SOLIDIFIED

Audit Report for Melonport. January 9, 2019.

Summary

Audit report prepared by Solidified for Melonport covering their Melon protocol and token migration contracts.

AMENDED [2019-2-12]: All reported issues were remediated or addressed to our satisfaction.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the contracts below. The debrief took place on January 8, 2019 and the final results are presented here.

Audited Files

The following files were covered during the audit:

```
contracts
 ├─── engine
 │    ├─── AmguConsumer.sol
 │    └─── Engine.sol
 ├─── exchanges
 │    ├─── EngineAdapter.sol
 │    ├─── EthfinexAdapter.sol
 │    ├─── ExchangeAdapter.sol
 │    ├─── KyberAdapter.sol
 │    ├─── MatchingMarketAdapter.sol
 │    └─── ZeroExV2Adapter.sol
 ├─── factory
 │    ├─── Factory.sol
 │    └─── FundFactory.sol
 ├─── fund
 │    ├─── Accounting.sol
 │    ├─── FeeManager.sol
 │    ├─── Hub.sol
 │    ├─── ManagementFee.sol
 │    ├─── PerformanceFee.sol
```

```
│   ├── Participation.sol
│   ├── Shares.sol
│   ├── Trading.sol
│   ├── Vault.sol
│   ├── UserWhitelist.sol
│   ├── AssetBlacklist.sol
│   ├── AssetWhitelist.sol
│   ├── MaxConcentration.sol
│   ├── MaxPosition.sol
│   ├── PriceTolerance.sol
│   ├── PolicyManager.sol
│   └── Policy.sol
├── migration
│   ├── Melon.sol
│   └── Alchemist.sol
├── version
│   ├── Version.sol
│   └── Registry.sol
└── prices
    ├── PriceSource.i.sol
    └── KyberPricefeed.sol
```

Intended Behavior

The purpose of these contracts is to facilitate the management of crypographic token assets in the context of investment funds. A full specification is available here.

The audit was based on commit `0d92a7cb24cd46b14aa75f59bb56ac04df47da6d`.

SOLIDIFIED

Audit Report for Melonport. January 9, 2019.

Issues Found

Critical

1. Engine does not track outflow of liquidEther

Description
In function `sellAndBurnMln()` of `Engine`, `liquidEther` is not decremented despite the intention of the sold ether being deducted from the `liquidEther` pool. This severely breaks the engine's intended behavior; and could lead to sale of supposedly frozen ETH, inflated premiums, and loss of funds for MLN sellers.

Recommendation
Deduct `ethToSend` from `liquidEther` in `sellAndBurnMln()`

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

2. Premium always computes to zero in `enginePrice()` of `Engine`

Description
A mistake in the integer arithmetic for calculating the premium causes the premium to always be zero.

Engine.sol / Line 85
```
uint premium = mul(ethPerMln, premiumPercent()) / 100;
```

Because `premiumPercent()` returns a percentage (< 100) the division will always floor to 0.

Recommendation
Reorder the operations as follows:
`uint premium = mul(ethPerMln, premiumPercent()) / 100;`

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

Major

3. Risk management policies can be bypassed by the manager through Participation contract

---

Description
When `PolicyManager` is invoked in `Participation` most of the arguments are given as `0x0`, despite the fact that investments can alter asset concentration, amount of positions, or include forbidden assets.

Participation.sol / Line 127

```
PolicyManager(routes.policyManager).preValidate(bytes4(sha3("executeRequestFor(address)")), [requestOwner, address(0), address(0), address(0), address(0)], [uint(0), uint(0), uint(0)], bytes32(0));
```

This means PolicyManager is effectively blind to risk management violations that occur in the form of an investment.

Recommendation
All actions which result in a change to the composition of the assets under management should be run through the policy manager (and extra care should be taken to make sure all relevant sigs should be configured to trigger said policies).

Melonport's Response:
Risk policies are to be seen only as pre-trade clearance. Not meant to be applied at participation. Although it does make sense to at least apply the asset whitelist/blacklist at participation. Other risk policies will not be applied at participation. We fixed for asset whitelist/blacklist and added `postValidate` to `Participation`.

Our Response:
`postValidate` in `requestInvestment` needs more info for risk policies to be possible to validate; alternatively, risk policies updates could have the same delay as investment, then policies would be validated in `executeRequestFor`.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

4. Fund can be misconfigured

Description
In `FundFactory`, `completeSetup()` can be called before individual setup steps have been completed.
There are checks in place that insure all the spokes have been created before there are added to a hub. This can lead to partially initialized funds being created.

Recommendation
Ensure `completeSetup()` can not be called before individual setup steps have been completed.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

5. Inconsistent usage of `makerAsset`/`takerAsset`

Description
Orders have (and risk policies are run on) a `makerAsset` and a `takerAsset`. The labels imply that the `makerAsset` is the asset held by the order maker and offered in exchange for the order taker's asset (`takerAsset`). How ever the intent of the majority of the code is evidently that `makerAsset` is to mapped to the asset held by the fund (source/outgoing asset), and `takerAsset` to the asset that will be received in (destination/incoming asset). This inconsistency produces confusion, which leads to the the incorrect asset being validated by the risk policies in `KyberAdapter` (possibly elsewhere), and potentially trading errors where a manager performs a trade with the opposite effect of what they intended.

Recommendation
Rename these to match their intent, e.g. `incomingAsset`/`outgoingAsset`. This should make it clear when the incorrect token is being passed around.

Melonport's Response:
We've added conditionals to our risk management policies that swap the assets when the policies are triggered through `makeOrder`.

Our Response:
Since incoming token is only identified as `addresses[3]` in risk policies if the function signature corresponds to "makeOrder", arguments in `preValidate()` and `postValidate()` are now incorrect when used in `Participation`. We strongly recommend changing the nomenclature to incoming/destination and outgoing/source token and refactoring to account for it, as it would reduce complexity for both the protocol itself and users.

In the same vein, it seems that abstracting the bookkeeping done in the exchange adapters by the `takeOrder`/`makeOrder` functions (e.g. `addAssetToOwnedAssets()`) to Trading would help with clarity and minimize the chance of asset confusion.

Melonport's Response:
We fixed the bug in above commit, but decided to stick with the maker/taker terminology because this could bring some complications on our 0x integration.
For 0x adapter, we pass in maker taker assets and other order parameters using our unified callOnExchange format. We use these parameters to reconstruct 0x orders in the actual struct format to submit to the exchange. If we adopt some other paradigm instead of maker / taker asset, it makes it much more confusing in the context of 0x adapter. Albeit maker / taker asset is arguably not as intuitive as src / destination or incoming / outgoing, we decided it was the better option and our team is more comfortable with that paradigm.
Therefore, we decided to unify the terminology across all exchange adapters (changed in KyberAdapter from src/dest to maker/taker).

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

6. Investment requests can be accidentally overwritten

Description
In `Participation` contract, because there is no check for existence of past request from the same sender address in `requestInvestment` function, existing request can be accidentally overwritten leading to a loss of funds.

Recommendation
Ideally, the contract would support multiple requests per sender address, but at the very least, a check preventing the creation of a new request from a sender address with an already pending request should be added so that.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

7. New investments can be used to arbitrage the fund

Description
In `Participation` contract, due to the fund selling shares at slightly outdated prices and the possibility of investment request being cancelled at any time, investors can take advantage of present market price knowledge to selectively execute investments at price levels that are advantageous relative to current market price, extracting value from funds without risk.

Recommendation
Ensure that all investment requests will be executed under normal conditions.

Melonport's Response:
Our solution is:

1. (execution incentive) there is a monetary incentive attached to each request, and anyone can execute any valid request
2. (execution price) the order you place is a sort of "limit order", where you give a price at request time, and your order can only execute when the real share price is equal or lower than yours
3. (token locking) tokens are locked at request time
4. (expiry) orders have an attached expiry time, after which they are no longer valid, but before which they can be executed
5. (cancellation) orders can only be cancelled after they expire, or under special circumstances (no price available or fund shuts down)

Our Response:
This should be sufficient as long as execution incentives stays above gas price of calling `executeRequestFor()`. Alternative solution would be to shift responsibility for executing investment requests on submitters and not allow cancellation on expiration.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

Minor

8. Eth paid to Engine is not frozen for a month

Description
The documentation for Engine states "The ETH received by the watermelon Engine is frozen for 1 month" but that's not true: if ETH is received just before unfreeze it will be unfrozen right away. All frozen ether, regardless of when it was received is eligible for release to liquid ETH 30 days (60*60*24*30 seconds) after the last thaw.

Recommendation
Update the documentation to reflect the actual behavior.

AMENDED [2019-2-12]:
The documentation was updated to reflect the actual behavior.

9. The lock and unlock functionality of the vault is not currently callable by anyone

Description
The vault spoke contains functions which are intended to manage access to withdrawal of assets managed by the fund. However, auth permissions to call these functions are never given to any spoke (or any other account). Therefore they provide no additional security.

Recommendation
No documentation on the conditions in which locking and unlocking the vault should occur, so this functionality should likely be removed to reduce confusion.

AMENDED [2019-2-12]:
The unused functionality is not present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

10. Multiple Issues with performance fee

A) Description
The frequency of payout of performance fee should not depend on when the payout is triggered.

Recommendation
This was addressed by adding `canUpdate` function to `PerformanceFee` that is used to enforce a limited payout window for each payout period.

B) Description
Because `highWaterMark` calculation in `updateState` function is based on result of `calcSharePrice` function of `Accounting` which takes into account unpaid fees, it will always end up being lower than `gavPerShare` that is used as measure of current performance in `feeAmount` function. This can lead to payout of higher performance fees than is fair.

Recommendation
This was addressed by unifying performance calculation between `feeAmount` and `updateState`, `Accounting.valuePerShare(Accounting.calcGav() / Shares.totalSupply())` is now used in both cases.

AMENDED [2019-2-12]:
These issues is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

11. REQUEST_INCENTIVE should not be static

Description
To be effective `REQUEST_INCENTIVE` needs to at least cover gas cost. Since gas cost can change, the incentive needs to be able to be changed as well.
Participation.sol / Line 26

```
uint constant public REQUEST_INCENTIVE = 10 finney;
```

Recommendation
Allow the council to set `REQUEST_INCENTIVE`.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

12. Be aware that the gas used by opcodes is subject to change

Description
Hard forks in ethereum often introduce changes to opcode cost. This means that the cost of using the protocol can change unexpectedly: not only in absolute cost, but in the cost of AMGU payable actions in relation to each others. Of course this is mitigated by those changes being

known well in advance, but the commotion surrounding Constantinople is strong evidence that those changes can still take a majority of developers by surprise. It's difficult for us to understand the justification for the tying protocol fees directly to the execution cost of the underlying implementation.

Recommendation
We recommend an alternative approach where the council sets a fee for each AMGU payable action separately, unrelated to their ethereum gas cost. This would make sure EVM changes don't unexpectedly alter the price of using the platform, allows for more fine tuning of the protocol cost, and reduces the complexity of the fee model (the code and the concept).

Melonport's Response:
The team is aware that a future hard fork on the Ethereum network could change gas used in opcodes and this would in turn affect the number of *amgu* (asset management gas units) used in a function on the Melon protocol. We have considered alternatives, including the recommendation, which we find sensible. Due to time constraints (February launch), we will put this recommendation as a MIP (Melon Improvement Proposal), that we can tackle post launch (no immediate security concern), based on the decision of the Melon Council. In the meantime, Ethereum changes would be known in advance (as you mentioned), so the Melon Council would have time to adjust *amgu* price and/or change the implementation.

13. Management fee calculation unfair to manager

Description
The following linear formula is used to calculate the portion of the fund that should be awarded to the fund manager as a management fee at any given point in time:

```
running_portion = yearly_portion * time_elapsed_since_last_payout_in_seconds / seconds_in_year
```

This formula leads to different fee payouts based on the time intervals in which the fee is rewarded: meaning that fund investors can reduce the effective fee paid to the manager simply by calling `rewardManagementFee` often. Given large enough parameters this loss can be significant. The linear formula can also theoretically lead to fee percentage accumulating over 100%.

Recommendation
We recommend replacing the linear formula with the following exponential formula that doesn't have said issues:

```
running_portion = 1 - (1 - yearly_portion)^(time_elapsed_since_last_payout_in_seconds / seconds_in_year)
```

Melonport's Response:
We had a long internal conversation debating the 2 formulas and their pros and cons. You make a very valid point, recommending the exponential formula, as it proves to be mathematically correct regardless of the management fee being 2% or 90%. However, using the exponential formula implies discriminating short-term investors (ie investors staying in the fund for shorter periods than a year).

It is important to us to give the freedom to the investor to redeem whenever he/she wants to redeem (Arguably one of the problems that worsened the last financial crisis for investors is that they were trapped into funds with minimum investment periods).

Ultimately this comes down to a trade-off between:
(a) pure mathematical correctness but unfairness to the investor
and
(b) mathematical imperfection but fair and equal treatment to all investors, regardless of the length of their holding period. (cost instead is borne by the manager)

The simulation you provided shows that with a 90% management fee, the deviation using the linear formula can become significant. For smaller numbers, the deviation is quite negligible (and always in favor of the investors, which are the most vulnerable actors in the ecosystem and whom we aim to protect the most).

On the other hand, charging the investors differently based on their holding period seems to be defeating the purpose of "redeem at any time" (which is close to our hearts).
In the asset management industry, most funds have a gross expected return of 6-8% and charge a management fee of 2%. As a consequence, any fund with a management fee of 4% or more would be considered as outrageous as the whole point of investing is to compound returns (or else no one would invest). Even more, the regulator would more than likely ban a fund with 50% management.

To double check our thinking, went back to look at how management fees are usually paid in the traditional hedge fund industry, and it is done in a linear fashion ie if a fund has 2% management fee per year, typically investors would pay a percentage of 0.02/12 per month. The common practice is that the same proportion is paid each month.
I think we could think about using the exponential formula for locked up funds that follow annual cycles which we will be introducing as a feature later on.

This formula usually caters better when compounding growth or returns ; but in the case of a tax or a fee, we shouldn't be compounding.

After much discussion we're unanimously comfortable making the following assumption: any fund with more than 5% management fee would have zero investors/traction and therefore would not occur in the first place. Assuming reasonable management fees, our formula outputs results with a negligible deviation, so we think it would be more appropriate for funds with open-investment/redemption horizons (ie. our current situation) because it renders fair treatment to investors and consistency with industry practices.

Our Response:
We agree that impact should be limited given the assumption of low management fees, however we disagree with the justification for the linear formula. Saying that it is unfair if an investor withdrawing after six months pays more than 2.5% fee with yearly fee being 5% is akin to saying that it is unfair for a debtor to pay less than 7.5% interest rate in six months on a loan with an annualised interest rate of 15%. The logic of charging fees by inflation is only consistent when the exponential formula is used. The linear formula leaves room for underexplored edge cases, which is risky in our view.

Melonport's Response:
The ideal management fee formula will increase linearly with time, so that is is equally fair for both manager and investor with no holding period bias, e.g. at 1.00% management fee terms, an investment managed for a quarter pays 0.25% of the investment value at the end of that holding period; an investment managed for 3/4 of a year will have paid 0.75% of the investment value at the end of that holding period.

The model proposed in Issue 13 pays a greater than unit proportion in early periods and less than unit proportion in later periods. For investors with holding periods of 1 year, there is no effective difference. However, investors holding for shorter periods will be disadvantaged, as their effective fee paid will be disproportionately greater. This creates a condition of unequal treatment of investors based on holding period and could influence investors to hold longer.

Equal treatment among investors is a cornerstone in investment management ethics and should be an objective to strive for to the extent possible. If calculation methodologies are imperfect, the preference is to benefit the investor.

It is the intention of the Melon protocol to maintain the unencumbered flexibility of investors subscribing and redeeming at arbitrary points in time. The legacy financial industry is unable to offer such flexibility, requiring defined windows of redemption and subscription. We believe this is a significant innovation and advantage for investors and investment managers.

See simulation in spreadsheet:
https://docs.google.com/spreadsheets/d/1g2Q5eKnv7try7FXiEnaRMUdd4NROJCjFUS06l7nZFvM/edit?pli=1#gid=0)

Notes, Improvements, & Optimizations

14. Redundant wrapped ether implementations

Description
There are two almost identical implementations for ERC-20 wrapped ETH: `WETH9` and `WETH`.

Recommendation
Code duplication should be minimized when possible.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

15. Two safe math libraries are used

Description
There are two different safe math libraries included in the smart contracts, sources from DappHub and OpenZeppelin.

Recommendation
Code duplication should be minimized when possible, we recommend using one or the other.

Melonport's Response:
Third party contracts use a different library than the one we use in our contracts.

16. Explicitly specify function visibility

Description
Throughout the code base there is inconsistent usage of the function visibility declarations. In some places the `public` keyword is used, whereas in other places, default public visibility is assumed. Additionally, many functions designated as `public` would be better suited as `external`.

Recommendation
It is best to stick to a single convention throughout the project. Specifying visibility explicitly should be the preferred option.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

17. Unify pragma statements for all contracts

Description
Pragma version statements vary across the the codebase.

Recommendation
We recommend using the same Solidity compiler version across all contracts and refactor the code accordingly, if required.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

18. Use of deprecated `var` keyword

Description
The deprecated keyword var is used in `EthFinexAdapter`, `MatchingMarketAdapter`, and `ZeroExV2Adapter`.

Recommendation
Type declarations should be made explicit.

AMENDED [2019-2-12]:
This issue is no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

19. Consider that 1 to 1 relation between managers and hubs may be limiting

Description
If a single account wants to manage multiple funds they'll need to use proxy contracts, since there is a strict 1 to 1 mapping between manager and hub.

Recommendation
This limitation is acceptable, however it may make ownership by smart contracts (i.e. DAOs) more complex from a UX perspective.

Melonport's Response:
Noted. It is currently a wanted behavior that one address can only manage one fund. If one person wants to manage several funds, they'll need to use different addresses. It is a limitation we deem acceptable for v1. That could change in the future.

20. Misc.: Dead/Unreachable Code, Outdated Comments, etc.

A. **KyberPriceFeed.sol / L146:** `baseDecimals` is never used
B. A few contracts inherit from `DBC` but do not use any of the functionality
C. **Acconting.sol:** Redundant call to `_addAssetToOwnedAssets()` in `updateOwnedAssets()`
D. **Alchemist.sol / L8:** `SPELL` never used
E. **AmguConsumer.sol / L22:** price conversion function should be used
F. **FundFactory.sol / L105:** comment incorrect
G. **Hub.sol / L105:** permission never used
H. **ManagementFee.sol / L19:** accounting never used
I. **Participation.sol / L107:** redundant, equivalent check already happens in `amguPayable` class
J. **Participation.sol / L185:** Since `getShareCostInAsset` is called right after `rewardManagementFee`, using `gavPerShareNetManagementFee` in calculations instead of `sharePrice` doesn't seem necessary: it should be 0.
K. **Participation.sol / L30:** `lockedAssetsForInvestor` is wasteful b/c an investor cannot use multiple assets in an investment request
L. **Policy.sol / L6,7:** comments are out of date, missing info about parameters
M. **Registry.sol:** comments and docs erroneously refer to ERC223 standard
N. **Registry.sol:** don't need to pass `_decimals` to `updateAsset` since it pulls this value from the token contract anyway to validate it
O. **Registry.sol:** all the asserts in Registry are redundant
P. **Trading.sol / L115:** since parameter `methodSignature` is not the selector, the selector is computed from `methodSignature` a few times below it. Would be best to refactor so that the method selector computation stays off chain and some gas is saved. Also, regardless of whether or not the computation stays on chain, it only needs to be computed once in that function.

AMENDED [2019-2-12]:
These issues are no longer present in commit `ddaedc4d82265ff9bfaac3becb4471698fd86d05`.

Audit Report for Melonport. January 9, 2019.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Melonport's products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.