PUBLIC

Code Assessment

of the Staking and Surplus Smart Contracts

January 7, 2022

Produced for

Angle

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Angle Team,

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

Angle implements a decentralized, over-collateralized stablecoin protocol. This report is an extension to the main audit report and reviews the new Angle staking and surplus extension. The staking functionality has been changed completely. The surplus extension introduces an additional fee in the PoolMaster contract where a part of the profit of a strategy is taken as surplus, converted into the selected token and deposited into the FeeDistributor. The FeeDistributor later distributes to veAngle holders (long term admins).

The most critical subjects covered in our audit are the security of the new contracts, the functional correctness and the impact of these changes on the existing system.

Contrary to the extensive documentation which exists for the main Angle Protocol, no documentation exists for the new functionality. This not only makes the understanding of the code more difficult but also prevents this review from cross checking if the implemented behavior matches the expected and documented behavior. Instead we had to make assumptions on the expected behavior. Based on our discussions, we assume that the documentation will be published in a timely manner since the changes impact the agents in the current system, notably the standard liquidity providers.

In the final iteration after the intermediate reports no issues remain open while two issues are acknowledged. The functional correctness is high.

Overall we find that the codebase in its current state provides a high level of security.

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

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

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

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

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	4
• Code Corrected	4
Low-Severity Findings	12
• Code Corrected	8
• Specification Changed	1
• Acknowledged	3

2 Assessment Overview

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

2.1 Scope

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

V	Date	Commit Hash	Note
1	December 7 2021	d70b2f1a5ec994ea4dc82f0371eed42ecbf84fa7	Initial Version
2	January 3 2022	bad70b4dd14a3913746cc42a14ff9907aeaf2018	Second Version
3	January 7 2022	9c86a1d5cd0b6202b8fb823ac16da7b5d6c163a6	Final Version

For the solidity smart contracts, the compiler version `0.8.7` was chosen to be in line with the core contracts already deployed. For the vyper smart contracts, the compiler version `0.2.15` was chosen. For the second version of the contracts, vyper compiler version `0.2.16` was chosen.

This review is an extension to the main audit of the Angle system. For the original audit, please refer to the report https://chainsecurity.com/security-audit/angle-protocol/.

This extension consists of two parts and following smart contracts were in scope for this review:

SurplusDistribution:

- BaseSurplusConverter.sol
- SurplusConverterSanTokens.sol
- SurplusConverterUniV2Sushi.sol
- SurplusConverterUniV3.sol
- PoolManager contracts (focusing on the new functionality / diff to the main audit)

Staking:

- AngleDistributor.sol
- LiquidityGaugeV4 (the diff with Curve's LiquidityGaugeV4)

2.1.1 *Excluded from scope*

All other smart contracts not listed above. Notably this includes the FeeDistributor, FeeDistributorMulti, veBoost, veAngle and GaugeController contracts which are part of the new extension. These contracts are unmodified forks which have been audited by third parties.

The whole system was reviewed as part of another report linked above. Changes since the previous audit unrelated to the new extension are not part of this review.

3 System Overview

For the original audit including an overview of the whole Angle system, please refer to the previous report
.

This extension introduces a new staking system and a new admin fee in form of surplus distribution.

3.1 New Staking System

The old staking contracts will be replaced by the new staking system consisting of the following contracts:

AngleDistributor: The AngleDistributor contract redistributes AngleRewards in the form of AngleTokens to so called gauges (staking contracts). Eligible gauge contracts receive rewards every week. This reward has to be claimed per gauge. The claiming functionality automatically handles all unclaimed reward epochs of the respective gauge. Every week the reward for this epoch decreases by a predefined factor.

Multiple different types of gauges are supported:

- GaugeType 0: Mainnet gauge such as the LiquidityGaugeV4. Fully trusted. Receives full approval to transfer the reward tokens from the AngleDistributor contract. Upon payout the distributor calls `deposit_reward_token()` on the gauge passing the reward token plus the reward amount. A more detailed description of the LiquidityGaugeV4 can be found below.
- GaugeType 1: The Perpetual contracts. As before in the initial staking system, perpetuals automatically stake the brought amount. This gauge type receives its reward tokens transfer before the AngleDistributor calls `notifyRewardAmount()` which it must implement.
- GaugeType 2: Supports a contract with an interface otherwise not supported by the AngleDistributor. Transfers the reward tokens to the gauge but does not call any function on the gauge.
- GaugeType >2: This means the gauge is on another chain. The AngleDistributor increases the allowance of this gauge address before calling `pullAndBridge()` passing the amount as parameter.

LiquidityGaugeV4: User can stake with their stable tokens and receive in exchange Angle tokens. It's possible that additional reward tokens are added.

More contracts are part of the new staking system (GaugeController, veBoostProxy) which however are not part of this review.

3.2 Surplus Distribution

Profits reported by a Strategy to the PoolManager are now split in three parts. A new admin fee has been introduced which receives a part of the profit or if possible has to cover a part of the reported loss. This new fee can be configured by setting `interestsForSurplus` which represents a percentage. Upon each execution of `PoolManager.report()`, `interestsAccumulated` is updated, the amount stored in this variable can at any time be transferred to a so called SurplusConverter (or to the FeeDistributor) contract using the unpermissioned `pullSurplus` function.

SurplusConverter contracts convert the input token, which usually is the pool token into the reward token set in the contract. Finally, the output token is transferred to the FeeDistributor which distributes the accrued fees to holder of veAngle (voting escrow Angle). Holders of veAngle tokens are the long term admin of the Angle system.

SurplusConverters are composable, instead of specifying the FeeDistributor as FeeDistributor one may set another SurplusConverter as FeeDistributor.

Generally a SurplusConverter implements following functionality:

- `buyback()`: Implements the logic to exchange the input token into the defined reward token of the contract. Permissioned function to be executed by the whitelisted role.
- `burn()`: Implemented to allow composability of SurplusConverter and FeeDistributor. Thus, adheres to the interface defined by the FeeDistributor. Pulls the tokens from `msg.sender` when called. Unpermissioned function.
- `sendToFeeDistributor()`: Forwards the reward tokens to the set feeDistributor. Unpermissioned function.

Furthermore the contract implements a pause functionality which is controlled by the Guardian Role. When the contract is paused, functions `sendToFeeDistributor` and `buyback` are halted.

Currently the following SurplusConverters exist:

SurplusConverters:

- SurplusConverterSanTokens.sol
    - Expects to receive the pool's underlying token, uses this token to deposit into the corresponding stable master and receive SanTokens in return. These SanToken are the reward token.
- SurplusConverterUniV2Sushi.sol
    - For the set reward token, the Guardian Role can add/remove exchange paths for any input token to be exchanged into the reward token. UniswapV2 and Sushiswap are used by this converter. If for the given input token paths for both exchanges are defined, the path with the favorable rate is taken.
- SurplusConverterUniV3.sol
    - For the set reward token, the Guardian Role can add/remove exchange paths for any input token to be exchanged into the reward token on UniswapV3.

More contracts are part of this surplus distribution (FeeDistributor, veAngle) which however are not part of this review.

3.3 Trust Model & Roles

3.3.1 Users of the System:

Users of the system are generally untrusted and expected to behave unpredictably. The user roles are:

Standard Liquidity Provider: Provides additional liquidity in return for interests. Untrusted Role.

Stakers/Perpetual Holders: Untrusted Role.

3.3.2 Privileged roles:

Whitelisted: User allowed to trigger the `buyback` function in the SurplusConverter. Note that veAngle holders which are long term admins of the system eventually profit from this surplus. This role is trusted to not exploit the trade within the buyback function (sandwich, arbitrage).

Governance: A MultiSig. It is expected to behave honestly and correctly, but some safeguards should be in place in the Staking and Surplus to prevent errors. Note also that also for the surplus module and the staking mechanism, the governance could be different from the core system.

Guardian: Expected to behave honestly and correctly at all times. Fully trusted role completing the Governance for time critical actions where the governance (a DAO) may be too slow such as pausing the system should a problem arise or similar actions.

GaugeController: Fully trusted, defines the allowed gauges eligible to claim rewards in the AngleDistributor contract.

LiquidityGaugeV4: The staking contract is fully trusted as it receives unlimited approval to transfer the reward token (Angle Token) from the Angle Distributor.

External DeFi Systems: UniswapV2, Sushiswap and UniswapV3 are expected to work properly as documented.

veAngle: Fully trusted. System relies on correctness of veAngle as its parameters are queried for several operations.

veBoostProxy: Fully trusted. LiquidityGaugeV4 as some operations rely on the correctness of this contract.

3.3.3 Changes in V2:

- AngleDistributor can now has a `delegateGauge` to which, if not set to zero, all the funds are transferred meant for the gauges of type 2 to reduce gas cost. If it is set to `address(0)`, the behaviour is as in V1.
- AngleDistributor now does not store the exact timestamp the gauge was paid but the timestamp rounded down to weeks to prevent possible lags in reward distributions.

4 Limitations and use of report

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

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

5 Terminology

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

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

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

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

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

6 Findings

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

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

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

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Incorrect Accounting in report() Acknowledged
- Checks on interestsForSurplus Acknowledged
- Governance Differences Acknowledged

6.1 Incorrect Accounting in `report()`

Design Low Version 2 Acknowledged

`PoolManager.report()` is called by a strategy to report the performance when harvesting. First, some parameters are updated and funds are transferred. Then, surplus for the administration is set aside and the gain or loss for SLPs is signaled to StableMaster.

However, the propagation of loss for SLPs could lead to accounting issues. Assume the following scenario:

- Neither interest for admins nor admin debt has been accumulated so far.
- The interests are shared 50/50 for surplus and the stablemaster (the SLPs)

1. Assume 10 SanDAI have been minted and the current rate is 3, meaning that each SanDAI is worth 3 DAI. The pool has 30 of the underlying available which can be used by the strategy.
2. The strategy now signals a loss of 20.
3. `loss` is 20 and is split equally: `lossForSurplus` is 10.
4. `lossForSurplus > interestsAccumulatedPreLoss` holds.
5. The admins cannot currently cover any loss. Hence, their debt is increased by setting `adminDebt` to 10.
6. A loss of 10 is signaled to StableMaster. The rate drops from 3 to 2.
7. However, the StableMaster accounts for a loss of 10 DAI while SLPs actually compensated with 20 DAI since they are temporarily covering some loss for the admins. Hence, the accounting of the StableMaster mismatches the SLPs balance in the PoolManager which holds only 10 DAI.
8. Only 5 SanDAI can be redeemed (at the current rate of 2), the remaining 5 SanTokens cannot as they are not backed by funds since PoolMaster holds only 10 underlying tokens (but the rate is 2).

Only after the `adminDebt` has been paid back, sufficient funds will be available in the StableMaster. In the meantime, further issues could arise.

Acknowledged:

Angle has acknowledged this issue since such scenarios are unlikely to occur. However, Angle will monitor the system. If such a scenario is detected, Angle will handle the situation appropriately.

6.2 Checks on `interestsForSurplus`

Design Low Version 1 Acknowledged

The new `interestsForSurplus` parameter in the pool manager contract allows to split the strategies' profits between SLPs and the fee distributor. However, that parameter can range from zero to 100%. A high choice may contradict with the specification that SLPs will earn more interest by depositing to protocols through Angle compared to direct deposits. Upper-bounding the aforementioned range further could increase trust and ensure splits are fair.

Acknowledged:

Angle replied:

```
We don't want to add an upper bound, and as in other part of the protocol we suppose
the guardians have aligned incentives with the protocol. Governance could for instance
decide to set 100% of the fees for veANGLE holders and at the same time redistribute
all the transaction fees to SLPs to counterbalance for that. We added comment to the
function setInterestForSurplus.
```

6.3 Governance Differences

Design Low Version 1 Acknowledged

The Angle distributor and the liquidity gauge both support only one governor. That diverges from the other contracts' capability of having multiple governors. That change of the governance system in the new module is undocumented.

Acknowledged:

Angle replied:

```
These differences are due to the change of paradigm of governance where we will
use snapshot voting implemented by a multisig instead of true on-chain governance.
Note that technically, the `AngleDistributor` could support multiple governors.

Another reason is that most Curve contracts we have forked support only one governor.
The `LiquidityGauge` forked from Curve had only one governor, we therefore decided to
keep it. The same will go for the other contracts of the protocol.

Most contracts of the protocol were coded to support multiple governors. One will
however be used in practice however.
```

Note that also only one governor is supported in the surplus contracts (following the same reasoning).

7 Resolved Findings

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

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

Critical-Severity Findings	0

High-Severity Findings	1

- Ineffective Protection Against Sandwich Attacks, Missing Slippage Protection Code Corrected

Medium-Severity Findings	4

- Potentially Stuck Funds After Collateral Removal Code Corrected
- Race Condition on Loss Code Corrected
- Reverting on setGaugeKilled() Code Corrected
- recoverERC20 Does Not Account for the New interestsAccumulated Code Corrected

Low-Severity Findings	9

- (Missing) Checks on Path Code Corrected
- Commented Code Code Corrected
- Confusing Naming Code Corrected
- Documentation Mismatches Specification Changed
- Gas Optimizations Code Corrected
- Guardian Powers Code Corrected
- Outdated Compiler Version Code Corrected
- burn in Surplus Converters Not Paused Code Corrected
- is_killed Remains Unused Code Corrected

7.1 Ineffective Protection Against Sandwich Attacks, Missing Slippage Protection

Security High Version 1 Code Corrected

The `buyback` functions of the SurplusConverter contracts is permissioned in an attempt to prevent sandwich attacks. This protection however is ineffective due to a phenomena know as Miner Extracted values. Please see this blogpost for more information. In short, while the permissioned function may prevent a sandwich attack e.g. from within a smart contract, for miner it's still possible to put a transaction just before and after this transactions in order to sandwich it. Such attacks can actually be observed on chain, this is more than just a theoretical threat.

The calls to the exchanges within the implementations of the `buyback()` functions have their slippage protection disabled by setting the minimum incoming amount to 0. This is very dangerous.

Code corrected:

Whenever there is potential slippage, the `buyback` function now features an additional parameter specifying the minimum amount to be received. In the current SurplusConverter contracts interacting with an exchange, this value is passed as minimum incoming amount parameter to the router contracts of the exchanges.

7.2 Potentially Stuck Funds After Collateral Removal

Design Medium Version 1 Code Corrected

Tokens of the underlying may be stuck in the SurplusConverterSanTokens contract should a collateral be removed from the StableMaster contract. `buyback()` will no longer work after the collateral has been deactivated in the StableMaster and it's no longer possible to exit the underlying held by the SurplusConverterSanToken contract. Should a collateral be removed, it's important to shut down the corresponding SurplusConverterSanTokens contract first.

Code corrected:

A function `recoverERC20`, callable only by a governor, has been implemented in the parent contract BaseSurplusConverter to enable withdrawing funds from all the converters.

7.3 Race Condition on Loss

Design Medium Version 1 Code Corrected

When a strategy made a loss, a race condition between veAngle holder / long term admins and Standard Liquidity Providers (SLPs) will arise:

Angle Stakers will attempt to call `pullSurplus()` to evacuate the surplus and protect their profits while SLPs will try to call `harvest()` on the strategy, which reports the loss to the PoolManager and at least partially tries to cover the loss using interestsAccumulated of the Angle Stakers.

However, the receivers of the funds can trick the SLPs by pulling surplus after each gain immediately.

Code corrected:

A new variable tracking the debt to be covered by the admins has been introduced. When the loss is too high such that the currently available `interestsAccumulated` are not enough to cover for the losses, this debt is accrued. If there is a gain, the gain is first going to reimburse the debt and only afterwards accrue new `interestsAccumulated`.

Note that upon the first loss to be reported the race condition still exists. Admins may withdraw the `interestsAccumulated` first but then have to cover the debt accrued by the reported loss with the next profit/profits reported.

7.4 Reverting on `setGaugeKilled()`

Correctness Medium Version 1 Code Corrected

The AngleDistributor has the ability to remove the approval for a gauge through `setGaugeKilled`. That function contains following line:

```
require(IGaugeController(controller).gauge_types(gaugeAddr)
== -1 && lastTimeGaugePaid[gaugeAddr] != 0, "112");
```

Note that the controller's `gauge_types` function is defined as follows:

```
gauge_type: int128 = self.gauge_types_[_addr]
assert gauge_type != 0
return gauge_type - 1
```

It reverts if the gauge type is 0 (return value -1). However, the first code snippet shows that the call reverts if the return type is not -1. Hence, `setGaugeKilled()` will revert always revert.

Code corrected:

The precondition has been simplified to

```
require(lastTimeGaugePaid[gaugeAddr] != 0, "112");
```

Moreover, it can now only be called by guardians.

7.5 `recoverERC20` Does Not Account for the New `interestsAccumulated`

Correctness | Medium | Version 1 | Code Corrected

Function `recoverERC20` of the PoolManager allows the Governor to recover ERC20 tokens held by this contract. Any amount of arbitrary ERR20 tokens held by the contract can be exited, however for the token of the pool there are some restrictions in order to prevent the Governor to pull funds belonging to the protocol. Due to onchain constraints there are some limitations however and this can only be seen as sanity check. The HA claims are not included as it's not feasible to calculate this amount.

The new functionality introduces `interestsAccumulated` which contains the amount of tokens reserved as surplus and which can be exited using the `pullSurplus()` function, however the check in `recoverERC20()` has not been updated to account for them.

Code corrected:

`interestsAccumulated` is now considered in the computations of `recoverERC20()`.

7.6 (Missing) Checks on Path

Design | Low | Version 1 | Code Corrected

Function `addToken` of the SurplusConverterUniV2Sushi contract which can only be called by the trusted Guardian role checks the given path:

```
require(pathLength >= 2 && path[pathLength - 1]
== address(rewardToken) && path[0] == token, "111");
```

The corresponding function of the SurplusConverterUniV3 contract doesn't do any check on the path.

Code corrected:

SurplusConverterUniV3 now also implements sanity checks for the path.

7.7 Commented Code

Design | Low | Version 1 | Code Corrected

In the surplus converter for UniswapV2 / Sushiswap, the `buyback` function has the following commented code:

```
// uint256 amount = IERC20(token).balanceOf(address(this));
```

For clarity this code could be removed.

However, the balance could be used to create a sanity check for a successful swap. Both, the UniswapV2/Sushiswap and the UniswapV3 SurplusConverter contracts do not revert early if the contract's balance is smaller than the specified buyback amount.

Code corrected:

The commented code has been removed.

7.8 Confusing Naming

Design | Low | Version 1 | Code Corrected

The contracts called `surplusConverters` are to be used as what is called `surplusDistributor` inside the PoolManager contract. Especially as there exists another contract called FeeDistributior, the naming may be confusing.

`PoolManager.pullSurplus()` actually pushes the surplus to the surplusDistributor.

While the structure of the contracts SurplusConverterUniV2Sushi and SurplusConvertUniV3 is identical, one has functions called `addToken/revokeToken` while in the other they are called `updateToken/revokeToken`.

Inside the SurplusConverterUniV2Sushi contract the complementary functions `addToken` and `revokeToken` take the parameter `_typePath` and `type` respectively. Although these parameters have different names, the meaning of their values is not aligned:

```
_typePath:   0: SushiswapPath
             1: uniswapPath

_typePath:   0: SushiswapPath and uniswapPath
             1: SushiswapPath
           >-2: UniswapPath
```

Clear and structured naming greatly improves readability of the code and helps to avoid confusion or potentially resulting coding errors.

Code corrected:

The naming has been changed.

7.9 Documentation Mismatches

Correctness | Low | Version 1 | Specification Changed

The documentation of some functions mismatches their implementations in several places:

- The documentation of `buyback()` for the UniswapV3 and SanToken converters, specify that it swaps on Uniswap or Sushiswap, which is incorrect (Copy&Paste error).
- The documentation the converter functions specifies the FeeDistributor as the recipient. However, the recipient of the funds could also be another converter (e.g. `setFeeDistributor`).

Specification changed:

The comments have been corrected accordingly.

7.10 Gas Optimizations

Design | Low | Version 1 | Code Corrected

The gas consumption of some functions could be reduced. For example:

- In `PoolManager.report()` in the `if(loss > 0)` branch, the variable `interestsAccumulated` is read four times from storage.
- In `PoolManager.pullSurplus()` `surplusDistributor` is read three times and `token` is read twice from storage.
- In `BaseSurplusConverter.setFeeDistributor()` `rewardToken` is read twice from storage.
- The Sushi / UniswapV2 surplus converter stores the length of the path. Since this can be retrieved from the array itself, that results in unnecessary storage writes.
- The buyback in the Sushi / UniswapV2 where both paths are present reads either the path for UniswapV2 or Sushiswap twice from storage. Similarly, the router address for one of these will be read twice from storage.
- `AngleDistributor.toggleDistributions()` has two storage reads. However, one could be sufficient.
- Even though not a gas optimization: BaseSurplusConverter imports IUniswapRouter which is unused.

This list of examples illustrates some inefficiencies in code which could increase the gas consumption of users. Some of these optimizations may or may not be done by the optimizer. As the exact behavior of the optimizer is unknown/undocumented the optimizations may be done manually, as is done in large parts of the Angle codebase already.

Code corrected:

All optimizations listed above have been implemented.

7.11 Guardian Powers

Design | Low | Version 1 | Code Corrected

The guardian has been specified as a role that can change system parameters. However, when transferring funds it is typically required that the governance performs such actions. Note, that in the new code the guardian can be overly powerful (e.g. setting the fee distributor).

Code corrected:

Now, the governor role has been introduced to the surplus converter contracts. Note that only governors can set the fee distributor. That restricts the permissions of the guardian.

7.12 Outdated Compiler Version

Design | Low | Version 1 | Code Corrected

The project uses an outdated version of the Vyper compiler.

```
# @version 0.2.15
```

At the time of writing the most recent Vyper release of version `0.2.x` is 0.2.16 which contains some bugfixes but no breaking changes.

Code corrected:

The compiler version has been updated to `0.2.16`.

7.13 `burn` in Surplus Converters Not Paused

Correctness | Low | Version 1 | Code Corrected

The BaseSurplus converter specifies the following for function `pause`:

```
/// @dev After calling this function, it is going to be impossible for whitelisted addresses to buyback
/// reward tokens or to send the bought back tokens to the `FeeDistributor`
```

However, as `burn()` has no `whenNotPaused` modifier, it is possible to send funds to the fee distributor. Without documentation the expected behavior is unclear.

Code corrected:

The `whenNotPaused` modifier has been added to `burn()`.

7.14 `is_killed` Remains Unused

Design | Low | Version 1 | Code Corrected

`is_killed` remains unused in code. Therefore, its setter has no effect on the system. That functionality could be removed to reduce code size and, hence, to reduce deployment cost.

Code corrected:

The code has been removed.

8 Notes

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

8.1 Arbitrage Opportunities for Whitelisted Role

Note **Version 1**

The buyback function swaps one token in another one and, hence, will change the prices of assets on one pool. Now pools will now price differences and thus arbitrage opportunities are created. The comments above the `buyback` function in the BaseSurplusConverter contract describe this. Described as a mitigation for this, the function is permissioned and can only be called by the whitelisted role.

The resulting arbitrage opportunity will anyway be taken advantage of, e.g. by bots. Depending on the amounts involved it could be worthwhile for the whitelisted role to do so / allow the code to do so. Note also that multiple buyback calls could increase the profit further by creating more arbitrage possibilities.

8.2 Vyper <-> Solidity Compatability

Note **Version 1**

Some of the new contracts are written in Vyper. The system now consists of interacting contracts written in Solidity and Vyper. While both compile to EVM bytecode and the interaction are normal low level calls, there might be incompatibilities: Encoding in one and decoding in the other may be problematic and there are concerns whether that works correctly in all circumstances. Hence, interaction between contracts written in Vyper and contracts with Solidity should be considered as experimental. The interaction of such contracts should be tested carefully.