ABDK CONSULTING

SMART CONTRACT AUDIT

PandiFi

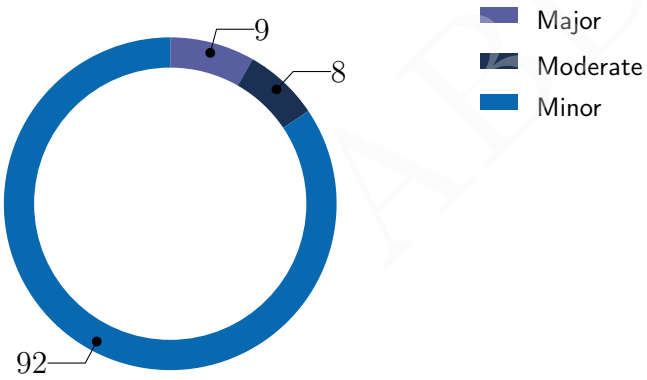
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
14th February 2022

We've been asked to review certain files in a Github repo. All critical and major issues as well as many other ones were fixed. Some issues are additionally commented by the client why they can be ignored.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Readability	Info
CVF-3	Major	Suboptimal	Info
CVF-4	Minor	Bad naming	Fixed
CVF-5	Minor	Overflow/Underflow	Fixed
CVF-6	Minor	Suboptimal	Fixed
CVF-7	Minor	Flaw	Fixed
CVF-8	Moderate	Flaw	Info
CVF-9	Minor	Suboptimal	Info
CVF-10	Minor	Suboptimal	Info
CVF-11	Minor	Suboptimal	Info
CVF-12	Moderate	Flaw	Fixed
CVF-13	Minor	Suboptimal	Info
CVF-14	Minor	Suboptimal	Info
CVF-15	Moderate	Flaw	Fixed
CVF-16	Minor	Suboptimal	Info
CVF-17	Minor	Suboptimal	Info
CVF-18	Minor	Suboptimal	Fixed
CVF-19	Moderate	Flaw	Fixed
CVF-20	Minor	Suboptimal	Info
CVF-21	Minor	Suboptimal	Info
CVF-22	Minor	Suboptimal	Info
CVF-23	Minor	Flaw	Info
CVF-24	Minor	Suboptimal	Info
CVF-25	Minor	Flaw	Fixed
CVF-26	Minor	Flaw	Info
CVF-27	Minor	Procedural	Info

ID	Severity	Category	Status
CVF-28	Minor	Procedural	Info
CVF-29	Minor	Bad datatype	Info
CVF-30	Minor	Bad datatype	Info
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Procedural	Fixed
CVF-33	Minor	Documentation	Fixed
CVF-34	Minor	Documentation	Fixed
CVF-35	Major	Suboptimal	Fixed
CVF-36	Minor	Procedural	Fixed
CVF-37	Minor	Procedural	Info
CVF-38	Minor	Documentation	Info
CVF-39	Minor	Documentation	Fixed
CVF-40	Minor	Procedural	Info
CVF-41	Minor	Procedural	Info
CVF-42	Minor	Suboptimal	Fixed
CVF-43	Minor	Suboptimal	Info
CVF-44	Minor	Suboptimal	Info
CVF-45	Minor	Readability	Fixed
CVF-46	Minor	Procedural	Info
CVF-47	Minor	Procedural	Info
CVF-48	Minor	Suboptimal	Info
CVF-49	Minor	Suboptimal	Info
CVF-50	Moderate	Flaw	Info
CVF-51	Minor	Procedural	Fixed
CVF-52	Minor	Readability	Fixed
CVF-53	Minor	Documentation	Fixed
CVF-54	Minor	Flaw	Fixed
CVF-55	Minor	Suboptimal	Fixed
CVF-56	Minor	Suboptimal	Fixed
CVF-57	Major	Flaw	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Suboptimal	Info
CVF-59	Minor	Unclear behavior	Fixed
CVF-60	Major	Suboptimal	Fixed
CVF-61	Major	Flaw	Info
CVF-62	Moderate	Flaw	Info
CVF-63	Moderate	Flaw	Info
CVF-64	Minor	Suboptimal	Fixed
CVF-65	Minor	Suboptimal	Info
CVF-66	Major	Suboptimal	Fixed
CVF-67	Minor	Documentation	Fixed
CVF-68	Moderate	Flaw	Fixed
CVF-69	Major	Suboptimal	Fixed
CVF-70	Minor	Procedural	Info
CVF-71	Minor	Suboptimal	Info
CVF-72	Minor	Bad datatype	Info
CVF-73	Minor	Bad datatype	Info
CVF-74	Minor	Overflow/Underflow	Info
CVF-75	Minor	Suboptimal	Info
CVF-76	Minor	Suboptimal	Info
CVF-77	Minor	Readability	Fixed
CVF-78	Minor	Suboptimal	Info
CVF-79	Minor	Suboptimal	Info
CVF-80	Major	Suboptimal	Fixed
CVF-81	Minor	Suboptimal	Info
CVF-82	Minor	Suboptimal	Fixed
CVF-83	Minor	Suboptimal	Fixed
CVF-84	Minor	Procedural	Info
CVF-85	Minor	Suboptimal	Info
CVF-86	Minor	Bad datatype	Info
CVF-87	Minor	Bad datatype	Info

ID	Severity	Category	Status
CVF-88	Minor	Bad datatype	Info
CVF-89	Minor	Bad datatype	Info
CVF-90	Minor	Suboptimal	Fixed
CVF-91	Minor	Suboptimal	Fixed
CVF-92	Minor	Suboptimal	Fixed
CVF-93	Minor	Overflow/Underflow	Fixed
CVF-94	Minor	Readability	Fixed
CVF-95	Minor	Suboptimal	Fixed
CVF-96	Minor	Procedural	Fixed
CVF-97	Minor	Documentation	Fixed
CVF-98	Minor	Documentation	Fixed
CVF-99	Minor	Suboptimal	Info
CVF-100	Major	Suboptimal	Fixed
CVF-101	Minor	Procedural	Info
CVF-102	Minor	Procedural	Info
CVF-103	Minor	Suboptimal	Fixed
CVF-104	Minor	Suboptimal	Info
CVF-105	Minor	Procedural	Info
CVF-106	Minor	Bad datatype	Info
CVF-107	Minor	Suboptimal	Info
CVF-108	Minor	Readability	Fixed
CVF-109	Minor	Suboptimal	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	January 11, 2022	D. Khovratovich	Initial Draft
0.2	January 12, 2022	D. Khovratovich	Minor revision
1.0	January 12, 2022	D. Khovratovich	Release
1.1	January 12, 2022	D. Khovratovich	Date fix
2.0	January 12, 2022	D. Khovratovich	Release
2.1	February 6, 2022	D. Khovratovich	Add client comments
3.0	February 6, 2022	D. Khovratovich	Release
3.1	February 14, 2022	D. Khovratovich	Add client comments
4.0	February 14, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the repository with the following files:

- base/EligibilityBase.sol
- base/ERC20LookthroughUpgradable.sol
- Digitizer.sol
- HomogenizerFactoryClone.sol
- HomogenizerUpgradable.sol

The fixes were provided at commit e12116a.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.

- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** ERC20LookthroughUpgradable.sol

Recommendation Should be "^0.8.0" according to a common best practice, unless there is something special about this particular version. Also relevant for the next files: Digitizer.sol, HomogenizerUpgradable.sol, HomogenizerFactoryClone.sol, EligibilityBase.sol.
Client Comment Changed to "0.8.6", mirroring the approached used in Uniswap V3.

Listing 1:

```
2 pragma solidity ^0.8.6;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** ERC20LookthroughUpgradable.sol

Recommendation These two calls are equivalent to: __ERC20_init(name, symbol);
Client Comment The recommendation is appropriate; but when implemented, it actually increases gas expense from 464,829 to 465,179 (when calling createHomogenizer).

Listing 2:

```
21 __Context_init_unchained();
__ERC20_init_unchained(name, symbol);
```

3.3 CVF-3

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description Calculating the full function signature and then it's 4-bytes selector on every invocation is a waste of gas.
Recommendation Consider calculating the selector once inside the "__ERC20Lookthrough_init" function and storing it instead of the function name. Then use the "abi.encodeWithSelector" function instead of "abi.encodeWithSignature".
Client Comment The recommendation is appropriate; but when implemented, it actually increases gas expense.

Listing 3:

```
33 string memory _fullFunctionSignature = string(abi.encodePacked(
     ↪ _underlyingBalanceFunction, '(address)'));
   bytes memory payload = abi.encodeWithSignature(
     ↪ _fullFunctionSignature, address(this));
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** ERC20LookthroughUpgradable.sol

Description The name is confusing as rounding an integer number doesn't make sense.
Recommendation Considier renaming to "round10" or something like this.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 4:

```
40 function round(uint256 value) private pure returns (uint256) {
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** ERC20LookthroughUpgradable.sol

Description Phantom overflow is possible here, i.e. a situation when the final result would fit into the destination type, while some intermediary calculations overflow. For example, for value = 2^256 - 2 the "value + 5" calcualtion will overflow, while the final output would be 2^256 - 6 which fits into the "uint256" type. Consider calcuation like this: uint r = value % 10; return value - r + r / 5 * 10;
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 5:

```
41 return ((value + 5) / 10) * 10;
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The "_totalSupply" value is read from the storage twice.
Recommendation Consider reading once and reusing.
Client Comment Implemented per recommendation.

Listing 6:

```
46 if (_totalSupply > 0) {
       return round((_balances[account] * totalSupply()) /
          ↪ _totalSupply);
```

3.7 CVF-7

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description Due to rounding errors, it is not guaranteed that the sum of all the balances equals to the total supply. Moreover, as rounding up is possible here, the sum of all the balances could be bigger than the total supply.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 7:

```
47 return round((_balances[account] * totalSupply()) / _totalSupply
   ↪ );
```

3.8 CVF-8

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description Most of existing DeFi applications, including Uniswap V3, wouldn't operate normally with tokens whose balances may changes by themselves, i.e. not as a result of a transfer.
Recommendation Consider keeping token balances stable and modifying the amount of assets per token.
Client Comment Homogenizers are designed to rebase. The "lookthrough" feature is what makes PandiFi's homgenizer tokens unique in the DeFi ecosystem. While they do follow the ERC20 API, we acknowledge that most DEXes today cannot accommodate this feature.

Listing 8:

```
47 return round((_balances[account] * totalSupply()) / _totalSupply
   ↪ );
```

3.9 CVF-9

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. as it is anyway not possible to transfer any tokens from zero address.
Client Comment The recommendation is appropriate, but this was done to mimic the Open Zeppelin parent contract i.e., ERC20Upgradeable.sol.

Listing 9:

```
55 require(sender != address(0), 'ERC20Lookthrough: transfer from
   ↪ the zero address');
```

3.10 CVF-10

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. There is nothing wrong in transferring tokens to zero address.
Client Comment The recommendation is appropriate, but this was done to mimic the Open Zeppelin parent contract i.e., ERC20Upgradeable.sol.

Listing 10:

```
56 require(recipient != address(0), 'ERC20Lookthrough: transfer to
   ↪ the zero address');
```

3.11 CVF-11

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. There is nothing wrong in transferring zero tokens when the total supply is zero.
Client Comment These are required because _totalSupply/uTotalSupply appear in the denominator of subsequent commands; transferring a 0 balance ERC20 is semantically vague.

Listing 11:

```
58 require(_totalSupply > 0 && uTotalSupply > 0, 'ERC20Lookthrough:
   ↪  there are no outstanding tokens to transfer');
```

3.12 CVF-12

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description The function calls "_beforeTokenTranfser" but doesn't call "_afterTokenTransfer".
Recommendation Consider calling "_afterTokenTranfser" at the end of the function.
Client Comment Implemented per recommendation.

Listing 12:

```
59 _beforeTokenTransfer(sender, recipient, amount);
```

3.13 CVF-13

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The "_totalSupply" value is read from the storage several times.
Recommendation Consider reading once and reusing.
Client Comment The recommendation is appropriate, but when implemented, it has no impact on gas expense. Perhaps more recent versions of the compiler handle this issue under the hood.

Listing 13:

```
58 require(_totalSupply > 0 && uTotalSupply > 0, 'ERC20Lookthrough:
   ↪  there are no outstanding tokens to transfer');

60 uint256 senderBalance = (_balances[sender] * uTotalSupply) /
   ↪ _totalSupply; // see spreadsheet example

62 _balances[sender] = ((senderBalance - amount) * _totalSupply) /
   ↪ uTotalSupply; // see spreadsheet example
_balances[recipient] += (amount * _totalSupply) / uTotalSupply;
   ↪ // see spreadsheet example
```

3.14 CVF-14

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The sender balance is not rounded to a factor of 10, so it could be different from what the "balanceOf" function returns for the sender.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 14:

```
60 uint256 senderBalance = (_balances[sender] * uTotalSupply) /
   ↪ _totalSupply; // see spreadsheet example
```

3.15 CVF-15

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description Due to rounding errors, the amount subtracted from the sender's balance could differ from the amount added to the recipient's balance, so the "_totalSupply" value could become inaccurate after the transfer.
Recommendation Consider calculating the scaled transfer amount as: uint256 scaledAmount = amount * _totalSupply / totalSupply (); and then transferring this precise amount.
Client Comment Implemented per recommendation.

Listing 15:

```
62 _balances[sender] = ((senderBalance - amount) * _totalSupply) /
   ↪ uTotalSupply; // see spreadsheet example
_balances[recipient] += (amount * _totalSupply) / uTotalSupply;
   ↪ // see spreadsheet example
```

3.16 CVF-16

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. There is nothing wrong in minting to the zero address.
Client Comment The recommendation is appropriate, but this was done to mimic the Open Zeppelin parent contract i.e., ERC20Upgradeable.sol.

Listing 16:

```
69 require(account != address(0), 'ERC20: mint to the zero address
   ↪ ');
```

3.17 CVF-17

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The "_totalSupply" value is read from the storage several times.
Recommendation Consider reading once and reusing.
Client Comment The recommendation is appropriate, but was not implemented as it has a de minimis impact on gas expense.

Listing 17:

```
72 if (uTotalSupply == 0 || _totalSupply == 0) {

74     _totalSupply += amount;

76     _balances[account] += (amount * _totalSupply) / uTotalSupply
       ↪ ;
       _totalSupply += (amount * _totalSupply) / uTotalSupply;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The expression "amount * _totalSupply / uTotalSupply" is calcualted twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 18:

```
76 _balances[account] += (amount * _totalSupply) / uTotalSupply;
   _totalSupply += (amount * _totalSupply) / uTotalSupply;
```

3.19 CVF-19

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description The sender and the recipient addresses are interchanged in this call.
Recommendation Should be: _afterTokenTransfer(address(0), account, amount);
Client Comment Implemented per recommendation.

Listing 19:

```
80 _afterTokenTransfer(account, address(0), amount); // added to
     ↪ mirror OZ library
```

3.20 CVF-20

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. It is anyway impossible to burn from the zero address.
Client Comment The recommendation is appropriate, but this was done to mimic the Open Zeppelin parent contract i.e., ERC20Upgradeable.sol.

Listing 20:

```
85 require(account != address(0), 'ERC20: burn from the zero
     ↪ address');
```

3.21 CVF-21

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The "_totalSupply" value is read from the storage several times.
Recommendation Consider reading once and reusing.
Client Comment See CVF-17.

Listing 21:

```
87 require(_totalSupply > 0 && uTotalSupply > 0, 'ERC20Lookthrough:
   ↪ there are no outstanding tokens to burn');

89 uint256 accountBalance = (_balances[account] * uTotalSupply) /
   ↪ _totalSupply;

91 _balances[account] = ((round(accountBalance) - round(amount)) *
   ↪ _totalSupply) / uTotalSupply;
   _totalSupply -= (amount * _totalSupply) / uTotalSupply;
```

3.22 CVF-22

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description This check is redundant. There is nothing wrong with burning zero tokens when the total supply is zero.
Client Comment See CVF-11.

Listing 22:

```
87 require(_totalSupply > 0 && uTotalSupply > 0, 'ERC20Lookthrough:
   ↪ there are no outstanding tokens to burn');
```

3.23 CVF-23

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description The account balance is not rounded to a factor of 10, so it could be different from what the "balanceOf" function returns for the account.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 23:

```
89 uint256 accountBalance = (_balances[account] * uTotalSupply) /
     ↪ _totalSupply;
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC20LookthroughUpgradable.sol

Description The expressions "round(accountBalance)" and "round(amount)" are calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 24:

```
90 require(round(accountBalance) >= round(amount), 'ERC20: burn
     ↪ amount exceeds balance');
_balances[account] = ((round(accountBalance) - round(amount)) *
     ↪ _totalSupply) / uTotalSupply;
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description Due to rounding errors, the "_balances[account]" decrease could differ from the "_totalSupply" decrease, so the "_totalSupply" value could become inaccurate after the burn.
Recommendation Consider applying the very same decrease to both, the balance and the total supply.
Client Comment Implemented per recommendation.

Listing 25:

```
91 _balances[account] = ((round(accountBalance) - round(amount)) *
   ↪ _totalSupply) / uTotalSupply;
_totalSupply -= (amount * _totalSupply) / uTotalSupply;
```

3.26 CVF-26

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** ERC20LookthroughUpgradable.sol

Description Transfer and mint operations don't round transaction amounts, while burn does round it.
Recommendation Consider using consistent rounding strategy across all the operations.
Client Comment round() was removed entirely; round() addressed an exception in an early version of the code which isn't applicable anymore.

Listing 26:

```
90 require(round(accountBalance) >= round(amount), 'ERC20: burn
   ↪ amount exceeds balance');
_balances[account] = ((round(accountBalance) - round(amount)) *
   ↪ _totalSupply) / uTotalSupply;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't review this file.
Client Comment Acknowledged.

Listing 27:

```
12 import './interfaces/IDigitizer.sol';
   import './interfaces/IHomogenizer.sol';
```

3.28 CVF-28

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't see the definition of "LoanSchema.TokenData" struct.
Client Comment Acknowledged.

Listing 28:

```
23 mapping(uint256 => LoanSchema.TokenData) private _tokenData; //
     ↪ public via getTokenData

62 function getTokenData(uint256 tokenId) public view override
     ↪ returns (LoanSchema.TokenData memory) {
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Digitizer.sol

Recommendation The type of this variable should be "IERC20".
Client Comment We decided not to implement this currently; note that we have "ERC20 stablecoin = ERC20(stablecoinAddress)" when the address is used, so we're effectively leveraging the type checking here.

Listing 29:

```
31 address public override stablecoinAddress;
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Digitizer.sol

Recommendation The type of the "stablecoinAddress_" should be "IERC20".
Client Comment We decided not to implement this currently; note that we have "ERC20 stablecoin = ERC20(stablecoinAddress)" when the address is used, so we're effectively leveraging the type checking here.

Listing 30:

```
55 constructor(string memory name, string memory symbol, address
   ↪ stablecoinAddress_) ERC721(name, symbol) {
```

3.31 CVF-31

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Recommendation Using the emitter's address as an event parameter is redundant, as every event already have an implicit indexed parameter containing the emitter's address.
Client Comment Implemented per recommendation.

Listing 31:

```
58 emit NewDigitizer(address(this));
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Digitizer.sol

Recommendation These functions should emit some events.
Client Comment Implemented per recommendation.

Listing 32:

```
74 function setTokenURI(uint256 tokenId, string calldata URI)
   ↪ external override {

80 function _setReservation(uint256 tokenId, address owner) private
   ↪ {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Digitizer.sol

Description The error message is inaccurate, as not only an admin, but also a servicer could modify a token URI.
Client Comment Implemented per recommendation.

Listing 33:

```
76 require(hasRole(SERVICER_ROLE, msg.sender) || hasRole(
   ↪ DEFAULT_ADMIN_ROLE, msg.sender), 'Digitizer: Only the
   ↪ admin can modify servicing data');
```

3.34 CVF-34

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Digitizer.sol

Description The error message is inaccurate, as not only an admin, but also a trader could modify a token reservation.
Client Comment Implemented per recommendation.

Listing 34:

```
81 require(hasRole(TRADER_ROLE, msg.sender) || hasRole(
   ↪ DEFAULT_ADMIN_ROLE, msg.sender), 'Digitizer: Only the
   ↪ admin can create a reservation');
```

3.35 CVF-35

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description Insider this function an authorization check is performed on every loop iteration, which is basically waste of gas.
Recommendation Consider performing the authorization check once before the loop.
Client Comment Implemented per recommendation.

Listing 35:

```
88 _setReservation(tokenIds[i], owner);
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Digitizer.sol

Recommendation This function should emit some event.
Client Comment Implemented per recommendation.

Listing 36:

```
93 function setStablecoinAddress(address newStablecoinAddress)
     ↪ external override nonReentrant {
```

3.37 CVF-37

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't see the definition of "LoanSchema.Original" struct.
Client Comment Implemented per recommendation.

Listing 37:

```
98 function _mintLoan(address to, LoanSchema.Original memory
     ↪ originalTokenData,

181 function mint(address to, LoanSchema.Original[] memory
     ↪ originalTokenDatas, LoanSchema.Current[] memory
     ↪ currentTokenDatas,
```

3.38 CVF-38

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Digitizer.sol

Description The semantics of the returned value is unclear.
Recommendation Consider giving it a descriptive name and/or adding a documentation comment.
Client Comment The documentation for mint() already has "returns an array of the new loan tokens' ERC721 IDs"

Listing 38:

```
99 LoanSchema.Current memory currentTokenData, string memory
     ↪ offChainDataURI) private returns (uint256) {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Digitizer.sol

Description The error message is inaccurate, as not only an admin, but also a minter could modify a token reservation.
Client Comment Implemented per recommendation.

Listing 39:

```
100 require(hasRole(MINTER_ROLE, msg.sender) || hasRole(
    ↪ DEFAULT_ADMIN_ROLE, msg.sender), 'Digitizer: Only the
    ↪ admin can mint a new NFT');
```

3.40 CVF-40

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't see the definition of "LoanSchema.Current" struct.
Client Comment Acknowledged.

Listing 40:

```
99 LoanSchema.Current memory currentTokenData, string memory
    ↪ offChainDataURI) private returns (uint256) {
```

3.41 CVF-41

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't see the definition of "LoanSchema.Current" struct.
Client Comment Acknowledged.

Listing 41:

```
99          LoanSchema.Current memory currentTokenData, string
              ↪ memory offChainDataURI) private returns (uint256)
              ↪ {

132 function _receiveCashFlow(uint256 tokenId, LoanSchema.Current
      ↪ memory newTokenData, LoanSchema.CashFlow memory
      ↪ cashFlowData, address collectionWallet) private {

181 function mint(address to, LoanSchema.Original[] memory
      ↪ originalTokenDatas, LoanSchema.Current[] memory
      ↪ currentTokenDatas,

203 function receiveCashFlow(uint256[] memory tokenIds, LoanSchema.
      ↪ Current[] memory newTokenDatas,
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "_tokenIds.current()" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 42:

```
102 uint256 newItemId = _tokenIds.current();
totalSupply = _tokenIds.current(); // alternative to
      ↪ ERC721Enumerable
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The "_tokenData[newItemId]" expression is calculated several times.
Recommendation Consider calculating once and reusing. Alternatively, using struct literal assignment: _tokenData[newItemId] = LoanSchema.TokenData ({ ... });
Client Comment The recommendation is appropriate, but when a struct literal version is implemented, it results in an increase from 454,442 to 457,662 in gas expense (mint function).

Listing 43:

```
105 _tokenData[newItemId].lastUpdate = uint40(block.timestamp);
    _tokenData[newItemId].dateAtIssuance = uint40(block.timestamp);
    _tokenData[newItemId].UPBAtIssuance = currentTokenData.UPB;

109 _tokenData[newItemId].current = currentTokenData;
110 _tokenData[newItemId].original = originalTokenData;
```

3.44 CVF-44

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Recommendation This selector is basically a constant and could be hardcoded rather than calculated.
Client Comment The PandiFi team tried implementing "bytes constant onForgoCashFlowSelector = abi.encodeWithSignature('onForgoCashFlowFromDigitizer()')"; in theory it should be better, but in practice it resulted in an increase from 136,945 to 136,968 in gas expense (receiveCashFlow function).

Listing 44:

```
120 bytes memory selector = abi.encodeWithSignature('
    ↪ onForgoCashFlowFromDigitizer()');
```

3.45 CVF-45

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Digitizer.sol

Recommendation This code could be simplified as: return success && bytes4(selector) == bytes4(returnData);
Client Comment Implemented per recommendation.

Listing 45:

```
122 if (success && bytes4(selector) == bytes4(returnData)) {
        return true;
    } else {
        return false;
    }
```

3.46 CVF-46

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description It is not ensured that the "returnData.length" is 4.
Recommendation Consider adding such check.
Client Comment This doesn't appear to be an issue; the default is that cash flows are NOT forgone i.e., if bytes4(selector) is not equal to bytes4(returnData) for ANY reason, then the cash flows are sent to the owner.

Listing 46:

```
122 if (success && bytes4(selector) == bytes4(returnData)) {
```

3.47 CVF-47

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Digitizer.sol

Description We didn't see the definition of "LoanSchema.CashFlow" struct.
Client Comment Acknowledged.

Listing 47:

```
132 function _receiveCashFlow(uint256 tokenId, LoanSchema.Current
    ↪ memory newTokenData, LoanSchema.CashFlow memory
    ↪ cashFlowData, address collectionWallet) private {

204         LoanSchema.CashFlow[] memory cashFlowDatas, address
              ↪ collectionWallet) external override nonReentrant {
```

3.48 CVF-48

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "_tokenData[tokenId]" is calculated several times.
Recommendation Consider calculating once and reusing.
Client Comment See CVF-43.

Listing 48:

```
133 if (newTokenData.creditScore != 0 && newTokenData.creditScore !=
    ↪  _tokenData[tokenId].current.creditScore) {
      _tokenData[tokenId].lastCreditScoreChangeDate = uint40(block
        ↪ .timestamp);

140   uint256 beginningUpb = _tokenData[tokenId].current.UPB;
      _tokenData[tokenId].current = newTokenData;

166   _tokenData[tokenId].lastUpdate = uint40(block.timestamp);
```

3.49 CVF-49

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "_tokenData[tokenId].current]" is calculated several times.
Recommendation Consider calculating once and reusing.
Client Comment See CVF-43.

Listing 49:

```
133 if (newTokenData.creditScore != 0 && newTokenData.creditScore !=
    ↪  _tokenData[tokenId].current.creditScore) {

140     uint256 beginningUpb = _tokenData[tokenId].current.UPB;
        _tokenData[tokenId].current = newTokenData;
```

3.50 CVF-50

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Digitizer.sol

Description In case the "newTokenData.creditScore" value is zero, the "_tokenData[_tokenId].current.creditScore" is set to zero but "_tokenData[tokenId].lastCreditScoreChangeDate" is not updated.
Recommendation Consider either not to set the token credit score to zero, or updating the last credit score change date.
Client Comment This doesn't appear to be an issue. "0" means there is no new credit score. Hence, we would not want to update the lastCreditScoreChangeDate.

Listing 50:

```
133 if (newTokenData.creditScore != 0 && newTokenData.creditScore !=
    ↪  _tokenData[tokenId].current.creditScore) {
        _tokenData[tokenId].lastCreditScoreChangeDate = uint40(block
        ↪ .timestamp);
    }

141     _tokenData[tokenId].current = newTokenData;
```

3.51 CVF-51

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Digitizer.sol

Recommendation This check should be done at the very beginning of the function.
Client Comment Implemented per recommendation.

Listing 51:

```
136 require(_exists(tokenId), 'Digitizer: Requested cash flow for a
    ↪ nonexistent token');
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Digitizer.sol

Recommendation This could be simplified as: require (isLocked[tokenId]);
Client Comment Implemented per recommendation.

Listing 52:

```
138 if (isLocked[tokenId]) {

168 } else {
        revert('Digitizer: Loan must be locked prior to distributing
          ↪  cash flows');
170 }
```

3.53 CVF-53

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Digitizer.sol

Description The error message is inaccurate, as not only an admin, but also a servicer could receive cash flow.
Client Comment Implemented per recommendation.

Listing 53:

```
139 require(hasRole(SERVICER_ROLE, msg.sender) || hasRole(
    ↪ DEFAULT_ADMIN_ROLE, msg.sender), 'Digitizer: Only the
    ↪ admin can modify servicing data');
```

3.54 CVF-54

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Digitizer.sol

Description The code above shouldn't be executed in case the token is not locked.
Client Comment Implemented per recommendation.

Listing 54:

```
138 if (isLocked[tokenId]) {
```

3.55 CVF-55

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "beginningUpb - newTokenData.UPB" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 55:

```
144 totalUPB -= (beginningUpb - newTokenData.UPB);
upbOfOwner[ownerOf(tokenId)] -= (beginningUpb - newTokenData.UPB
    ↪ );
```

3.56 CVF-56

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "newTokenData.UPB - beginningUpb" is calculated twice.
Recommendation Consider calculation once and reusing.
Client Comment Implemented per recommendation.

Listing 56:

```
147 totalUPB += (newTokenData.UPB - beginningUpb);
upbOfOwner[ownerOf(tokenId)] += (newTokenData.UPB - beginningUpb
    ↪ );
```

3.57 CVF-57

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** Digitizer.sol

Recommendation The conditions, divider, and multiplier here should be precomputed once the stable coin is set. Calculating these values and especially obtaining the stable coin decimals every time is waste of gas.
Client Comment "uint256 stablecoinDecimals = stablecoin.decimals()" was implemented; the full version as recommended by auditor actually has higher gas expense.

Listing 57:

```
150 if (stablecoin.decimals() < PANDIFI_DECIMALS) {
        cashFlow = cashFlow / 10**(PANDIFI_DECIMALS - stablecoin.
            ↪ decimals());
    } else if (stablecoin.decimals() > PANDIFI_DECIMALS) {
        cashFlow = cashFlow * 10**(stablecoin.decimals() -
            ↪ PANDIFI_DECIMALS);
```

3.58 CVF-58

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Description This variable is redundant. Just use "stablecoin" instead as its type already implements the "IERC20" interface.
Client Comment The original version implements "using SafeERC20 for IERC20" i.e., we're explicitly trying to avoid the pitfalls described https://soliditydeveloper.com/safe-erc20 and https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca

Listing 58:

```
155 IERC20 safeStablecoin = IERC20(stablecoinAddress);
```

3.59 CVF-59

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Digitizer.sol

Description The transfer should be made only when cashFlow > 0, i.e. under the "if" statement.
Client Comment Implemented per recommendation.

Listing 59:

```
156 safeStablecoin.safeTransferFrom(collectionWallet, address(this),
    ↪ cashFlow); // must get approval first
if (cashFlow > 0) {
```

3.60 CVF-60

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description Cash flow is transferred twice: once from the collection wallet to this contract and another time from this contract to the final recipient.
Recommendation Consider performing a single transfer directly from the collection wallet to the final recipient.
Client Comment Implemented per recommendation.

Listing 60:

```
156 safeStablecoin.safeTransferFrom(collectionWallet, address(this),
    ↪ cashFlow); // must get approval first

159         safeStablecoin.safeTransfer(getRoleMember(SWEEPER_ROLE,
                ↪ 0), cashFlow);

161         safeStablecoin.safeTransfer(ownerOf(tokenId), cashFlow);
```

3.61 CVF-61

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** Digitizer.sol

Description This basically implements a part of the homogenizer logic.
Recommendation Consider removing this logic form here, so the cash flow will go to homogenizer, and them in the homogenizer code implement a logic to forward the cash flow to the sweeper.
Client Comment At best, this seems to be a subjective suggestion; at worst, forwarding the cash flow through a homogenizer is a waste of gas. ABDK advises: "Homogenizer doesn't need to forward each tranche of a cash flow separately, but may rather accumulate the received cash until the sweeper will withdraw it." However, there could be an infinite number of potential homogenizers, so this is impractical.

Listing 61:

```
158 if (_checkOnForgoCashFlowFromDigitizer(ownerOf(tokenId))) {
        safeStablecoin.safeTransfer(getRoleMember(SWEEPER_ROLE, 0),
            ↪ cashFlow);
```

3.62 CVF-62

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Digitizer.sol

Description Taking all the cash flow from the owners of the homogenizer tokens is unfair. This will stimulate users deploying their own versions of homogenizers.
Recommendation Consider giving at least part of the cash flow to the owners of homogenizer tokens. This doesn't necessary requires distributing the cash flow among all the token holders. The money could just be added to the homogenizer portfolio increasing the value of each tokens, so token holders will get their parts of the cash flow when selling or redeeming homogenizer tokens.
Client Comment It's OK if people create their own versions of homogenizers. A critical feature of our homogenizers is that they are not investable assets / securities. One way to achieve this is to strip them of cash flow rights.

Listing 62:

```
158 if (_checkOnForgoCashFlowFromDigitizer(ownerOf(tokenId))) {
        safeStablecoin.safeTransfer(getRoleMember(SWEEPER_ROLE, 0),
            ↪ cashFlow);
```

3.63 CVF-63

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Digitizer.sol

Description It is not ensured that the token with given ID does exist.
Recommendation Consider adding such check.
Client Comment True, but note super._transfer also doesn't have this check. See Open Zeppelin's ERC721 implementation.

Listing 63:

```
174 function _transfer(address from, address to, uint256 tokenId)
    ↪ internal override {
```

3.64 CVF-64

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The expression "_tokenData[tokenId].current.UPB" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 64:

```
175 upbOfOwner[from] -= _tokenData[tokenId].current.UPB;
    upbOfOwner[to] += _tokenData[tokenId].current.UPB;
```

3.65 CVF-65

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Digitizer.sol

Description Passing a single array of structs with two fields would be more efficient than two parallel arrays, and also would make the length check unnecessary.
Client Comment The recommendation is appropriate, but we're going to keep the code as is for V1 as the impact isn't significant.

Listing 65:

```
181 function mint(address to, LoanSchema.Original[] memory
    ↪ originalTokenDatas, LoanSchema.Current[] memory
    ↪ currentTokenDatas,
          string[] memory offChainDataURIs) public override
            ↪ nonReentrant returns (uint256[] memory) {

203 function receiveCashFlow(uint256[] memory tokenIds, LoanSchema.
    ↪ Current[] memory newTokenDatas,
          LoanSchema.CashFlow[] memory cashFlowDatas, address
            ↪ collectionWallet) external override nonReentrant {
```

3.66 CVF-66

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The "_mintLoan" call performs an authorization check on every iteration of the loop, which is waste of gas.
Recommendation Consider performing the authorization check once before the loop.
Client Comment Implemented per recommendation.

Listing 66:

```
187 newItemIds[i] = _mintLoan(to, originalTokenDatas[i],
    ↪ currentTokenDatas[i], offChainDataURIs[i]);
```

3.67 CVF-67

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Digitizer.sol

Description The error message is inaccurate, as not only an admin, but also a servicer could lock loans.
Client Comment Implemented per recommendation.

Listing 67:

```
194 require(hasRole(SERVICER_ROLE, msg.sender) || hasRole(
    ↪ DEFAULT_ADMIN_ROLE, msg.sender), 'Digitizer: Only the
    ↪ admin can lock loans');
```

3.68 CVF-68

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Digitizer.sol

Description The locked token counter is increased even for those tokens that are already locked.
Recommendation Consider either forbidding locking already locked tokens or incrementing the counter only for those tokens that were not not already locked.
Client Comment Implemented per recommendation.

Listing 68:

```
195 lockedTokenCount += tokenIds.length;

198     isLocked[tokenIds[i]] = true;
```

3.69 CVF-69

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Digitizer.sol

Description The "_receiveCashFlow" call performs an authorization check on every iteration of the loop, which is waste of gas.
Recommendation Consider performing the authorization check once before the loop.
Client Comment Implemented per recommendation.

Listing 69:

```
207 _receiveCashFlow(tokenIds[i], newTokenDatas[i], cashFlowDatas[i
    ↪ ], collectionWallet);
```

3.70 CVF-70

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** HomogenizerUpgradable.sol

Description We didn't review these files.
Client Comment Acknowledged.

Listing 70:

```
6 import './interfaces/IHomogenizer.sol';
  import './interfaces/IDigitizer.sol';
```

3.71 CVF-71

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description The same constant is defined in the "Digitizer" contract.
Recommendation Consider defining it in a single place and using from there.
Client Comment The recommendation is appropriate, but we're going to keep the code as is for V1. Moreover, it's not clear that the recommendation won't increase gas expense. See CVF-76.

Listing 71:

```
15 uint256 private constant PANDIFI_DECIMALS = 6;
```

3.72 CVF-72

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerUpgradable.sol

Recommendation The type of this variable should be "IERC20".
Client Comment The recommendation is appropriate, but we're going to keep the code as is for V1. Note that we have "ERC20 stablecoin = ERC20(stablecoinAddress)" when the address is used, so we're effectively leveraging the type checking here.

Listing 72:

```
16 address public stablecoinAddress;
```

3.73 CVF-73

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerUpgradable.sol

Recommendation The type of this variable could be more specific.
Client Comment It's not clear what you'd gain by making it more specific. No change made.

Listing 73:

```
17 address private _homogenizerFactoryCloneAddress;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** HomogenizerUpgradable.sol

Description Overflow is possible here.
Client Comment No change was made because recent versions of the Solidity compiler should handle overflow issues.

Listing 74:

```
38 homogenizerParameters.maxRate = uint32(maxRate);
   homogenizerParameters.minRate = uint32(minRate);
40 homogenizerParameters.buydownRatio = uint32(buydownRatio);
```

3.75 CVF-75

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description This call only logs an event but doesn't change the blockchain state.
Recommendation Consider removing this call.
Client Comment The recommendation was not implemented. Homogenizers are created infrequently, so the potential for excessive gas expense is minimal.

Listing 75:

```
43 _mint(msg.sender, 0); // initial supply is 0
```

3.76 CVF-76

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Recommendation Consider using the expression instead of the s hardcoded value. Solidity compiler is smart enough to calculate constant expressions at compile time.
Client Comment This recommendation doesn't work in practice. When the change is implemented, gas expense increases e.g., caling homogenize() increases gas expense from 511,025 to 511,819.

Listing 76:

```
48 return 8; // = PANDIFI_DECIMALS + 2
```

3.77 CVF-77

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** HomogenizerUpgradable.sol

Description This code could be simplified as: return noteRateCriteria && _isEligibleAndValid(tokenId, _underlyingContractAddress, homogemizerParametes.eligibilityContractAddress);
Recommendation This code could be simplified as: return noteRateCriteria && _isEligibleAndValid(tokenId, _underlyingContractAddress, homogemizerParametes.eligibilityContractAddress);
Client Comment Implemented per recommendation.

Listing 77:

```
71 if (noteRateCriteria &&
       _isEligibleAndValid(tokenId, _underlyingContractAddress,
         ↪ homogenizerParameters.eligibilityContractAddress)
   ) {return true;} else {return false;}
```

3.78 CVF-78

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Recommendation Consider using expressions in denominators, rather than hardcoded values. Solidity compiler is smart enough to calculate constant expressions at compile time.
Client Comment See CVF-76; expressions (even with constants) appear to use more gas.

Listing 78:

```
81 return digitizer.getTokenData(tokenId).original.llpaPercentage *
     ↪ digitizer.getTokenData(tokenId).current.UPB /10**8 + //
     ↪ 8=PANDIFI_DECIMALS+2

83   (homogenizerParameters.maxRate - _rate(tokenId) + digitizer.
       ↪ getTokenData(tokenId).current.servicingRate) / 10**14;
       ↪ // 14=2*PANDIFI_DECIMALS+2
```

3.79 CVF-79

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description A function that just returns its own selector looks weird. Those who know how to call this function already know its result.
Client Comment This is subjective. We're copying the ERC721 pattern "onERC721Received" i.e., as desribed in the Open Zeppelin documentation, "This function MUST return the function selector".

Listing 79:

```
88 function onForgoCashFlowFromDigitizer() public virtual override
     ↪ returns (bytes4) {
     return this.onForgoCashFlowFromDigitizer.selector;
90 }
```

3.80 CVF-80

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description The conditions, the divider, and the multiplier should be calculated once during initialization.
Recommendation Calculating them on every usage is waste of gas.
Client Comment Added "uint256 stablecoinDecimals = stablecoin.decimals()". Also see CVF#57.

Listing 80:

```
95  if (stablecoin.decimals() != PANDIFI_DECIMALS) {
        if (stablecoin.decimals() < PANDIFI_DECIMALS) {
            return value / 10**(PANDIFI_DECIMALS - stablecoin.
              ↪ decimals());

99          return value * 10**(stablecoin.decimals() -
              ↪ PANDIFI_DECIMALS);
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description The value "cashHeld[tokenid]" that was just written to the storage is read back, which is waste of gas.
Recommendation Consider caching the value in a local variable and reusing.
Client Comment The recommendation is appropriate, but the impact on gas expense is de minimis in practice.

Listing 81:

```
114 cashHeld[tokenId] = currentCashRequired(tokenId);

116 safeStablecoin.safeTransferFrom(ownerOfTokenId, address(this),
      ↪ _convertDecimals(cashHeld[tokenId])); // get approval
      ↪ first
```

3.82 CVF-82

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description The expression "digitizer.reservationOwner(tokenId)" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 82:

```
131 require(digitizer.reservationOwner(tokenId) == address(0) || //
      ↪ No reservations
          digitizer.reservationOwner(tokenId) == endUserAddress ||
            ↪  // Reservation owned by endUserAddress
```

3.83 CVF-83

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerUpgradable.sol

Description The expression "cashHeld[tokenId]" is calculated several times.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 83:

```
142 if (cashHeld[tokenId] > homogenizationFeeCurrent) {

145         _convertDecimals(cashHeld[tokenId] -
              ↪ homogenizationFeeCurrent)

149     safeStablecoin.safeTransfer(endUserAddress, _convertDecimals
          ↪ (cashHeld[tokenId]));
```

3.84 CVF-84

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** HomogenizerFactoryClone.sol

Description We didn't review this file.
Client Comment Acknowledged.

Listing 84:

```
7 import './interfaces/IHomogenizerFactory.sol';
```

3.85 CVF-85

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description The same constant is defined in the "Digitizer" contract.
Recommendation Consider defining it in a single place and using from there.
Client Comment The recommendation is appropriate, but we're going to keep the code as is for V1. Also see CVF-71.

Listing 85:

```
13 uint256 private constant PANDIFI_DECIMALS = 6;
```

3.86 CVF-86

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerFactoryClone.sol

Recommendation The value type for this mapping could be more specific.
Client Comment It's not clear what you'd gain by making it more specific. No change made.

Listing 86:

```
14 mapping(bytes => address) private uniqueHomogenizerAddress; //
    ↪ see getUniqueHomogenizerAddress below
```

3.87 CVF-87

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerFactoryClone.sol

Recommendation The types for these variables could be more specific.
Client Comment It's not clear what you'd gain by making it more specific. No change made.

Listing 87:

```
15 address immutable homogenizerImplementation; // address of the
    ↪ homogenizer contract proxy implementation using ERC-1167
```

3.88 CVF-88

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerFactoryClone.sol

Recommendation The type of this variable should be "IDigitizer".
Client Comment See CVF-29.

Listing 88:

```
18 address public override digitizerAddress;
```

3.89 CVF-89

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** HomogenizerFactoryClone.sol

Recommendation The argument type could be more specific.
Client Comment It's not clear what you'd gain by making it more specific. No change made.

Listing 89:

```
30 constructor(address digitizerAddress_) {
```

3.90 CVF-90

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Recommendation Using the emitter's address as an event parameter is redundant, as every event already have an implicit indexed parameter containing the emitter's address.
Client Comment Implemented per recommendation.

Listing 90:

```
40 emit NewFactory(address(this), digitizerAddress_);
```

3.91 CVF-91

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description This function is very inefficient as it copies the string data byte by byte and perform range checks on every byte.
Recommendation Here is how this functions could be optimized: https://gist.github.com/3sGgpQ8H/07c7949c81228578556837400b153776
Client Comment This code is not needed once the version recommended per CVF-92 is implemented.

Listing 91:

```
43 function _substring(string memory str, uint256 startIndex,
   ↪ uint256 endIndex) private pure returns (string memory) {
```

3.92 CVF-92

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description This function is very inefficient.
Recommendation Here is how it could be optimized: https://gist.github.com/3sGgpQ8H/68ce6113eb1e2e303aa9ba5187d37ddd
Client Comment Implemented per recommendation.

Listing 92:

```
52 function _rateToString(uint256 rate, uint256 digitsToDisplay,
   ↪ string memory endCharacter,
```

3.93 CVF-93

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** HomogenizerFactoryClone.sol

Description This will underflow in case digitsToDisplay > storageDecimals.
Client Comment Implemented per recommendation; see CVF-92.

Listing 93:

```
55 string memory rateTruncString = (rate / 10**(storageDecimals -
   ↪ digitsToDisplay)).toString(); // i.e., truncated past 3
   ↪ decimals
```

3.94 CVF-94

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** HomogenizerFactoryClone.sol

Recommendation This could be simplified as: bytes memory rateTruncBytes = bytes (rateTruncString);
Client Comment Implemented per recommendation; see CVF-92.

Listing 94:

```
58 bytes memory rateTruncBytes = abi.encodePacked(bytes(
   ↪ rateTruncString));
```

3.95 CVF-95

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description This code is very inefficient as it copies the whole string at every loop iteration.
Client Comment Implemented per recommendation; see CVF-92.

Listing 95:

```
59 for (uint256 i = 0; i < minLength - length; i++) {
60     rateTruncBytes = abi.encodePacked('0', rateTruncBytes);
   }
```

3.96 CVF-96

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** HomogenizerFactoryClone.sol

Description These functions should emit some events.
Client Comment Implemented per recommendation.

Listing 96:

```
74 function setEligibilityContractRestrictions(
   ↪ EligibilityContractRestrictions calldata
   ↪ eligibilityContractRestrictions_) external override
   ↪ nonReentrant {

83 function setHomogenizerRestrictions(HomogenizerRestrictions
   ↪ calldata homogenizerRestrictions_) external override
   ↪ nonReentrant {
```

3.97 CVF-97

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** HomogenizerFactoryClone.sol

Description The error message is inaccurate, as not only an admin but also a trader could set eligibility restrictions.
Client Comment Implemented per recommendation.

Listing 97:

```
78 'HomogenizerFactory: Only admin can call
   ↪ setEligibilityContractRestrictions');
```

3.98 CVF-98

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** HomogenizerFactoryClone.sol

Description The error message is inaccurate, as not only an admin but also a trader could set homogenizer restrictions.
Client Comment Implemented per recommendation.

Listing 98:

```
87 'HomogenizerFactory: Only admin can call
    ↪ setHomogenizerRestrictions');
```

3.99 CVF-99

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description The expression "eligibilityContractRegistry[symbol]" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment The recommendation is appropriate, but gas expenses actually increases when implemented.

Listing 99:

```
95 require(bytes(eligibilityContractRegistry[symbol].name).length
    ↪ == 0 && eligibilityContractRegistry[symbol].
    ↪ contractAddress == address(0),
```

3.100 CVF-100

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description The "digitizer.totalSupply()" expression is calculated multiple times.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 100:

```
102 if (digitizer.totalSupply() >= eligibilityContractRestrictions.
    ↪ sampleSizeOfTestedLoans) {
      uint256 startIndex = digitizer.totalSupply() -
        ↪ eligibilityContractRestrictions.
        ↪ sampleSizeOfTestedLoans;

105   for (uint256 i = startIndex; i < digitizer.totalSupply(); i
        ↪ ++) {
```

3.101 CVF-101

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** HomogenizerFactoryClone.sol

Description Only one load is tested here, while more loans could be available.
Recommendation Consider testing all the existing loans.
Client Comment The recommendation isn't clear; sampleSizeOfTestedLoans is the number of loans tested.

Listing 101:

```
103 uint256 startIndex = digitizer.totalSupply() -
    ↪ eligibilityContractRestrictions.sampleSizeOfTestedLoans;
```

3.102 CVF-102

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** HomogenizerFactoryClone.sol

Recommendation This variable should be declared inside the loop where it is only used.
Client Comment The recommendation is approrpriate, but gas expense actually increases when implemented.

Listing 102:

```
104 uint256 startGas;
```

3.103 CVF-103

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Description The homogenizer key is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Implemented per recommendation.

Listing 103:

```
137 require(uniqueHomogenizerAddress[abi.encodePacked(
    ↪ eligibilityContractSymbol, maxRate, minRate, buydownRatio,
    ↪  useArmMargin, useCurrentRate)] == address(0),

166 uniqueHomogenizerAddress[abi.encodePacked(
    ↪ eligibilityContractSymbol, maxRate, minRate, buydownRatio,
    ↪  useArmMargin, useCurrentRate)] = joinedAddresses[0];
```

3.104 CVF-104

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** HomogenizerFactoryClone.sol

Recommendation It would be more efficient to pass a single array of structures with two fields instead of two parallel arrays. This would also make the length check redundant.
Client Comment The recommendation is appropriate, but we're going to keep the code as is for V1 as the impact is de minimis. Moreover, it's not clear that the gas won't end up increasing. See CVF-76.

Listing 104:

```
174 function homogenizeBatch(uint256[] memory tokenIds, address[]
    ↪ memory homogenizerAddresses, bool allOrNone) external
    ↪ override {

190 function dehomogenizeBatch(uint256[] memory tokenIds, address[]
    ↪ memory homogenizerAddresses, bool allOrNone) external
    ↪ override {

208     uint256[] calldata tokenIds,
        address[] calldata eligibilityContractAddresses

220 function isEligibleByHomogenizer(uint256[] calldata tokenIds,
    ↪ address[] calldata homogenizerAddresses) external view
    ↪ override returns (bool[] memory) {
```

3.105 CVF-105

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** EligibilityBase.sol

Description We didn't review this file.
Client Comment Acknowledged.

Listing 105:

```
3 import '../interfaces/IEligibilityBase.sol';
```

3.106 CVF-106

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** EligibilityBase.sol

Recommendation The types of the "digitizerAddress" and "eligibilityContractAddress" should be more specific.
Client Comment It's not clear what you'd gain by making it more specific. No change made.

Listing 106:

```
8 function _isEligibleAndValid(uint256 tokenId, address
    ↪ digitizerAddress, address eligibilityContractAddress)
    ↪ public view override returns (bool) {
```

3.107 CVF-107

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** EligibilityBase.sol

Recommendation This utility function doesn't need to be public.
Client Comment This was done deliberately in case a third party application or a UI wants to check eligibililty in real time.

Listing 107:

```
8 function _isEligibleAndValid(uint256 tokenId, address
    ↪ digitizerAddress, address eligibilityContractAddress)
    ↪ public view override returns (bool) {
```

3.108 CVF-108

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** EligibilityBase.sol

Recommendation These lines could be simplified as: return abi.decode(returnData, (bool));
Client Comment Implemented per recommendation.

Listing 108:

```
11 bool eligibilityCriteria = abi.decode(returnData, (bool));
if (eligibilityCriteria && success) {return true;} else {return
    ↪ false;}
```

3.109 CVF-109

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** EligibilityBase.sol

Description The "&& success" part is redundant, as "success" is guaranteed to be true here.
Client Comment Implemented per recommendation.

Listing 109:

```
12 if (eligibilityCriteria && success) {return true;} else {return
   ↪ false;}
```