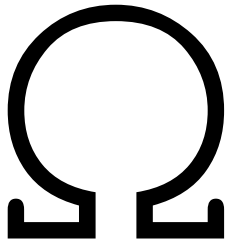
Prime DAO V2
Audit Report

July 21, 2021
Updated on August 3, 2021



Team Omega

Summary

PrimeDAO has asked Team Omega to audit the contracts that define the behavior of the PrimeDAO Seed contracts and its deployment process.

We found no critical issues.
We classified two issues as “medium” - issues we believe that you should definitely address but that do not lead to loss of funds - and 24 issues classified as “low”.

An additional 11 issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Methods Used

Code Review

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

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

Automatic analysis

We have used several automated analysis tools, including Slither and Remix 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.

Scope of the Audit

The audit concerns the contracts committed here:
https://github.com/PrimeDAO/contracts-v2/commit/37ce790ffd65338ab98619ad2de059bb5d935f5f
And specifically the following contracts:

```
contracts
├── seed
│   ├── Seed.sol
│   └── SeedFactory.sol
└── utils
    ├── Signer.sol
```

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Resolution

After delivery of an earlier version of this report, PrimeDAO has addressed most of the issues mentioned here in commit d45d8e7e8c85d9f905fe284fb832307a9481d85f. These resolutions are integrated in the report below.

Findings

General

Test coverage

Running `npx hardhat coverage` gives the following output:

```
-------------------|----------|----------|----------|----------|----------------|
File               |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------|----------|----------|----------|----------|----------------|
 seed/             |      100 |    87.23 |      100 |      100 |                |
  Seed.sol         |      100 |    86.36 |      100 |      100 |                |
  SeedFactory.sol  |      100 |      100 |      100 |      100 |                |
 test/             |      100 |       50 |      100 |      100 |                |
  ERC20Mock.sol    |      100 |      100 |      100 |      100 |                |
```

```
  Imports.sol       |      100 |      100 |      100 |      100 |                |
  PrimeToken.sol    |      100 |       50 |      100 |      100 |                |
 utils/             |    94.44 |      100 |    85.71 |    90.48 |                |
  CloneFactory.sol  |       50 |      100 |       50 |       50 |          38,39 |
  Signer.sol        |      100 |      100 |      100 |      100 |                |
 utils/interface/   |      100 |      100 |      100 |      100 |                |
  ISAFE.sol         |      100 |      100 |      100 |      100 |                |
--------------------|----------|----------|----------|----------|----------------|
All files           |    99.35 |     87.5 |    97.06 |    98.79 |                |
--------------------|----------|----------|----------|----------|----------------|
```

The incomplete branch coverage of 86.36% in `Seed.sol` is about 12 untested `require` statements, for example in `allowedToBuy` or `allowedToClaim` methods.
Although `CloneFactory.sol` is out of scope of the audit, the 50% of functions tested in CloneFactory concerns the unused `isClone` function, which can be safely removed from the code.
Severity: Low
Recommendation: Extend the tests to cover all branches in `Seed.sol`. Remove the `isClone` function from the CloneFactory as it is not used.
Status: Resolved: the `Seed.sol` contract has now 100% test coverage, as do all the other files

30 warnings from linter

Running `npm run lint` gives 30 warnings, `npx hardhat check` gives 4 linter-related warnings
Severity: Info
Recommendation: Change the linter configuration to silence the warnings (or change the code)
Status: Resolved

Set Solidity version to 0.8.6

The solidity version used in the contracts is 0.8.4. There is a newer release available, version 0.8.4.
Severity: Low
Recommendation: Upgrade to 0.8.6
Status: Resolved

Fix versions of dependencies of imported Solidity code

The code uses Gnosis and OpenZeppelin code. The version numbers are specified as minimum versions in `package.json`

```
  "dependencies": {
    "@openzeppelin/contracts": "^4.1.0",
```

```
    "axios": "^0.21.1",
    "dotenv": "^10.0.0",
    "openzeppelin-solidity": "^4.1.0"
  }
```

The versions that are actually used for the openzeppelin packages is 4.2.0 (according to `package-lock.json`) .
It is recommended to pin the dependencies to an exact version, so that solidity code is not changed accidentally.
Severity: Low
Recommendation: Pin exact versions of solidity dependencies
Status: Resolved

Reversed variables in tests

In 2 cases, hardcap and softcap are reversed, this does not seem to currently cause a problem, but should be fixed to not make unexpected testing issues in the future.
This happens in `seed-factory.js:155` and `seed-factory.js:173`
Severity: Info
Recommendation: Flip the the variables to have them in the correct order
Status: Resolved. This was intentional, a comment has been added.

Seed.sol

No clear failsafe mode [medium]

Often, contracts like these include a failsafe mode that can be enabled by the owner in the case of an emergency or if a bug has been found. Typically, such a mode restricts calls to functions, and only leaves, or enables, refund functionality.
There is no such failsafe mode present. There is, however, a `pause`() function, that disables most interaction with the pool except for the withdraw functionality - specifically, it does not allow for users to retrieve their funds
Severity: Medium
Recommendation: implement a failsafe mechanism that disables all interaction with the contract except to call `retrieveFundingTokens`.
Status: A failsafe mode was not implemented, but it was considered that the close() and pause() functions suffice

PCT_BASE has confusing name and doc string

On l. 46, it says:

```
uint256 constant internal PCT_BASE = 10 ** 18; // // 0% = 0; 1% = 10 **
16; 100% = 10 ** 18
```

Both the name as well as the doc string suggest that this constant somehow is used in the context of percentages. It is not; it is used for the precision in the price calculation instead, while the fee, which is a percentage, does not.
Severity: Low
Recommendation: Change the constant name and doc string - perhaps to `PRICE_PRECISION`. Also, consider our remarks below on the precision of the `price` and the `fee` variables.
Status: Resolved - the variable is now called `PRECISION`

Superfluous information in event

On l.69, the `TokensClaimed` event includes both the `amount` and the `feeAmount`.

```
event TokensClaimed(address indexed recipient,uint256 amount,address
indexed beneficiary,uint256 feeAmount);
```

As the `feeAmount` can be calculated from the `amount`, there is no need to include this value explicitly in the event, and some gas can be saved by removing it.
Severity: Low
Recommendation: Remove the `feeAmount` value from the `TokensClaimed` event
Status: Not Resolved

feeClaimed state variable can removed

The value of the state variable `feeClaimed` that is defined on line 61 can be calculated as `totalClaimed * fee`
Recommendation: remove the variable to save some gas
Severity: Low
Status: Resolved

FunderPortfolio struct stores superfluous information

In l. 73, the `FunderPortfolio` struct stores `seedAmount`, `fundingAmount` and `fee`, and it stores `totalClaimed` and `feeClaimed`.
These values depends on each other (for example, `seedAmount = fundingAmount / price`, f`eeClaimed = totalClaimed * fee`) , and so there is no need to store each of these separately.
Also note that `feeClaimed` is never used in the code - so there is no need to store that value on chain at all.
Recommendation: remove `seedAmount`, `fee` and `feeClaimed` from the `FunderPortfolio` struct - it is cheaper to calculate these when needed.
Severity: Low
Status: Resolved

Distribution period and claim period can overlap

Lines 98-110: the definition of `allowedToBuy` requires that `endTime >= block.timestamp` while `allowedToClaim` requires that `endTime <= block.timestamp`.
This allows for the possibility that users can `buy` and `claim` in the same block. This does not seem intended.
Severity: Low - we see no attack factors that apply
Recommendation: Require that `endTime < block.timestamp` in `allowedToClaim`
Status: Resolved

Superfluous require statement in allowedToWithdraw

In l. 120, `allowedToWithdraw` requires that the contract is not paused. As only the admin is allowed to withdraw, and the admin can unpause the contract at will, this requirement does not add any security, and at best will just be an annoyance for the admin who wants to withdraw funds.
Severity: Info
Recommendation: remove the `require` statement
Status: Resolved

Use external modifier where that is applicable

There are a number of function definitions that are marked `public`, but are not called internally. Some gas can be saved (and some clarity gained) by marking these functions is `external`. This applies to the following functions:

```
initialize
buy
claim
retrieveFundingTokens
pause
unpause
close
whitelist
whitelistBatch
unwhitelist
withdraw
updateMetaData
```

Severity: Low
Recommendation: Mark these functions `external`
Status: Resolved

Documentation of price parameter is wrong

The `price` state variable is used to calculate the amount of fundingTokens a user must contribute to claim a seedToken, as on l. 197 of the code:

```
uint256 seedAmount = _fundingAmount(*PCT_BASE)/price;
```

The price is expressed with a precision of `PCT_BASE` - i.e. if the value of `price` is `3.14 * PCT_BASE`, this means a user will need to contribute 3.14 funding tokens to claim a single seed token.

Given this, the doc string that claims that `_price` represents the price "of fundingTokens" is confusing, as this is not the price of funding tokens but of seed tokens

```
@param _price The price in wei of fundingTokens when exchanged for
seedTokens
```

While in the SeedFactory, the doc string also suggests that the price is expressed in seed tokens:

```
* @param _price 1 Funding Token = _price amount of Seed Token
```

Please also note that the writing "the price in wei" is not appropriate - the concept of a "wei" refers to the smallest unit of ETH, which happens to be 10**18th part of a "single ETH". This convention has nothing to do with how the relation between the seed tokens and funding tokens (which can either or both use a different convention) is expressed.
Recommendation: make these doc strings more clear, for example:

> The price of a SeedToken, expressed in fundingTokens, with a precision of 10**18

Severity: Info
Status: Resolved

Precision of fee seems too low

The price state variable has a precision of `10**18` (which is the value of `PCT_BASE`) - meaning that it is possible to specify the exchange rate between `fundingToken` and `seedToken` to a precision of 18.
The `fee` state variables has a precision of `10**2`, meaning that it is possible to specify a fee of `0.01` (i.e. 1%), but not of `0.015` (which would be 1.5%).
It seems to us that the precision of the price is larger than needed (which is mostly harmless), but that the `fee` parameter lacks in precision.

Recommendation: Use a single parameter for expressing the precision of both `fee` and `price`. Do not call it `PCT_BASE`, but perhaps `PRECISION`. A reasonable value seems 10**8, but also 10**18 will do.
Severity: Low
Status: Resolved

Unnecessary use of variable

On l. 200 the `fundingBalance` variable is declared and initialized as:

```
// Funding Token balance of this contract
uint256 fundingBalance = fundingCollected;
```

This variable is not really used, it just copies the value of `fundingCollected` but is never changed. It's name and docstring are confusing, as `fundingCollected` does *not* represent the token balance - funding tokens could be sent directly to the contract, or some could have already been withdrawn by the administrator.
Recommendation: Refactor the code to not use this variable
Severity: Info
Status: Resolved

Do external calls after state changing operations

On lines 192, 219 in the `buy` function, calls to external contracts are made. To protect against reentry attacks, it is best practice to do external calls after state-changing functions.
Recommendation: move these functions calls to just before the emission of the events
Severity: Low
Status: Not resolved

Wrong error message

On line 219:

```
require(fundingToken.transferFrom(msg.sender, address(this),
_fundingAmount), "Seed: no tokens");
```

The error message is not correct: there can be other reasons for that the `transferFrom` calls fails (for example, when no allowance is set)

Similarly on line 266:

```
require(seedToken.transfer(_funder, _claimAmount), "Seed: no tokens");
```

Recommendation: Change the error messages to “Seed: funding token transferFrom failed” and “Seed: seed token transfer failed”
Severity: Low
Status: Resolved

Unnecessary values passed to _addFunder

In lines 233 and 235, the values for `totalClaimed` and `feeClaimed` are always set to 0, so there is no need to set these values explicitly.
Recommendation: Pass “0” for these values in `_addFunder`, or just remove these parameters from the `_addFunder` signature, and initialize to 0.
Severity: Info
Status: Resolved - the helper function was removed

Unnecessary copying of data from storage to memory

On line 257, the `funderPortfolio` array is copied to memory, then values are changed, and subsequently it is written back to storage.

```
FunderPortfolio memory tokenFunder = funders[_funder];
```

The same pattern is used in `retrieveFundingTokens` on line 279ff . And similarly on line 398

Some gas can be saved by writing directly to storage.

Recommendation: Change the code to:

```
FunderPortfolio storage tokenFunder = funders[_funder];
tokenFunder.totalClaimed    += _claimAmount
tokenFunder.feeClaimed      += feeAmountOnClaim;
```

And apply a similar change for lines 279ff and 398.
Severity: Low (some gas can be saved)
Status: Resolved

Rename FundingReclaimed event

in the function and modifier names is the language used is “retrieve Funding”, so for consistency, this event should be renamed to “`FundingRetrieved`”
Severity: Info
Status: Not resolved

The close() function does not necessarily return the seed tokens to the funder

The `close()` function sends remaining seed tokens to the admin of the contract. However, the admin is not necessarily the same as the address that provided the seed tokens to the contract in the first place: these tokens could come from any other address.
Recommendation: Consider implementing a separate `fund()` function that keeps track of a list of addresses that funded the contract and for how much
Severity: Info
Status: Resolved

The value of close is not always true after calling close()

If `minimumReached` is true, the value of the state variable `close` is not set
Recommendation: set the value of `close` to true also in the given case
Severity: Low
Status: Resolved

unwhitelist function seems overly restrictive

L: 364: the `unwhitelist` function can only be called if the contract is not paused. This seems unnecessary restrictive: it does not provide any security, as the admin can always unpause the contract, while it does exclude certain emergency scenarios such as this one:

- a whitelisted buyer misbehaves
- Admin pauses the contract to gain some time to find the address of the culprit
- The admin unwhitelists the buyer
- The admin can now safely unpause the contract

Recommendation: Remove the requirement that the contract should not be paused when calling the `whitelist` function
Status: Resolved

Unclear error message on whitelist-related functions

The error message in lines 344, 354 and 365 are confusing, as they suggest that there is a module (?) that should be whitelisted:

```
require(permissionedSeed == true, "Seed: module is not whitelisted");
```

Recommendation: Use a better error message, like “Seed: pool does not have a whitelist”
Severity: Info
Status: Not resolved

updateMetadata can be called by anyone if the contract is not initialized

On l. 385, it is explicitly allowed that `updateMetadata` can be called by anyone if the contract is not initialized yet. We can not think of any scenarios where this would be useful (nor can we think of an attack vector if the contracts are deployed by the `seedFactory`).
Recommendation: remove this option (and adapt the `TokenFactory.deploySeed` function accordingly). Or explain why permissions are set like this
Severity: Info
Status: Not resolved

checkWhitelist function is redundant

The `checkWhitelist` function is redundant, as the solidity compiler already provides a `whitelist(address)` function for getting the value
Recommendation: Remove this function
Severity: Low
Status: Resolved

_currentTime() is used inconsistently

The `_currentTime()` function that is defined on line 432 is used in some, but not all, of the times that the `block.timestamp` is read in the code.
Recommendation: Either use `_currentTime()` or `block.timestamp` in the code, but not both
Severity: Info
Status: Resolved

The _feeClaimed parameter is not documented in _addFunder

Severity: Info
Status: Resolved, the function was removed

totalFunderCount does not represent the total amount of funders

On line 464, `totalFunderCount` is updated also when an address already has added funds before. So it does not represent the count of funders, but rather a count of funding events.
Recommendation: As the variable is not used in the code, it is probably best to omit it, also to save some gas costs. For the UI, the amount of funders can be derived from the events
Severity: Low (as some gas can be saved)
Status: Resolved

SeedFactory.sol

Wrong value passed to SeedCreated event

On line 103:

```
emit SeedCreated(address(_newSeed), msg.sender);
```

This should be `_beneficiary` instead of `msg.sender`
Recommendation: Change to beneficiary
Severity: Low
Status: Resolved

CloneFactory.sol

Licensing violation

The CloneFactory contract was originally published under the MIT license here:
https://github.com/optionality/clone-factory/blob/master/contracts/CloneFactory.sol
The MIT license states explicitly that:

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

Neither the copyright notice, nor the permission notice, is included in this copy, which is in violation of the original license.
Recommendation: include the npm package https://www.npmjs.com/package/@optionality.io/clone-factory just as you do with the gnosis and openzeppeling contracts, instead of copying the file to the source tree.
If you want to keep the file: Instead of changing the license text and adding the primedao logo, you can include the original file without any changes (i.e. including the original MIT license and copyright notice). You are allowed to release the primedao-v2 software under an LGPL license also if the source tree includes a file with the MIT license.
Severity: Low
Status: Resolved

Signer.sol

isValidSignature does not check if the data is signed [medium]

The implementation of `isValidSignature` checks that the message hash is in the `approvedSignatures` array, but does not do any checks of the data provided in the `_hash` parameter. In other words, given a registered signature, it will consider any transaction hash that is provided as valid.

We have done a quick scan of the implementation of the Transaction Service, and believe this is not checked there is no check there if the relationship between the signature and the transaction data when validation is delegated to a contract (see https://github.com/gnosis/gnosis-py/blob/8ffdfc3bc276934e3c3f1b24a3d206a1c3e33b96/gnosis/safe/safe_signature.py#L168)
If that is true, then an attacker can listen to the `SignatureCreated` event, extract the signature, and submit it with arbitrary transaction hash to the Safe transaction queue: in other words, they can queue any transactionto be signed in the Safe UI. This renders the `Signer` contract completely useless as a gatekeeper.
Recommendation: Check if the data `_hash` submitted in `isValidSignature` is indeed signed by the provided `signature`.
Severity: Medium
Status: Resolved

Inherit from ISignatureValidator

The EIP-1271 standard has been updated since the Gnosis implementation - the first argument for `isValidSignature` has been changed from `bytes` (for passing the transaction data) to `bytes32` (for passing a hash of the data); and consequently also the “Magic Value” has changed. Please see to https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md

This is somewhat confusing, as the PrimeDAO code currently refers to the EIP-1271 standard but does not (strictly) adhere to either to outdated or latest version of that standard.
Recommendation: Inherit from `@gnosis/safe-contracts/contracts/interfaces/ISignatureValidator.sol`. This will make sure that you use the right value for `EIP1271_MAGIC_VALUE` and the correct signature for the `isValidSignature` method.
Severity: Info
Status: Resolved

Mark unchangeable state variables as immutable

On lines 35 and 36, the state variables `safe` and `seedFactory` should be marked `immutable`, as they cannot be changed
Severity: Low
Status: Resolved

generateSignature is calleable by anyone

As `generateSignature` is callable by anyone, anyone can submit a request to deploy a pool to the Gnosis Safe. This makes it possible for an attacker to spam the safe (for a relatively low gas cost) or to submit doctored transactions.

As the intended use is that this method is called by a dApp (instead of being called directly by users), consider making `generateSignature` permissioned, i.e. only callable by a chosen account, and route requests from the dApp through a signer.
Recommendation: Review the access policy for `generateSignature`, and consider limiting access.
Severity: Low
Status: Resolved. This is intentional, and so was not changed

Redundant parameters in generateSignature

The `_to` parameter in `generateSignature` is redundant, as it must always be equal to the seedFactory address.
Similarly, the value of `operation` will always be `CALLDATA` - no reason for the user to be able to change this, by mistake or intention.
Finally, it not clear clear what the intended use case is for providing a non-default value for `refundReceiver`
Recommendation: remove these parameters
Severity: Low
Status: Resolved: it was decided to leave the function signature unchanged, and to add `require` statements that fix the behavior

Unnecessary _value parameter in generateSignature

The `generateSignature` function allows for a user to send ether with the transaction by providing a non-zero value for the `_value` parameter. There is no use case that we can see for that, while it enlarges the attack surface
Recommendation: remove the `_value` parameter, and set the `transaction.value` to 0
Severity: Low
Status: Resolved: it was decided to leave the function signature unchanged, and to add `require` statements that fix the behavior

Wrong error message

On line 95, the function signature of the submitted calldata is checked:

```
require(_getFunctionHashFromData(_data) == SEED_FACTORY_MAGIC_VALUE,
"Signer: cannot sign invalid function call");
```

This error message is wrong: the check is not for validity of the call data, but if the `calldata` represents a call to `deploySeed`.
Recommendation: Change the error message to `"Signer: can only sign calls to deploySeed"`
Severity: Low
Status: Resolved

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	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion