CONSENSYS Diligence

mStable 1.1

Date	July 2020

1 Executive Summary

ConsenSys Diligence has conducted an audit of the mStable 1.1 protocol. mStable is a protocol for creating “meta assets” (mAssets) that are backed by baskets of tokenized assets of the same peg, such as stablecoins or commodities (referred to as “bAssets”). mAssets are always backed 1:1. bAssets supplied by users are automatically deposited into lending platforms (currently Aave and Compound). mAsset holders who stake in mStable’s SAVE contract earn 100% of the yield from lending platforms plus fees generated by users swapping between bAssets in the basket. The first mAsset, mUSD, is already active on the Ethereum mainnet and currently holds $56m in reserves.

The review was conducted over the course of 3 ½ weeks, from July 1st to July 24th by Bernhard Mueller and Valentin Wuestholz. A total of 35 person-days were spent. The audit consisted of:

- A detailed manual review of the codebase with a focus on generic smart contract security bugs as well as protocol-specific security issues;
- An in-depth fuzzing campaign to verify key security properties of the mStable protocol.

Mitigations and feedback on the audit report were reviewed from July 29th to July 30th. All major issues have been resolved. Some non-blocking minor will be addressed in future releases.

1.1 Results and Observations

- Code quality is high and the code is well-documented. Test coverage is outstanding with near-100% branch coverage. The code adheres to best practices and does not contain trivial bugs. However, it appears that some planned features are only partially implemented, leading to some dead code as well as unused variables and enum values that make the code more difficult to understand than necessary - this should be cleaned up in later iterations but doesn’t necessitate any urgent updates.

- It is apparent that the mStable team has put effort into considering threat vectors and possible failure modes. Planned functionality for re-collateralization in case of permanent loss (when an asset in the basket permanently loses its peg) has not yet been implemented. However, permanent loss is capped via max basket weights and mStable has contingency plans for manually handling basket failures.

- Some security issues were discovered during this audit, two of which were independently reported via the bug bounty program which was ongoing at the same time as this audit. All major issues have been addressed on in the current mainnet version.

2 Scope

Our review focused on the commit hash `6faf3a2387439271e8bbab4ebb74942e0645974c`. The list of files in scope can be found in the Appendix.

2.1 Objectives

The following priorities were identified for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Verify assumptions related to the core basket logic (minting, redeeming, balances) and review newly added functionality such as the rewards contracts.

3 System Overview

The audited system consisted of a set of modular, upgradeable smart contracts referred to as "modules" that are interconnected via an configurable registry (the "Nexus") that identifies each module via a bytes32 key. The Nexus itself is immutable.

3.1 mAsset

The mAsset is an ERC20 token backed 1:1 by a basket of bAssets of the same peg. A single mAsset is composed of several modules: The mAsset ERC20 token with functionality for depositing bAssets and minting or redeeming mAssets, a BasketManager that holds the basket configuration and exposes governance functionality (e.g. marking an asset as below peg), and a ForgeValidator that validates attempts to mint or redeem mAssets.

3.2 Savings Contract

The mStable savings smart contract allows mAsset holders to lock up their mAsset and receive the interest and swap fees generated from the Basket proportional to the amount of tokens locked.

3.3 Staking Rewards

The staking rewards contracts, which at the time of writing this report were not yet deployed on mainnet, will allow users to earn META tokens in return for staking LP tokens (such as Uniswap or Balancer pool tokens). mStable uses the StakingRewards contracts originally developed by Synthetix with some modifications. Notably, mStable has added the capability of locking up tokens in a rewards vault instead of airdropping them directly to users, and the capability of also rewarding users with a secondary token (e.g. liquidity mining rewards from another platform).

3.4 Meta Token

Meta (MTA) is an ERC20 token issued as a reward to liquidity providers and will allow holders to take part in governance (via a future Aragon DAO). MTA will also serve as collateral of last resort in case a basket becomes permanently undercollateralised. However, both the DAO and recollerateralisation mechanism are medium-term roadmap items and will not launch with the current iteration of the system.

The token is based on a standard OpenZeppelin mintable ERC20 implementation with role-based access (`MinterRole`) that has been extended to interface with the mStable governance system. Specifically, the contract maintains a list of minters that can be modified by the governor address configured in the mStable Nexus.

3.5 Governance and Upgrades

Most mStable components are upgradeable via governance decision. mStable's implementation is based on OpenZeppelin's ProxyAdmin with an added one-week time delay. The system governor can propose upgrades to any modules and execute the upgrades after the one week period has passed.

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are listed below with their respective abilities:

Users

- Mint mAssets by depositing bAssets into the basket
- Redeem mAssets for specific bAssets (if basket weights permit) or propotionate amounts of the bAssets in the basket
- Swap bAssets at a 1:1 rate
- Deposit mAssets to the SAVE contract to earn swap fees and interest returned by the integrated lending pools

Governor

The system governor has the privilege of upgrading and initializing smart contracts (modules) in the mStable system. They also can:

- Transfer governance to another address
- Add & remove bAssets to / from an mAsset
- Mark a basket as "failed" to stop minting, redeeming and swapping in the basket
- Change swap fees and other configuration options
- "Lock" modules to mark them as non-upgradeable
- Manage whitelist of fund managers responsible for handling staking rewards

Fund Manager

Fund managers are capable of sending ERC20 tokens (in this context, usually MTA rewards) to the StakingRewards contract.

4.2 Trust Model

Currently, governance functions are approved via a Gnosis multisig smart contract controlled by the mStable team. Users must trust the mStable team not to abuse administrative functions.

Smart contract updates enforce a one week delay during which users can opt out of the system. However, it would still be possible for admins to drain users' funds, for instance by adding an arbitrary (worthless) ERC20 token with 100% max weight and swapping it for the real assets in the basket.

This trust requirement will exist until governance rights are transferred to a DAO which is planned for a later iteration.

4.3 Important Security Properties

The following is a non-exhaustive list of security properties that were verified in this audit

- Loss of collateral or stealing of funds from the mAsset, resulting in it becoming under-collateralised
- Loss of collateral or unfair payouts in any REWARD contracts
- Unfair payouts through SAVE, MINT, REDEEM or SWAP functionalities
- Manipulating or circumvention of mStable governance mechanism
- Locking or freezing of any of the mStable contracts or inability to upgrade
- Ineffective or error prone forge validation mechanisms

5 Automated property checking and fuzzing

As part of the audit, we performed several fuzzing campaigns using Harvey, our in-house greybox fuzzer for smart contracts (see https://arxiv.org/pdf/1905.06944.pdf for more details), to check six custom properties. Five of these properties capture critical correctness and security properties outlined in the audit brief provided by the mStable team. In order to fuzz the entire contract system, we used mStable's existing deployment scripts to set up an initial state for the fuzzer containing the following contracts:

- Mock1 **(under test)**
- Mock2 **(under test)**
- Mock3 **(under test)**
- Mock4 **(under test)**
- MockAToken1 **(under test)**
- MockAToken2 **(under test)**
- MockAToken3 **(under test)**
- MockAave **(under test)**
- MockCToken **(under test)**
- Nexus **(under test)**
- ForgeValidator
- DelayedProxyAdmin
- mUSD **(under test)**
- mUSD impl.
- BasketManager **(under test)**
- BasketManager impl.
- AaveIntegration
- CompoundIntegration
- SavingsManager
- SavingsContract **(under test)**
- StakingRewards
- StakingToken1
- StakingRewardsWithPlatformToken
- StakingToken2
- PlatformToken
- Meta **(under test)**
- RewardsVault
- RewardsDistributor

We extended the deployment scripts to distribute mock tokens to several known users such that they could interact with the system after approving the mUSD contract. We also made a small number of changes to the code to improve the effectiveness of the fuzzer.

For checking the properties, we selected a subset of all contracts as being “under test” (see above). The fuzzer was set up to invoke functions of these contracts directly. However, functions in many other contracts (ForgeValidator, mUSD impl., etc.) were also invoked indirectly during our fuzzing campaigns.

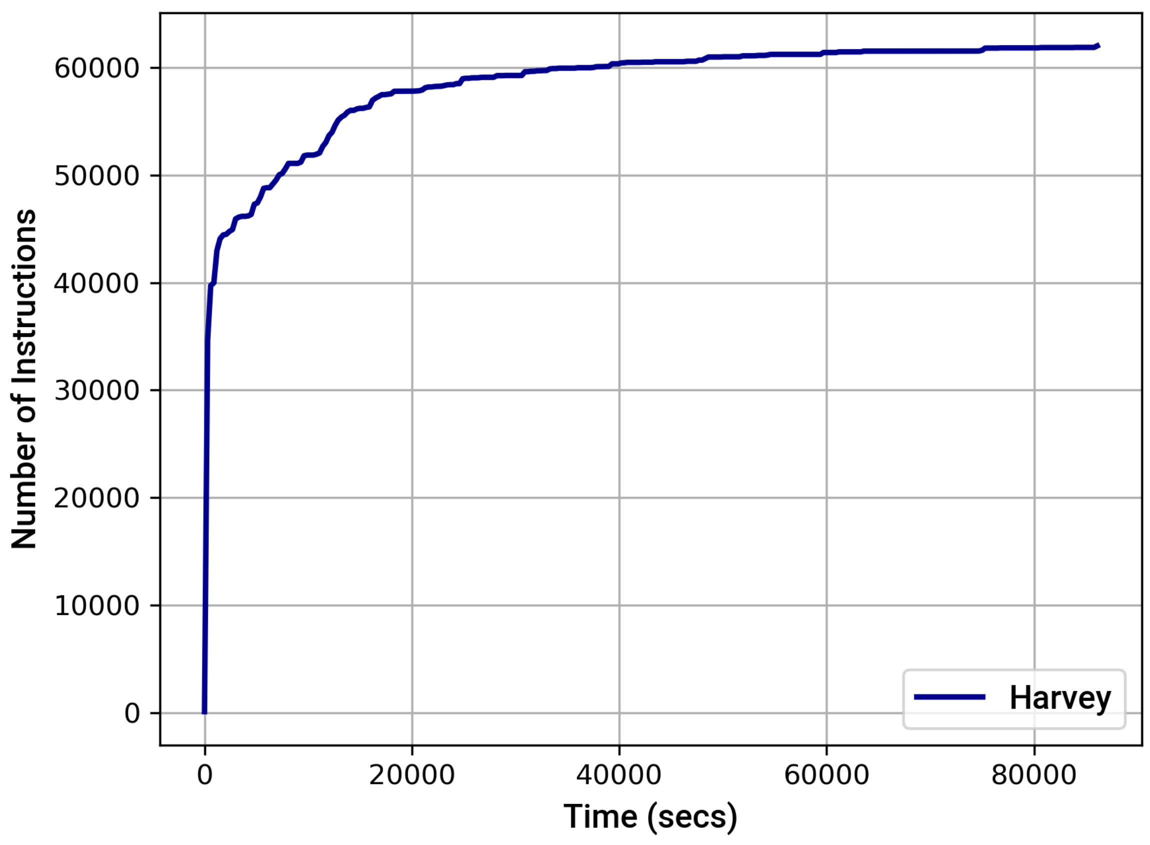
We formalized the following (informal) properties and instrumented the contracts with corresponding checks:

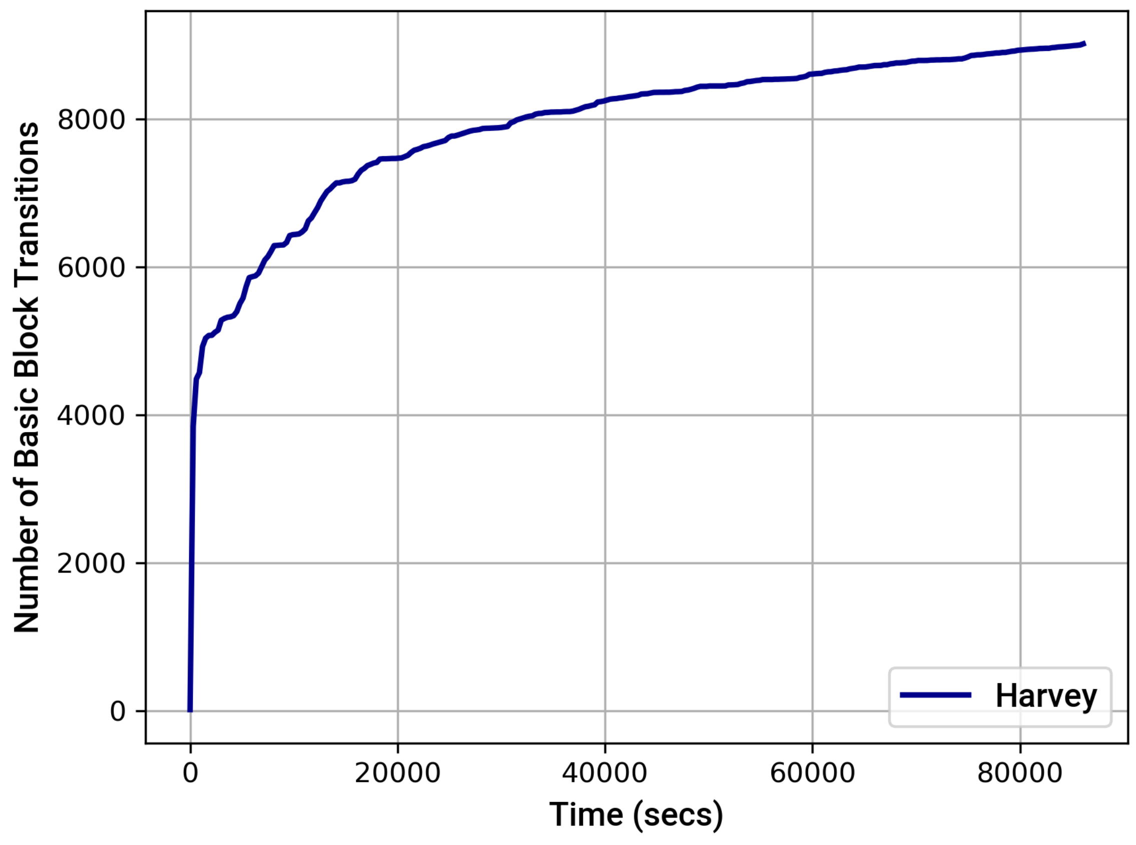
- **P1**: the total value of bAssets held in the vaults should always be greater or equal to the total supply of an mAsset (* the collateralisation ratio)
- **P2**: the bAsset quantities present in the Platform Integrations should always be greater or equal to the amounts written to storage in the BasketManager
- **P3**: actors should never receive more output than the input (at 1:1 ratio) for operations such as minting, swapping, and redeeming
- **P4**: only the mAsset and BasketManager are allowed to withdraw and deposit from the platform integrations
- **P5**: the platform integrations should never return more than is asked for to the receiver
- **P6**: exchange rate in the savings contract is ever increasing

Our final 24-hour fuzzing campaign was able to detect a property violation for P2. The “issues” section describes this finding in more detail. The fuzzer was not able to violate any of the other properties.

The graphs below provide an indication of the instruction and basic block transition coverage achieved by Harvey over time. After 24 hours, Harvey achieved the following coverage:

- EVM instruction coverage: 62002
- Path coverage: 11231
- EVM basic block transition coverage: 9026


Number of Instructions
60000
50000
40000
30000
20000
10000
0
0
20000
40000
60000
80000
Time (secs)
Harvey


Number of Basic Block Transitions
8000
6000
4000
2000
0
0
20000
40000
60000
80000
Time (secs)
Harvey

6 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 Swap fees can be bypassed using redeemMasset Major ✓ Addressed

Resolution

This issue was reported independently via the bug bounty program and was fixed early during the audit. The fix has already been deployed on mainnet using the upgrade mechanism

Description

Part of the value proposition for liquidity providers is earning fees incurred for swapping between assets. However, traders can perform fee-less swaps by providing liquidity in one bAsset, followed by calling `redeemMasset()` to convert the resulting mAssets back into a proportional amount of bAssets. Since removing liquidity via `redeemMasset()` does not incur a fee this is equivalent to doing a swap with zero fees.

As a very simple example, assuming a pool with 2 bAssets (say, DAI and USDT), it would be possible to swap 10 DAI to USDT as follows:

1. Add 20 DAI to the pool, receive 20 mUSD
2. call redeemMasset() to redeem 10 DAI and 10 USDT

Examples

The boolean argument `applyFee` is set to `false` in `_redeemMasset`:

code/contracts/masset/Masset.sol:L569

```
_settleRedemption(_recipient, _mAssetQuantity, props.bAssets, bAssetQuantities,
props.indexes, props.integrators, false);
```

Recommendation

Charge a small redemption fee in `redeemMasset()`.

6.2 Users can collect interest from SavingsContract by only staking mTokens momentarily Major ✓Addressed

Resolution

The blocker on collecting interest more than once in 30 minute period. A new APY bounds check has been added to verify that supply isn't inflated by more than 0.1% within a 30 minutes window.

Description

The SAVE contract allows users to deposit mAssets in return for lending yield and swap fees. When depositing mAsset, users receive a “credit” tokens at the momentary credit/mAsset exchange rate which is updated at every deposit. However, the smart contract enforces a minimum timeframe of 30 minutes in which the interest rate will not be updated. A user who deposits shortly before the end of the timeframe will receive credits at the stale interest rate and can immediately trigger and update of the rate and withdraw at the updated (more favorable) rate after the 30 minutes window. As a result, it would be possible for users to benefit from interest payouts by only staking mAssets momentarily and using them for other purposes the rest of the time.

Examples

code/contracts/savings/SavingsManager.sol:L141-L143

```
// 1. Only collect interest if it has been 30 mins
uint256 timeSinceLastCollection = now.sub(previousCollection);
if(timeSinceLastCollection > THIRTY_MINUTES) {
```

Recommendation

Remove the 30 minutes window such that every deposit also updates the exchange rate between credits and tokens. Note that this issue was reported independently during the bug bounty program and a fix is currently being worked on.

6.3 Internal accounting of vault balance may diverge from actual token balance in lending pool Medium Won't Fix

Resolution

After discussion with the team the risk of this invariant violation was considered negligible as the gas cost increase for querying constantly querying the lending pool would outweigh the size of the accounting error of only 1 base unit.

Description

It is possible that the vault balance for a given bAsset is greater than the corresponding balance in the lending pool. This violates one of the correctness properties stated in the audit brief. Our Harvey fuzzer was able to generate a transaction that mints a small amount (0xf500) of mAsset. Due to the way that the lending pool integration (Compound in this case) updates the vault balance it ends up greater than the available balance in the lending pool.

More specifically, the integration contract assumes that the amount deposited into the pool is equal to the amount received by the mAsset contract for the case where no transaction fees are charged for token transfers:

code/contracts/masset/platform-integrations/CompoundIntegration.sol:L45-L58

```
quantityDeposited = _amount;

if(_isTokenFeeCharged) {
    // If we charge a fee, account for it
    uint256 prevBal = _checkBalance(cToken);
    require(cToken.mint(_amount) == 0, "cToken mint failed");
    uint256 newBal = _checkBalance(cToken);
    quantityDeposited = _min(quantityDeposited, newBal.sub(prevBal));
} else {
    // Else just execute the mint
    require(cToken.mint(_amount) == 0, "cToken mint failed");
}

emit Deposit(_bAsset, address(cToken), quantityDeposited);
```

For illustration, consider the following scenario: assume your current balance in a lending pool is 0. When you deposit some amount X into the lending pool your balance after the deposit may be less than X (even if the underlying token does not charge transfer fees). One reason for this is rounding, but, in theory, a lending pool could also charge fees, etc.

The vault balance is updated in function `Masset._mintTo` based on the amount returned by the integration.

code/contracts/masset/Masset.sol:L189

```
basketManager.increaseVaultBalance(bInfo.index, integrator, quantityDeposited);
```

code/contracts/masset/Masset.sol:L274

```
 uint256 deposited = IPlatformIntegration(_integrator).deposit(_bAsset,
quantityTransferred, _erc20TransferFeeCharged);
```

This violation of the correctness property is temporary since the vault balance is readjusted when interest is collected. However, the time frame of ca. 30 minutes between interest collections (may be longer if no continuous interest is distributed) means that it may be violated for substantial periods of time.

code/contracts/masset/BasketManager.sol:L243-L249

```
uint256 balance = IPlatformIntegration(integrations[i]).checkBalance(b.addr);
uint256 oldVaultBalance = b.vaultBalance;

// accumulate interest (ratioed bAsset)
if(balance > oldVaultBalance && b.status == BassetStatus.Normal) {
    // Update balance
    basket.bassets[i].vaultBalance = balance;
```

The regular updates due to interest collection should ensure that the difference stays relatively small. However, note that the following scenarios is feasible: assuming there is 0 DAI in the basket, a user mints X mUSD by depositing X DAI. While the interest collection hasn't been triggered yet, the user tries to redeem X mUSD for DAI. This may fail since the amount of DAI in the lending pool is smaller than X.

Recommendation

It seems like this issue could be fixed by using the balance increase from the lending pool to update the vault balance (much like for the scenario where transfer fees are charged) instead of using the amount received.

6.4 Missing validation in Masset._redeemTo Medium Acknowledged

Resolution

An explicit check will be added with the next Masset proxy upgrade.

Description

In function `_redeemTo` the collateralisation ratio is not taken into account unlike in `_redeemMasset` :

code/contracts/masset/Masset.sol:L558-L561

```
uint256 colRatio = StableMath.min(props.colRatio, StableMath.getFullScale());

// Ensure payout is related to the collateralised mAsset quantity
uint256 collateralisedMassetQuantity = _mAssetQuantity.mulTruncate(colRatio);
```

It seems like `_redeemTo` should not be executed if the collateralisation ratio is below 100%. However, the contracts (that is, `Masset` and `ForgeValidator`) themselves don't seem to enforce this explicitly. Instead, the governor needs to ensure that the collateralisation ratio is only set to a value below 100% when the basket is not "healthy" (for instance, if it is considered "failed"). Failing to ensure this may allow an attacker to redeem a disproportionate amount of assets. Note that the functionality for setting the collateralisation ratio is not currently implemented in the audited code.

Recommendation

Consider enforcing the intended use of `_redeemTo` more explicitly. For instance, it might be possible to introduce additional input validation by requiring that the collateralisation ratio is not below 100%.

6.5 Removing a bAsset might leave some tokens stuck in the vault

Minor Acknowledged

Resolution

The issue was acknowledged and downgraded to 'minor' risk as only very small token amounts can be affected. A fix will be triaged for a future update.

Description

In function `_removeBasset` there is existing validation to make sure only "empty" vaults are removed:

code/contracts/masset/BasketManager.sol:L464

```
require(bAsset.vaultBalance == 0, "bAsset vault must be empty");
```

However, this is not necessarily sufficient since the lending pool balance may be higher than the vault balance. The reason is that the vault balance is usually slightly out-of-date due to the 30 minutes time span between interest collections. Consider the scenario: (1) a user swaps out an asset 29 minutes after the last interest collection to reduce its vault balance from 100 USD to 0, and (2) the governor subsequently remove the asset. During those 29 minutes the asset was collecting interest (according to the lending pool the balance was higher than 100 USD at the time of the swap) that is now “stuck” in the vault.

Recommendation

Consider adding additional input validation (for instance, by requiring that the lending pool balance to be 0) or triggering a swap directly when removing an asset from the basket.

6.6 Unused parameter in BasketManager._addBasset Minor Won't Fix

Resolution

While the parameter is not currently used it will be used in future mAssets such as mGOLD.

Description

It seems like the `_measurementMultiple` parameter is always `StableMath.getRatioScale()` (1e8). There is also some range validation code that seems unnecessary if the parameter is always 1e8.

code/contracts/masset/BasketManager.sol:L310

```
require(_measurementMultiple >= 1e6 && _measurementMultiple <= 1e10, "MM out of
range");
```

Recommendation

Consider removing the parameter and the input validation to improve the readability of the code.

6.7 Unused event BasketStatusChanged Minor Won't Fix

Resolution
This event will be used in future releases.

Description

It seems like the event `BasketManager.BasketStatusChanged` event is unused.

Recommendation

Consider removing the event declaration to improve the readability of the code.

6.8 Assumptions are made about interest distribution Minor Won't Fix

Description

There is a mechanism that prevents interest collection if the extrapolated APY exceeds a threshold (MAX_APY).

code/contracts/savings/SavingsManager.sol:L174

```
require(extrapolatedAPY < MAX_APY, "Interest protected from inflating past maxAPY");
```

The extrapolation seems to assume that the interest is payed out frequently and continuously. It seems like a less frequent payout (for instance, once a month/year) could be rejected since the extrapolation considers the interest since the last time that `collectAndDistributeInterest` was called (potentially without interest being collected).

Recommendation

Consider revisiting or documenting this assumption. For instance, one could consider extrapolating between the current time and the last time that **(non-zero) interest was actually collected**.

6.9 Assumptions are made about Aave and Compound integrations

Minor Acknowledged

Resolution

it was acknowledged that unexpected changes in behaviour by the integrated lending pools could potentially cause issues; However, it was decided that the risk is minor since the current lending pool behaviour is known and the fact that lending pools might introduce severe changes is accounted for by keeping the integrations separate and upgradable such that governance can react these changes in time.

Description

The code makes several assumptions about the Aave and Compound integrations. A malicious or malfunctioning integration (or lending pool) might violate those assumptions. This might lead to unintended behavior in the system. Below are three such assumptions:

1) function `checkBalance` reverts if the token hasn't been added:

code/contracts/masset/BasketManager.sol:L317

```
IPlatformIntegration(_integration).checkBalance(_bAsset);
```

2) function `withdraw` is trusted to not fail when it shouldn't:

code/contracts/masset/Masset.sol:L611

```
IPlatformIntegration(_integrators[i]).withdraw(_recipient, bAsset, q,
_bAssets[i].isTransferFeeCharged);
```

3) the mapping from mAssets to pTokens is fixed:

code/contracts/masset/platform-integrations/InitializableAbstractIntegration.sol:L119

```
require(bAssetToPToken[_bAsset] == address(0), "pToken already set");
```

The first assumption could be avoided by adding a designated function to check if the token was added.

The second assumption is more difficult to avoid, but should be considered when adding new integrations. The system needs to trust the lending pools to work properly; for instance, if the lending pool would blacklist the integration contract the system may behave in unintended ways.

The third assumption could be avoided, but it comes at a cost.

Recommendation

Consider revisiting or avoiding these assumptions. For any assumptions that are there by design it would be good to document them to facilitate future changes. One should also be careful to avoid coupling between external systems. For instance, if withdrawing from Aave fails this should not prevent withdrawing from Compound.

6.10 Assumptions are made about bAssets Minor Acknowledged

Description

The code makes several assumptions about the bAssets that can be used. A malicious or malfunctioning asset contract might violate those assumptions. This might lead to unintended behavior in the system. Below there are several such assumptions:

1) Decimals of a bAsset are constant where the decimals are used to derive the asset's ratio:

code/contracts/masset/BasketManager.sol:L319

```
uint256 bAsset_decimals = CommonHelpers.getDecimals(_bAsset);
```

2) Decimals must be in a range from 4 to 18:

code/contracts/shared/CommonHelpers.sol:L23

```
require(decimals >= 4 && decimals <= 18, "Token must have sufficient decimal
places");
```

3) The governor is able to foresee when transfer fees are charged (which needs to be called if anything changes); in theory, assets could be much more flexible in when transfer fees are charged (for instance, during certain periods or for certain users)

code/contracts/masset/BasketManager.sol:L425

```
function setTransferFeesFlag(address _bAsset, bool _flag)
```

It seems like some of these assumptions could be avoided, but there might be a cost. For instance, one could retrieve the decimals directly instead of "caching" them and one could always enable the setting where transfer fees may be charged.

Recommendation

Consider revisiting or avoiding these assumptions. For any assumptions that are there by design it would be good to document them to facilitate future changes.

6.11 Unused field in ForgePropsMulti struct Minor Won't Fix

Resolution

The field is currently used but will be used in a future version.

Description

The `ForgePropsMulti` struct defines the field `isValid` which always seems to be true:

code/contracts/masset/shared/MassetStructs.sol:L78-L84

```
/** @dev All details needed to Forge with multiple bAssets */
struct ForgePropsMulti {
    bool isValid; // Flag to signify that forge bAssets have passed validity check
    Basset[] bAssets;
    address[] integrators;
    uint8[] indexes;
}
```

If it is indeed always true, one could remove the following line:

code/contracts/masset/Masset.sol:L518

```
if(!props.isValid) return 0;
```

Recommendation

If the field is indeed always true please consider removing it to simplify the code.

6.12 BassetStatus enum defines multiple unused states Minor Won't Fix

Resolution

The states will potentially be used in future releases.

Description

The `BassetStatus` enum defines several values that do not seem to be assigned in the code:

- Default (different from “Normal”?)

- Blacklisted
- Liquidating
- Liquidated
- Failed

code/contracts/masset/shared/MassetStructs.sol:L59-L69

```
/** @dev Status of the Basset - has it broken its peg? */
enum BassetStatus {
    Default,
    Normal,
    BrokenBelowPeg,
    BrokenAbovePeg,
    Blacklisted,
    Liquidating,
    Liquidated,
    Failed
}
```

Since some of these are used in the code there might be some dead code that can be removed as a result. For example:

code/contracts/masset/forge-validator/ForgeValidator.sol:L46-L47

```
_bAsset.status == BassetStatus.Liquidating ||
_bAsset.status == BassetStatus.Blacklisted
```

Recommendation

If those values are indeed never used please consider removing them to simplify the code.

6.13 Potential gas savings by terminating early Minor Acknowledged

Resolution

acknowledged that gas savings are possible, might be moved changed in a future version.

Description

If a function invocation is bound to revert, one should try to revert as soon as possible to save gas. In `ForgeValidator.validateRedemption` it is possible to terminate more early:

code/contracts/masset/forge-validator/ForgeValidator.sol:L264

```
if(atLeastOneBecameOverweight) return (false, "bAssets must remain below max weight",
false);
```

Recommendation

Consider moving the require-statement a few lines up (for instance, after assigning to `atLeastOneBecameOverweight`).

6.14 Discrepancy between code and comments Minor ✓ Addressed

Resolution

The comments have been updated.

Description

There is a discrepancy between the code at:

code/contracts/masset/BasketManager.sol:L417

```
require(weightSum >= 1e18 && weightSum <= 4e18, "Basket weight must be >= 100 && <=
400%");
```

And the comment at:

code/contracts/masset/BasketManager.sol:L409

```
* @dev Throws if the total Basket weight does not sum to 100
```

Recommendation

Update the code or the comment to be consistent.

6.15 Outdated Solidity version Minor Won't Fix

Resolution

the issue was deemed acceptable because an update to solc 0.5.17 would not fix any relevant security bugs.

Description

The codebase is using an outdated version of the Solidity compiler.

Recommendation

Please consider using an up-to-date version (ideally 0.6.12 or at least 0.5.17).

Appendix 1 - Files in Scope

This audit covered the following files:

File Name	SHA-1 Ha
contracts/governance/ClaimableGovernor.sol	91e4a4e8acafefd6422e47ab
contracts/governance/DelayedClaimableGovernor.sol	17b99b5cd2ae3b8f93117a9c
contracts/governance/Governable.sol	662d7c466ab21aa7e543241
contracts/governance/InitializableGovernableWhitelist.sol	6f812e30dbaf6fac3bd35f21
contracts/masset/BasketManager.sol	33a202f061b9c38402c463e
contracts/masset/Masset.sol	5a144c4ea012470751c9502
contracts/masset/forge-validator/ForgeValidator.sol	91e0a40fe1c7fdcec42762ab
contracts/masset/forge-validator/IForgeValidator.sol	127cc3f88e9b62ae8438f3da
contracts/masset/platform-integrations/AaveIntegration.sol	3a6c7ff9967fc3ef42491f3a1
contracts/masset/platform-integrations/CompoundIntegration.sol	a7df6bab424dc584ae0a022
contracts/masset/platform-integrations/IAave.sol	330265aae09012d5a542bac
contracts/masset/platform-integrations/ICompound.sol	03ef5d33236f0e351cc65b03
contracts/masset/platform-integrations/InitializableAbstractIntegration.sol	5246077f46cd727d6f100a2
contracts/masset/shared/MassetHelpers.sol	aa37fe4aa35f39ab19242f8c
contracts/masset/shared/MassetStructs.sol	cfd811141edc454bf105507f2
contracts/meta-token/MetaToken.sol	201dc9c47eb3e1d37cd4393
contracts/meta-token/GovernedMinterRole.sol	06d4d7662c4b2fbb9afebae
contracts/nexus/Nexus.sol	2c3b95e46b14fa57ba0eada1
contracts/rewards/RewardsDistributionRecipient.sol	53dc93112fb8ee81ff105949
contracts/rewards/RewardsDistributor.sol	65c5502f5bac5f988da07a8a
contracts/rewards/RewardsVault.sol	4feeca1da637464e2d18ea0b
contracts/rewards/staking/LockedUpRewards.sol	f5e17975e8d3a33af3d27780
contracts/rewards/staking/PlatformTokenVendor.sol	d53868b690f35ec6e9abc38
contracts/rewards/staking/StakingRewards.sol	dc0a65c82bfbbbb25dee622
contracts/rewards/staking/StakingRewardsWithPlatformToken.sol	1dec44910c527e3777dfb57e

File Name	SHA-1 Ha
contracts/rewards/staking/StakingTokenWrapper.sol	77e60c0b7d1343229f9bf687
contracts/savings/SavingsContract.sol	0ed2a92496141b97b437f46
contracts/savings/SavingsManager.sol	badc8bb6fdcab432c2c7dda6
contracts/shared/CommonHelpers.sol	096e1539d2543efdd40224f
contracts/shared/IBasicToken.sol	ae9690b0ff945845ac947de7
contracts/shared/InitializableERC20Detailed.sol	ae624d989a7601d2f1ee4d8
contracts/shared/InitializableModule.sol	aba5ef8393850bc3efcef725
contracts/shared/InitializableModuleKeys.sol	e2499d9cc7c8315a40b196b
contracts/shared/InitializablePausableModule.sol	18148822e789260b6a77fc3
contracts/shared/InitializableReentrancyGuard.sol	e57f8898b8c2acd26eef71e4
contracts/shared/InitializableToken.sol	207b09a9f5ea379deec34f6d
contracts/shared/Module.sol	81b5b30403e2ab74afedb31
contracts/shared/ModuleKeys.sol	8a4fcb2c267522aea5c3615
contracts/shared/PausableModule.sol	6dbb757c98ba6f21f622c638
contracts/shared/StableMath.sol	48a87b6081f82b4cddb4bb5
contracts/upgradability/DelayedProxyAdmin.sol	e6d5534bf9c91ad3673309
contracts/upgradability/Proxies.sol	52f0c256ef8b1c429c830e6f

Appendix 2 - Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

A.2.1 Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
governance/	100	100	100	100
ClaimableGovernor.sol	100	100	100	100
DelayedClaimableGovernor.sol	100	100	100	100
Governable.sol	100	100	100	100
InitializableGovernableWhitelist.sol	100	100	100	100
masset/	100	99.4	100	100
BasketManager.sol	100	98.86	100	100
Masset.sol	100	100	100	100
masset/forge-validator/	100	100	100	100
ForgeValidator.sol	100	100	100	100
IForgeValidator.sol	100	100	100	100
masset/platform-integrations/	100	94.74	100	100
AaveIntegration.sol	100	100	100	100
CompoundIntegration.sol	100	88.89	100	100
IAave.sol	100	100	100	100
ICompound.sol	100	100	100	100
InitializableAbstractIntegration.sol	100	100	100	100
masset/shared/	100	100	100	100
MassetHelpers.sol	100	100	100	100
MassetStructs.sol	100	100	100	100
meta-token/	100	100	100	100
GovernedMinterRole.sol	100	100	100	100
MetaToken.sol	100	100	100	100

File	% Stmts	% Branch	% Funcs	% Lines
nexus/	100	100	100	100
Nexus.sol	100	100	100	100
rewards/	100	100	100	100
RewardsDistributionRecipient.sol	100	100	100	100
RewardsDistributor.sol	100	100	100	100
RewardsVault.sol	100	100	100	100
rewards/staking/	100	100	100	100
LockedUpRewards.sol	100	100	100	100
PlatformTokenVendor.sol	100	100	100	100
StakingRewards.sol	100	100	100	100
StakingRewardsWithPlatformToken.sol	100	100	100	100
StakingTokenWrapper.sol	100	100	100	100
savings/	100	95	100	100
SavingsContract.sol	100	90.91	100	100
SavingsManager.sol	100	100	100	100
shared/	100	100	100	100
CommonHelpers.sol	100	100	100	100
IBasicToken.sol	100	100	100	100
InitializableERC20Detailed.sol	100	100	100	100
InitializableModule.sol	100	100	100	100
InitializableModuleKeys.sol	100	100	100	100
InitializablePausableModule.sol	100	100	100	100
InitializableToken.sol	100	100	100	100
Module.sol	100	100	100	100
ModuleKeys.sol	100	100	100	100
PausableModule.sol	100	100	100	100
StableMath.sol	100	100	100	100
upgradability/	100	92.86	100	100
DelayedProxyAdmin.sol	100	92.86	100	100

File	% Stmts	% Branch	% Funcs	% Lines
Proxies.sol	100	100	100	100
All files	**100**	**98.57**	**100**	**100**

Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.