PUBLIC

Limited Review

of the Protocol Smart Contracts

March 18, 2025

Produced for

FUME

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Fume with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Protocol according to Scope to support you in forming an opinion on their security risks.

Fume implements an on-chain administration platform for funds. Investments into such funds can be handled either off-chain or on-chain.

The most critical subjects covered in our audit are functional correctness, access control, precision of arithmetic operations.

Access control is working as expected.

Several issues with the correct handling of investment stages have been found in previous versions of this report. Furthermore, depending on the chosen base currency of a fund, the contracts were exposed to a certain degree of Precision loss. These problems have been mitigated.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	12
• Code Corrected	12
Low-Severity Findings	9
• Code Corrected	7
• Risk Accepted	1
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Protocol repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	20 November 2024	82ee4770aa4742465fd204a35a0120f458040dce	Initial Version
2	15 December 2024	5389e6404f9b0505cb09314c138ec74eb0e1a75f	After 1st Report
3	10 February 2025	0c1a6c2052c9becfea6b24023644be9701608b9c	After 2nd Report
4	27 February 2025	d3184fc284f24b025c98efb393cfd746c1e402cc	After 3rd Report

The following files were in scope:

- contracts/FumeStorage.sol
- contracts/FumeStorageConstructionNoEvents.sol
- contracts/FumeRegistrarConstructionWithEvents.sol
- contracts/SuperFundManager.sol
- contracts/OffChainTA.sol
- contracts/FumeRegistrarConstructionNoEvents.sol
- contracts/FumeStorageConstructionWithEvents.sol
- contracts/NAVCalculator.sol
- contracts/OnchainTA.sol
- contracts/FumeRegistrar.sol
- contracts/FundRegistry.sol
- contracts/FumeBeacon.sol

For the solidity smart contracts, the compiler version `0.8.28` was chosen.

2.1.1 Excluded from scope

Generally, any parts of the codebase not mentioned above were out-of-scope. This includes third party libraries.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Fume offers a fund administration platform that can be used with on- as well as off-chain investments. For each fund, a set of contracts based on the requirements of the fund is deployed. Fund managers can onboard investors off-chain and allow them to either deposit funds in the contracts or send funds off-chain which are then accounted for by the manager. On-chain funds are settled on the platform while off-chain funds are settled directly between the parties. Off-chain settlements are accounted via events in the contracts. The holdings are updated in discrete time intervals on the contracts.

The deployment for each individual fund consists of two storage contracts (`FumeRegistrar` and `FumeStorage`) containing default values and all necessary state for accounting, one or more *TA* contracts that handle user interactions (either directly or through the fund manager), the `NAVCalculator` that is responsible for calculating the current value of the fund and processing investments and the `FumeBeacon` that acts as a repository for the other contracts.

The contracts are not behind proxies. To maintain upgradeability, new contracts can be deployed and set in the `FumeBeacon` (which is the only contract owned by Fume). If state contracts have to be upgraded, certain child contracts for `FumeRegistrar` and `FumeStorage` exist that allow to setup the state in new deployments. This mechanism can also be used to create funds that already have a history outside of the Fume ecosystem.

2.2.1 FumeBeacon

`FumeBeacon` holds the addresses of all associated contracts. Funds can choose to deploy either `OnchainTA`, `OffchainTA` or another, unspecified `TA` contract. They can also choose to us any combination of them. Furthermore, funds can choose to use a `SuperFundManager`.

Funds can be closed through the function `closeDownFund()` by Fume at any time. All current investments are converted to redemptions which are then processed by the fund manager.

2.2.2 FumeRegistrar

`FumeRegistrar` holds the state of all investments, as well as the KYC status of individual investors and the share amounts that belong to investors/investments.

For each investment, a different set of shares is created as an ERC-6909 token. Shares can be transferred to other investors which splits investments (and their corresponding shares) apart (by creating a new set of shares and a new investment).

Share transfers are not activated in all funds. Furthermore, share transfers can be paused by the fund managers.

If this contract should be deployed as an upgrade to an existing registrar, `FumeRegistrarConstructionNoEvents` can be used to set up initial state such as share balances and investments.

If this contract should be deployed for a fund that already has a history, `FumeRegistrarConstructionWithEvents` can be used. It allows to emit events for past activity (which is not strictly required for upgrades as the old contract's events are still on the blockchain).

2.2.3 FumeStorage

`FumeStorage` holds configuration parameters like management and performance fees as well as the current NAV (equity per share) and different arrays that represent the different stages of investments. During its lifecycle, an investment can go through the following stages:

- **Whitelisted**: The fund manager created a new investment with a pre-defined amount and whitelisted it.
- **ReceivedCapital**: Users transferred the given amount of tokens to a whitelisted investment using `OnchainTA`.

- **ReceivedOffchain**: Users transferred the given amount to the fund manager which then updated the users' investments using `OffchainTA`.
- **Liberated**: Funds of investments in **ReceivedCapital** state have been released to the fund manager before subscription.
- **Subscribed**: Investments have been subscribed to the fund and are accruing interest.
- **RedemptionRequested**: Users have requested the redemption of their **Subscribed** investments. These investments are still part of the fund.
- **RedemptionPending**: Investments in **RedemptionRequested** state have been successfully processed and their final withdrawal value has been determined.
- **Redeemed**: Users have successfully received their funds.
- **Default**: Investments have been rejected in either **Whitelisted** or **ReceivedCapital** stages.

Similar to `FumeRegistrar`, two child contracts exist for `FumeStorage` that allow to initialize the state of the contract: `FumeStorageConstructionNoEvents` and `FumeStorageConstructionWithEvents`.

2.2.4 OnchainTA

`OnchainTA` is used for handling investments that are settled on-chain in underlying tokens (base currency) of the fund. This is the only contract that exposes functions users can interact with:

- `subscribe()`: After the fund manager has created a whitelisted investment for a given investor, the investor can call this function to deposit funds and move their investment to the **ReceivedCapital** stage.
- `redeem()`: Investments in **Subscribed** stage can be requested to be redeemed with this function. The investments are moved to the **RedemptionRequested** stage. After the next NAV calculation, these investments can be redeemed. This function can be paused through the `pause()` function in `FumeRegistrar`.

The fund manager interacts with the following functions:

- `whitelist()` / `unWhitelist()`: Creates / removes a whitelisted investment with a pre-defined amount that can then be deposited by the respective investor. Additionally sets the KYC status of the investor to `true`.
- `addKYC()` / `removeKYC()`: While whitelisting creates investments, their investors hold a KYC status on their own. These states can be added / removed at any point in time. Investments of investors without KYC status are no longer part of the fund until their KYC status returns.
- `sendBackOneSubscription()`: Investments in **ReceivedCapital** stage can be rejected and the supplied funds sent back to the investor using this function.
- `liberateSubscriptions()`: Supplied funds of investments in **ReceivedCapital** stage can be sent to the fund manager before the investments are subscribed.
- `settleLiabilitiesByID()`: Funds of investments in **RedemptionPending** stage are sent to the individual investors.
- `addLiability()`: Emits a `Liability` event in case the fund managers must add an extra liability to their books.

2.2.5 OffchainTA

For funds that are settled off-chain (e.g., bank transfers), fund managers can manage the investments on behalf of their investors using `OffchainTA`:

- `whitelist()`: Same as in `OffchainTA` in case the fund does not use the `OnchainTA` module. An `unWhitelist()` function is not present.

- `moveInvestmentToStageTwo()`: Moves an investment to **ReceivedCapital** stage after the funds have arrived.
- `moveInvestmentToStageFour()`: Moves an investment to **RedemptionRequested** stage after a user has requested a redemption from the fund manager.
- `moveToStageSix()`: Moves an investment to **Redeemed** stage after the fund manager has sent the respective funds to the investor.
- `sentBackSubscription()`: Called after an investment in **ReceivedCapital** stage has been sent to the respective investor and therefore been rejected.
- `addLiability()` / `settleLiability()`: Emits `Liability` / `LiabilityPaid` events for correct accounting.

2.2.6 NAVCalculator

`NavCalculator` lets fund managers open a fund by calling `openFund()`. After that, functions are enabled on all contracts and existing investments in **ReceivedCapital**, **ReceivedOffchain** and **Liberated** stages are converted to the **Subscribed** stage.

The fund manager periodically calls `calculateNAV()` with the current equity of the fund. This calculates the equity per share (after taking management and performance fees) and converts all new investments, for which capital has been received, to **Subscribed** stage. Furthermore, it calculates the final withdrawal value for investments in **RedemptionRequested** stage and moves them to **RedemptionPending** stage.

2.2.7 FundRegistry

Registers all `FumeBeacon` contracts that belong to Fume.

2.2.8 SuperFundManager

If deployed, allows transfers between any investments of the fund using `forceShareTransfer()` and direct replacements of the current NAV value using `amendLastNAV()`.

2.2.9 Changes in Version 3

In Version 3 of the smart contracts, partially redeeming an investment now creates a new ID for the redemption request and keeps the old ID for the subscribed investment.

2.2.10 Roles & Trust Model

All contracts in the `FumeBeacon` can be replaced by Fume at any time during the runtime of a fund. Fume is therefore fully trusted.

Fund managers are fully trusted as they have access to all funds (at the bare minimum through the function `recover()` in `OnchainTA`). Funds settled off-chain are also under the full control of the fund managers. Furthermore, the fund managers have to be trusted to report the correct equity during NAV calculation and actually send funds for **RedemptionPending** investments (this is also true for on-chain funds as they cannot be redeemed by the investors themselves).

Funds using a `SuperFundManager` can transfer shares of individual investors and change the NAV to any value after the fact.

Fund deployment is performed by Fume opaquely. Furthermore, the state of deployments with `setupMode` depends on the correct initialization by the fund managers.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

5.1 Redemption Denial of Service

Security Low Version 1 Risk Accepted

CS-FMP-022

Redemption requests are capped by the constant `_MAX_INVESTORS_TRANSFERRING`: It is not possible that more than `_MAX_INVESTORS_TRANSFERRING` redemption requests exist at the same time. While actual redemptions decrease the amount of redemption requests so that new requests can be created, such new requests can still be blocked by any active user of the fund. Frontrunning the creation of such a request (or backrunning a redemption) with a partial share transfer (which also increases the amount transferring investors) results in the limit being reached again, blocking other users from creating redemption requests.

Risk accepted:

Fume accepts the risk with the following statement:

> Wallets need to be whitelisted for transfers, and bad actors can be suspended to prevent them from acting again.

5.2 Uneven Redemptions

Correctness Low Version 1 Acknowledged

CS-FMP-019

Users can partially redeem their investments. When a redemption is requested, the amount of shares that is redeemed is stored in `amountSharesToBeRedeemed`. Redemption requests are then processed at the end of a NAV calculation.

Since the NAV calculation can decrease the amount of shares a user holds but `amountSharesToBeRedeemed` is not updated accordingly, the redemption possibly yields a larger chunk than the user originally anticipated. Consider the following example:

1. A user holds 1000 shares.
2. The user requests a redemption of 50% of their investment (i.e., 500 shares).
3. NAV is calculated and decreases the user's shares to 800 to accommodate for their share of the performance fees.
4. The redemption of 500 shares is processed and yields an amount that is 62.5% of their total investment.

Acknowledged:

Fume acknowledges the behavior and gives the following statement:

> That is correct, and expected behaviour. The redemption are always specified in number of shares or dollars amount (we don’t support this yet). Redeeming 50% of an investment is not something investors can request (at least in those terms).

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Denial of Service Code Corrected

Medium-Severity Findings	12

- Investment Denial of Service Code Corrected
- Investment Not Transferred Code Corrected
- Funds Can Be Closed Down Multiple Times Code Corrected
- Incorrect Accounting Code Corrected
- Multiple Settlements Possible Code Corrected
- Performance Fees of un-KYCed Accounts Code Corrected
- Precision Loss Code Corrected
- Send Back Possible After Liberation Code Corrected
- Sent Back Subscription Not Fully Removed Code Corrected
- Transfer of un-KYCed Accounts Code Corrected
- Unchecked Investment Transfer Code Corrected
- Unvalidated Input Code Corrected

Low-Severity Findings	7

- Division Before Multiplication Code Corrected
- Investment Amount Inflation Code Corrected
- Missing Checks Code Corrected
- Missing Events Code Corrected
- Missing Share Initialization Code Corrected
- Off-chain Investments Deleted Code Corrected
- transferFrom() Limited by Msg.Sender Balance Code Corrected

6.1 Denial of Service

Security High Version 1 Code Corrected

CS-FMP-001

NAV calculation relies on the iteration over unbounded arrays for investments in different stages. In particular, it iterates over all currently subscribed investments and performs some rather gas intensive logic on each iteration.

Users are able to create a high amount of additional subscribed investments by simply calling `FumeRegistrar.transfer()` with an amount of 1 wei shares to themselves. The amount of created investments is only limited by the amount of wei of shares they hold and the amount of gas they are willing to spend.

Creating enough such investments can cause the NAV calculation to use more gas than available in a block and thus the whole fund to be permanently DoSed.

Code corrected:

Each fund now has a maximum of 1000 investments that can be created. It is thus impossible to create enough investments to reach the gas limit.

6.2 Investment Denial of Service

Security | Medium | Version 3 | Code Corrected

CS-FMP-015

Each time a new investment is created (via whitelisting, partial redemption or partial transfer), the `_nextInvestmentId` in the `FumeRegistrar` contract is incremented by one. The smart contracts define a constant `_MAX_INVESTORS` that limits the amount of such increments to 1000. This means that no new investments can be added, nor can partial transfers or partial redemptions be performed, after the `_nextInvestmentId` reached this threshold.

Since users can arbitrarily increment `_nextInvestmentId` via transfers and redemption requests, the limit can easily be reached, blocking the aforementioned functionality permanently.

Code corrected:

The `_MAX_INVESTORS` limit is now enforced against `_subscribedInvestments`. Once the limit is reached, it can be deflated by investments being moved to the redemption phase.

For pending redemptions, there now also exists a limit `_MAX_INVESTORS_TRANSFERRING` that can in turn be deflated by users redeeming legit requests. The same limit is also in place for investments that have received capital.

While these changes ensure that DoSed contracts can be brought back into a state that allows further processing, the DoS vector is still available as redemptions can be backrun with partial transfers. See Redemption Denial of Service for details.

6.3 Investment Not Transferred

Correctness | Medium | Version 3 | Code Corrected

CS-FMP-018

`FumeRegistrar.transfer()` and `transferFrom()` copy the investment data of a given ID to the local memory variable `localInvestment`. On full transfers, shares are then sent to the new recipient but the `investor` of the investment is only changed in the local copy and not in storage:

```
Investment memory localInvestment = investments[id];
...
if (balanceOf[msg.sender][id] == amount) {
    localInvestment.investor = receiver;
```

```
    ...
}
```

The investment is thus never given to the receiver. To redeem an investment using `OnchainTA`, however, the `msg.sender` must be equal to the investment's `investor`. The receiver will therefore not be able to redeem the received shares.

Code corrected:

The code now updates the correct storage variable.

6.4 Funds Can Be Closed Down Multiple Times

Correctness | Medium | Version 1 | Code Corrected

CS-FMP-002

`FumeBeacon.closeDownFund()` converts all investments to pending redemptions and closes the fund. Investments are pushed into the `_investmentsRedemptionPending` array but are not removed from the respective arrays of their current stage. Furthermore, the function has the modifier `onlyOpen` which only checks that the fund has been opened before but not that it was closed already. This allows the function to be run again (e.g., by accident).

Since, in the first call, state arrays have not been correctly updated and the function also does not check the current stage of investments, the second call will create another set of liabilities for all liberated investments (for subscribed investments, shares are burnt in the first call which leads to 0-value liabilities in the second call).

Code corrected:

`closeDownFund()` now reverts if the fund has been closed before.

6.5 Incorrect Accounting

Correctness | Medium | Version 1 | Code Corrected

CS-FMP-003

`NAVCalculator._equalize()` calculates the current NAV based on the current equity (minus management fees) of the fund and the total outstanding shares:

```
nav = ((equityBeforeCommissions_ * 100000000 - mngmtFee * 100000000 - fumeFeeManagement * 100000000)
    * (10 ** fundDecimals)) / (registrar.totalOutstandingShares() * 100000000);
```

Performance fees are then taken for each individual investment based on its high watermark. Since the NAV is based on the equity without the performance fees removed, the individual shares of the investments have to be adapted accordingly so that the following equation is true:

$$equity - managementfees - performancefees = totaloutstandingshares * NAV$$

To achieve this, performance fees of the first investment (investment with the lowest index in either `_investmentsSubscribed` or `_investmentsRedemptionRequested`) are calculated and then subtracted from the NAV:

```
nav =
    (
        (equityBeforeCommissions_ * 1000000000 - mngmtFee * 1000000000 - fumeFeeManagement
            * 1000000000 - (perfFee + fumeFeePerformance) * 1000000000) * (10 ** fundDecimals))
        / (registrar.totalOutstandingShares()
        * 1000000000
    );
```

The performance fees for the rest of the investments are then calculated based on this adjusted NAV value. If the high watermark of these investments is lower than the high watermark of the first investment, their shares are reduced:

```
uint256 localOldShares = registrar.balanceOf(localInvestor, localId);
uint256 localNewShares = (localOldShares *
    (nav * _BPS_TO_PERCENTAGE * 1000000000 -
        ((nav * 1000000000 - localHighWaterMark * 1000000000) * storage_.performanceFee())))
    / (nav * _BPS_TO_PERCENTAGE * 1000000000);
registrar.burn(localInvestor, localId, localOldShares - localNewShares);
```

This approach has several flaws:

1. The performance fee of the first investment is based on a different NAV than the performance fee of all other investments. This results in a calculation of less performance fees than expected.
2. The shares are not updated correctly in all circumstances. Consider the following example:
    - The fund contains two investments.
    - Both investments have a high watermark of 1.
    - Since investment 2 has the same high watermark as investment 1, no shares are reduced at all.
    - Since the NAV has only been reduced by the performance fees of the first investment and the total supply hasn't changed, the equation mentioned above does not hold anymore: The total value of the fund is inflated.
3. The shares of the first investment are never adjusted. If we assume that the shares of investment 2 in the previous example are reduced according to the presented code, the ratios between the two investments would change: Investment 1 now holds a larger part of the share since its shares are not reduced but still based on an NAV that includes the performance fees of investment 2 (since the equity part of the NAV has only been reduced by the performance fees of investment 1 at this point). Investment 2 therefore pays some of the fees of investment 1.
4. The share reduction only accounts for the manager performance fees but not for the Fume performance fees.

Code corrected:

Performance fees and high watermarks of all investments starting from the second investment are now based on the regular NAV from which the performance fee of the first investment has not been removed yet. Furthermore, the final NAV is calculated by removing all performance fees from the equity and dividing by the total amount of shares that has been reduced by burns of shares in different series.

The Fume performance fee is now also taken into account when shares are burnt.

6.6 Multiple Settlements Possible

Correctness | Medium | Version 1 | Code Corrected

CS-FMP-004

`OnchainTA.settleLiabilitiesByID()` does not check the current stage of an investment. If the passed `investmentsIDs` contain investments in the wrong stage (e.g., by accident), the investments are marked as redeemed and a liability paid event is created even though there might be no corresponding liability event. In particular, it is also possible that an investment is redeemed multiple times, leading to inconsistent state.

The inner call to `FumeStorage.removeRedemptionPendingInvestment()` also does not revert in case an ID is not found inside the respective array.

The same is true for `OffchainTA.moveToStageSix()`.

Code corrected:

Both functions now check that the chosen investment is in the correct stage.

6.7 Performance Fees of un-KYCed Accounts

Correctness | Medium | Version 1 | Code Corrected

CS-FMP-013

`NAVCalculator._equalize()` neither calculates performance fees nor updates the shares of investments that belong to an account without KYC. If the principal (or current equity) of such investments is still part of the fund's equity, then fees should be calculated and shares adapted. If it is not part of the equity, then the equity would be reduced and therefore the shares of such investments should no longer be reflected in the total outstanding shares.

Code corrected:

Investments are no longer skipped during NAV calculation if the associated investor has lost their KYC status.

6.8 Precision Loss

Correctness | Medium | Version 1 | Code Corrected

CS-FMP-005

The precision of mathematical operations involving NAV and shares is dependent on the decimals of the underlying base currency as well as the initial NAV. The project's deployment scripts specify the value for the `INITIALNAV` to be:

```
100 * 10 ** baseCurrencyDecimals
```

This initial value defines how shares and NAV are calculated. Consider the following example using GUSD with two decimals:

1. A fund is deployed with no fees.
2. The `INITIALNAV` is set to 100 * 10 ** 2 = 10,000.
3. An investment of 999.99 GUSD (i.e., 99,999 wei GUSD) is subscribed.
4. The fund is opened, awarding 999 shares to the investor. 0.99 GUSD were lost.

The higher the initial NAV, the higher the precision loss in share calculation.

Now let's consider a lower initial NAV value in the same example:

1. A fund is deployed with no fees.
2. The `INITIALNAV` is set to 1 * 10 ** 2 = 100.
3. An investment of 1,000 GUSD (i.e., 100,000 wei GUSD) is subscribed.
4. The fund is opened, awarding 100,000 shares to the investor.
5. After some time, the equity of the fund has gone down and the new NAV is calculated with an equity of 999.99 GUSD. The new NAV is set to 99. 9.99 GUSD were lost.

The lower the initial NAV, the higher the precision loss in NAV calculation.

If the underlying token has higher decimals (e.g., USDC), the same rounding errors are introduced but with lower impact. In the first example, the investor would lose 0.000099 USDC, in the second example, the fund would lose 0.000999 USDC.

Code corrected:

The contracts now use a fixed denomination of 18 decimals for shares/NAV.

6.9 Send Back Possible After Liberation

Correctness **Medium** **Version 1** **Code Corrected**

CS-FMP-006

`OnchainTA.sendBackOneSubscription()` can be used by the fund manager reject an investment and send the already transferred principal back to the investor.

When an investment is liberated by the fund manager, it is moved to a different array (`_investmentsLiberated`) but its state stays the same (`ReceivedCapital`). `sendBackOneSubscription()` only checks for the state of an investment but not for the current array it resides in. This makes it possible to call the function on an investment that has already been liberated (and thus the funds are no longer on the contract), sending funds that belong to other investments to the given investor.

Furthermore, depending on what `indexInStateArray` is set to, another un-associated investment might get removed from the `_investmentsReceivedCapital` array, leading to inconsistent state.

Code corrected:

The function now checks if a given investment is in the appropriate stage (array).

6.10 Sent Back Subscription Not Fully Removed

Correctness **Medium** **Version 1** **Code Corrected**

CS-FMP-007

`OffchainTA.sentBackSubscription()` sets an investment in `ReceivedCapital` state to `Default` (i.e., removed) state. However, the function does not remove the investment from the `_investmentsReceivedOffchain` array. The investment, therefore, is still processed in the next NAV calculation and thus decreases the NAV of the fund as shares are emitted while the associated equity is no longer part of the fund's equity.

Code corrected:

The function now removes the investment from the respective array.

6.11 Transfer of un-KYCed Accounts

Correctness Medium Version 1 Code Corrected

CS-FMP-008

`FumeRegistrar.approve()` does not perform any KYC checks on the account that is giving the approval. Furthermore, `transferFrom()` only performs such KYC checks on the `msg.sender` and the receiver. An account that has lost their KYC status can approve another, KYCed account. This account can then transfer their shares to their own account.

In contrast, the `transfer()` function does not allow an account without active KYC to transfer their shares to another account.

Code corrected:

KYC checks are now also performed on the `sender` account.

6.12 Unchecked Investment Transfer

Correctness Medium Version 1 Code Corrected

CS-FMP-009

`FumeRegistrar.transferFrom()` can be called by any KYCed account on any other account with an amount of 0. In this case, the function does not revert in any of the given checks, particularly:

- The allowance is only checked to be smaller than the amount.
- The balance of the `msg.sender` is only checked to be smaller than the amount.

If the balance of the `msg.sender` for the share ID of the `sender` is equal to the amount (by default true as the `msg.sender` does not hold any such shares), the `investor` for the given investment is set to the `receiver` specified by the `msg.sender`.

While no shares are transferred, the original owner is no longer capable of redeeming the shares themselves. Furthermore, if the owner would request the fund manager for a redemption through `OffchainTA`, the submitted `LiabilityPaid` event would contain the wrong investor account.

Code corrected:

Transfers of 0 amount now return early, creating no storage changes. Furthermore, full investment transfers are now performed based on the balance of the `sender` instead of the `msg.sender`.

6.13 Unvalidated Input

Security Medium Version 1 Code Corrected

CS-FMP-010

`OnchainTA.subscribe()` and `redeem()` have an argument `indexInStateArray` that is not validated. Since these two functions can be called by untrusted or partially trusted users (in case the fund performed a real KYC off-chain), this is problematic:

For example, a user calling `redeem()` with the wrong index could remove another, un-associated investment from the `_investmentsSubscribed` array leading to wrong accounting during NAV calculation.

Please note that some of the functions that are only accessible by the fund manager are not validating this argument as well which could also potentially lead to inconsistent state due to manual error or errors in the respective off-chain applications.

Code corrected:

All functions now validate the `indexInStateArray` argument (if in place).

6.14 Division Before Multiplication

Correctness | Low | Version 1 | Code Corrected

CS-FMP-011

`NAVCalculator._computeMngmtFee()` computes the management fee in the following way:

```
uint256 part1 = (equityBeforeCommissions_ * timeCoefficient) /
    (12 * _BPS_TO_PERCENTAGE);
uint256 part2 = (((daysInCurrentMonth * 1000000000) /
    fullDaysInCurrentMonth +
    (daysInLastMonth * 1000000000) /
    fullDaysInLastMonth) *
    equityBeforeCommissions_) / (12 * _BPS_TO_PERCENTAGE * 1000000000);
mngmtFee = (part1 + part2) * storage_.managementFee();
```

Both parts are divided by `_BPS_TO_PERCENTAGE` before being multiplied with `storage_.managementFee()`. This can introduce rounding errors that would not be present if the multiplication were to be performed before the division.

Code corrected:

The code has been changed so that the multiplication is performed first:

```
uint256 part1 = equityBeforeCommissions_ * timeCoefficient;
uint256 part2 = (((daysInCurrentMonth * _PRECISION_INCREASE) /
    fullDaysInCurrentMonth +
    (daysInLastMonth * _PRECISION_INCREASE) /
    fullDaysInLastMonth) *
    equityBeforeCommissions_) / _PRECISION_INCREASE;

mngmtFee = (part1 + part2) * storage_.managementFee() / (12 * _BPS_TO_PERCENTAGE);
```

6.15 Investment Amount Inflation

Correctness | Low | Version 1 | Code Corrected

CS-FMP-012

`FumeRegistrar.transfer()` and `transferFrom()` allow users to transfer investments between each other. The investment amounts (i.e., the original amounts that are used to determine the amount of tokens a user must send during subscription) are not correctly updated in this process.

If a user partially transfers an investment to another user, the amount of the new investment created for the new user is set to the amount of transferred shares. Additionally, the old investment's amount is not reduced by the transferred amount.

Additionally, redemption requests can be created based on these inflated investment amounts (by sending shares back and forth).

Because shares are always transferred correctly and the investment amounts of subscribed investments are no longer used by the rest of the contracts (redemptions are capped by the maximum amount of actual shares a user holds), this is only a problem in case external applications make use of these amounts.

Code corrected:

Amounts of the shares transferred are now correctly converted to underlying before being written to the investment amount field. On partial transfers, the transferred amount is now correctly removed from the old investment.

6.16 Missing Checks

Correctness | Low | Version 1 | Code Corrected

CS-FMP-014

The following parts of the code miss certain checks that could compromise the integrity of the contracts' state:

1. `FumeRegistrar` and `FumeStorage` can be deployed in setup mode even when the parent contract (in contrast to, for example, `FumeRegistrarConstructionNoEvents`) is used. The parent contracts lack finalization methods leaving such deployments permanently unusable.
2. `FumeStorage` checks that entry and exit fee are not higher than 100% in the constructor. However, the sum of the manager fees and the Fume fumes should not be greater than 100% in order to make sure that the contracts do not revert on some actions. Furthermore, management and performance fees have no bounds.
3. `OnchainTA.whitelist()` and `setLockupPeriodForParticularInvestment()` do not check that the lockup period for an investment is smaller than or equal to the lockup period defined in `FumeStorage` (the lockup period in `FumeStorage` is therefore currently not used at all).
4. `NAVCalculator._equalize()` only processes investments if their respective investor is still KYCed. However, this is not true for the first investment in either `_investmentsSubscribed` or `_investmentsRedemptionRequested`.

Code corrected:

All missing checks have been sufficiently implemented or mitigated.

6.17 Missing Events

Design | Low | Version 1 | Code Corrected

CS-FMP-021

The following state-changing functions are missing events:

1. `FumeBeacon.lowerFumeFeeManagement()`.
2. `FumeStorage.setLegalInfo()`.
3. `NAVCalculator.openFund()` (the functions emits events but does not emit an event specific to the opening of the fund).

Code corrected:

The first 2 events have been added. As for the third event, Fume states the following:

> OpenFund is the only way to start a fund, and does emit the first NAV event (with the pre-set initialNAV value). Therefore the first NAV event always indicated when the fund has been started.

6.18 Missing Share Initialization

Correctness | Low | Version 1 | Code Corrected

CS-FMP-016

Partial share transfers with `FumeRegistrar.transfer()` and `transferFrom()` result in a new set of shares being created for the receiver. However, `initializeShares()` is never called to set up the metadata for these shares.

Furthermore, `OnchainTA.redeem()` and `OffchainTA.moveToStageFour()` create new investments on partial redemptions. The shares of these new investments are not initialized either.

Code corrected:

All functions that create new investment IDs now successfully initialize the corresponding shares.

6.19 Off-chain Investments Deleted

Correctness | Low | Version 1 | Code Corrected

CS-FMP-017

`NAVCalculator._liberateAndProcessSubscriptions()` processes off-chain investments and moves them to **Liberated** state before creating subscriptions. Only off-chain investments that have a `joinTimestamp` before the date of the NAV are considered. However, in contrast to the processing of all other investment stages, all off-chain investment are removed from the respective storage array - instead of only the processed ones. This means, that such investments will not be processed in the next NAV calculation and have to be re-added by the fund manager.

Code corrected:

The function now only removes processed investments.

6.20 transferFrom() Limited by Msg.Sender Balance

Correctness | Low | Version 1 | Code Corrected

CS-FMP-020

`FumeRegistrar.transferFrom()` allows a third party to transfer funds on behalf of a user in case the user has given prior approval. The function, however, reverts if the balance of this third party (`msg.sender`) is lower than the transfer amount even when the user's balance and given allowance are higher.

Code corrected:

The transfer amount is now checked against the balance of the `sender`.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Early Subscription

Informational Version 1 Acknowledged

CS-FMP-023

Fund managers can liberate investments at any point in time using `OnchainTA.liberateSubscriptions()`. If a fund manager were to call this function prior to a NAV calculation and with a later date than used in the NAV calculation, investments that joined after the NAV date but before the date used in the function call would still get subscribed as `NAVCalculator._liberateAndProcessSubscriptions()` does not check for the join date of liberated investments.

Acknowledged:

Fume acknowledges the described behavior and gives the following statement:

> That is correct behavior. If the liquidity of an investment is made available to the fund, then this investment is considered subscribed. If an investment is liberated it should always be processed in the next NAV calculation (since its capital is already available). Moreover, the user is expected to provide correct dates when calling these functions (we also have guardrails in the frontend).

7.2 Pure Offchain Funds Cannot Be Closed

Informational Version 1 Acknowledged

CS-FMP-024

`FumeBeacon.closeDownFund()` directly calls a function on `OnchainTA`. If a fund has been deployed without an `OnchainTA` contract, the fund cannot be closed down until one is added via `setOnchainTA()`.

Acknowledged:

Fume acknowledges the described behavior and gives the following statement:

> We foresee almost zero complete off chain funds (we provide little benefit in this case). We might change this function in the future to accommodate for this fact however.

7.3 Uncommon Style Choices

Informational Version 1 Acknowledged

CS-FMP-025

The project's code does not follow best practices on several occasions:

1. Interfaces like `FumeMetadata` usually start with a capital I.

2. Some variables with uppercase-only names can be changed (e.g., `FUMEWALLET`). This naming style is typically chosen for constants only. Furthermore, some variables are treated as constant (e.g., `_SHARESNAME`). Such variables are typically still not named uppercase-only.
3. The `_BPS_TO_PERCENTAGE` constant is defined separately in multiple contracts.
4. Some contracts make use of magic constants.
5. Some functions are defined but never used (e.g., `FumeRegistrar.setInvestorForInvestment()`)
6. Parts of the code are copied instead of extracted out (e.g., the functionality for subscribed investments in `_equalize()` is repeated for redemption-requested investments).
7. During redemption processing, `amountSharesToBeRedeemed` is re-purposed to hold the actual tokens that are redeemed.
8. The naming of functions with similar functionality varies greatly between `OnchainTA` and `OffchainTA`. Furthermore, the function `OffchainTA.moveToStageSix()` is not named in the same style as the other functions in the contract.

Acknowledged:

Fume acknowledges the described points and states that they might be changed in the future.

7.4 Unnecessary Event Emissions

Informational | Version 1 | Acknowledged

CS-FMP-026

Some events are emitted even though their contained data is meaningless:

1. `NAVCalculator._equalize()` emits liability events and their associated payment events with 0-values if the contract contains no shares or the equity is 0.
2. The whole project emits liability events with 0-values if calculated fees in a particular step are 0.

Acknowledged:

Fume acknowledges the described behavior and gives the following statement:

> The fact that 0-value logs are emitted is for accounting purposes, as it is then much simpler to get them as a list rather than having to add 0-value events where they should be if they weren't emitted. Potential improvement however.

7.5 Unused Argument

Informational | Version 1 | Acknowledged

CS-FMP-027

`NAVCalculator.calculateNAV()` takes an argument `assets_`. This argument is only checked to be bigger than or equal to `equityBeforeCommissions_` but not used for any computations or events.

Acknowledged:

Fume acknowledges the described behavior and states that it might be changed in the future.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Base Currency Change Considerations

Note Version 1

`FumeBeacon.setBaseCurrency()` allows Fume to change the base currency of a fund during its runtime. This function should be handled with care to avoid the following pitfalls:

1. The currency should only be changed to another currency with equal value.
2. The currency should not be changed to another currency with different decimals as all stored amounts will be wrong after the change.
3. The base currency should only be changed right after a NAV calculation or liberation (atomically) to ensure that the funds of on-chain investments in **ReceivedCapital** state can be correctly transferred out. Otherwise, it is possible that such investments cannot be processed (as there are not enough underlying tokens to process the transfers when liberating these investments) further which then requires manual action (token recovery and exchange).

8.2 Reentrancy

Note Version 1

Under the following (very specific) circumstances, it is possible for a KYCed user to create inconsistent state:

- The underlying base currency calls the recipient of a transfer (e.g., ERC-777).
- The fund manager is a multi-sig wallet on which an action can be executed without access control once all required signatures are known (e.g., Safe).
- The fund manager creates transactions for `NAVCalculator.calculateNAV()` and `OnchainTA.sendBackOneSubscription()` at the same time.
- The user has an investment in **ReceivedCapital** stage that is getting sent back by the fund manager.

When `sendBackOneSubscription()` is executed, execution context is given to the user in the token's callback before the stage of the user's investment is updated. If another transaction executing `calculateNAV()` has been published at the same time, the user could execute this transaction in the callback by using its signatures that can now be found in the mempool. They could use an MEV service to guarantee that the transactions are executed in the correct order.

By doing this, the user executes `calculateNAV()` on a state that will process their investment even though it is sent back directly after. When the execution context returns to `sendBackOneSubscription()`, their investment is set to `Default` stage and an unrelated investment is potentially removed from then `_investmentsReceivedCapital` array, leading to inconsistent state.

8.3 Trusted Deployment

Note Version 1

Deployment of funds is handled through an off-chain script either run by Fume or through a trusted application. If some actors that cannot be fully trusted gain access to the deployment keys, it cannot be guaranteed that a fund is deployed in a secure manner. For example, such an attacker could theoretically mint unlimited shares by creating a liberated investment before completing the deployment which will then receive newly minted shares after the first NAV calculation.

Many different malicious deployments are possible that might not be immediately obvious. It is therefore advised to run the deployment scripts on sufficiently hardened servers (if publicly available). Fund managers might want to consider performing deployment validation before activating a deployed fund.

8.4 Trusted Setup

Note Version 1

Both storage contracts `FumeRegistrar` and `FumeStorage` can be initialized by the fund manager with arbitrary storage. This can be done with functions that do not necessarily guarantee full consistency of the data. For example, `FumeRegistrarConstructionNoEvents.setupInvestment()` allows to create investments, while another function `FumeStorageConstructionNoEvents.setupInvestments()` allows for the insertion of such investments into the respective arrays in the storage contract. If an investment were to be added to the `_investmentsReceivedCapital` array but no corresponding investment would be created in the registrar, the NAV calculation will revert, leading to permanent DoS after finalization of the setup.

The security of such deployments is therefore dependent on off-chain factors that cannot be evaluated in this review.

8.5 Unsupported Tokens

Note Version 1

The project does not support the following tokens as base currency:

1. Rebasing tokens: Liberation of investments in **ReceivedCapital** stage transfers the amount of tokens that has been deposited at an earlier point in time. Since rebasing tokens could cause the amount of tokens to increase since then, an incorrect amount would be liberated.
2. Fee-taking tokens: Subscription requires users to deposit a pre-defined amount. With fee-taking tokens, the actual amount that reaches the contract's balance would be less than this amount.