Zomma Protocol

Audit Report

Wed Nov 29 2023

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

Zomma Protocol Audit Report

1 Executive Summary

1.1 Project Information

Description	Options on chain with the highest capital efficiency
Type	Options
Auditors	ScaleBit
Timeline	Mon Aug 21 2023 - Wed Nov 29 2023
Languages	Solidity
Platform	zkSync Era
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/zomma-protocol/zomma-contracts-hardhat
Commits	a912c068e4f7aeefd8630b7a9b1b2edf77d1013a d467085e33e89dae75bd049eac2a712add78e627 9b97f36afea05b3244dfca535b0ebad3fd08da43 af931ef38d75686d64d47705174145924c93d4be 42daf8bdbcb7edbdfdbb910a2029110e0f033133

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
OMA	contracts/OptionMarket.sol	8ffc0ea575979026e7a1e1bdcb8133208cf77389
SDM	contracts/libraries/SafeDecimalMath.sol	1bdd6b5a622597d57f5ed7fb8d6571ff4963f9a0
SSDM	contracts/libraries/SignedSafeDecimalMath.sol	b074e5b3a91b5861a54af036dbe65f2f6567c09d
LED	contracts/Ledger.sol	926466bc994e3707d9fc791bf61f91a41b15a051
TIM	contracts/utils/Timestamp.sol	735665e83cfb44199527b9f348c7b2d150488252
VPR	contracts/VaultPricer.sol	511e579f3abc9a32349ddd99783d3fbd6a294dc9
OPR	contracts/option-pricer/OptionPricer.sol	1938817618f816c5f96a18ba272bb759b0435c80
SOP	contracts/option-pricer/SignedOptionPricer.sol	51aa1dcdf8b7818652105fcecebbbda82b376861
ICH	contracts/interfaces/IChainlink.sol	6e4350e888fbeab5d50af514d54dbc70af5b51bf
IVA	contracts/interfaces/IVault.sol	c1d2ca360aa4054e77e56c0ec12e24451687d7be
IOP	contracts/interfaces/IOptionPricer.sol	20d8d0dc0fcb5772a609dd6f520218e8397b1318

SPO	contracts/signed/pools/SignedPool.sol	3ce71459012a43b0ed91509bf5691e08b4a53e30
SVA	contracts/signed/SignedVault.sol	69129ce20c4be981c15999a567766520203064b7
PTO	contracts/pools/PoolToken.sol	705e6f408483d915b7c2aa8e006d5b1bf95d9fa9
VAU	contracts/Vault.sol	c8ade9b9f20399ce0fd2fd9c114af1fd8231d050
SVA	contracts/SignatureValidator.sol	2c1fbdd25efa6488e35a97d29cef9bdc551d1df2
RDI	contracts/RewardDistributor.sol	b48b1e16f6506f55d0d22209dfdac929275b99f7
SSP	contracts/signed/SignedSpotPricer.sol	831a114ee9dfaca895774365a4f2fefaa1019593
SPR	contracts/SpotPricer.sol	03e41bce3a819fb98324c6eadbbf086fe1f31653
CON	contracts/Config.sol	c1c3cbb1f943711d42e9a496135594c40afc16e5
POO	contracts/pools/Pool.sol	541097869cae3ab09f0a917b7df5e93ef4c29684
POW	contracts/pools/PoolOwner.sol	91a9000d73ef9047d5f68637baf364b391a096ac
VOW	contracts/VaultOwner.sol	c94aa2819835865306aa1caecd10a332e8f30b69

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	29	23	6
Informational	0	0	0
Minor	14	12	2
Medium	2	2	0
Major	13	9	4
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Zomma Protocol to identify any potential issues and vulnerabilities in the source code of the Zomma Protocol smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 29 issues of varying severity, listed below.

ID	Title	Severity	Status
CON-1	The `removePool()` Function can be Front-Run to Prevent the Owner From Removing a Pool	Major	Fixed
CON-2	Hacked Owner or Malicious Owner can Steal Assets on the Platform	Major	Acknowledged
CON-3	Optimizing Pool Removal Process in the `removePool()` Function	Minor	Acknowledged
CON-4	Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable	Minor	Fixed
CON-5	Splitting `require()` Statements That Use `&&` Saves Gas	Minor	Fixed
CON-6	Using Bools for Storage Incurs Overhead	Minor	Fixed
CON-7	Use Custom Errors	Minor	Fixed
CON-8	Don't Initialize Variables with Default Value	Minor	Fixed

CON-9	Use Assembly To Check For Address(0)	Minor	Acknowledged
ISP-1	`latestRoundData` May Return Stale or Incorrect Price	Major	Fixed
LED-1	For Operations That will not Overflow, You could Use Unchecked	Minor	Fixed
PFA-1	Re-org Attack in Factory	Major	Fixed
PFA-2	Use Calldata Instead of Memory for Function Arguments That Do not Get Mutated	Minor	Fixed
POO-1	`SafeApprove` Deprecated	Medium	Fixed
SDM-1	Using Private rather than Public for Constants, Saves Gas	Minor	Fixed
SET-1	Cache Array Length Outside of Loop	Minor	Fixed
SPR-1	Use of Deprecated Chainlink API	Major	Fixed
SPR-2	`getRoundData` Does Not Check For The Freshness of The `answer`	Major	Fixed
SPR-3	Use `!= 0` instead of `> 0` for Unsigned Integer Comparison	Minor	Fixed
SVA-1	Signature Malleability	Major	Fixed
SVA-2	Signature Replay Attack	Major	Fixed
SVA-3	Use Fixed Compiler Version	Medium	Fixed
VAU-1	Insufficient Circle USDC Liquidity for User Withdrawals After Bridged	Major	Acknowledged

	USDC Conversion		
VAU-2	Single-step Ownership Transfer Can be Dangerous	Major	Acknowledged
VAU-3	Loss of Precision Caused by Division Followed by Multiplication	Major	Acknowledged
VAU-4	Missing Deadline Checks Allow Pending Transactions To be Maliciously Executed	Major	Fixed
VAU-5	`++i` Costs Less Gas than `i++`, especially When It's Used in For-loops (`--i` / `i--` too)	Minor	Fixed
VAU-6	Use Shift Right/Left instead of Division/Multiplication if Possible	Minor	Fixed
VOW-1	Missing Approve After `setQuote`	Major	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Zomma Protocol Smart Contract:

Admin

- The admin can refresh the market quote by calling the `refreshQuote()` function.
- The admin can set the reserve rate through the `setReservedRate()` function.
- The admin is able to set the ZLM rate using the `setZlmRate()` function.
- The admin has the privilege to set the bonus rate via the `setBonusRate()` function.
- The admin can set the withdrawal fee rate using the `setWithdrawFeeRate()` function.
- The admin can set the free withdrawable rate through the `setFreeWithdrawableRate()` function.
- The admin can extract ERC20 tokens from the pool by calling `withdrawToken`, and can also withdraw ETH by calling `withdraw`.
- The admin has the capability to set crucial parameters such as `InitialMarginRiskRate`, `LiquidateRate`, `ClearRate`, and others in the config contract.
- The admin can invoke the `setPoolPaused` function to establish the paused status of the pool.
- The admin has the authority to call the `addPool` and `removePool` functions to add or remove pools, respectively.
- The admin has the ability to invoke the `setIv` function to set the IV (Implied Volatility).
- The admin can utilize the `setTradeDisabled` and `setExpiryDisabled` functions to configure the disabled status of trading and expiry, respectively.
- The admin has the capability to invoke the `setAddresses` function to change contract addresses.

User

- Users can invoke the `deposit` function to deposit funds and receive corresponding shares.
- Users can utilize the `withdrawBySignature` function to extract a specific quantity of shares by providing a signature.

- Users have the option to call the `withdraw` function to burn shares and withdraw funds.
- Users can use the `trade` function to execute batch trades.
- Users can invoke the `settle` function to settle all positions associated with the account and expiry.
- Users can use the `liquidate` function to liquidate a position.
- Users have the ability to call the `clear` function to clear an account.

4 Findings

CON-1 The `removePool()` Function can be Front-Run to Prevent the Owner From Removing a Pool

Severity: Major

Status: Fixed

Code Location:

contracts/Config.sol#217-232

Descriptions:

The `vault.removePool()` is designed to remove a specified pool from a list of pools, provided the contract owner calls it. Inside the function, it checks whether the specified pool has any associated expiries listed in the vault. If the length of the list of expiries for the given pool is not zero, it throws an error message. If a bad actor monitors transactions in the mempool and purchases an option before owner calls the `removePool()` fucntion, this check will fail.

```
function removePool(address pool) external onlyOwner {
    require(vault.listOfExpiries(pool).length == 0, "position not empty");
    require(poolAdded[pool], "pool not found");
    uint length = pools.length;
```

Suggestion:

It is recommended to consider temporarily disabling the pool before proceeding with its removal.

Resolution:

This issue has been fixed. The client has followed the suggestion.

CON-2 Hacked Owner or Malicious Owner can Steal Assets on the Platform

Severity: Major

Status: Acknowledged

Code Location:

contracts/Config.sol#182,188

Descriptions:

Having a single EOA as the only owner of contracts is a large centralization risk and a single point of failure. A single private key may be taken in a hack, or the sole holder of the key may become unable to retrieve the key when necessary. Consider changing to a multi-signature setup, or having a role-based authorization model.

Suggestion:

It is recommended to consider changing to a multi-signature setup, or having a role-based authorization model.

CON-3 Optimizing Pool Removal Process in the `removePool()` Function

Severity: Minor

Status: Acknowledged

Code Location:

contracts/Config.sol#217-232

Descriptions:

The function `removePool()` can be optimized by enhancing the process when `pools[i]` is equal to `pool`. Instead of iterating through the loop, it could directly swap `pools[I]` with the last element `pools[length-1]` and then reduce the length of the array by one using `pop()`. This can be achieved more efficiently.

```
for (uint i = 0; i < length; i++) {
    if (found) {
      pools[i - 1] = pools[i];
    } else if (pools[i] == pool) {
      found = true;
    }
  }
  pools.pop();
  poolAdded[pool] = false;
```

Suggestion:

It is recommended to implement the solution as follows:

```
  for (uint i = 0; i < length; i++) {
      if (pools[i] == pool) {
          pools[i] = pools[length - 1];  // Swap with the last element
          pools.pop();                   // Reduce the array length by one
          poolAdded[pool] = false;
          emit RemovePool(pool);
          return;                        // Exit the function after performing the swap and pop
      }
  }
```

CON-4 Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable

Severity: Minor

Status: Fixed

Code Location:

contracts/Config.sol#L99

Descriptions:

If a function modifier such as onlyOwner is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are CALLVALUE(2),DUP1(3),ISZERO(3),PUSH2(3),JUMPI(10),PUSH1(3),DUP1(3),REVERT(0),JUMPDEST(1),PO which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

Suggestion:

It is recommend that functions guaranteed to revert when called by normal users can be marked payable.

Resolution:

This issue has been fixed. The client has followed the suggestion.

CON-5 Splitting `require()` Statements That Use `&&` Saves Gas

Severity: Minor

Status: Fixed

Code Location:

contracts/Config.sol#L106

Descriptions:

Instead of using operator `&&` on single require check. Using double require check can save more gas,there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper by 3 gas.

Suggestion:

It is recommended to implement the following example.

```
require(_liquidateRate <= MAX_LIQUIDATE_RATE , "exceed the limit");
require(clearRate <= _liquidateRate, "exceed the limit");
```

Resolution:

This issue has been fixed. The client has used this method to optimize.

CON-6 Using Bools for Storage Incurs Overhead

Severity: Minor

Status: Fixed

Code Location:

contracts/Config.sol#31,32;

contracts/Ledger.sol#23

Descriptions:

Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from `false` to `true`, after having been `true` in the past. See source.

```
mapping(address => bool) public poolAdded;
mapping(address => bool) public poolEnabled;
mapping(address => mapping(uint => mapping(uint => mapping(bool => Position))))
internal accountPositions;
```

Suggestion:

It is recommended to use `uint256(1)` and `uint256(2)` for true/false.

CON-7 Use Custom Errors

Severity: Minor

Status: Fixed

Code Location:

contracts/Config.sol#100,106,112,118,134,142,148,154

Descriptions:

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.Source

```
require(_initialMarginRiskRate <= MAX_INITIAL_MARGIN_RISK_RATE, "exceed the limit");
require(_liquidateRate <= MAX_LIQUIDATE_RATE && clearRate <= _liquidateRate, "exceed the limit");
```

Suggestion:

It is recommed to use Custom Errors.

```
require(_initialMarginRiskRate <= MAX_INITIAL_MARGIN_RISK_RATE, "ETL");
```

Resolution:

This issue has been fixed. The client has followed the suggestion.

CON-8 Don't Initialize Variables with Default Value

Severity: Minor

Status: Fixed

Code Location:

contracts/Config.sol#221,222;

contracts/Ledger.sol#54,116,128

Descriptions:

Uninitialized variables are assigned with the types default value. Explicitly initializing a variable with it's default value costs unnecessary gas.

```
bool found = false;
for (uint i = 0; i < length; i++) {
```

Suggestion:

It is recommended not to use default values to initialize variables.

Resolution:

This issue has been fixed. The client has followed the suggestion.

CON-9 Use Assembly To Check For Address(0)

Severity: Minor

Status: Acknowledged

Code Location:

contracts/Config.sol#L189

Descriptions:

Saves 6 gas per instance if using assembly to check for address(0).

Suggestion:

It is recommended to use assembly to check for address(0).

```
assembly {
 if iszero(_addr) {
  mstore(0x00, "zero address")
  revert(0x00, 0x20)
 }
}
```

ISP-1 `latestRoundData` May Return Stale or Incorrect Price

Severity: Major

Status: Fixed

Code Location:

contracts/interim/InterimSpotPricer.sol#55-58

Descriptions:

ChainlinkOracle should use the `updatedAt` value from the `latestRoundData()` function to make sure that the latest answer is recent enough to be used. In the current implementation of `SpotPricer.sol`, there is no freshness check. This could lead to stale prices being used. Moreover, chainlink aggregators have a built-in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value (i.e. LUNA crash) the price of the oracle will continue to return the minPrice instead of the actual price of the asset and vice versa.

```
function getPrice() public view virtual returns (uint) {
  (, int256 answer, , , ) = oracle.latestRoundData();
  return uint(answer) * 10**18 / 10**oracle.decimals();
}
```

Suggestion:

It is recommended to consider using the following checks:

```
// minPrice check
require(answer > minPrice, "Min price exceeded");
// maxPrice check
require(answer < maxPrice, "Max price exceeded");
require(block.timestamp - updatedAt < validPeriod, "freshness check failed.")
```

The `validPeriod` can be based on the Heartbeat of the feed.

Resolution:

This issue has been fixed. The client has implemented the specified check.

LED-1 For Operations That will not Overflow, You could Use Unchecked

Severity: Minor

Status: Fixed

Code Location:

contracts/Ledger.sol#116;

contracts/OptionMarket.sol#58

Descriptions:

For Operations that will not overflow, you could use unchecked

```
for (uint i = 0; i < length; i++) {
```

Suggestion:

It is recommended to use Solidity's `unchecked` block to save the overflow checks.

```
for (uint i; i < length;) {

unchecked { ++i; }
}
```

Resolution:

This issue has been fixed. The client has followed the suggestion.

PFA-1 Re-org Attack in Factory

Severity: Major

Status: Fixed

Code Location:

contracts/pools/PoolFactory.sol#L21-29

Descriptions:

The contract creates new pools and pooltokens through the clone function from openzeppelin. The address is determined by the create address derivation, which depends on the contract nonce. This is a risky approach as re-orgs can occur in all EVM chains. An attacker could potentially steal funds through a reorg attack if the pool is funded within a few blocks of being created. Furthermore, the reorg could last for several minutes, providing ample time to create the pool and transfer funds to that address, especially when using a script instead of manual operations. Any significant reorg incident creates an opportunity for users' funds to be stolen. Additionally, the use of a small number of confirmations in user transactions can lead to a loss of money.

Imagine that Alice creates pool and pool token, and then user deposits assets into the pool. Bob sees that the network block reorg happens and calls `create()`. Thus, it creates a pool and token clone with an address to which user sends funds. Then user's transactions are executed and user transfers funds to Bob's contract.

Here are some examples of block reorganizations:

- Ethereum Beacon Chain Blockchain Reorg
- Polygon Hit by 157 Block Reorg Despite Hard Fork to Reduce Reorgs
- PolygonScan Block 39599624

Suggestion:

It is recommended to use the cloneDeterministic function from openzeppelin to create pools and pool tokens. This function uses a deterministic computation to derive the address of the new contract, which can help to prevent potential reorg attacks. By using cloneDeterministic, the address of the new contract is determined in a predictable manner,

reducing the risk of unexpected address changes due to reorgs. This can enhance the security of the contract and protect users' funds.

Resolution:

This issue has been fixed. The client has already removed the part of the code.

PFA-2 Use Calldata Instead of Memory for Function Arguments That Do not Get Mutated

Severity: Minor

Status: Fixed

Code Location:

contracts/pools/PoolFactory.sol#21

Descriptions:

Mark data types as calldata instead of memory where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as calldata. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies memory storage.

```
function create(address _vault, string memory name, string memory symbol) external returns(address clonedPool, address clonedPoolToken) {
```

Suggestion:

It is recommended to use calldata instead of memory.

Resolution:

This issue has been fixed. The client has followed the suggestion.

POO-1 `SafeApprove` Deprecated

Severity: Medium

Status: Fixed

Code Location:

contracts/pools/Pool.sol#L65

Descriptions:

The OpenZeppelin SafeERC20 `safeApprove()` function has been deprecated, as seen in the comments of the OpenZeppelin code. Using this deprecated function can lead to unintended reverts and potentially the locking of funds.

Suggestion:

It is recommended to replace `safeApprove()` with `safeIncreaseAllowance()`.

Resolution:

This issue has been fixed. The client has already used `safeIncreaseAllowance`.

SDM-1 Using Private rather than Public for Constants, Saves Gas

Severity: Minor

Status: Fixed

Code Location:

contracts/libraries/SafeDecimalMath.sol#9,10;

contracts/libraries/SignedSafeDecimalMath.sol#9,10

Descriptions:

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves 3406-3606 gas in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

```
uint8 public constant PRECISION = 18;
uint public constant UNIT = 10**uint(PRECISION);
```

Suggestion:

It is recommed to use private rather than public for constants.

Resolution:

This issue has been fixed. The client has followed the suggestion.

SET-1 Cache Array Length Outside of Loop

Severity: Minor

Status: Fixed

Code Location:

contracts/Settler.sol#8;

contracts/Vault.sol#478,485,625,629

Descriptions:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

```
for (uint i = 0; i < accounts.length; ++i) {

for (uint i = 0; i < expiries.length; ++i) {
```

Suggestion:

It is recommended to cache the array length before entering the loop.

Resolution:

This issue has been fixed. The client has followed the suggestion.

SPR-1 Use of Deprecated Chainlink API

Severity: Major

Status: Fixed

Code Location:

contracts/SpotPricer.sol#L35;

contracts/SpotPricer.sol#L45

Descriptions:

According to Chainlink's documentation, the `latestAnswer` and `getAnswer` function is deprecated. This function might suddenly stop working if Chainlink stop supporting deprecated APIs. And the old API can return stale data

Suggestion:

It is recommended to switch to `latestRoundData()` as described here.

Resolution:

This issue has been fixed. The client has already used `latestRoundData` .

SPR-2 `getRoundData` Does Not Check For The Freshness of The `answer`

Severity: Major

Status: Fixed

Code Location:

contracts/SpotPricer.sol#38-49

Descriptions:

While the `latestRoundData` function no longer needs to check for round completeness, the `getRoundData` function needs to check for `answeredInRoun` d and `roundId` for price freshness and `latestTimestamp` for round completeness. `answeredInRound` is the combination of `aggregatorAnsweredInRound` and `phaseId` .

`aggregatorAnsweredInRound` : The round the answer was updated in. You can check `answeredInRound` against the current `roundId` . If `answeredInRound` is less than `roundId` , the `answer` is carried over. Also, you need to validate that the timestamp on that round is not 0.

```
function settle(uint expiry, uint80 roundId) external {
  if (settledPrices[expiry] != 0) {
    revert Settled();
  }
  if (!checkRoundId(expiry, roundId)) {
    revert InvalidRoundId();
  }
  (, int256 answer, , , ) = oracle.getRoundData(roundId);
  uint price = uint(answer) * 10**18 / 10**oracle.decimals();
  settledPrices[expiry] = price;
  emit SettlePrice(expiry, price, roundId);
}
```

Suggestion:

It is recommended to check for price freshness:

```
require(answeredInRound == roundId, "the price is not fresh")
```

```
require(latestTimestamp > 0, "Round not complete");
```

Resolution:

This issue has been fixed. The client has implemented the specified check.

SPR-3 Use `!= 0` instead of `> 0` for Unsigned Integer Comparison

Severity: Minor

Status: Fixed

Code Location:

contracts/SpotPricer.sol#52

Descriptions:

When dealing with unsigned integer types, comparisons with != 0 are cheaper than with > 0.

```
return timestamp > 0 && expiry >= timestamp && expiry < timestamp2;
```

Suggestion:

It is recommend to use `!= 0` instead of `> 0` for unsigned integer comparison.

Resolution:

This issue has been fixed. The client has followed the suggestion.

SVA-1 Signature Malleability

Severity: Major

Status: Fixed

Code Location:

contracts/signed/SignedVault.sol#L73

Descriptions:

The elliptic curve used in Ethereum for signatures is symmetrical, hence for every [v,r,s] there exists another [v,r,s] that returns the same valid result. Therefore two valid signatures exist which allows attackers to compute a valid signature without knowing the signer's private key. ecrecover() is vulnerable to signature malleability [1, 2] so it can be dangerous to use it directly.An attacker can compute another corresponding [v,r,s] that will make this check pass due to the symmetrical nature of the elliptic curve.

Suggestion:

It is recommended to use OpenZeppelin’s ECDSA.sol library and reading the comments above ECDSA's tryRecover() function provides very useful information on correctly implementing signature checks to prevent signature malleability vulnerabilities. When using OpenZeppelin's ECDSA library, special care must be taken to use version 4.7.3 or greater, since previous versions contained a signature malleability bug.

Resolution:

This issue has been fixed. The client has integrated OpenZeppelin's EIP712Upgradeable.

SVA-2 Signature Replay Attack

Severity: Major

Status: Fixed

Code Location:

contracts/signed/SignedVault.sol#L67-100

Descriptions:

The contract does not adhere to EIP-712, exposing a vulnerability to signature replay attacks. To prevent such attacks, smart contracts must implement the following measures:

- Maintain a Nonce:

Smart contracts should keep track of a nonce, a unique number associated with each transaction or action performed by the contract.

- Provide Current Nonce to Signers:

The current nonce should be made available to signers before they generate a signature.

- Validate Signature with Nonce:

During signature validation, the contract must verify the signature using the current nonce. This ensures that the signature is only valid for the specific transaction or action associated with that nonce.

- Store Used Nonces:

Once a nonce has been used, the contract should store this information in storage. This prevents the same nonce from being used again, effectively rendering replayed signatures invalid. Incorporate Nonce in Signatures:

Signers are required to include the current nonce when signing their messages. As a result, signatures that have already been used cannot be replayed, as the corresponding nonce will be marked as used in storage. An illustrative example of this concept can be found in OpenZeppelin's ERC20Permit implementation.

```
function permit(
    address owner,
    address spender,
    uint256 value,
```

```
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public virtual override {
    // ...
    bytes32 structHash = keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender,
value, _useNonce(owner), deadline));
    // incorporates chain_id (ref next section Cross Chain Replay)
    bytes32 hash = _hashTypedDataV4(structHash);
    // ...
}

function _useNonce(address owner) internal virtual returns (uint256 current) {
    Counters.Counter storage nonce = _nonces[owner];
    current = nonce.current();
    nonce.increment();
}
```

Furthermore, the smart contracts operate across multiple blockchain networks using the same contract . However, due to the absence of chain-specific verification in the contracts, a valid signature used on one chain could be replicated by an attacker on another chain. This would grant the attacker unauthorized access to the same user and contract address.

To mitigate cross-chain signature replay attacks, it is crucial for smart contracts to validate signatures using the chain_id. Additionally, users must include the chain_id in the message they sign. This practice ensures that signatures are specific to each chain and cannot be replayed across different chains, enhancing security and protecting against unauthorized access.

Suggestion:

It is recommended to implement the methods mentioned in the description approach to prevent signature replay attacks, further insights can be obtained from Ethereum Improvement Proposal EIP-712. This EIP provides additional information and guidance on structuring data encoding, signing, and enhancing the overall security of cryptographic signatures within smart contracts.

Resolution:

This issue has been fixed. The client has added a nonce value and has already integrated OpenZeppelin's EIP712Upgradeable.

SVA-3 Use Fixed Compiler Version

Severity: Medium

Status: Fixed

Code Location:

contracts/signed/SignedVault.sol#L2

Descriptions:

All in scope contracts use ^0.8.11 as compiler version.

They should use a fixed version,to make sure the contracts are always compiled with the intended version. Using versions of Solidity that are not as expected could potentially result in security issues. This might include the usage of versions with known vulnerabilities, such as the ones mentioned in the Solidity 0.8.15 release announcement: Solidity 0.8.15 Release Announcement.

Suggestion:

It is recommended to use fixed and the latest compiler versions.

Resolution:

This issue has been fixed. The client is using a fixed compiler version.

VAU-1 Insufficient Circle USDC Liquidity for User Withdrawals After Bridged USDC Conversion

Severity: Major

Status: Acknowledged

Code Location:

contracts/Vault.sol#223-241

Descriptions:

In the `Vault.withdraw()` function, the protocol extracts funds from the configured quote address in the config and transfers them to the `msg.sender` address. However, when the bridged USDC is changed to Circle USDC, user balances remain unchanged, potentially leading to insufficient Circle USDC in the protocol to support user withdrawals. This results in the `withdraw()` function being unable to execute.

```
function transfer(address to, uint amount) private {
  IERC20(config.quote()).safeTransfer(to, (amount * 10**config.quoteDecimal()) / ONE);
}
```

Suggestion:

It is recommended to implement a function that allows only the owner to withdraw funds. Before changing the bridged USDC to Circle USDC, the vault should be paused. Subsequently, all bridged USDC funds should be withdrawn, exchanged for Circle USDC, and then injected back into the vault. This ensures that there is sufficient Circle USDC in the protocol to support user withdrawals.

Resolution:

The client has confirmed that they will address this issue in the newly deployed contract.

VAU-2 Single-step Ownership Transfer Can be Dangerous

Severity: Major

Status: Acknowledged

Code Location:

contracts/Vault.sol#130-134

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract.

```
function checkOwner() internal view {
  if (msg.sender != owner) {
    revert NotOwner();
  }
}
```

`import "@openzeppelin/contracts/access/Ownable.sol";`

Below is the official documentation explanation from OpenZeppelin

https://docs.openzeppelin.com/contracts/4.x/api/access

`Ownable is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests but projects with production concerns are likely to outgrow it .`

Suggestion:

It is recommended to use a two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise the old owner still has control of the contract.

VAU-3 Loss of Precision Caused by Division Followed by Multiplication

Severity: Major

Status: Acknowledged

Code Location:

contracts/Vault.sol#L287-324

Descriptions:

There is a precision loss issue in the calculations within the following function.

```
 function reducePosition(address account, int amountToRemove, TxCache memory
txCache, PositionInfo memory positionInfo, AccountInfo memory accountInfo) private
returns (int) {
    uint rate = uint(amountToRemove.decimalDivRound(accountInfo.equity));
    // sold position risk
    uint ratedRisk = accountInfo.initialMargin - uint(-positionInfo.sellValue);
    uint riskDenominator = ratedRisk + uint(positionInfo.buyValue);
    // total risk want to remove
    uint riskDenominatorToRemove = riskDenominator.decimalMul(rate);
```

We can simplify these calculations in order to make the observation clearer:

```
riskDenominatorToRemove=riskDenominator*rate
rate=amountToRemove/equity
```

```
riskDenominatorToRemove=(amountToRemove/equity  *riskDenominator
```

By observing the calculations, we can clearly see the precision loss caused by division followed by multiplication.

Suggestion:

It is recommended to always maintain the calculation order of multiplication followed by division.

VAU-4 Missing Deadline Checks Allow Pending Transactions To be Maliciously Executed

Severity: Major

Status: Fixed

Code Location:

contracts/Vault.sol#L781;

contracts/pools/Pool.sol#L101;

contracts/pools/Pool.sol#L137

Descriptions:

Lack of control over the `deadline` parameter during protocol transactions, deposits, or withdrawals may lead to potential slippage losses or susceptibility to malicious attacks.This is actually how uniswap implemented the Deadline, this protocol also need deadline check like this logic. https://github.com/Uniswap/v2-periphery/blob/0335e8f7e1bd1e8d8329fd300aea2ef2f36dd19f/contracts/UniswapV2Router02.sol#L

The point is the deadline check.

```
modifier ensure(uint deadline) {
    require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
    _;
}
```

The deadline check ensure that the transaction can be executed on time and the expired transaction revert.

Suggestion:

It is recommended to add a `deadline` parameter check.

Resolution:

This issue has been fixed. The client has implemented a `deadline` check.

VAU-5 `++i` Costs Less Gas than `i++`, especially When It's Used in For-loops (`--i` / `i--` too)

Severity: Minor

Status: Fixed

Code Location:

contracts/Vault.sol#546;

contracts/Ledger.sol#116,128

Descriptions:

`++i` costs less gas than `i++`, especially when it's used in for-loops (--i/i-- too).

```
for (uint i = 0; i < length; i++) {
```

Suggestion:

It is recommed to use `++i`.

Resolution:

This issue has been fixed. The client has followed the suggestion.

VAU-6 Use Shift Right/Left instead of Division/Multiplication if Possible

Severity: Minor

Status: Fixed

Code Location:

contracts/Vault.sol#447;

contracts/option-pricer/OptionPricer.sol#69,70,73

Descriptions:

A division/multiplication by any number x being a power of 2 can be calculated by shifting log2(x) to the right/left.

While the DIV opcode uses 5 gas, the SHR opcode only uses 3 gas. Furthermore, Solidity's division operation also includes a division-by-0 prevention which is bypassed using shifting.

```
int pivot = arr[uint(left + (right - left) / 2)].notional;

utilization = (utilization + utilizationAfter) / 2;
```

Suggestion:

It is recommended to use shift Right/Left instead of division/multiplication.

Resolution:

This issue has been fixed. The client has followed the suggestion.

VOW-1 Missing Approve After `setQuote`

Severity: Major

Status: Fixed

Code Location:

contracts/VaultOwner.sol#39-45

Descriptions:

The `setQuote` function reconfigures the protocol token address and decimal places. However, after updating the token address, it lacks the necessary approval operation. Although `safeIncreaseAllowance` is performed on initialization, this approval becomes ineffective after updating the quote. It is crucial to reapprove to prevent potential failures in the crucial transfer of account balances between the vault and pool.

```
function initialize(address _vault) external initializer {
  __Ownable_init();
  _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
  vault = _vault;
  Config config = Vault(_vault).config();
  IERC20(config.quote()).safeIncreaseAllowance(_vault,
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff);
 }
```

Suggestion:

It is recommended to implement a functionality similar to `refreshQuote()` in the pool.

Resolution:

This issue has been fixed. The client has added an additional function to approve a new token.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.