ABDK CONSULTING

SMART CONTRACT AUDIT

Primitive V2

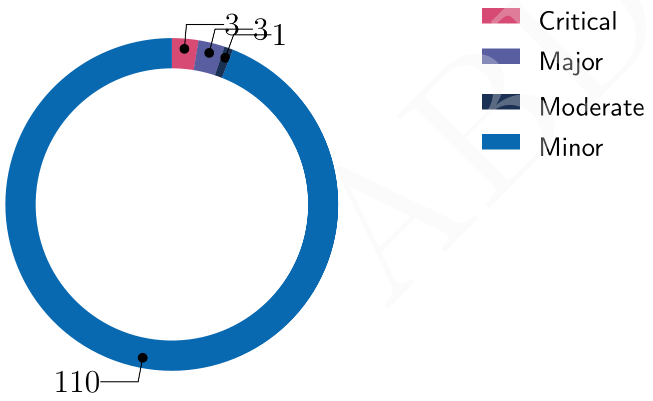
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
1st December 2021

We've been asked to review the 18 files in a Github repo. We found 3 critical, 3 major, and a few less important issues. 3 critical and 2 major issues were fixed.



Findings

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

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

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

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Info
CVF-89	Minor	Procedural	Info
CVF-90	Minor	Procedural	Fixed
CVF-91	Minor	Documentation	Info
CVF-92	Minor	Bad naming	Fixed
CVF-93	Minor	Bad datatype	Fixed
CVF-94	Minor	Suboptimal	Info
CVF-95	Minor	Bad datatype	Info
CVF-96	Minor	Bad datatype	Info
CVF-97	Minor	Suboptimal	Info
CVF-98	Minor	Suboptimal	Fixed
CVF-99	Minor	Bad datatype	Fixed
CVF-100	Minor	Flaw	Fixed
CVF-101	Minor	Readability	Fixed
CVF-102	Minor	Documentation	Fixed
CVF-103	Minor	Documentation	Fixed
CVF-104	Minor	Unclear behavior	Fixed
CVF-105	Minor	Readability	Fixed
CVF-106	Minor	Readability	Fixed
CVF-107	Minor	Readability	Fixed
CVF-108	Minor	Readability	Fixed
CVF-109	Minor	Suboptimal	Fixed
CVF-110	Minor	Suboptimal	Fixed
CVF-111	Minor	Readability	Fixed
CVF-112	Minor	Bad naming	Opened
CVF-113	Minor	Documentation	Opened
CVF-114	Minor	Procedural	Fixed
CVF-115	Minor	Readability	Fixed
CVF-116	Minor	Bad naming	Info
CVF-117	Minor	Suboptimal	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	October 30, 2021	D. Khovratovich	Initial Draft
0.2	October 31, 2021	D. Khovratovich	Minor revision
1.0	October 31, 2021	D. Khovratovich	Release
1.1	December 1, 2021	D. Khovratovich	Add repository links
2.0	December 1, 2021	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 following files:

- CumulativeNormalDistribution.sol
- Reserve.sol
- IPrimitiveEngineEvents.sol
- IPrimitiveEngineActions.sol
- PrimitiveEngine.sol
- IPrimitiveEngineErrors.sol
- PrimitiveFactory.sol
- ReplicationMath.sol
- IPrimitiveEngineView.sol
- IERC20.sol
- IPrimitiveRepayCallback.sol
- IPrimitiveEngine.sol
- IPrimitiveFactory.sol
- Transfers.sol
- SafeCast.sol
- Units.sol
- Position.sol
- Margin.sol

The fixes were provided in the repository.

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
- **Category** Procedural
- **Status** Info
- **Source** CumulativeNormalDistribution.sol

Description Should be “0̂.8.0” or “0̂.8.6’ according to a common best practice. Also relevant for the next files: Reserve.sol, IPrimitiveEngineEvents.sol, IPrimitiveEngineActions.sol, PrimitiveEngine.sol, IPrimitiveEngineErrors.sol, PrimitiveFactory.sol, ReplicationMath.sol, IPrimitiveEngineView.sol, IERC20.sol, IPrimitiveSwapCallback.sol, IPrimitiveRepayCallback.sol, IPrimitiveLiquidityCallback.sol, IPrimitiveDepositCallback.sol, IPrimitiveCreateCallback.sol, IPrimitiveBorrowCallback.sol, IPrimitiveEngine.sol, IPrimitiveFactory.sol, Transfers.sol, SafeCast.sol, Units.sol, Position.sol, Margin.sol.
Client Comment No changes to compiler version

Listing 1:

```
solidity 0.8.6;
```

3.2 CVF-2

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

Recommendation These variables should be turned into compile-time constants.

Listing 2:

```
int128 p = 0x53dd02a4f5ee2e46;
int128 one = uint256(1).fromUInt();
int128 two = uint256(2).fromUInt();
int128 a3 = 0x16a09e667f3bcc908;

    int128 a3 = 0x16a09e667f3bcc908;
    int128 a4 = -0x17401c57014c38f14;
    int128 a5 = 0x10fb844255a12d72e;

    int128 one = ABDKMath64x64.fromUInt(1);
    int128 a1 = 0x413c831bb169f874;
    int128 a2 = -0x48d4c730f051a5fe;

int128 half = 0x8000000000001060; // 0.5

int128 a0 = 0x26A8F3C1F21B39C0; // 0.151015506
int128 a1 = -0x87C57E5DA70D0FE0; // -0.530357263
int128 a2 = 0x15D71F57212414CA0; // 1.365020123
int128 b0 = 0x21D0A04B0E9BA0F0; // 0.132089632
int128 b1 = -0xC2BF5D74C7247680; // -0.760732499
```

3.3 CVF-3

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

Description The precision of the hardcoded quotients seems to be worse than the maximum possible of 64.64-bit binary fixed-point numbers.

Recommendation Consider calculating the quotients more precisely, for example using this calculator: https://keisan.casio.com/calculator

Listing 3:

```
int128 p = 0x53dd02a4f5ee2e46;

int128 a3 = 0x16a09e667f3bcc908;

    int128 a3 = 0x16a09e667f3bcc908;
    int128 a4 = -0x17401c57014c38f14;
    int128 a5 = 0x10fb844255a12d72e;

    int128 a1 = 0x413c831bb169f874;
    int128 a2 = -0x48d4c730f051a5fe;

int128 a0 = 0x26A8F3C1F21B39C0; // 0.151015506
int128 a1 = -0x87C57E5DA70D0FE0; // -0.530357263
int128 a2 = 0x15D71F57212414CA0; // 1.365020123
int128 b0 = 0x21D0A04B0E9BA0F0; // 0.132089632
int128 b1 = -0xC2BF5D74C7247680; // -0.760732499
```

3.4 CVF-4

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

Recommendation These constants are not used in this function (only in the erf computation) so this comment should be removed.

Listing 4:

```
//a1 = 0.254829592, a2 = - 0.284496736, a3 = 1.421413741, a4 = - 1.453152027, a5 = 1.061405429/
```

3.5 CVF-5

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

Recommendation This variable should not be called a3 since the a3 value by formula is 1.421... so, as we divide by sqrt(2), it should be called sqrt2 or smth like this

Listing 5:

```
int128 a3 = 0x16a09e667f3bcc908;
```

3.6 CVF-6

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** CumulativeNormalDistribution.sol

Recommendation The subexpression “one.div (two)” should be turned into a compile-time constant.

Listing 6:

```
int128 result = (one.div(two)).mul(one.add(erf));
```

3.7 CVF-7

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

Recommendation This value is incorrect as a3 is not equal to sqrt(2). It should be 0x16be1c55bae156b65.

Listing 7:

```
int128 a3 = 0x16a09e667f3bcc908;
```

3.8 CVF-8

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

Description It is actually e(̂-z2̂), not e-̂2z in the code.
Recommendation Fix the comment

Listing 8:

```
int128 result; // 1 - t * (step2 * e^-2z)
```

3.9 CVF-9

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

Description The brackets around “z” are redundant.
Recommendation Consider removing them.

Listing 9:

```
result = one.sub(t.mul(step2.mul(((z).pow(2).neg()).exp())));
```

3.10 CVF-10

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

Description Probably this function should revert on invalid inputs.

Listing 10:

```
function getInverseCDF(int128 p) internal pure returns (int128) {
```

3.11 CVF-11

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

Recommendation The value is not precise. The precise value for 0.5 would be "0x8000000000000000".

Listing 11:

```
int128 half = 0x8000000000001060; // 0.5
```

3.12 CVF-12

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

Recommendation "r.mul(r)" would be more efficient than "r.pow(2)".

Listing 12:

```
int128 result = q.mul(a2.add((a1.mul(r).add(a0)).div((r.pow(2).add(b1.mul(r)).add(b0)))));
```

3.13 CVF-13

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

Recommendation This error should probably be moved to "IPrimitiveEngineErrors.sol".

Listing 13:

```
/// @notice Thrown on attempting to supply more liquidity than is allowed
error LiquidityError();
```

3.14 CVF-14

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Reserve.sol

Description The name is too generic.
Recommendation Consider renaming to "ReserveData" or "ReserveState".
Client Comment No changes, this library struct is only gettable by using Reserve.Data, which makes it clearer.

Listing 14:

```
struct Data {
```

3.15 CVF-15

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

Description These function signatures are formatted differently from the other functions in this file.
Recommendation Consider using consistent formatting across the code.
Client Comment No changes.

Listing 15:

```
function update(Data storage res, uint32 blockTimestamp)
    ↪ internal {

function addFloat(Data storage reserve, uint256 delLiquidity)
    ↪ internal {

function removeFloat(Data storage reserve, uint256 delLiquidity)
    ↪  internal {

function borrowFloat(Data storage reserve, uint256 delLiquidity)
    ↪  internal {

function repayFloat(Data storage reserve, uint256 delLiquidity)
    ↪ internal {
```

3.16 CVF-16

- **Severity** Critical
- **Category** Overflow/Underflow
- **Status** Fixed
- **Source** Reserve.sol

Description Overflow is desired in “+=” operations (though, not practically possible). However, overflow is actually possible and not desired in “*” operations, as the multiplications performed module 21̂28 here.
Recommendation Consider explicitly casting reserve and liquidity amounts to the “uint256“ type before multiplying them by “deltaTime”.

Listing 16:

```
// overflow is desired

        res.cumulativeRisky += res.reserveRisky * deltaTime;
        res.cumulativeStable += res.reserveStable * deltaTime;
        res.cumulativeLiquidity += res.liquidity * deltaTime;
```

3.17 CVF-17

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

Recommendation This assignment should also be under the “if” statement.

Listing 17:

```
res.blockTimestamp = blockTimestamp;
```

3.18 CVF-18

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

Description This function basically implements two functions: one to swap risky tokens to stable token and another to swap back.
Recommendation Splitting into two functions would make the code more efficient and easier to read.
Client Comment No changes.

Listing 18:

```
function swap(
    Data storage reserve,
    bool riskyForStable,
    uint256 deltaIn,
    uint256 deltaOut,
    uint32 blockTimestamp
) internal {
```

3.19 CVF-19

- **Severity** Critical
- **Category** Procedural
- **Status** Fixed
- **Source** Reserve.sol

Recommendation The cumulative values should be updated before updating the reserve and liquidity amounts. Otherwise the new amounts are applied to the time interval before the new amounts were actually set.

Listing 19:

```
update(reserve, blockTimestamp);

update(reserve, blockTimestamp);

update(reserve, blockTimestamp);
```

3.20 CVF-20

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** Reserve.sol

Recommendation The values 1000 and 800 should be named constants or even immutable variables set in the constructor.

Listing 20:

```
if ((reserve.float * 1000) / reserve.liquidity > 800) revert LiquidityError();
```

3.21 CVF-21

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

Recommendation Events are usually named via nouns such as: "Creation", "CurveCreation", "NewCurve", or just "Curve"; "TimestampUpdate", "Deposit", "Withdrawal", etc.

Listing 21:

```
event Created(address indexed from, uint256 indexed strike, uint256 sigma, uint256 indexed maturity);

event UpdatedTimestamp(bytes32 indexed poolId, uint32 indexed timestamp);

event Deposited(address indexed from, address indexed recipient, uint256 delRisky, uint256 delStable);

event Withdrawn(address indexed from, address indexed recipient, uint256 delRisky, uint256 delStable);

event Allocated(

event Removed(address indexed from, bytes32 indexed poolId, uint256 delRisky, uint256 delStable);

event Supplied(address indexed from, bytes32 indexed poolId, uint256 delLiquidity);

event Claimed(address indexed from, bytes32 indexed poolId, uint256 delLiquidity);

event Borrowed(address indexed recipient, bytes32 indexed poolId, uint256 delLiquidity, uint256 premium);

event Repaid(
```

3.22 CVF-22

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

Description The way how pool identifier is generated is insignificant implementation details.
Recommendation Consider just referring to it as pool ID in comments.
Client Comment No changes

Listing 22:

```
/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with

/// @param poolId      Keccak hash of the option parameters of a curve to interact with
```

3.23 CVF-23

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

Description Indexing a timestamp parameter looks useless.
Recommendation Consider no indexing it.

Listing 23:

```
event UpdatedTimestamp(bytes32 indexed poolId, uint32 indexed timestamp);
```

3.24 CVF-24

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

Description These events are formatted differently from the other events in this file.
Recommendation Consider using consistent formatting across the code.

Listing 24:

```
event Allocated(
    address indexed from,
    address indexed recipient,
    bytes32 indexed poolId,
    uint256 delRisky,
    uint256 delStable
);

event Swap(
    address indexed from,
    bytes32 indexed poolId,
    bool indexed riskyForStable,
    uint256 deltaIn,
    uint256 deltaOut
);

event Repaid(
    address indexed from,
    address indexed recipient,
    bytes32 indexed poolId,
    uint256 delLiquidity,
    uint256 premium
);
```

3.25 CVF-25

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

Description The number formats for these values are unclear.
Recommendation Consider explaining in the documentation comment.

Listing 25:

```
/// @param strike      Strike price of the option to calibrate to
/// @param sigma       Volatility of the option to calibrate to

/// @param delta       Call option delta, change in option value wrt to a 1% change in underlying value

    uint256 strike,
    uint64 sigma,

    uint256 delta,
```

3.26 CVF-26

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

Description The way how pool identifier is generated is insignificant implementation details.
Recommendation Consider just referring to it as pool ID in comments.
Client Comment No changes.

Listing 26:

```
/// @return poolId       Keccak256 hash of the parameters (engine, strike, sigma, and maturity)

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with

/// @param  poolId       Keccak hash of the option parameters of a curve to interact with
```

3.27 CVF-27

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

Description The returned values don't have names, however they are referred by named in the documentation comment.
Recommendation Consider giving names to the returned values.

Listing 27:

```
/// @return delRisky     Amount of risky tokens that were allocated
/// delStable            Amount of stable tokens that were allocated

) external returns (uint256, uint256);
```

3.28 CVF-28

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

Description These function definitions are formatted differently from the other functions in this interface.
Recommendation Consider using consistent formatting across the code.
Client Comment Updated, no changes to remove.

Listing 28:

```
function remove(bytes32 poolId, uint256 delLiquidity) external returns (uint256 delRisky, uint256 delStable);

function supply(bytes32 poolId, uint256 delLiquidity) external;

function claim(bytes32 poolId, uint256 delLiquidity) external;
```

3.29 CVF-29

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

Description It is unclear how swap directions are mapped to the boolean argument values.
Recommendation Consider explicitly explaining what directions are mapped to "true" and "false" values.

Listing 29:

```
/// @param riskyForStable Whether to do a risky to stable token swap, or stable to risky swap
```

3.30 CVF-30

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

Description This argument controls not only where the premium will be charged from, but also where the stable asset will be transferred to.
Recommendation Consider explaining this in the comment. Also, consider introducing another argument to control these two behaviors separately.

Listing 30:

```
/// @param fromMargin Use margin risky balance to pay premium?

/// @param fromMargin Whether the `msg.sender` uses their margin balance, or must send tokens
```

3.31 CVF-31

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

Recommendation This variable should have type IPrimitiveFactory.
Client Comment Data types, we are keeping to using addresses.

Listing 31:

```
address public immutable override factory;
```

3.32 CVF-32

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

Recommendation This low-level functionality should be moved to a separate utility contract inherited by this contract.
Client Comment No changes, not used anywhere else in codebase.

Listing 32:

```
uint8 private unlocked = 1;

modifier lock() {
```

3.33 CVF-33

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

Recommendation These functions are overcomplicated. The common practice is to just call “balanceOf” on the token contracts.
Client Comment No changes, these are gas-optimized to reduce an extra extcodesize opcode.

Listing 33:

```
function balanceRisky() private view returns (uint256) {

function balanceStable() private view returns (uint256) {
```

3.34 CVF-34

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

Recommendation This code should be moved to a library similar to the "Transfers" library.
Client Comment No changes.

Listing 34:

```
(bool success, bytes memory data) = risky.staticcall(
    abi.encodeWithSelector(IERC20.balanceOf.selector, address(this))
);
if (!success && data.length < 32) revert BalanceError();
return abi.decode(data, (uint256));

(bool success, bytes memory data) = stable.staticcall(
    abi.encodeWithSelector(IERC20.balanceOf.selector, address(this))
);
if (!success && data.length < 32) revert BalanceError();
return abi.decode(data, (uint256));
```

3.35 CVF-35

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

Description Overflow is possible here.
Recommendation Consider using safe conversion.
Client Comment Overflow is known, no changes.

Listing 35:

```
blockTimestamp = uint32(block.timestamp);
```

3.36 CVF-36

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

Description There is no explicit range check for the “delta” argument.
Recommendation Consider adding such check.

Listing 36:

```
uint256 delta,

delRisky = 1e18 - delta; // 0 <= delta <= 1
```

3.37 CVF-37

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

Description There is no explicit range check for the “delLiquidity” argument.
Recommendation Consider adding such check.

Listing 37:

```
uint256 delLiquidity,

positions.fetch(msg.sender, poolId).allocate(delLiquidity - 1000); // burn 1000 wei, at cost of msg.sender
```

3.38 CVF-38

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

Description This will revert in case the maturity time is in the past, however this will not throw the “PoolExpiredError” designed specifically for this situation.
Recommendation Consider explicitly checking that the maturity time is no in the past and throwing PoolExpiredError in case it is.

Listing 38:

```
uint32 tau = cal.maturity - timestamp; // time until expiry
```

3.39 CVF-39

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

Description There is a named constant for the 1e18 value.
Recommendation Consider using it.

Listing 39:

```
delRisky = 1e18 - delta; // 0 <= delta <= 1

delRisky = (delRisky * delLiquidity) / 1e18;
delStable = (delStable * delLiquidity) / 1e18;

    uint256 nextRisky = ((resRisky + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;

        1e18);

    uint256 nextStable = ((resStable + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;

        1e18;

uint256 reserveRisky = (res.reserveRisky * 1e18) / res.liquidity; // risky per 1 liquidity
uint256 reserveStable = (res.reserveStable * 1e18) / res.liquidity; // stable per 1 liquidity
```

3.40 CVF-40

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

Recommendation Consider adding a comment next to the zero value, with the argument name this value is passed for.
Client Comment No Changes.

Listing 40:

```
delStable = ReplicationMath.getStableGivenRisky(0, delRisky, cal.strike, cal.sigma, tau).parseUnits();
```

3.41 CVF-41

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

Description In these lines, multiplication and division for WAD numbers are implemented several times.

Recommendation Consider extracting utility functions for operations with WAD numbers.

Client Comment No Changes.

Listing 41:

```
delRisky = (delRisky * delLiquidity) / 1e18;
delStable = (delStable * delLiquidity) / 1e18;

    uint256 nextRisky = ((resRisky + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;
    uint256 nextStable = ((getStableGivenRisky(details.poolId, nextRisky).parseUnits() * reserve.liquidity) /
        1e18);

    uint256 nextStable = ((resStable + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;
    uint256 nextRisky = (getRiskyGivenStable(details.poolId, nextStable).parseUnits() * reserve.liquidity) /
        1e18;

uint256 reserveRisky = (res.reserveRisky * 1e18) / res.liquidity; // risky per 1 liquidity
uint256 reserveStable = (res.reserveStable * 1e18) / res.liquidity; // stable per 1 liquidity
```

3.42 CVF-42

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

Description The multiplication here could overflow and thus revert the transaction. Also, safe multiplication used here is quite expensive.
Recommendation Consider doing straighforward check: delRisky == 0 —— delStable == 0

Listing 42:

```
if (delRisky * delStable == 0) revert CalibrationError(delRisky, delStable);
```

```
if (delRisky * delStable == 0) revert ZeroDeltasError();
```

```
if (delRisky * delStable == 0) revert ZeroDeltasError();
```

3.43 CVF-43

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

Description It is a good practice to call untrusted external contracts after updating contract's state.

Recommendation Consider putting the callback invocation and subsequent checks after all the state updates.

Listing 43:

```
138 IPrimitiveCreateCallback(msg.sender).createCallback(delRisky,
      ↪ delStable, data);
    if (balanceRisky() < delRisky + balRisky) revert
      ↪ RiskyBalanceError(delRisky + balRisky, balanceRisky());
140 if (balanceStable() < delStable + balStable) revert
      ↪ StableBalanceError(delStable + balStable, balanceStable())
      ↪ ;

159 if (delRisky > 0) balRisky = balanceRisky();
160 if (delStable > 0) balStable = balanceStable();
    IPrimitiveDepositCallback(msg.sender).depositCallback(delRisky,
      ↪ delStable, data); // agnostic payment
    if (balanceRisky() < balRisky + delRisky) revert
      ↪ RiskyBalanceError(balRisky + delRisky, balanceRisky());
    if (balanceStable() < balStable + delStable) revert
      ↪ StableBalanceError(balStable + delStable, balanceStable())
      ↪ ;

202     IPrimitiveLiquidityCallback(msg.sender).allocateCallback(
          ↪ delRisky, delStable, data); // agnostic payment
        if (balanceRisky() < balRisky + delRisky) revert
          ↪ RiskyBalanceError(balRisky + delRisky, balanceRisky())
          ↪ ;
        if (balanceStable() < balStable + delStable)
            revert StableBalanceError(balStable + delStable,
              ↪ balanceStable());

290             IPrimitiveSwapCallback(msg.sender).swapCallback(
                  ↪ details.deltaIn, 0, data); // agnostic payment
                if (balanceRisky() < balRisky + details.deltaIn)
                    revert RiskyBalanceError(balRisky + details.
                      ↪ deltaIn, balanceRisky());
```

(... 301, 372, 374, 417, 419)

3.44 CVF-44

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

Description The expression “delRisky + balRisky” is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 44:

```
if (balanceRisky() < delRisky + balRisky) revert RiskyBalanceError(delRisky + balRisky, balanceRisky());
```

3.45 CVF-45

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

Description The expression “delStable + balStable” is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 45:

```
if (balanceStable() < delStable + balStable) revert StableBalanceError(delStable + balStable, balanceStable());
```

3.46 CVF-46

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** PrimitiveEngine.sol

Recommendation The “1000” value should be a compile-time constant.

Listing 46:

```
positions.fetch(msg.sender, poolId).allocate(delLiquidity - 1000); // burn 1000 wei, at cost of msg.sender
```

3.47 CVF-47

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

Description These variables are not initialized.
Recommendation Consider explicitly initializing them to 0 for readability.
Client Comment Initializing to 0 costs gas.

Listing 47:

```
uint256 balRisky;
uint256 balStable;
```

3.48 CVF-48

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

Description This check should be performed only when delRisky is not zero: if (delRisky ¿ 0 && ...)

Listing 48:

```
if (balanceRisky() < balRisky + delRisky) revert RiskyBalanceError(balRisky + delRisky, balanceRisky());
```

3.49 CVF-49

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

Description "This check should be performed only when delStable is not zero: if (delStable ¿ 0 && ...)"

Listing 49:

```
if (balanceStable() < balStable + delStable) revert StableBalanceError(balStable + delStable, balanceStable());
```

3.50 CVF-50

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

Description This event is emitted even if the deposit is 0.

Listing 50:

```
emit Deposited(msg.sender, recipient, delRisky, delStable);
```

3.51 CVF-51

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

Description This function signature is formatted differently from the other functions in this file.
Recommendation Consider using consistent formatting across the code.
Client Comment No changes.

Listing 51:

```
function remove(bytes32 poolId, uint256 delLiquidity)
```

3.52 CVF-52

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

Recommendation Consider moving all structure definitions to the beginning of the file to simplify code navigation.
Client Comment Since this struct is only used in swap, we are keeping it close by, no changes.

Listing 52:

```
struct SwapDetails {
    bytes32 poolId;
    uint256 deltaIn;
    bool riskyForStable;
    bool fromMargin;
}
```

3.53 CVF-53

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

Description Reverting on operations with zero amounts makes it more dangerous using the contract from other contracts.
Recommendation Consider just doing nothing on zero amounts.

Listing 53:

```
if (deltaIn == 0) revert DeltaInError();

if (delLiquidity == 0) revert ZeroLiquidityError();

if (delLiquidity == 0) revert ZeroLiquidityError();

if (delLiquidity == 0) revert ZeroLiquidityError();
```

3.54 CVF-54

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

Description According to the documentation comment, the "PoolExpiredError" is thrown when calling ‘create‘ with a maturity that is less than the current block.timestamp. However, here it is thrown from the "swap" function.
Recommendation Consider either fixing the documentation comment or trowing some other error here.

Listing 54:

```
if (timestamp > calibrations[details.poolId].maturity + 120)
    ↪ revert PoolExpiredError();
```

3.55 CVF-55

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

Description It would be more efficient to use just "poolId" instead of "details.poolId".
Recommendation Consider either fixing the documentation comment or trowing some other error here.
Client Comment The use of this struct is to avoid stack too deep errors, no changes.

Listing 55:

```
if (timestamp > calibrations[details.poolId].maturity + 120) revert PoolExpiredError();
calibrations[details.poolId].lastTimestamp = timestamp;
emit UpdatedTimestamp(details.poolId, timestamp);

int128 invariant = invariantOf(details.poolId);
Reserve.Data storage reserve = reserves[details.poolId];

    uint256 nextStable = ((getStableGivenRisky(details.poolId, nextRisky).parseUnits() * reserve.liquidity) /

    uint256 nextRisky = (getRiskyGivenStable(details.poolId, nextStable).parseUnits() * reserve.liquidity) /

    int128 nextInvariant = invariantOf(details.poolId); // 4. Important: do invariant check

    emit Swap(msg.sender, details.poolId, details.riskyForStable, details.deltaIn, amountOut);
```

3.56 CVF-56

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PrimitiveEngine.sol

Recommendation The value 9985 should be a compile-time constant.

Listing 56:

```
uint256 nextRisky = ((resRisky + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;

uint256 nextStable = ((resStable + ((details.deltaIn * 9985) / 1e4)) * 1e18) / reserve.liquidity;
```

3.57 CVF-57

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

Description The next values are first translated into delta values, and then (inside the "swap" function) these delta values are added to the current values ending with the next values that were already calculated.
Recommendation Consider refactoring to make the code more efficient.

Listing 57:

```
271 deltaOut = resStable − nextStable;

276 deltaOut = resRisky − nextRisky;

309 reserve.swap(details.riskyForStable, details.deltaIn, amountOut,
      ↪ timestamp);
```

3.58 CVF-58

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

Recommendation Consider implementing an option for a user to deposit the swap outcome into the user's margin account.

Listing 58:

```
286 IERC20(stable).safeTransfer(msg.sender, amountOut); // send
      ↪ proceeds, for callback if needed

297 IERC20(risky).safeTransfer(msg.sender, amountOut); // send
      ↪ proceeds first, for callback if needed
```

3.59 CVF-59

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

Description The expression "balRisky + details.deltaIn" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 59:

```
291 if (balanceRisky() < balRisky + details.deltaIn)
        revert RiskyBalanceError(balRisky + details.deltaIn,
          ↪ balanceRisky());
```

3.60 CVF-60

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

Description These checks look redundant. They may fail only in case a transfer call will transfer more tokens than requested. It is possible for some token contracts that charge transfer fee, but just reverting is a poor way to handle this.
Recommendation Consider removing these checks.

Listing 60:

```
        if (balanceStable() < balStable - amountOut)
            revert StableBalanceError(balStable - amountOut,
                ↪ balanceStable());

        if (balanceRisky() < balRisky - amountOut)
            revert RiskyBalanceError(balRisky - amountOut,
                ↪ balanceRisky());

if (balanceStable() < balStable - delStable)
    revert StableBalanceError(balStable - delStable,
        ↪ balanceStable());

if (balanceStable() < balStable + delStable)
    revert StableBalanceError(balStable + delStable,
        ↪ balanceStable());
```

3.61 CVF-61

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

Description The expression "balStable + details.deltaIn" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 61:

```
if (balanceStable() < balStable + details.deltaIn)
    revert StableBalanceError(balStable + details.deltaIn,
        ↪ balanceStable());
```

3.62 CVF-62

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

Description This condition may never be true, as if invariant ¿ nextInvariant, then nextInvariant.sub (invariant) is negative and thus is guaranteed to be less than Units.MANTISSA_INT, which is positive.
Recommendation Fix the condition.

Listing 62:

```
if (invariant > nextInvariant && nextInvariant.sub(invariant) >=
  ↪  Units.MANTISSA_INT)
```

3.63 CVF-63

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

Description In some cases this variable can be used without explicit initialization.
Recommendation Consider explicitly initializing to zero.
Client Comment Explicitly setting tau to 0 will incur gas, no changes.

Listing 63:

```
uint256 tau;

uint256 tau;

uint256 tau;
```

3.64 CVF-64

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

Description These lines look like plain assignments, while they actually return values.
Recommendation Consider using explicit "return" statements.
Client Comment Not explicitly returning variables, assigning them to the return variables instead. no changes.

Listing 64:

```
reserveStable = ReplicationMath.getStableGivenRisky(invariantLast, reserveRisky, cal.strike, cal.sigma, tau);

reserveRisky = ReplicationMath.getRiskyGivenStable(invariantLast, reserveStable, cal.strike, cal.sigma, tau);

invariant = ReplicationMath.calcInvariant(reserveRisky, reserveStable, cal.strike, cal.sigma, tau);
```

3.65 CVF-65

- **Severity** Critical
- **Category** Overflow/Underflow
- **Status** Fixed
- **Source** PrimitiveEngine.sol

Description The multiplication here is performed in 128-bit numbers which could lead to overflow and revert transaction.
Recommendation Do calculations in 256-bit numbers.

Listing 65:

```
uint256 reserveRisky = (res.reserveRisky * 1e18) / res.liquidity; // risky per 1 liquidity
uint256 reserveStable = (res.reserveStable * 1e18) / res.liquidity; // stable per 1 liquidity
```

3.66 CVF-66

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

Recommendation It would be more helpful and informative to have a separate event for each parameter.

Listing 66:

```
error CalibrationError(uint256 delRisky, uint256 delStable);
```

3.67 CVF-67

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

Recommendation The last value type should be IPrimitiveEngine
Client Comment Data types... keeping to addresses, no changes.

Listing 67:

```
mapping(address => mapping(address => address)) public override getEngine;
```

3.68 CVF-68

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

Recommendation This field should have type “IPrimitiveFactory”.
Client Comment Data types... keeping to addresses, no changes.

Listing 68:

```
address factory;
```

3.69 CVF-69

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

Recommendation These fields should have type “IERC20”.
Client Comment Data types... keeping to addresses, no changes.

Listing 69:

```
address risky;
address stable;
```

3.70 CVF-70

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

Description It is not explicitly checked that the engine for given risky and stable tokens is not yet deployed.
Recommendation Consider performing an explicit check and throw a custom error with descriptive name.

Listing 70:

```
getEngine[risky][stable] = engine;
```

3.71 CVF-71

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

Description While constructor arguments do affect the address of a contract deployed via CREATE2, the address still remains predictable, so, it is possible to use constant (say zero) salt, pass risky and stable token addresses as the constructor arguments, and basically have the same behavior. With this approach, constructor argument will play the salt role. Passing the factory address is anyway redundant, as the engine may obtain it inside the constructor as “msg.sender”.
Client Comment While the constructor args still allow the address to be predictable, doing that calculation on chain is less feasible since we’d need to append the contructor args to the bytecode, per engine, to calculate the address of it. Taking this approach of not using constructor args allows us to avoid this. See: https://github.com/primitivefinance/primitive-v2-core/pull/195 Ref 5.13:

Listing 71:

```
/// @dev              Engine contract should have no constructor
    ↪ args, because this affects the deployed address
```

3.72 CVF-72

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

Description The comment is confusing. Does it mean that the value is in basis point and 1% is represented as 100?

Listing 72:

```
/// @param sigma Volatility scaled by Percentage Mantissa of 1e4, where 1 bip = 100
```

3.73 CVF-73

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

Recommendation This could be simplified as: vol = sigma.divi (10000).mul (sqrtTau);

Listing 73:

```
vol = sigma.fromUInt().mul(sqrtTau).div(Units.PERCENTAGE_INT); // scales down from Mantissa
```

3.74 CVF-74

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

Recommendation Probably some range check for this parameter is needed.
Client Comment Range check for reserve values, if out of the range it will cause a revert.

Listing 74:

```
uint256 reserveRisky,
```

3.75 CVF-75

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

Description The comment is useless.
Recommendation Consider removing it.

Listing 75:

```
int128 input = phi.add(vol); // phi + vol
```

3.76 CVF-76

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

Description These arguments are not documented.
Recommendation Consider describing them in the documentation comment.

Listing 76:

```
uint256 strike,
uint256 sigma,
uint256 tau
```

3.77 CVF-77

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

Description Intermixing two formats for fractional numbers makes code harder to read, less efficient, and more error-prone.
Recommendation Consider using the same format everywhere.

Listing 77:

```
int128 reserve2 = getStableGivenRisky(0, reserveRisky, strike, sigma, tau);
invariant = reserveStable.parseUnits().sub(reserve2);
```

3.78 CVF-78

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

Description The way how pool identifier is generated is insignificant implementation details.
Recommendation Consider just referring to it as pool ID in comments.
Client Comment No changes.

Listing 78:

```
/// @param  poolId      Keccak256 hash of engine, strike price, volatility, and maturity timestamp

/// @param  poolId      Keccak256 hash of engine, strike price, volatility, and maturity timestamp

/// @param poolId       Keccak256 hash of engine, strike price, volatility, and maturity timestamp

/// @param  poolId      Keccak256 hash of engine, strike price, volatility, and maturity timestamp

/// @param  posId       Keccak256 hash of owner address and poolId
```

3.79 CVF-79

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

Description The explanation is misleading. One would think that these arguments are token amounts in reserves, while actually they are token shares in reserves represented as decimal fixed-point numbers with 18 decimals.
Recommendation Consider explaining this in the documentation comment and probably renaming the arguments.

Listing 79:

```
/// @param  reserveRisky Current reserve of risky tokens

/// @param  reserveStable Current reserve of stable tokens
```

3.80 CVF-80

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

Description The current reserves are passed as decimal fixed point numbers with 18 decimals, while the expected reserves are returned as binary fixed-point numbers with 64 binary digits in fractional part. This is confusing and error-prone.
Recommendation Consider using the same fractional format across the code for consistency.

Listing 80:

```
/// @param  reserveRisky Current reserve of risky tokens
/// @return reserveStable Expected stable token reserve

/// @param  reserveStable Current reserve of stable tokens
/// @return reserveRisky Expected risky token reserve
```

3.81 CVF-81

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

Recommendation The return type should be "IPrimitiveFactory".
Client Comment Return types as IPrimitiveFactory, no changes, keep to addresses.

Listing 81:

```
function factory() external view returns (address);
```

3.82 CVF-82

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

Recommendation The return types should be "IERC20".
Client Comment Return types as IERC20, no change.

Listing 82:

```
function risky() external view returns (address);

function stable() external view returns (address);
```

3.83 CVF-83

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** IPrimitiveEngineView.sol

Description The word "block" is redundant. This is just a timestamp when the comulative values were last updated.
Recommendation Consider renaming to "cumulativeTimestamp".
Client Comment BlockTimestamp naming, no changes.

Listing 83:

```
/// blockTimestamp      Timestamp when the cumulative reserve values were last updated

        uint32 blockTimestamp,
```

3.84 CVF-84

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

Description These function declarations are formatted differently from other functions in this file.
Recommendation Consider using consistent formatting across the code.
Client Comment Line length exceeds the limit, so prettier-solidity formats it, on purpose.

Listing 84:

```
function reserves(bytes32 poolId)

function calibrations(bytes32 poolId)

function positions(bytes32 posId)
```

3.85 CVF-85

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

Description Token amounts are usually represented by the "uint256" type. Using "uint128" here doesn't reduce gas cost.
Recommendation Consider using "uint256" for token amounts. Additional comment: Yes, this is about the return types chosen for token amounts. Even if we know that the returned amount will never exceed 128 bits, it would be better to return them as uint256 values because this doesn't cost extra gas, is more conventional, and is less error prone. Note, that when a function returning uint128 is used in an expression, then in come cases Solidity may decide to perform calculations with the returned values module 21̂28 which could cause some hard to find errors.
Client Comment A uint128 vars don't reduce gas, comment: I thought there is gas reductions from packing in the struct? Or are we just talking about return type for the view function?

Listing 85:

```
            uint128 reserveRisky,
            uint128 reserveStable,
            uint128 liquidity,
            uint128 float,
            uint128 debt,

            uint128 float,
            uint128 liquidity,
            uint128 debt

function margins(address account) external view returns (uint128
    ↪ balanceRisky, uint128 balanceStable);
```

3.86 CVF-86

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

Description Block timestamp is a "uint256" value.
Recommendation Consider using "uint256" for timestamps.
Client Comment A uint32 used for timestamp to pack in struct.

Listing 86:

```
uint32 blockTimestamp,
```

3.87 CVF-87

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

Description The number formats of these arguments are unclear.
Recommendation Consider explaining in the documentation comments.

Listing 87:

```
/// @return strike        Strike price of the pool
/// sigma                 Volatility of the pool

            uint128 strike,
            uint64 sigma,
```

3.88 CVF-88

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

Description In Ethereum, timestamps are "uint256" values. Using narrower types here doesn't save gas.
Recommendation Consider using "uint256" for timestamps.
Client Comment A uint32 is used for timestamp to tightly pack the Calibration struct

Listing 88:

```
uint32 maturity,
uint32 lastTimestamp
```

3.89 CVF-89

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** IERC20.sol

Description This function declaration is formatted differently from the other functions declared in this file.
Recommendation Consider using consistent formatting across the code.
Client Comment Formatting in IERC20 is different because of line length exceeding limit, no changes

Listing 89:

```
function transferFrom(
```

3.90 CVF-90

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

Description The formatting is different from other callback.
Recommendation Consider using consistent formatting across the code.

Listing 90:

```
function repayCallback(uint256 delStable, bytes calldata data) external;
```

3.91 CVF-91

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

Recommendation It is a good practice to put a comment into an empty block to explain, why the block is empty.
Client Comment Empty code block, no changes.

Listing 91:

```

```

3.92 CVF-92

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

Recommendation Events are usually named via nouns, such as “Deployment” or “NewEngine”.

Listing 92:

```
event Deployed(address indexed from, address indexed risky, address indexed stable, address engine);
```

3.93 CVF-93

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** IPrimitiveFactory.sol

Recommendation The arguments type should be “IERC20”. The return type should be “IPrimitiveEngine”.

Listing 93:

```
function deploy(address risky, address stable) external returns (address engine);
```

3.94 CVF-94

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

Description This value is redundant, as the engine may obtain it as “msg.sender”.
Client Comment Typings same as 97, no change.

Listing 94:

```
address factory,
```

3.95 CVF-95

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

Recommendation The type of these returned values should be “IERC20”.
Client Comment The factiry parameter, no changes.

Listing 95:

```
address risky,
address stable
```

3.96 CVF-96

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

Recommendation The type of arguments should be "IERC20". The return type should be "IPrimitiveEngine".
Client Comment Typings risky and stable as IERC20, no change.

Listing 96:

```
function getEngine(address risky, address stable) external view returns (address engine);
```

3.97 CVF-97

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

Description This function is redundant here. The ownership model is an insignificant implementation details here.
Recommendation Consider removing this function.
Client Comment The owner(), will remove at the end if not used.

Listing 97:

```
function owner() external view returns (address);
```

3.98 CVF-98

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

Description Actually, the returned value is always true, as the function revert in case of failed transfer.
Recommendation Consider removing the returned value.

Listing 98:

```
/// @return         Whether or not the call was successful
```

3.99 CVF-99

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** SafeCast.sol

Recommendation The return type should be “uint64”.

Listing 99:

```
function toUint64(uint256 x) internal pure returns (uint128 z) {
```

3.100 CVF-100

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

Description This value is actually 364 days, and could be rendered in Solidity as “364 days”. However, a calendar year is either 365 or 366 days, so 364 days year looks weird.

Listing 100:

```
uint256 internal constant YEAR = 31449600; // 1 year in seconds
```

3.101 CVF-101

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

Recommendation These values could be rendered as “1e18”, “1e8”, and “1e4” respectively.

Listing 101:

```
uint256 internal constant DENOMINATOR = 10**18; // wei
uint256 internal constant MANTISSA = 10**8;
uint256 internal constant PERCENTAGE = 10**4;
```

3.102 CVF-102

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

Description This value is approximately 1e9 * 2̂64. The precise value would be 18446744073709551616000000000.
Recommendation Consider using the precise value and explaining what the value is in a comment.

Listing 102:

```
int128 internal constant MANTISSA_INT =
    ↪ 18446744073709500000000000000;
```

3.103 CVF-103

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

Description This value is 1e4 * 2̂64.
Recommendation Consider explaining what the value is in a comment.

Listing 103:

```
int128 internal constant PERCENTAGE_INT =
    ↪ 184467440737095516160000;
```

3.104 CVF-104

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

Description These lines looks like plain assignments, while they actually return values from the functions.
Recommendation Consider using “return” statements.

Listing 104:

```
y = x.divu(DENOMINATOR);

y = (fromInt(x) * 1e18) / MANTISSA;

y = x > 0 ? (x).toUInt() : uint256(0);
```

3.105 CVF-105

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

Recommendation This function could be simplified as: y = x.mulu (DENOMINATOR);

Listing 105:

```
y = (fromInt(x) * 1e18) / MANTISSA;
```

3.106 CVF-106

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

Recommendation This function could be simplified as: return denorm.divu (PERCENTAGE);

Listing 106:

```
int128 numerator = denorm.fromUInt();
int128 denominator = PERCENTAGE.fromUInt();
return numerator.div(denominator);
```

3.107 CVF-107

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

Recommendation This function could be simplified as: return denorm.mulu (PERCENTAGE);

Listing 107:

```
uint256 numerator = denorm.mul(PERCENTAGE_INT).toUInt();
return numerator;
```

3.108 CVF-108

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

Recommendation This function could be simplified as: return quantitySeconds.divu (YEAR);

Listing 108:

```
int128 time = quantitySeconds.fromUInt();
int128 units = YEAR.fromUInt();
return time.div(units);
```

3.109 CVF-109

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

Recommendation This variable should be turned into a compile-time constant.

Listing 109:

```
int128 units = YEAR.fromUInt();
```

3.110 CVF-110

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

Description This function is redundant.
Recommendation Consider removing it.

Listing 110:

```
function fromInt(int128 x) internal pure returns (uint256 y) {
```

3.111 CVF-111

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

Recommendation These lines could be simplified as: y = x.mulu (1e9)

Listing 111:

```
x = x.mul((MANTISSA).fromUInt());
y = x > 0 ? (x).toUInt() : uint256(0);
```

3.112 CVF-112

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** Position.sol

Description The name is too generic.
Recommendation Consider renaming to “PositionData“ or “PositionState“.

Listing 112:

```
struct Data {
```

3.113 CVF-113

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Position.sol

Description The way how pool identifier is generated is insignificant implementation details.
Recommendation Consider just referring to it as pool ID in comments.

Listing 113:

```
/// @param  poolId      Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)

/// @param poolId       Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)

/// @param poolId       Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)

/// @param poolId       Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)

/// @param poolId       Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)

/// @param   poolId     Keccak256 hash of the engine address and pool parameters (strike, sigma, maturity)
```

3.114 CVF-114

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

Description While all these functions are very similar, some of them accept a Data structure as an argument, while other accept a mapping of Data structures and a key in this mapping.
Recommendation Consider making all the functions to accept a Data structure for consistency and let the caller to fetch the Data structure from the mapping.

Listing 114:

```
function fetch(

function allocate(Data storage position, uint256 delLiquidity)
    ↪ internal {

function remove(

function borrow(

function supply(

function claim(

function repay(Data storage position, uint256 delLiquidity)
    ↪ internal {
```

3.115 CVF-115

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

Description This statement looks like a plain assignment while it actually returns value from the function.
Recommendation Consider using a "return" statement instead for readability.

Listing 115:

```
posId = keccak256(abi.encodePacked(account, poolId));
```

3.116 CVF-116

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** Margin.sol

Recommendation The name is too generic. Consider renaming to “MarginData” or “MarginState”.
Client Comment The Margin lib is accessed by the PrimitiveEngine contract, and the struct is accessed with Margin.Data, this is now the only struct named this, and its clear its from the Margin lib.

Listing 116:

```
struct Data {
```

3.117 CVF-117

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

Description When updating two fields stored in a single storage slot, Solidity will perform 4 SLOAD and 2 SSTORE operations, wile one SLOAD and one SSTORE would be enough.
Recommendation Consider storing a single uint256 value and packing/unpacking uint128 values into it manually.
Client Comment Suboptimal storage of two uint128s. Not fixed yet, but will consider.

Listing 117:

```
if (delRisky > 0) margin.balanceRisky += delRisky.toUint128();
if (delStable > 0) margin.balanceStable += delStable.toUint128()
    ↪ ;

if (delRisky > 0) margin.balanceRisky -= delRisky.toUint128();
if (delStable > 0) margin.balanceStable -= delStable.toUint128()
    ↪ ;
```