Enzyme Audit

OpenZeppelin | security

Introduction

The Enzyme team engaged us to audit the Enzyme protocol. We carefully reviewed the system and our findings are published here.

Scope

We audited the repository avantgardefinance/protocol. The audited commit was c2215678c25aae326eb480488f0dc7cc0aa047d4. The following files were in scope.

- persistent/external-positions/compound-debt/CompoundDebtPositionLibBase1.sol
- persistent/vault/VaultLibBase2.sol
- persistent/external-positions/ExternalPositionFactory.sol
- persistent/external-positions/ExternalPositionProxy.sol
- persistent/protocol-fee-reserve/ProtocolFeeReserveProxy.sol
- persistent/protocol-fee-reserve/ProtocolFeeReserveLib.sol
- persistent/protocol-fee-reserve/bases/ProtocolFeeReserveLibBase1.sol
- persistent/protocol-fee-reserve/bases/ProtocolFeeReserveLibBaseCore.sol
- persistent/protocol-fee-reserve/utils/ProxiableProtocolFeeReserveLib.sol
- release/core/fund/comptroller/ComptrollerLib.sol
- release/core/fund/comptroller/ComptrollerProxy.sol
- release/core/fund/vault/VaultLib.sol
- release/core/fund-deployer/FundDeployer.sol
- release/extensions/external-position-manager/ExternalPositionManager.sol
- release/extensions/fee-manager/FeeManager.sol
- release/extensions/integration-manager/IntegrationManager.sol
- release/extensions/policy-manager/PolicyManager.sol
- release/extensions/utils/ExtensionBase.sol
- release/extensions/utils/PermissionedVaultActionMixin.sol
- release/infrastructure/asset-finality/AssetFinalityResolver.sol
- release/infrastructure/gas-relayer/GasRelayPaymasterFactory.sol
- release/infrastructure/gas-relayer/GasRelayPaymasterLib.sol
- release/infrastructure/gas-relayer/GasRelayRecipientMixin.sol
- release/infrastructure/gas-relayer/bases/GasRelayPaymasterLibBase1.sol
- release/infrastructure/price-feeds/derivatives/AggregatedDerivativePriceFeedMixin.sol
- release/infrastructure/protocol-fees/ProtocolFeeTracker.sol
- release/infrastructure/price-feeds/primitives/ChainlinkPriceFeedMixin.sol
- release/infrastructure/value-interpreter/ValueInterpreter.sol
- release/utils/beacon-proxy/BeaconProxy.sol
- release/utils/beacon-proxy/BeaconProxyFactory.sol
- release/utils/AddressArrayLib.sol
- release/utils/AssetHelpers.sol
- release/utils/FundDeployerOwnerMixin.sol

- release/utils/MakerDaoMath.sol
- release/utils/MathHelpers.sol
- release/utils/NonUpgradableProxy.sol
- release/extensions/external-position-manager/.../CompoundDebtPositionLib.sol
- release/extensions/external-position-manager/.../CompoundDebtPositionParser.sol
- release/extensions/fee-manager/fees/PerformanceFee.sol

All other project files and directories (including tests), along with external dependencies and projects, game theory, and incentive design, were also excluded from the scope of this audit. External code and contract dependencies were assumed to work as documented.

High-level overview of the system

Enzyme is a decentralized on-chain asset management protocol based on the Ethereum blockchain. The protocol offers functionalities for users or entities to setup or invest in different funds. These funds are highly customizable vaults for asset management maintained by trusted managers. Each fund is capable of holding two types of assets: tracked assets and external positions. By investing into a fund, investors will be issued shares representing their investment position and exposure to the fund's holding assets. Shares can be redeemed for the fund's assets proportionally or for specific assets. However, since external positions are not divisible, the value held in external positions will be lost if an investor chooses to redeem shares proportionally.

Project design and maturity

We carefully reviewed the files listed in scope, and we are very pleased with the general impression of the system. The codebase is fairly complicated with carefully designed upgradability systems to serve multiple moving parts and the system is well engineered to avoid vulnerabilities and attack vectors. However, we do think there are ways for improvement in order to make it less complicated and easier to understand while maintain necessary functions and generality.

During the audit, there have been 2 commit changes and several contracts were left behind in order to accommodate the scope to the time frame. Some of these contracts are inherited by in-scope contracts, further bugs may be found in those contracts and in the functionalities implemented by adapters, extensions, and policies that were not reviewed, please refer to the Scope section above for the final audit scope.

Privileged Roles

There are several privileged roles in the system including the system administrators, fund owners, position deployers, and fund managers. These privileged roles are able to perform actions putting the protocol at great risk, for instance:

- stealing funds from the protocol.

- maliciously changing/hiding fund's assets composition.
- update certain critical contract storages.
- alter Funds' policies and bypass safety checks.
- break the correct functionality of the fund.

More details are provided by the Enzyme Team under their Known Risks & Mitigations section in their General Spec document.

Users/investors must fully trust that the entities controlling those roles will always behave correctly and in the best interest of the system and its users/investors. However, the Enzyme team do have a detailed plan to decentralize the administrator of the protocol.

Client-reported findings

The Enzyme team disclosed a vulnerability to us during the audit. The issue is related to Compound debt positions not offering any functions to accrued `COMP` tokens back into the corresponding fund. The team will be adding extra functions to address this issue after our audit.

Critical severity

None.

High severity

[H01] Denial of service on the gas relayer

The protocol allows the usage of a trusted forwarder to relay certain transactions.

In order to activate the usage for a particular fund, the owner of the `ComptrollerProxy` contract calls the `deployGasRelayPaymaster` function which will deploy a new gas relayer linked to the current `VaultProxy` contract and deposit the initial ETH for operation. The `deposit` function from the `GasRelayPaymasterLib` contract is only callable from the `ComptrollerProxy` contract, and its task is to deposit as many ETH as needed to complete the `DEPOSIT` constant of 0.2 ETH.

However, because addresses of deployed contracts are predictable, any user could scan the mempool for the gas relayer deployment and call the `depositFor` function from the `RelayHub` contract on behalf of the future gas relayer contract to send more than the `DEPOSIT` value causing the `SafeMath` `sub` method used to calculate the missing ETH value to revert and the deployment to fail. Since this is not a restricted operation, a malicious user could attack every fund that request a gas relayer deployment.

Furthermore, if a fund already has a gas relayer, a user could also duplicate this attack vector on relayed transactions that have set the `shouldTopUpDeposit` flag during the execution of the `postRelayedCall` hook, preventing the execution of a particular relayed transaction in order to take advantage of it(for instance, a trading opportunity).

Consider validating that the balance of the gas relayer is less than the `DEPOSIT` value before proceeding with the calculation of the remaining value needed.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d, added a check to ensure the balance of the gas relayer is less than DEPOSIT value.*

Medium severity

[M01] Lack of error validation

The `CompoundDebtPositionLib` contract implements the functionalities allowing `VaultProxy` to interact with the Compound Protocol. During a repayment of borrowed assets, the contract accrues interest of the respective `cToken` contract in order to calculate the current borrowed balance.

However, because the `accrueInterest` function from the `cToken` contract returns an error code without reverting the transaction, it might cause the repayment process to continue as usual even the accrue interest process failed on Compound. Consider validating the error code as it was done with other function from Compound.

Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d, a check was added to ensure no error code is returned from Compound when accuring interest.

Low severity

[L01] Constants not declared explicitly

There are several occurrences of literal values with unexplained meaning in the codebase. For example:

- Line 271 of `PolicyManager.sol`.
- Line 509 of `ComptrollerLib.sol`.

Literal values in the code base without an explained meaning make the code harder to read, understand and maintain, thus hindering the experience of developers, auditors and external contributors alike.

Developers should define a constant variable for every magic value used (including booleans), giving it a clear and self-explanatory name. Additionally, for complex values, inline comments explaining how they were calculated or why they were chosen are highly recommended. Following Solidity's style

guide, constants should be named in `UPPER_CASE_WITH_UNDERSCORES` format, and specific public getters should be defined to read each one of them.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[L02] Lack of event emission after sensitive actions

The following function do not emit relevant events after executing sensitive actions.

- The `activateForFund` function from the `ExernalPositionManager` contract.

Consider emitting events after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[L03] Missing docstrings

Some of the contracts, functions, events, and constants in the codebase lack documentation, for example the ones in the `BeaconProxyFactory` and in the `BeaconProxy` contracts and in the `AddressArrayLib` library. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

[L04] Rounding effect when converting quantities

The `MathHelpers` contract implements the `__calcRelativeQuantity` function that calculates the proportionnal value based on a known ratio.

However, such function is susceptible to rounding errors depending on the quantities involved. In the worst case, if the function is used to calculate assets, amount could be small enough that the corresponding asset amount is zero, which may let the user extract funds with no cost or, viceversa, receive nothing in exchange. Since this is only important for very small values, the transaction cost of running this attack may be prohibitive.

Furthermore, when calculating the next share price in the `PerformanceFee` contract, a similar rounding effect may occur when the `_gav.mul(sharesDecrease)` value is less than the total supply of shares.

Nevertheless, in the interest of reducing the attack surface, consider ensuring the calculated amounts are non-zero, when convenient, and documenting when this does not impose a risk to the protocol. Moreover, consider rounding up when calculating the the number of assets involved during an exchange.

Update: *The Enzyme decided to not implement a fix, they have added extra comments to inform the caller.*

[L05] Lack of access control could lead to data pollution for policy contracts

The `setConfigForFund` function from the `PolicyManager` contract is in charge of enabling policies for the calling `ComptrollerProxy` contract. It does this by calling the `__enablePolicyForFund` function for each given `policies[]` decoded out of the `_configData` data parameter.

However, because there is no access control for the `setConfigForFund` function the safety is ensured by passing on the `msg.sender` address as the `comptrollerProxy` parameter, thus all corresponding actions will be recorded under the `msg.sender` address. This works well to ensure the safety of the system.

However because the `__enablePolicyForFund` function calls into each given policy's `addFundSettings()` to register the calling fund, where this function is different from policy to policy and could affect local storage, allowing any address to call it could lead to data pollution for current and future policy contract storages.

Although it currently does not cause a risk, future policies might still choose to interact with other's storage types through this function. Consider restricting the access to only the respective contracts or taking this into consideration when building new policies.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[L06] Duplicated external positions

Whenever an external position needs to be added, the external position manager calls the `callOnExtension` function from the `ComptrollerLib` contract which calls the `deploy` function from the `ExternalPositionFactory` contract to deploy a new external position proxy.

However, the `ExternalPositionFactory` contract does not check if this given asset has already been deployed with the same configuration, meaning that it is possible to deploy multiple equal external positions for a same `VaultProxy` contract. This could further lead to incorrect calculation of the fund's Gav or hinder the protocol's functionality.

Whether this should be allowed or not, consider validating that no duplicated external positions can be deployed or documenting the reasons to allow it.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d. Further notes are added to explain the reason to allow duplicated external positions and how they are managed.*

[L07] Fund could get soft-bricked

The `payFee` function from the `ProtocolFeeTracker` contract marks and calculates the number of shares that should be minted during a payment. The calculation of such shares fall on the `__calcSharesDueForVault` function where it uses the intrinsic constants of the protocol along with the elapsed time since the last time paid and the respective fee rate for the vault.

However, depending on the constants and the time elapsed, the result could become greater than the current total supply of shares for that vault. In such scenario, the transaction will revert due to the `SafeMath` shield.

Because this function could be called by several functions from the `ComptrolerLib` contract, such as the `buyShares` function, if the condition is exceeded the operation will be interrupted.

It is unlikely this situation will ever happen because it requires the elapsed time being considerable large, plus there are ways to prevent this from happening e.g. the `FundDeployer` contract owner can always set a new last-paid time.

Consider checking the `rawSharesDue` value to be lower than the total supply of shares.

[L08] Leftover ComptrollerProxy during migration of fund

When a migrator of the fund sends a request to migrate it to a newer version, the functionality of the `FundDeployer` contract deploys a new `ComptrollerProxy` contract and it links it to the respective `VaultProxy` contract.

However, because the `Dispatcher` contract does not check if the `VaultProxy` has a pending migration process, the migrator could send a new migration request through the `FundDeployer` contract without cancelling the previous migration which would not only not overwrite the migration request in the `Dispatcher` contract but also it would leave a real `ComptrollerProxy` contract partially linked to the respective `VaultProxy` contract.

Furthermore, because the data has been overwritten, cancelling the migration request would only destroy the latest `ComptrollerProxy` contract.

Even though this contract will not have the powers to control the `VaultProxy` due to its partial linkage, consider asserting that there is no other migration process active before being able to submit a new request in order to prevent the creation of multiple `ComptrollerProxy` contracts which increases the possible attack surface.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[L09] Missing critical checks in some functions

The protocol is missing critical checks in some functions as listed below. Although some of these functions are only accessible via fully trusted privileged roles, it is still a good practice to enforce function-level safety checks.

Missing checks for functions available to privileged roles

- Ensure an asset cannot be white-listed as both a primitive and derivative.
- Double check the given aggregator is the ETH/USD aggregator during the call to the `__setEthUsdAggregator` function from the `ChainlinkPriceFeedMixin` contract because the current `__validateAggregator` function only checks if there is a return value greater than zero.
- Ensure the given `_nextStaleRateThreshold` parameter value is not zero in the `__setStaleRateThreshold` function from the `ChainlinkPriceFeedMixin` contract to avoid an unexpected price stale.
- Ensure vault shares can never be used in any external positions as a tracking asset or interacting asset because it could affects the Gav value used during the fees settlement.
- Ensure the given `_nextAutoProtocolFeeSharesBuyback` parameter is not same as the existing one in the `setAutoProtocolFeeSharesBuyback` function from the `ComptrollerLib` contract to avoid unnecessary event emissions.

Missing checks for general functions

- Check that `_quantity1` is not zero in line 28 of `MathHelpers.sol` to avoid the EVM consuming all the remaining gas with an invalid opcode revert.
- Check for return value when calling the `removeStorageItem` method from the `AddressArrayLib` library to ensure the removal is correctly handled in lines 524 and 535 of `VaultLib.sol`, and lines 129 and 171 of `CompoundDebtPositionLib.sol`

Update: *Not fixed. The Enzyme team has provided further documented reasons for their decision.*

[L10] Policies removal status are not being check

The `PolicyManager` contract manages which policies are active or enabled for all Funds. However when disabling a policy for a fund, the fund's owner calls the `disablePolicyForFund` function with the respective policy to disable, and the function checks all the implemented hooks for that policy before removing them from the respective fund. At last the function will check the removal operation is successful by requiring the `disabled` variable value to be true.

However, because the requirement is outside the for loop and the variable is overwritten in each loop, only the latest result from the `removeStorageItem` method of the `AddressArrayLib` library will be validated, this means the operation will be treated as successful even if certain hooks were not correctly removed.

In order to fully validate that all hooks were successfully removed, consider either adding the requirement inside the for loop to validate each removal or apply a logical `AND` operation over the `disabled` variable with the new result.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where the result of removing individual hook is checked and a separated event is emitted. Downgraded to Low.*

[L11] Protocol may fail to issue the correct amount of shares

The `__buyShares` function from the `ComptrollerLib` contract implements the functionality to allow the investor to transfer assets and issue shares back to them.

During this process, it safe transfers deposits in line 767 from `ComptrollerLib.sol` with

```
ERC20(denominationAssetCopy).safeTransferFrom(
    _canonicalSender,
    vaultProxyCopy,
    _investmentAmount
);
```

and use the `_investmentAmount` value for accounting and issue shares back to the depositer.

However, it is not a good practice to use transferred amount for accounting purpose due to certain tokens might charge a fee during this process e.g. USTD (although not turned on at the moment) and STA (which is used in a recent hack on Balancer). This could cause the actual received amount of token to be less than the `_investmentAmount` value, which means that the protocol could issue an incorrect amount of shares.

Note even tokens that are currently safe choose to implement a similar process, possibly causing this issue in the future.

Consider always using the before and after delta balance of a token transfer as the final amount for accounting purpose, especially when calculating the number of assets to give in exchange.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where a new function `__transferFromWithReceivedAmount` is added to compute the before and after balance of the token transfer, which is then used to issue shares.*

[L12] Arbitrary call execution not handling returned data

The `callOnContract` function from the `VaultLib` contract implements the functionality to allow the `ComptrollerProxy` contract to call any arbitrary contract with the respective calldata.

However, the function does not return any data information in case of a successful transaction.

Consider implementing the functionality to handle return data from these arbitrary calls.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where the data result from the arbitrary call is passed on as the return value.*

[L13] Users can forgo assets by default during the redemption process

The `ComptrollerLib` contract implements the functionality to buy and redeem shares of a certain fund, among others.

One of the ways of redeeming the shares is by specifying the assets that the user would like to receive in exchange and the percentages associated to them. This method of redeeming also allows share holders to forgo a part of the value owed by setting as the zero address one of the assets.

However, because the zero address is used to trigger that action, an user could accidentally send a transaction with the default zero value as one of the assets and that value will not be transferred to the user.

To prevent the accidental forgo of assets on behalf of the users, consider using a different address from the default zero one to forgot a part of the value owed.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where a pre fixed const value is used instead of the 0 address to forgo an asset.*

[L14] UUPS-type proxies are susceptible to being owned

Throughout the codebase, several contracts use a proxy pattern to either reduce the gas cost or to allow new features through upgrades. Among them are the vaults, the comptrollers, the gas relayers, and the protocol fee reserve contract.

A malicious user could potentially get access to the un-used storages in the implementation contracts of the vault, gas relayer, and protocol fee reserve by calling the `init` function directly on the implementation contract. This can be used to grant the malicious user critical roles in different implementation storages e.g. The `creator` role in the vault library and further get access to `accessor` and `owner`.

This is largely safe for now since there is no ways to perform a call back from implementation contract to the proxy, however future protocol updates could introduce critical attack vector possibilities if not being careful enough, please refer to our recent blog post about this issue here.

Nevertheless, in favor of reducing the attack surface of the protocol, consider taking the ownership of all the library contracts to prevent a malicious user from doing the same. Even better, consider encapsulating under the same transaction the initialization of the libraries' contracts during its deployment.

Update: *Not fixed. The Enzyme team further explained their intention to call `init` function in their deployment scripts to avoid this issue.*

Notes & Additional Information

[N01] Misleading documentation

Some of the functions in the codebase have misleading documentation, for example:

- Line 151 of `GasRelayPaymasterLib.sol`, the revert message indicates it is coming from the function `shutdownRelayer`, but it actually is the `withdrawBalance` function.
- Line 246 of `ComptrollerLib.sol` the function is passing on the caller to `extension.receiveCallFromComptroller(caller, , )`, however the extension interface indicates this parameters is the `_comptrollerProxy` in line 23 of `IExtension.sol`.

Consider correctly documenting these functions to improve the readability of the codebase.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[N02] Unused imports

Certain contracts in the codebase imports other contracts but never make use of them. For instance:

- Line 16 of `FeeManager.sol` imports `ERC20.sol` but it is never used.
- Line 12 of `CompoundDebtPositionParser.sol` imports `IExternalPosition.sol` but it is never used.

Consider removing these un-used imports.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

[N03] Contract-address validation may fail during constructor

The `__isContract` function from the `Dispatcher` contract uses the `extcodesize` operation to determine if the queried address is a contract or not, but this will not work if it is called within a contract's constructor.

There is no immediate risk since this helper is currently only used in controlled functions, however future updates need to be extremely careful when using this helper function. Consider taking this into account when using this validation on newer versions of the protocol.

[N04] Hard-coded gas limits

Throughout the codebase there are paces where the gas cost of certain operations are fixed to a particular estimated value. In particular:

- Line 509 from `ComptrollerLib.sol`, where it is passed the gas value of `200000` in order to pay the protocol fees.
- Line 523 from `ComptrollerLib.sol`, where it is passed the gas value of `300000` which currently can "accommodates up to 8 such fees".

However, if gas costs for Ethereum operations are changed in the future or the gas needed for the action operations changes in newer releases, the transaction could reach the gas limit without completing the execution and due to the nature of the `destructActivated` function some funds may get lost.

Consider adding the ability for the owner to change, or manually set, the gas limit for such operations and in all places that a gas limit has been imposed.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where functions are added for admin to update these gas limits.*

[N05] Inconsistent use of named return variables

Named return variables are used inconsistently. While most functions name their return variables, there are a few functions that do not, for example the `transferFrom` function from the `VaultLib` contract. Moreover, the `deployProxy` function from the `BeaconProxyFactory` contract names the returned variable as `proxy_` but it also uses the `return` operation unnecessarily.

To improve both explicitness and readability of the project, consider removing all named return variables, explicitly declaring them as local variables, and adding the necessary return statements where appropriate, or maintaining the consistency along the whole codebase.

Update: *Particially fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d where 2 cases in contract `VaultLib` has been updated with named return variable.*

[N06] Unbounded loops

Throughout the codebase there are unbounded loops used. In particular:

- Lines 572 and 583 from `ComptrollerLib.sol`.
- Lines 128, 503, and 156 from `VaultLib.sol`.
- Line 288 from `ExternalPositionManager.sol`.
- Lines 130 and 155 from `IntegrationManager.sol`.

Although none of these loops impose an immediate risk, consider monitoring these array lengths as the protocol evolves to prevent an out-of-gas situation.

[N07] Typographical error

There is a typographical error in the code:

- Line 94 of `GasRelayPaymasterLib.sol`, "ALways" should be "Always".

Consider running a spell check throughout the codebase.

Update: *Fixed in e99215b3515f8266ca0b3228a7fbb7ed827c937d.*

Conclusions

One high severity issue was found. Some changes were proposed to follow best practices and reduce the potential attack surface.