Brava Labs

Brava - Smart Contracts

Security Assessment Report

Version: 2.1

January, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Brava Labs smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Brava Labs smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Brava Labs smart contracts in scope.

Overview

The Brava protocol is a set of pools built on top of the Safe smart wallet platform that enable the use of self-custodied investment strategies, intended for consumption by institutional clients as well as less technical end-users.

Brava achieves this with a system of pools with delegated actions that act directly on each Safe smart wallet, avoiding the need to combine user funds or create undesirable trust assumptions.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Brava Labs repository.

The scope of this time-boxed review was strictly limited to the following files and directories at commit 6556134:

- `actions/*`
- `auth/*`
- `SequenceExecutor.sol`

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

The fixes of identified issues were assessed at commit 29d4211.

Approach

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 16 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- High: 3 issues.
- Medium: 3 issues.
- Low: 4 issues.
- Informational: 5 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Brava Labs smart contracts in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
BRAV-01	Unrestricted Role Revocation in `AccessControlDelayed`	Critical	Resolved
BRAV-02	Protocol Fees Can Be Bypassed By Users	High	Resolved
BRAV-03	Protocol Fees Can Block Withdrawals	High	Resolved
BRAV-04	Disabling Fee Module From The Safe Wallet Prevents Fee Collection	High	Resolved
BRAV-05	Multiple Methods To Avoid Paying Fees On Withdrawals	Medium	Resolved
BRAV-06	Protocol Fees Can Block Further Deposits	Medium	Resolved
BRAV-07	Yearn Vault Withdrawal Amount Discrepancy	Medium	Resolved
BRAV-08	Supply Action Will Revert If User Already Has Assets	Low	Resolved
BRAV-09	Excess Insurance Coverage Payments Can Get Temporarily Stuck	Low	Resolved
BRAV-10	`proposeFeeConfig()` Can Overwrite Ongoing Fee Proposal	Low	Closed
BRAV-11	Proposals Can Be Reused To Add Pools	Low	Resolved
BRAV-12	`SendToken` Event Emitted Amount Can Be Wrong	Informational	Resolved
BRAV-13	`ERC-4626` Pools Do Not Make Use Of `maxDeposit()`	Informational	Resolved
BRAV-14	Georestrictions On Some Supported Protocols	Informational	Resolved
BRAV-15	Some Roles Cannot Be Granted To New Users	Informational	Resolved
BRAV-16	Miscellaneous General Comments	Informational	Resolved

BRAV-01	Unrestricted Role Revocation in `AccessControlDelayed`
Asset	`auth/AccessControlDelayed.sol`
Status	**Resolved:** See Resolution
Rating	Severity: Critical · Impact: High · Likelihood: High

Description

As there is no access control on the function `revokeRole()`, any user can revoke roles of other users.

The function `revokeRole()` does not implement restrictions on who can call it and directly uses the `super._revokeRole()` internal function, which also does not enforce any access control mechanisms.

```
function revokeRole(bytes32 role, address account) public override(AccessControl) {
    require(hasRole(role, account), Errors.AdminVault_NotGranted());
    super._revokeRole(role, account);
    // no need to check if the proposal exists, we're just setting it to 0
    delete proposedRoles[keccak256(abi.encodePacked(role, account))];
    LOGGER.logAdminVaultEvent(404, abi.encode(role, account));
    }
```

Any user can call this function to revoke critical roles, such as owner or admin, which are essential for the protocol's security and operation. Removing crucial roles could lead to a complete denial of service (DoS) on the protocol.

Note, this function can also be utilised to remove role proposers, posing an additional risk to the protocol.

Recommendations

Use the external function `super.revokeRole()` instead of `super._revokeRole()` in `AccessControlDelayed.revokeRole()`.

Resolution

The use of `super._revokeRole()` has been replaced with `super.revokeRole()` enforcing correct access control on `revokeRole()`.

BRAV-02	Protocol Fees Can Be Bypassed By Users		
Asset	`auth/AdminVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Fee timestamps are crucial for calculating the payments users owe to Brava during interactions, however, users can directly manipulate these timestamps, potentially avoiding any fee payments.

A user can interact directly with the `AdminVault` contract through their Safe wallet by sending a transaction to the `setFeeTimestamp()` function, specifying the pool addresses where they hold active deposits.

There are no access restrictions on this function, allowing the user to update the `lastFeeTimestamp` for both their Safe wallet and the specified pool to the current timestamp. Because `lastFeeTimestamp` plays a critical role in the `_calculateFee()` function, which assesses fees for nearly all user actions, this could enable a user to maliciously update the timestamp before withdrawing funds, thereby evading fee payments.

Recommendations

Users should not have unrestricted access to alter their fee timestamp.

Addressing this vulnerability would likely require significant modifications to the fee protocol's design.

Resolution

Brava Labs have added a guard to the Safe wallets that end users deploy. This restricts calls made from the Safe wallet to only be sent to the `SequenceExecutor` contract which, in turn, checks that only approved actions are performed from the Safe wallet.

BRAV-03	Protocol Fees Can Block Withdrawals		
Asset	`actions/across/AcrossWithdraw.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

It is possible for the payment of accrued fees to cause withdrawals on the same vault to fail.

When processing withdrawals in `AcrossWithdraw`, the withdrawal amount is set using `sharesBefore`, which is recorded prior to the removal of accrued fees. This means that a withdrawal can be processed for quantity of shares that the account no longer holds, which would then cause the withdrawal call to revert.

This will happen with proportionally large withdrawals and, as such, is highly likely to occur as users will usually opt to remove all assets when withdrawing.

Note, this issue has the same accounting cause as BRAV-06, but it has been recorded as a separate finding due to different impact and likelihood of occurring.

This finding has been rated a medium impact as only one supported protocol is affected (Across), and the issue could be resolved by temporarily setting protocol fees to zero.

Recommendations

Alter the accounting flow such that the share balance is recorded after fees have been processed, but before the withdrawal action occurs.

Resolution

The withdrawal amount in `AcrossSupply.sol` has been adjusted to take into account any protocol fees paid during the function call.

BRAV-04	Disabling Fee Module From The Safe Wallet Prevents Fee Collection		
Asset	`auth/FeeTakeSafeModule.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Safe wallet owners can disable the fee-taking module from their wallets by utilising the built-in module management functionality. Consequently, bots will be unable to collect fees from these wallets.

The fee collection mechanism relies on whitelisted bots invoking the `FeeTakeSafeModule.takeFees()` function to extract fees from Safe wallets. This function initiates a zero-amount deposit which is executed in the context of the Safe wallet using the function `execTransactionFromModule()`:

```
bool success = ISafe(_safeAddr).execTransactionFromModule(
    SEQUENCE_EXECUTOR_ADDR,
    msg.value,
    sequenceData,
    Enum.Operation.DelegateCall
);
```

Malicious Safe wallet owners can bypass this mechanism by simply disabling the fee-taking module from their wallet using `disableModule()` function. This action effectively prevents bots from executing the `takeFees()` function and collecting the intended fees.

Recommendations

Safe wallet users should not be able to disable the fee module from their wallet to prevent regular fee collection.

Addressing this vulnerability would likely require significant modifications to the design of the Safe fee module.

Note, the users who disable the fee module would still end up paying fees to Brava, but only when they interact with the protocol through new deposits or withdrawals.

Resolution

Brava Labs have added a guard to the Safe wallets that end users deploy. This restricts calls made from the Safe wallet to only be sent to the `SequenceExecutor` contract which, in turn, checks that only approved actions are performed from the Safe wallet.

BRAV-05	Multiple Methods To Avoid Paying Fees On Withdrawals		
Asset	/*		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Users are intended to pay an annual fee based on deposit size to Brava, this annual fee can be bypassed via various means which remove the liquidity provider tokens from the protocol.

1. Some pool withdraw action contracts include a function called `exit()`. This function was intended as an escape hatch should a code error cause a DoS situation to occur with fee payments. However, as this function has no access control, it can be used at any time to exit the system without paying accrued interest to Brava.
2. Likewise, the `SendToken`s action can be used to remove the tokenised representations of liquidity such as `aTokens` used by Aave. Once removed, these positions can then be exited using the frontend of the underlying protocol, again, avoiding payment of fees to Brava.
3. More advanced users would also be able to design their own calls that make use of the Safe wallet's existing ability to `call()` or `delegatecall()` other contracts to achieve the same result of removing supplied assets without paying any fees to Brava.

Recommendations

Completely preventing this issue is likely to be impossible due to the flexibility afforded by a Safe wallet to the end user. However, for Brava's intended target audience the following steps may help enforce intended action flows:

- Restrict the `SendToken` action from sending known liquidity tokens supported by the pools. A list of these tokens could be stored in `AdminVault` which could then be checked against on transfers.
- Remove the `exit()` function or add an emergency exit state that must be triggered globally prior to it being available to users.

Resolution

Multiple steps have been taken by the Brava team to solve this issue:

- Action contracts with the `exit()` function have had it removed as it was a stale design.
- When calling the `SendToken` action it is checked that the `feeTimestamp` has been updated at the same time. If not then the call to `SendToken` will revert, prompting the user to pay the owed fees.
- Brava Labs have added a guard to the Safe wallets that end users deploy. This restricts calls made from the Safe wallet to only be sent to the `SequenceExecutor` contract which, in turn, checks that only approved actions are performed from the Safe wallet.

BRAV-06	Protocol Fees Can Block Further Deposits		
Asset	`actions/across/AcrossSupply.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Due to system accounting, it is possible for the payment of accrued fees to Brava to cause further deposits to fail.

When supplying assets, accrued fees owed to Brava are first removed before processing the deposit. As this occurs after taking the initial share balance, it is possible that the overall share balance may decrease after a deposit, for example, if a user has not interacted with the protocol in a long time, such as 1 year. If the share balance decreases, then the accounting on line [79] will cause an overflow revert, which in turn will cause the deposit to fail.

This revert can also occur as part of the slippage control in `AcrossSupply` with the check on line [80]:

```
require(
    shares >= _inputData.minSharesReceived,
    Errors.Action_InsufficientSharesReceived(
        protocolName(),
        uint8(actionType()),
        shares,
        _inputData.minSharesReceived
    )
);
```

This check is intended to prevent a supply issuing too few shares for a deposit and so is more likely to occur than the previously mentioned overflow.

Note, while only Across can revert due to this issue, the other protocols are affected by this for the accounting aspect only. Other protocols do not revert, but the `sharesBefore` and `sharesAfter` emitted by their events do not accurately represent the deposit size.

Recommendations

Alter the accounting flow such that the share balance is recorded after fees have been processed, but before the supply or withdrawal action occurs.

Alternatively, ensure regular use of the fee taking bot, so that accrued fees are always proportionally very small compared to user deposits.

Resolution

The share balance calculations in `AcrossSupply` have been adjusted to remove the potential for protocol fee payment to cause an overflow revert.

The Brava team have noted that the delta of `sharesBefore` and `sharesAfter` emitted by events is not intended to represent the size of a user's deposit and so other contracts have not been altered.

BRAV-07	Yearn Vault Withdrawal Amount Discrepancy		
Asset	`action/yearn/YearnWithdraw.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

Users withdrawing tokens from the Yearn Protocol would not receive the exact amount requested.

Yearn vaults process withdrawals in `yvToken` rather than the underlying asset. Therefore, if a user requests an amount to withdraw from a Yearn vault using the Brava protocol, they will end up receiving a different amount due to the exchange rate between the `yvToken` and the underlying asset.

This design differs from other pool designs that Brava is integrating. Moreover, the NatSpec comment for the function `ERC4626Withdraw._executeWithdraw()` on line [**108**] states that the parameter amount represents *"the amount of the underlying token to withdraw"*. However, this is inaccurate in the context of the Yearn vault.

Recommendations

To have a similar withdrawal design as other pools, override the function `_executeWithdraw()` for `YearnWithdraw` contract so it performs the conversion of the requested underlying amount to the corresponding amount of `yvTokens` before calling the yearn vault `withdraw()`.

Necessary changes should also be made to the function `YearnWithdraw._getMaxWithdraw()`.

Resolution

The Brava team have added a new withdrawal standard that handles pools which specify withdrawals in terms of shares rather underlying tokens.

BRAV-08	Supply Action Will Revert If User Already Has Assets		
Asset	`actions/ActionBase.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

If a user holds a liquidity providing token to a platform integrated by Brava in their Safe wallet, then supply actions for that protocol will revert.

When supplying to a new protocol using Brava, there is a check performed in `_processFee()` to initialise the fee timestamp. However, in the event that a user has already supplied liquidity to a protocol, externally to Brava, this check will malfunction and assume the fee timestamp has already been initialised due to the non-zero `_shareBalance`. This will then trigger a revert on line [91] due to the fee timestamp not being initialised.

The development team has highlighted that their target market consists of non-technical users and institutions. With the protocol set to automatically deploy a new Safe wallet for each user, the likelihood of this issue occurring has been marked as low.

Recommendations

Implement other checks than the share balance to determine if the fee timestamp needs initialising, for example:

- Verify if the current value stored as the fee timestamp is zero.
- Introduce a boolean storage value to indicate if a user has deposited to the chosen protocol before.

Resolution

Fee timestamp initialization has been decoupled from the user's share balance and so operates correctly if the user has already supplied liquidity prior to using Brava's system.

BRAV-09	Excess Insurance Coverage Payments Can Get Temporarily Stuck		
Asset	`actions/nexus-mutual/BuyCover.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When purchasing deposit insurance cover from Nexus Mutual, it is possible to send more ETH for the payment than is required. This extra ETH would then remain in the Safe wallet and not be returned to the user.

When purchasing Nexus Mutual cover, it is possible to pay the cost using a transfer of ETH sent with the `BuyCover()` action. However, the amount forwarded to Nexus Mutual is only `params.maxPremiumInAsset` which can be less than `msg.value`, as such, any excess ETH will remain in the Safe wallet.

Additionally, this can occur when `params.maxPremiumInAsset` is excessive for the period of insurance cover desired, as any excess premium is returned to the caller which, in this case, is the Safe wallet.

It should be noted that retrieving this stuck ETH from the Safe wallet is possible, but is not part of any normal action offered by Brava. A user would have to notice the ETH balance on the Safe wallet and create a custom call to retrieve it.

Recommendations

There are multiple possible resolution methods:

- Check that `params.maxPremiumInAsset == msg.value` when processing the `BuyCover` action. It should be noted that this will prevent users from creating a sequence that includes multiple `BuyCover` actions and that this does not protect against sending an excessive `params.maxPremiumInAsset` amount to Nexus Mutual. This solution may therefore be coupled with another solution to help reduce gas usage and simplify most insurance purchase occurrences.
- Forward any ETH balance in the Safe wallet after the insurance purchase back to the user.
- Create an action similar to `SendToken` for sending ETH in the Safe wallet to the wallet owner.

Resolution

The `SendToken` action can now also send ETH allowing users to reclaim any ETH in their smart wallet.

BRAV-10	`proposeFeeConfig()` Can Overwrite Ongoing Fee Proposal		
Asset	`auth/AdminVault.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The function `proposeFeeConfig()` does not check if there is an ongoing fee proposal. This allows users with the `FEE_PROPOSER_ROLE` to submit a new fee proposal, overwriting the current one.

Malicious actors with the `FEE_PROPOSER_ROLE` could repeatedly overwrite proposals to intentionally sabotage the fee adjustment process.

Recommendations

Add a check in the function `proposeFeeConfig()` to verify if there is an ongoing proposal. This can be achieved by checking the `proposalTime` of the struct `pendingFeeConfig`.

Resolution

The Brava team have acknowledged that this is possible for accounts with the correct role to overwrite existing fee proposals and have opted to keep this design to reduce operational friction.

BRAV-11	Proposals Can Be Reused To Add Pools		
Asset	`auth/AdminVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Pool proposals are not being deleted after a pool is added. As a result, old proposals can be reused to add the pool without requiring a new proposal.

Adding a pool requires a new proposal, and the proposal delay period must lapse before it can be executed. However, in the `addPool()` function, once the pool is added, the corresponding proposal is not deleted.

As a result, if the pool is ever removed using the `removePool()` function, the old proposal can be reused by the `POOL_EXECUTOR_ROLE` to add the pool back without requiring the `POOL_PROPOSER_ROLE` to propose it, and without going through the delay period.

Recommendations

Delete the proposal once the pool is added.

Resolution

Pool proposal and action proposals mappings are now deleted as part of the flow in `addPool()` and `addAction()` preventing reuse of old proposals.

BRAV-12	`SendToken` Event Emitted Amount Can Be Wrong
Asset	`actions/utils/SendToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `SendToken` amount logged in its event can be incorrect in some circumstances.

The `SendToken` action can accept `type(uint256).max` as a special amount input, when received, it will transfer the full balance of the Safe wallet for that token.

This is not reflected in the event logging emitted alongside the transfer, as it uses `inputData.amount` which would still be `type(uint256).max` .

Recommendations

Alter the logic of `executeAction()` such that the adjustment for a `type(uint256).max` amount is done prior to calling `_sendToken()` so it is also reflected in the event logging.

Resolution

The `SendToken` action's event now reports the correct amount of tokens sent.

BRAV-13	`ERC-4626` Pools Do Not Make Use Of `maxDeposit()`
Asset	`action/common/ERC4626Supply.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `ERC-4626` vault specification defines a `maxDeposit()` function that returns *"the maximum amount of underlying assets that can be deposited in a single deposit call"*.

If a deposit exceeds this size the call will revert, this feature of the specification is not used currently by deposits.

Recommendations

Make use of the `maxDeposit()` function to reduce incidents of reverting supply actions.

As users will likely be using Brava's frontend to make these calls, this could occur offchain in the frontend rather than at the smart contract level.

Resolution

The `maxDeposit()` function has been integrated to the ERC4626 pool deposit flow.

BRAV-14	Georestrictions On Some Supported Protocols
Asset	`actions/nexus-mutual/BuyCover.sol, actions/strike/*, actions/spark/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Some of the supported protocols that Brava offers are not available globally. For example, Nexus Mutual insurance is not available to users located in Korea, Mexico, India, China, Japan, Russia and other countries. Likewise, Spark Finance products are not available to users located in USA or the UK. Other protocols may also have geographic restrictions or introduce them in the future.

While Nexus Mutual makes use of a whitelist to ensure only compliant users can access their products, other protocols do not use such stringent measures. Due to the use of Safe wallets and shifting regulatory environments, it may be possible for users to interact with protocols that then later restrict access for legal reasons.

Recommendations

Brava should make end users aware that they need to check protocols they intend to use for geo-restrictions, as any misuse may lead to their assets being locked.

Resolution

The Brava team have acknowledged this issue and will include warnings to the end user in their frontend client.

BRAV-15	Some Roles Cannot Be Granted To New Users
Asset	`auth/AdminVault.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The roles `ROLE_CANCELER_ROLE` , `ROLE_DISPOSER_ROLE` , `ROLE_EXECUTOR_ROLE` and `ROLE_PROPOSER_ROLE` do not have an admin and, as such, cannot be granted to new users.

Recommendations

Add an admin for these roles, so they can be granted to new users.

Resolution

Some roles have been removed to simplify the access control.

BRAV-16	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Stale Or Redundant Code**
   Related Asset(s): Multiple files
   Some parts of the codebase were noted as not being used or redundant (note, the list below is non-exhaustive):

   - In `actions/actionBase.sol`, the check on line [**91**] has already been checked in the `getLastFeeTimestamp()` call the line beforehand.
   - In `action/nexus-mutual/BuyCover.sol`, `COVER_NFT` is never used.
   - In `action/nexus-mutual/BuyCover.sol`, the function `_assetIdToTokenAddress()` has an `if` clause for when `_assetId` is zero. However, this will never be triggered as the only use of `_assetIdToTokenAddress()` is in the `_buyCover()` function which has a different execution path for when `_assetId` is zero.
   - In `auth/AccessControlDelayed.sol`, the function `_checkProposalWaitTime()` is never used.
   - In `auth/AdminVault.sol`, the `lastFeeTimestamp` mapping is accessed twice in the same function on line [**332**] and line [**334**] rather than being stored in a local variable.

   Remove stale or redundant code where possible to reduce code complexity.

2. **Typos And Inaccurate Comments**
   Related Asset(s): Multiple files
   There are some typos in some files and some NatSpec comments are incorrect (note, the list below is non-exhaustive):

   - The action contract `UwULendWithdraw` is incorrectly named `UsULendWithdraw.sol`.

   A non-exhaustive list of incorrect comments is:

   - The comment for `AdminVault.setFeeTimestamp()` states this function must only be called when the user has zero balance but this function is used when the user has a non-zero balance as part of normal functioning.
   - `SendToken._sendToken()` incorrectly states the function can be used to send ETH.
   - `SendToken` contains a field `to` that has the NatSpec comment *"Where the tokens are sent"* but the restriction on line [**30**] in `SendToken.executeAction()` enforces this to only be the Safe wallet owner.
   - The NatSpec comment for `PullToken._pullToken()` incorrectly states it converts amounts of `type(uint).max` to the user's full token balance amount.

   These comments and typos should be corrected to aid future development.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have resolved this issues where necessary.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 7 tests for test/tests-fork/AccessControlDelayed.t.sol:AccessControlDelayedTest
[PASS] test_cancelRoleProposal() (gas: 50131)
[PASS] test_changeDelay_case1() (gas: 38930)
[PASS] test_changeDelay_case2() (gas: 76374)
[PASS] test_constructor() (gas: 12032)
[PASS] test_grantRole() (gas: 73041)
[PASS] test_proposeRole() (gas: 59916)
[PASS] test_revokeRole_vuln() (gas: 26038)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 1.10s (3.58ms CPU time)

Ran 3 tests for test/tests-fork/FluidTest.t.sol:FluidTest
[PASS] test_action_constructors() (gas: 16643)
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 363985)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 446486)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.12s (22.21ms CPU time)

Ran 5 tests for test/tests-fork/SparkTest.t.sol:SparkTest
[FAIL. Reason: revert: delegatecall from Smart wallet failed] test_SendTokenETH_Vuln() (gas: 122610)
[PASS] test_action_constructors() (gas: 16643)
[FAIL. Reason: User has removed sDAI deposit from Smart Wallet: 0 != 178342486371669261227] test_exitVaultBypass_Vuln() (gas:
      ↪  581011)
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 396728)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 481746)
Suite result: FAILED. 3 passed; 2 failed; 0 skipped; finished in 1.12s (25.26ms CPU time)

Ran 3 tests for test/tests-fork/StrikeTest.t.sol:StrikeTest
[PASS] test_action_constructors() (gas: 16621)
[PASS] test_compoundSupply_through_sequenceExecutor() (gas: 420292)
[PASS] test_compoundWithdraw_through_sequenceExecutor() (gas: 515304)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.12s (26.48ms CPU time)

Ran 5 tests for test/tests-fork/UtilsTest.t.sol:UtilsTest
[FAIL. Reason: User has removed sDAI deposit from Smart Wallet: 0 != 178271827240842639012] test_SendTokenVaultBypass_Vuln() (gas:
      ↪  777538)
[PASS] test_pullTokens() (gas: 488172)
[PASS] test_pullTokens_fails_without_approval() (gas: 467481)
[PASS] test_sendTokens() (gas: 485351)
[PASS] test_sendTokens_fails_on_insufficient_balance() (gas: 466077)
Suite result: FAILED. 4 passed; 1 failed; 0 skipped; finished in 1.12s (24.10ms CPU time)

Ran 15 tests for test/tests-fork/AdminVaultTest.t.sol:AdminVautTest
[PASS] test_addAction() (gas: 89544)
[PASS] test_addPool() (gas: 116350)
[FAIL. Reason:  shouldn't be able to add, so it should be address(0): 0xc7183455a4C133Ae270771860664b6B7ec320bB1 !=
      ↪  0x0000000000000000000000000000000000000000] test_addingPool_afterRemoving_without_proposal_vuln() (gas: 147943)
[PASS] test_cancelActionProposal() (gas: 50348)
[PASS] test_cancelFeeConfigProposal() (gas: 102071)
[PASS] test_cancelPoolProposal() (gas: 52806)
[PASS] test_constructor() (gas: 21911)
[PASS] test_proposeAction() (gas: 60373)
[PASS] test_proposeFeeConfig() (gas: 124521)
[PASS] test_proposePool() (gas: 62305)
[PASS] test_removeAction() (gas: 76438)
[PASS] test_removePool() (gas: 103554)
[PASS] test_setFeeConfig() (gas: 133016)
[PASS] test_setFeeTimestamp() (gas: 148931)
[FAIL. Reason: User should not be able to update feetimestamp directly: 1732887767 != 1732974167] test_setFeeTimestamp_Vuln() (gas:
      ↪  1274939)
Suite result: FAILED. 13 passed; 2 failed; 0 skipped; finished in 1.12s (27.78ms CPU time)

Ran 3 tests for test/tests-fork/MorphoTest.t.sol:MorphoTest
[PASS] test_action_constructors() (gas: 16643)
```

```
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 457582)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 595867)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.12s (29.39ms CPU time)

Ran 5 tests for test/tests-fork/AaveV3Test.t.sol:AaveV3Test
[PASS] test_action_constructors() (gas: 14330)
[PASS] test_supply_through_SequenceExecutor() (gas: 667834)
[PASS] test_supply_through_executeAction() (gas: 409343)
[PASS] test_withdraw_through_SequenceExecutor() (gas: 1018941)
[PASS] test_withdraw_through_executeAction() (gas: 785350)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 1.13s (65.75ms CPU time)

Ran 3 tests for test/tests-fork/ClearpoolTest.t.sol:ClearpoolTest
[PASS] test_action_constructors() (gas: 16643)
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 517282)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 632230)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.13s (31.01ms CPU time)

Ran 8 tests for test/tests-fork/AcrossTest.t.sol:AcrossTest
[PASS] test_action_constructors() (gas: 19306)
[FAIL. Reason: revert: delegatecall from Smart wallet failed] test_feeCollectionRevertsSupplyWithSlippage_Vuln() (gas: 789125)
[FAIL. Reason: revert: delegatecall from Smart wallet failed] test_feeCollectionRevertsSupply_Vuln() (gas: 776310)
[FAIL. Reason: revert: delegatecall from Smart wallet failed] test_feeCollectionRevertsWithdrawAllShares_Vuln() (gas: 654309)
[PASS] test_supplyToVault_through_executeAction() (gas: 324561)
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 336411)
[PASS] test_withdrawFromVault_through_executeAction() (gas: 417742)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 424378)
Suite result: FAILED. 5 passed; 3 failed; 0 skipped; finished in 1.13s (37.63ms CPU time)

Ran 7 tests for test/tests-fork/AaveV2Test.t.sol:AaveV2Test
[FAIL. Reason: revert: delegatecall from Smart wallet failed] test_Supply_Vuln() (gas: 1191870)
[FAIL. Reason: User has redeemed original token deposit without paying fees: 10600000000000000000 != 0] test_Withdraw_Vuln() (gas:
      ↪  1120328)
[PASS] test_action_constructors() (gas: 14330)
[PASS] test_supply_through_SequenceExecutor() (gas: 733296)
[PASS] test_supply_through_executeAction() (gas: 467835)
[PASS] test_withdraw_through_SequenceExecutor() (gas: 1091946)
[PASS] test_withdraw_through_executeAction() (gas: 858355)
Suite result: FAILED. 5 passed; 2 failed; 0 skipped; finished in 1.13s (61.83ms CPU time)

Ran 3 tests for test/tests-fork/YearnTest.t .sol:YearnTest
[PASS] test_action_constructors() (gas: 16643)
[PASS] test_supplyToVault_through_sequenceExecutor() (gas: 350721)
[PASS] test_withdrawFromVault_through_sequenceExecutor() (gas: 440261)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 2.19s (1.09s CPU time)

Ran 5 tests for test/tests-fork/UwuLendTest.t.sol:UwuLendTest
[PASS] test_action_constructors() (gas: 14330)
[PASS] test_supply_through_SequenceExecutor() (gas: 855023)
[PASS] test_supply_through_executeAction() (gas: 589870)
[PASS] test_withdraw_through_SequenceExecutor() (gas: 1175668)
[PASS] test_withdraw_through_executeAction() (gas: 942077)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 63.11s (58.16ms CPU time)

Ran 4 tests for test/tests-fork/NexusMutualTest.t.sol:NexusMutualTest
[PASS] test_action_constructors() (gas: 9719)
[PASS] test_buyCover_through_sequenceExecutor_DAI() (gas: 1213111)
[PASS] test_buyCover_through_sequenceExecutor_ETH() (gas: 1510874)
[FAIL. Reason: User's Safe wallet should return excess ETH to user: 1996840038687925384 != 1000000000000000000]
      ↪  test_buyCover_through_sequenceExecutor_ETH_Vuln() (gas: 1066858)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 63.11s (172.48s CPU time)

Ran 14 test suites in 63.13s (140.75s CPU time): 65 tests passed, 11 failed, 0 skipped (76 total tests)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium

Likelihood

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].