/17db9bc8a93d710c308a5e642e7f5ab5e924e733

- **Status:** Resolved.
- **Updates:**
  - [PWN Finance, 14/10/2024]: Fixed in commit 5565162bfb35916c287056c1df49fad3656196e3[5]

4.5 | [INFO] Minimum health factor concerns

- **Location(s):** AaveAdapter.sol
- **Description:** Within `AaveAdapter.sol` the function `setMinHealthFactor()` allows to set a value, different for every user, in the mapping `minHealthFactor`. This value, `_minHealthFactor`, is passed as input and should be greater than `1e18`.

```
function setMinHealthFactor(uint256 _minHealthFactor) external {
    if (_minHealthFactor < 1e18) {
        revert InvalidMinHealthFactor(_minHealthFactor);
    }

    minHealthFactor[msg.sender] = _minHealthFactor;
}
```

  In the withdraw() function it's checked if the current `healthFactor` retrieved from Aave is greater than this minimum value, if it was set as described above, or if it's greater than `DEFAULT_MIN_HEALTH_FACTOR = 1.2e18` if a minimum threshold was not set before.

  The `healthFactor` retrieved from Aave is a key indicator of the safety of a loan relative to the value of the deposited collateral. It is a numerical value that measures the risk of liquidation of a certain position: the higher the Health Factor, the safer the position is. Depositing more collateral increases the Health Factor while withdrawing collateral lowers it down. Particularly a health factor below 1 (HEALTH_FACTOR_LIQUIDATION_THRESHOLD = 1e18[6]) represents a borrow position that is eligible for liquidation. Note that the Health Factor changes also depending on collateral and loaned asset price fluctuations.

  However, `setMinHealthFactor()` is designed to allow a minimum threshold greater than or equal to 1e18 otherwise withdraws are blocked in `withdraw()`. This seems to be useless since the withdraw process it's already blocked from Aave logic itself in `ValidationLogic.validateHealthFactor()` here[7].

- **Recommendation:** If the goal is to prevent withdrawing if the Health Factor is near to 1e18, consider just keep `DEFAULT_MIN_HEALTH_FACTOR` as the minimum value allowed to be set in AaveAdapter when `setMinHealthFactor()` is called. Explicitly disallowing set values lower than `1e18` in `setMinHealthFactor()` is useless as explained above.
- **Status:** Acknowledged.
- **Updates:**

[5] https://github.com/PWNDAO/pwn_contracts_periphery/commit/5565162bfb35916c287056c1df49fad3656196e3
[6] https://github.com/aave/aave-v3-core/blob/782f51917056a53a2c228701058a6c3fb233684a/contracts/protocol/libraries/logic/ValidationLogic.sol#L53C27-L53C69
[7] https://github.com/aave/aave-v3-core/blob/782f51917056a53a2c228701058a6c3fb233684a/contracts/protocol/libraries/logic/ValidationLogic.sol#L572

- `[PWN Finance, 14/10/2024]`: The client acknowledged the issue and stated: "The goal is to protect users but also to give them the ability to "turn it off" by setting min HF to 1. The value 1 is intentionally set not to give the false impression that the HF can be below 1 just by using our protocol."

5 | Test Coverage

5.1 | Solidity tests

```
forge test
[] Compiling...
No files changed, compilation skipped

Running 2 tests for test/unit/AaveAdapter.t.sol:AaveAdapter_SetMinHealthFactor_Test
[PASS] testFuzz_shouldFail_whenNewHealthFactorLessThanOne(uint256)
        (runs: 256, : 15126, ~: 14839)
[PASS] testFuzz_shouldUpdateMinHealthFactor(uint256) (runs: 256, : 37621, ~: 37346)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 40.38ms

Running 2 tests for test/unit/CompoundAdapter.t.sol:CompoundAdapter_Withdraw_Test
[PASS] testFuzz_shouldFail_whenCallerNotActiveLoan(address)
        (runs: 256, : 25886, ~: 25886)
[PASS] testFuzz_shouldWithdrawFromCompound(address,address,uint256)
        (runs: 256, : 22582, ~: 22582)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 41.14ms

Running 2 tests for test/unit/AaveAdapter.t.sol:AaveAdapter_Supply_Test
[PASS] testFuzz_shouldApprovePool(uint256) (runs: 256, : 23672, ~: 23680)
[PASS] testFuzz_shouldDepositToPool(address,uint256) (runs: 256, : 21850, ~: 21873)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 43.92ms

Running 3 tests for test/unit/ERC4626Adapter.t.sol:ERC4626Adapter_Supply_Test
[PASS] testFuzz_shouldApproveVault(uint256) (runs: 256, : 24073, ~: 24081)
[PASS] testFuzz_shouldDepositToVault(address,uint256) (runs: 256, : 20732, ~: 20745)
[PASS] testFuzz_shouldFail_whenVaultAssetNotSupplyAsset(address)
       (runs: 256, : 21516, ~: 21516)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 65.55ms

Running 3 tests for test/unit/ERC4626Adapter.t.sol:ERC4626Adapter_Withdraw_Test
[PASS] testFuzz_shouldFail_whenCallerNotActiveLoan(address)
       (runs: 256, : 25868, ~: 25868)
[PASS] testFuzz_shouldFail_whenVaultAssetNotWithdrawAsset(address)
       (runs: 256, : 29022, ~: 29022)
[PASS] testFuzz_shouldWithdrawFromVault(address,uint256)
       (runs: 256, : 23699, ~: 23699)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 77.82ms

Running 2 tests for test/unit/CompoundAdapter.t.sol:CompoundAdapter_Supply_Test
[PASS] testFuzz_shouldApproveCompound(uint256) (runs: 256, : 23596, ~: 23612)
[PASS] testFuzz_shouldDepositToCompound(address,uint256)
       (runs: 256, : 22025, ~: 22048)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 22.80ms

Running 5 tests for test/unit/AaveAdapter.t.sol:AaveAdapter_Withdraw_Test
[PASS] testFuzz_shouldFail_whenCallerNotActiveLoan(address)
       (runs: 256, : 25923, ~: 25923)
[PASS] testFuzz_shouldFail_whenOwnerHealthFactorBelowDefautlMin_whenNotSet(uint256)
       (runs: 256, : 41416, ~: 41137)
[PASS] testFuzz_shouldFail_whenOwnerHealthFactorBelowMin_whenSet(uint256)
       (runs: 256, : 62075, ~: 61781)
[PASS] testFuzz_shouldTransferATokensToAdapter(address,uint256)
       (runs: 256, : 36052, ~: 36052)
```

```
[PASS] testFuzz_shouldWithdrawFromPool(address,address,uint256)
       (runs: 256, : 37621, ~: 37621)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 77.71ms
```

A | Disclaimers

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purpose

A.1 | Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

A.2 | Proprietary Information

The content of this document should be considered proprietary information. Extropy gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

B | Slither results

```
slither .
'forge clean' running
'forge config --json' running
'forge build --build-info --skip */test/** */script/** --force' running

INFO:Detectors:
AaveAdapter.withdraw(address,address,address,uint256)
 ↪  (src/pool-adapter/AaveAdapter.sol#68-90) ignores return value by
 ↪  (None,None,None,None,None,healthFactor) =
 ↪  IAavePoolLike(pool).getUserAccountData(owner)
 ↪  (src/pool-adapter/AaveAdapter.sol#81)
AaveAdapter.withdraw(address,address,address,uint256)
 ↪  (src/pool-adapter/AaveAdapter.sol#68-90) ignores return value by
 ↪  IAavePoolLike(pool).withdraw(asset,amount,owner)
 ↪  (src/pool-adapter/AaveAdapter.sol#89)
ERC4626Adapter.withdraw(address,address,address,uint256)
 ↪  (src/pool-adapter/ERC4626Adapter.sol#34-49) ignores return value by
 ↪  IERC4626Like(vault).withdraw(amount,owner,owner)
 ↪  (src/pool-adapter/ERC4626Adapter.sol#48)
ERC4626Adapter.supply(address,address,address,uint256)
 ↪  (src/pool-adapter/ERC4626Adapter.sol#54-65) ignores return value by
 ↪  IERC4626Like(vault).deposit(amount,owner)
 ↪  (src/pool-adapter/ERC4626Adapter.sol#64)
Reference:
 ↪  https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

INFO:Detectors:
Parameter AaveAdapter.setMinHealthFactor(uint256)._minHealthFactor
 ↪  (src/pool-adapter/AaveAdapter.sol#105) is not in mixedCase

INFO:Detectors:
AaveAdapter.hub (src/pool-adapter/AaveAdapter.sol#54) should be immutable
CompoundAdapter.hub (src/pool-adapter/CompoundAdapter.sol#26) should be immutable
ERC4626Adapter.hub (src/pool-adapter/ERC4626Adapter.sol#23) should be immutable
Reference: https://github.com/crytic/slither/wiki/
       Detector-Documentation#state-variables-that-could-be-declared-immutable
```

extropy