Audit Report for Prime DAO

by Team Omega

authors	**Oren Sokolowsky** **Ben Kaufman - ben.kaufman10@gmail.com** **Jelle Gerbrandy - jellegerbrandy@gmail.com**
date	
Version 0.1	**2021-04-14 - Midterm Report**
Final report	**2021-06-**

Executive Summary 5

Scope 5

Methodology 5

Code Review 5

Automatic analysis 6

Severity definitions 6

Detailed Results 6

General 6

Incomplete documentation and specifications [resolved] 6

Inconsistent Licenses [resolved] 7

Duplicate contract implementations for SafeERC20 [resolved] 7

Use more recent versions of Arc and solidity [low, not resolved] 7

Pin a specific version of the solidity compiler [resolved] 7

The deployed code from clones cannot be verified on etherscan [resolved] 8

Outdated or incomplete installation instructions [resolved] 8

Compiler Warnings relative to ABIEncoderV2 [resolved] 8

Formatting issues [info, not resolved] 9

FarmFactory.sol 9

notifyRewardAmount is not called with the correct amount in _increaseReward [resolved] 9

Missing check for success in _increaseReward [resolved] 9

setRewardDistribution cannot be called on StakingRewards clones [resolved] 10

_increaseReward may have unintended effects [resolved] 10

FarmFactory.changeParent has clumsy ownership management [resolved] 10

Permissions pattern for scheme registration is not standard, and can be simplified [low, not addressed] 11

increaseReward provides a way to send funds to any contract address [low, not addressed] 11

Test coverage 11

Events are not searchable without index keyword [resolved] 11

Naming [resolved] 12
Code style - place state variable declarations at the start of the class definition [resolved] 12
Missing Documentation [resolved] 12
StakingRewards.sol 12
Missing Attribution to synthetix [resolved] 13
Unexpected use of starttime 13
Unused state variable: _initReward [resolved] 13
Superfluous function call in getReward [resolved] 13
Use of “protected” modifier is superfluous 14
Seed.sol 14
The contract does not guarantee a sufficient supply of seed tokens [resolved] 14
Token locks are overwritten by subsequent calls to the buy function [resolved] 14
Fee value is calculated in fundingToken but paid in seedToken [resolved] 15
Administrator can take all the money by calling close() [not resolved] 15
Potential overflow at buy function [resolved] 15
Dead code [resolved] 16
Admin can block any user from buying their tokens back [not resolved] 16
Redundant storage: recipient [resolved] 16
Inconsistency between docs and implementation for _amount in the buy() function [resolved] 17
Redundant storage: vestingDuration [resolved] 17
Redundant storage: vestingCliff [resolved] 17
Check if vestingDuration > 0 in initialize [low, not resolved] 17
Semantics of “price” is unexpected [more or less resolved] 17
Duplicate call to balanceOf(..) [resolved] 18
Use explicit type declarations [resolved] 18
Redundant check that startTime is not 0 [resolved] 18
Unused state variable [resolved] 18
Meaningless require(...) [not resolved] 19
Withdraw modifiers are overly restrictive [resolved] 19
Naming and documentation [resolved] 19
SeedFactory.sol 19
deploySeed sends “cap” amount of seed tokens to the contract, but the cap is denominated in funding tokens [resolved] 19
deploySeed assumes that the admin of the Seed contract is also the funder of the seed contract [resolved] 20
Unused import of Controller and Avatar [resolved] 20
deploySeed sets msg.sender as the beneficiary of the Seed contract [resolved] 20
New stuff - not part of our audit :) 21

Executive Summary

Prime DAO has asked Team Omega to audit 4 of their contracts.

We found 5 issues that we marked as “critical” - these are issues that can lead to the user losing funds. We classified 15 issues as “medium” and 23 issues as “low”.

On May 31, PrimeDAO addressed these issues in the branch https://github.com/PrimeDAO/contracts/tree/omega-seed-reaudit, commit hash `e4bec454a7242aa725d3cebedc78366d85db41c3`

Scope

The audit concerns four contracts:

FarmFactory.sol
https://github.com/PrimeDAO/contracts/blob/ee9c961e10843a5aaf354453c9479bcb0ddb2ee3/contracts/schemes/FarmFactory.sol

StakingRewards.sol
https://github.com/PrimeDAO/contracts/blob/ee9c961e10843a5aaf354453c9479bcb0ddb2ee3/contracts/incentives/StakingRewards.sol

SeedFactory.sol
https://github.com/PrimeDAO/contracts/blob/63984dadefea331ff4aa406cdc5d934fe869886f/contracts/seed/SeedFactory.sol

Seed.sol
https://github.com/PrimeDAO/contracts/blob/63984dadefea331ff4aa406cdc5d934fe869886f/contracts/seed/Seed.sol

Methodology

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a Ganache test environment, both manually and through the test suite provided.

Automatic analysis

We have used several automated analysis tools, including Slither and MythX to detect common potential vulnerabilities. No (true) high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the Solidity pragma version setting and function visibility, were found and we have included them below in the appropriate parts of the report.

Severity definitions

Critical	vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Vulnerabilities that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Detailed Results

General

Incomplete documentation and specifications [resolved]

There are no complete specifications that describe the functionality of the contracts, and the in-line documentation in the contract code is mostly inexistent. This means that there is no good way to check if the code does what was intended, and it also makes it more likely to introduce bugs because it can be unclear what a method is intended to do.
Recommendation: Document all non-trivial functions
Severity: Medium
Status : Fixed

Inconsistent Licenses [resolved]

General: Licensing information is inconsistent: the readme and in the spdx-license-identifier say the repository is published under LGPL, but the LICENSE file in the repository is MIT. The Synthetix contract that `StakingRewards.sol` is forked from is released under an MIT licence.
Recommendation: Provide consistent licensing information, and choose a license that is compatible with the used code
Severity: Medium
Status: Fixed

Duplicate contract implementations for SafeERC20 [resolved]

The contracts use two different copies of `SafeERC20`, namely `StakingRewards.sol` uses `@daostack/arc/contracts/libs/SafeERC20.sol`, while `TokenVesting.sol` imports `openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol;`

This is confusing, and may lead to unpredictable behavior
Recommendation: Choose a single implementation - i.e. the one of Open Zeppelin implementation
Severity: medium
Status: `TokenVesting.sol` was removed

Use more recent versions of Arc and solidity [low, not resolved]

There are more recent versions available of `daostack/arc` (version 57) and of the Solidity compiler (5.17).
Recommendation: use the latest 0.5.XX version of solidity (i.e. 0.5.17) and the latest arc version
Severity: low
Status: This issue was not addressed

Pin a specific version of the solidity compiler [resolved]

Some contracts in the repository use compiler version declarations that allow for a range of versions, e.g. “pragma solidity >=0.5.13;” For contracts such as these, that are to be compiled and deployed, it is better to fix a single version, so that the output of the compilation is more deterministic and can more easily be verified
Recommendation: Specify a single version of solc
Severity: low
Status: All files now specify a specific compiles version

The deployed code from clones cannot be verified on etherscan [resolved]

Etherscan provides an independent way to verify the source code of the deployed contracts, and a way to interact directly with the deployed contracts. It is, in practice, the standard. Etherscan does not support this for clones created with the CloneFactory
Recommendation: Use OpenZeppelin's pattern, which _is_ supported
Severity: low
Status: Etherscan added support for this proxy pattern

Outdated or incomplete installation instructions [resolved]

Running `truffle compile` as described in the README only works if truffle is globally installed, and there is no guarantee that the locally installed truffle version is compatible with the package.

The README file omits instructions to install and run ganache, so running the tests as described in the README fails.

The README contains examples of commands that are not defined in package.json, such as npm run setup:pool:create:kovan

Recommendation:
- add a script in package.json to run truffle compile, or update the doc and use `npx truffle compile`
- add instructions to install ganache
- remove references to non-existing commands

Severity: Low
Status: README file was updated

Compiler Warnings relative to ABIEncoderV2 [resolved]

Compiling the contracts gives the following warning

```
Warning: Experimental features are turned on. Do not use experimental features
on live deployments.
pragma experimental ABIEncoderV2;
      ^-------------------------------^
```

Recommendation: None
Severity: Info
Status: Our recommendation was implemented

Formatting issues [info, not resolved]

The code uses tabs instead of spaces, which makes the rendering of the code somewhat unpredictable - e.g. on github spacing looks arbitrary

Recommendation: use spaces not tabs
Severity: Info

FarmFactory.sol

`Farmfactory.sol` is a helper contract that is used to deploy and initialize clones of the StakingRewards contract.

notifyRewardAmount is not called with the correct amount in _increaseReward [resolved]

In FarmFactory:193, in the `_increaseReward` function `notifyRewardAmount` is called and passed the current total balance of the rewardToken, calculated like this:

```
_amount.add(oldBalance)
```

However, on l. 118 of the StakingRewards contract, if the farming has not finished yet, the rewardRate is calculated on the basis of this new value PLUS any leftover tokens that are still to be claimed:

```
    rewardRate = reward.add(leftover).div(duration)
```

It will then check on l. 126 if there are enough tokens available to pay out all the rewards, which will fail. The netto effect is that `_increaseReward` will revert if the farming is still active.
Recommendations: just pass `_amount`
Severity: ***medium***
Status: This was fixed

Missing check for success in _increaseReward [resolved]

In FarmFactory:189, the return value for the call to `notifyRewardAmount`, which re-sets a number of state variables that make the reward available, is not checked. If the call to the target contract reverts, the call to `increaseReward` itself will *not* revert.
Recommendations: Require that the return value is true

*Severity: **medium***
Status: The check was added

setRewardDistribution cannot be called on StakingRewards clones [resolved]

The StakingRewards contract inherits from the Ownable contract, and clones are created by calling `createClone` in FarmFactory.sol:102 and then calling `initialize` on the new instance in FarmFactory.sol:114.
The clone will not have its owner set, because the constructor of the `Ownable` contract is never called. In particular, this means that `setRewardDistribution`, which is guarded by `onlyOwner`, can never be called on these cloned contracts.

Recommendations: Either (a) set the owner in the `StakingRewards.initialize(..)` function, or (b) avoid the ownership altogether and to not inherit from `Ownable` at all (and use `onlyRewardDistribution` instead of `onlyOwner`).
In addition, the test suite should be extended to test not the `masterCopy` of the `stakingRewards` contract, but on a clone of the contract.
Severity: medium
Status: Our second recommendation was implemented

_increaseReward may have unintended effects [resolved]

The `increaseReward` function calls `notifyRewardAmount` on the StakingRewards contract. The function is not documented, so it is hard to know what the intended effect is, but it should be noted that, in addition to increasing the reward, it also resets the farming period.
Recommendation: specify what this function is expected to do
Severity: medium
Status: The farming period is not reset anymore in `notifyRewardAmount`

FarmFactory.changeParent has clumsy ownership management [resolved]

The `changeParent` method calls a `transferOwnership` method on the new StakingRewards contract.

```
        parent.transferOwnership(address(avatar));
```

This will only work if the current FarmFactory is the owner of newParent - which is sort of inconvenient as (presumably) the deployer of the parent contract would first need to transfer ownership to the FarmFactory before it can register it as the new basis for Staking rewards
Recommendation: Do not call `transferOwnership`, but instead check if the parent is already owned by the avatar before setting this a parent

Severity: Medium
Status: The `changeParent` function is replaced by `changeMasterCopy`, and it does not call `transferOwnership` anymore

Permissions pattern for scheme registration is not standard, and can be simplified [low, not addressed]

The Farmfactory needs to be registered as a Scheme for the `increaseRewards` and `rescueTokens` functions to be usable. This gives the FarmFactory.sol a lot of power, which adds to the security risk. It is not clear why these two functions are needed, as these operations can also be called directly on the StakingRewards contract by the controller.
Recommendations: remove these functions, and avoid registering the FarmFactory as a scheme.
Severity: low
Status: This was not addressed

increaseReward provides a way to send funds to any contract address [low, not addressed]

The `_increaseReward` function on Farmfactory:166 sends ERC20 tokens to any address - this represents some risk to send funds to a new address via a proposal that was made by the DAO by mistake or by bad intention.
Recommendation: to whitelist all farms which are created by the factory at the `createFarm` function using a mapping, and validate that upon each call to `increaseReward` and `rescueTokens`.
Severity: low
Status: Our recommendation was not taken

Test coverage

The function `changeParent(..)` in FarmFactory.sol:71 is not covered by any test (and the problem with the implementation mentioned above would probably have been caught with a proper test)
Recommendation: write a test for the behavior of this function
Severity: low

Events are not searchable without index keyword [resolved]

Some event definitions do not use the `indexed` keyword and so will not be searchable, for example in lines 40 and 41:

```
    event TokenRescued(address farm, address token, address to);
    event RewardIncreased(address farm, uint amount);
```

Recommendation: use the “indexed” keyword

Severity: low
Status: Fixed

Naming [resolved]

- The StakingRewards contract that serves as the basis for creating clones in FarmFactory is called "`parent`" (in FarmFactory.sol:36, and in function names such as `changeParent`. The name suggests that the StakingReward contract is somehow a "parent" of the FarmFactory - while instead it is the "parent" of the farms that will be deployed by the factory
  Recommendation: rename `parent` to something more meaningful. A more explicit name that we have seen in other context is `masterCopy`
- `IRewardDistributionRecipient` is not an interface, as its name suggests, but actually implements logic, which is confusing
  Recommendation: Loose the "I"

Severity: info
Status: Our recommendations were implemented

Code style - place state variable declarations at the start of the class definition [resolved]

FarmFactory.sol:82 and further contains some property declarations after the initialize() function. It is a convention to define properties _before_ methods in the call.
Recommendation: Move the properties down
Severity: info
Status: Our recommendations were implemented

Missing Documentation [resolved]

- The state variables in FarmFactory.sol are not documented
- `FarmFactory.createFarm`: Please add that the `duration` is in days
- `FarmFactory.sol:76ff` The docstring does not include description of the return value

Severity: info
Status: Our recommendations were implemented

StakingRewards.sol

StakingRewards.sol is a contract that was forked from Synthetix, it allows users to stake tokens in the contract and returns reward-tokens in exchange. We reviewed the code where it deviates from the original contracts. We did not find any major issues.

Missing Attribution to synthetix [resolved]

StakingRewards.sol is based on Synthetix's contracts, and it would be correct to mention that in the contract's code as well as in the README (especially because the solidity file's first 10 lines are dedicated to an attribution to PrimeDAO)

Recommendations: Add the attribution
Severity: low
Status: Our recommendations were implemented

Unexpected use of starttime

The `starttime` that is set in the initializer is used to restrict access to a number of methods (`withdraw`, `stake`, `getReward`). The value is not taken into account when calculating values such as the `rewardRate` and `periodFinish`.
E.g. on l. 129

```
periodFinish = block.timestamp.add(duration);
```

This may produce unexpected results, as perhaps the caller of createFarm will expect the farming to last `duration` of days (i.e. start at `starttime` and end on `starttime + duration*days`. Instead, the farming is available only between `starttime` and `current time + duration*days`.
Recommendation: No specific one
Severity: low

-

Unused state variable: _initReward [resolved]

The `_initReward` state variable is not used, and some gas and complexity can be saved by removing it
Severity: low
Status: The variable was removed

Superfluous function call in getReward [resolved]

On line 200 there is a call to the `earn` function to get the `msg.sender`'s reward. This should instead get the reward directly from the `rewards` array, instead of performing an additional calculation which would cancel out.
Recommendation: Change the line to: `uint256 reward = rewards[msg.sender];`
Severity: low
Status: Our recommendation was implemented

Use of “protected” modifier is superfluous [not resolved]

One recurring change with respect to Synthetix’s original contract is the addition of the `protected` modifier to some (but not all) methods, which checks if the contract is initialized. Given that this instances of this contract will be, presumably, created with the Factory contract, which guarantees that the contract is initialized, these checks can be removed for a small savings in gas costs
Recommendations: remove the `protected` modifier
Severity: low
Status: The implementer chose to keep the modifier

Seed.sol

The Seed.sol contract allows the deployer to sell tokens at a fixed price, which will gradually become available to the buyer during a vesting period.

The contract does not guarantee a sufficient supply of seed tokens [resolved]

When a buyer claims her tokens once the minimum threshold is reached, there is no guarantee there is enough `seedToken` in the contract to pay her. The call to `claimLock` will revert because of the lack of seedTokens. She cannot rescue her fundingTokens using `buyBack` either, because that will throw an error after the threshold is reached. Effectively, the user’s fundingTokens are stuck in the contract.

Recommendation: require, in the `buy` function, that there are enough seedTokens available for the user to buy (as well as to pay the fees)
Severity: Critical
Status: The `buy` function now as the recommended check. Also, the `buyBack` function, which was renamed to `retrieveFundingTokens`, can now also be called after the threshold has been reached, so the user’s funding Tokens can be retrieved.

Token locks are overwritten by subsequent calls to the buy function [resolved]

In `Seed.sol:169`, if the `buy` function is called for a second time, a lot of the information on the earlier lock is lost.

If there already is an earlier lock for this user in place, then:

(a) the earlier `fundingAmount` will be simply overwritten with the new `_amount` value (instead of adding it)

(b) the `amount` (of seed tokens locked) is calculated like this:

```
        (_lockTokens.add(_amount)).mul(price).div(PCT_BASE)
```

Instead, this probably should be:

```
        _lockTokens.add((_amount.mul(price).div(PCT_BASE))
```

(c) `startTime` is overwritten, which means that the buyer loses any vesting rights that she had obtained for tokens that were locked earlier.

Recommendation: Fix this code.
Severity: Critical
Status: this code was completely refactored

Fee value is calculated in fundingToken but paid in seedToken [resolved]

Seed.sol:186: the `fee` is paid in `seedToken`, but is calculated on line 166 on as a percentage of the amount of `fundingToken`. This is clearly a mistake.
Secondly, the decision to pay the fee in the seed token leads to a potential loss of funds for the user, because the contract will run out of funds as it needs to pay `cap` amount of seed tokens to the buyers in addition to paying the fees to the beneficiary.
Recommendation: make it clear in the contract code and the specs that the fees are paid in `fundingToken` (or in `seedToken)`
Severity: Critical
Status: fixed

Administrator can take all the money by calling close() [critical, not resolved]

Seed.sol:229: admin can call `close()` and get all the money (both seed tokens as well as funding tokens), at any time during the sale. This is an unnecessary

amount of power given to the administrator

Recommendation: adapt the `close` method so that the fundingTokens that were contributed remain in the contract and can be claimed back by the contributors
Severity: Critical (if this was not intended)
Status : Our recommendation was implemented, and now only `seedToken` is taken when calling `close()`. However, the admin can still take all the funding tokens by calling `withdraw`, and so can take all the money in any case. (Note also that the fact `allowedToWithdraw` modifier is pretty much meaningless, as the admin can at any time add funds herself to set minimumReached to true, and then withdraw all funding tokens).

Potential overflow at buy function [resolved]

In line 159:

```
      require((fundingToken.balanceOf(address(this)) + _amount) <= cap, "Seed:
amount exceeds contract sale cap");
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85d
b41c3/contracts/seed/Seed.sol#L259-L260
```

If `_amount` is big enough it will overflow and pass over the `cap`. Even if the amount which the user will use to buy will be big, and there is no clear exploit, it is better to protect here against overflow.

Recommendation: use `SafeMath`

Severity: Medium

Status: Fixed

Dead code [resolved]

On line 358, the condition is always false, and so this code can be removed. The comment on line 357 however suggests that perhaps this code does not what it is intended to do, and that it perhaps is not the start time of the lock, but rather the start time of the Seed sale, that was meant to be checked.
Recommendation: remove the code (if this was the intention), or fix the code (if the intention was different)

Severity: medium

Status: dead code was removed

Admin can block any user from buying their tokens back [not resolved]

One line 194, to call the `buyBack` function, a user needs to be whitelisted. This check does not provide any particular security, but it allows the admin (who controls the whitelist) to block the user from buying the tokens back by un-whitelisting the user.
Recommendation: remove the `checked` modifier

Severity: medium

Status : `buyBack` was replaced by `retrieveFundingTokens` , which does not have this check

Redundant storage: recipient [resolved]

Seed.sol:103, 187, 188, 311, etc. : Recipient is always the key to the mapping, which makes it completely redundant and wasteful of gas to store it in the `Lock` struct and read it from storage.

Recommendation: remove `recipient` from the `Lock` struct

Severity: Medium

Status: Fixed

Inconsistency between docs and implementation for _amount in the buy() function [resolved]

Seed.sol:158: the documentation says that `_amount` is "The amount of tokens to buy", as it should be according to the specifications. But in the code, `_amount` corresponds to the amount of fundingTokens that are being offered.
Recommendation: Fix either the documentation or the code
Severity: medium
Status: Fixed

Redundant storage: vestingDuration [resolved]

Line 340: `vestingDuration` is global var and cannot be changed, so there is not need to store this separately
Recommendation: do not save `vestingDuration` in the lock
Severity: medium
Status: Fixed

Redundant storage: vestingCliff [resolved]

Line 341: `vestingCliff` is global var and cannot be changed, so there is not need to store this separately
Recommendation: do not save `vestingCliff` in the lock
Severity: medium
Status : Fixed

Check if vestingDuration > 0 in initialize [low, not resolved]

The initialize function does not check the input for sanity. This is ok, as the initialize function is called by the deployer, who presumably can be trusted. However, there are some gotcha's here, and one of them is that if `vestingDuration` = 0, the buy function will always revert.
Recommendation: Check that `vestingDuration` > 0, and perhaps add some other sanity checks
Severity: low
Status: Not fixed.

Semantics of "price" is unexpected [more or less resolved]

Seed.sol:123. The `_price` argument is not documented. The `price` variable is only used in on place, on line 169, which is problematic (see above), but there the intention seems to be to write:

```
    seedTokens = fundingTokens * price
```

I.e. `price` here is understood as "the amount of seed tokens you get if you pay 1 funding token". This is not wrong, but may be confusing, as it is more natural to express the price in fundingTokens/seedTokens rather than the other way around.
Recommendation: change the calculation on line 169 to obtain a more natural reading of `price`
Severity: low
Status : `price` is now expressed in fundingTokens/seedTokens as recommended, but the documentation remains confusing

Duplicate call to balanceOf(..) [resolved]

There is a duplicate external contract call of f`undingToken.balanceOf(address(this))` at line 162.
Recommendation: This call can be avoided, which leads to a small improvement in gas costs
Severity: low
Status: fixed

Use explicit type declarations [resolved]

Recommendation: on line 166, change `uint` to `uint256`
Severity: low
Status: fixed

Redundant check that startTime is not 0 [resolved]

Seed.sol:338 checks if `_startTime` is 0. This case never occurs, so this check not necessary
Recommendations: remove the check
Severity: low
Status : `startTime` is removed from lock.

Unused state variable [resolved]

The variable defined on line 37, `endTime`, it not used anywhere in the contract
Recommendation: remove it
Severity: low
Status: the variable `endTime` is is now used in `allowedToBuy` and `allowedToClaim`

Meaningless require(...) [not resolved]

Seed.sol:335: this requirement is basically equivalent to checking that `amount` >= `vestingDuration` - it is not clear what the purpose of this requirement is
Recommendation: Remove this line, or explain it
Severity: low
Status: The line is still there:
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L448

Withdraw modifiers are overly restrictive [resolved]

Seed.sol:274: the withdraw function is "protected", and as a result a call to this function will revert if the sale is closed. This means that if a user accidentally sends tokens after the sale has been closed, the tokens will remain locked in the contract
Recommendation: remove the "protected" statement on l. 274
Severity: Low
Status: fixed

Naming and documentation [resolved]

- Seed.sol:45 the doc string of `PCT_BASE` is hard to understand
- Seed.sol:75: `protected` is a meaningless word here, rename this to something more explicitly, like `saleIsActive`
- Seed.sol:81: rename `checked` to `senderIsWhitelisted` or `senderIsAllowedToBuy`
- `isWhitelisted`: is a confusing variable name for the Seed class, as it is not the Seed itself that is whitelisted.
- Seed.sol:160: the error message when `transferFrom` fails here will be at best confusing

Severity: info
Status: fixed

SeedFactory.sol

The SeedFactory is a utility contract that helps deploy Seed.sol clones

deploySeed sends "cap" amount of seed tokens to the contract, but the cap is denominated in funding tokens [resolved]

On line 101, the `cap` for the Seed is set, and on line 113, seed tokens are sent to the Seed contract to the amount of `cap`. However, in the seed contract, the cap is denoted in *funding tokens*, so, depending on the price, either too much or too little seed tokens are sent. This can lead to a loss of user's funds, as

they may get locked in the contract when the Seed contract runs out of seed tokens before the user has claimed hers.
Recommendation: calculate the right amount of seed tokens to send
Severity: critical
Status: No tokens are sent anymore in the `deploySeed` function, so the problem does not occur

deploySeed assumes that the admin of the Seed contract is also the funder of the seed contract [resolved]

On line 113, seedTokens to the amount of the `cap` of the Seed are sent from the address given by the `_admin` argument. This seems unnecessarily restrictive, as being an admin and being the funder are logically distinct roles, and there may be use cases in which the Seed funder is not the same address as the admin
Recommendation: add a `seedTokenHolder` argument to `deploySeed` as the address to send the tokens from, or just remove lines 112-115 and send the tokens separately.
Severity: low
Status: The lines were removed as recommended

Unused import of Controller and Avatar [resolved]

SeedFactory.sol:16-17 The imports of `Controller.sol` can be removed (as it is not used at all), and the code can be easily refactored by removing `Avatar.sol` (because the interface is not used at all)
Recommendation: remove these unused dependencies
Severity: low
Status: fixed

deploySeed sets msg.sender as the beneficiary of the Seed contract [resolved]

On line 97, `msg.sender` is set as the beneficiary of the Seed contract. The beneficiary can not be changed later. Although the `deploySeed` function has a `protected` modifier, which guarantees that `msg.sender == avatar`, it is better to make this explicit
Recommendation: change `msg.sender` to `avatar`
Severity: info
Status : `beneficiary` is now passed as an explicit argument to the `deploySeed` function

New stuff - not part of our audit :)

Redundent casting
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/SeedFactory.sol#L99

Redundant require
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L206

Redundant return var. Why return the input param ?
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L267

Redundant storage
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L259-L260 these values can be calculated offchain using events.

iDo first mul then div to prevent rounding errorrs:
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L176

iDo first mul then div to prevent rounding errorrs:
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L408

These two lines are equivalent to checking that _seedAmount > vestingDuration.
https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L448

Storing _seedAmount seems superfluous:

https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L452

Why ? can be simpler..

https://github.com/PrimeDAO/contracts/blob/e4bec454a7242aa725d3cebedc78366d85db41c3/contracts/seed/Seed.sol#L259-L260