OMNISCIA

October 31, 2024

SMART CONTRACT AUDIT REPORT

Tempest Finance

Ambient Liquidity Management Strategies Round 2


omniscia.io


info@omniscia.io


Online report: tempest-finance-ambient-liquidity-management-strate

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.


omniscia.io


info@omniscia.io

Online report: tempest-finance-ambient-liquidity-management-strategies-round-2

Ambient Liquidity Management Strategies Round 2 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
3767bb08e1	October 23rd 2024	2d86e4ef91
715e11a37e	October 29th 2024	37f6c09f2a
715e11a37e	October 31st 2024	48e4b56278

Audit Overview

We were tasked with performing an audit of the Tempest Finance codebase and in particular their Ambient Liquidity Management Strategies refactored after our previous audit round.

Over the course of the audit, we identified an issue in the sequencer-aware `BridgeOracle` as well as a potential profit miscalculation within the `LstAmbientLiquidity` contract.

We advise the Tempest Finance team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Tempest Finance team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Tempest Finance and have identified that all exhibits outlined within the feedback document shared to us have been addressed adequately.

All remaining exhibits are considered safely acknowledged and no remediative action is pending by the Tempest Finance team.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	32	18	0	14
Minor	2	2	0	0
Medium	1	1	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **14 findings utilizing static analysis** tools as well as identified a total of **21 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- 🎯 **Scope**
- 💻 **Compilation**
- 🔍 **Static Analysis**
- 👁️ **Manual Review**
- 🖌️ **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/Tempest-Finance/tempest_smart_contract**
- Commit: 3767bb08e1a1d399fb65f6c99a041fe2033137be
- Language: Solidity
- Network: Ethereum
- Revisions: **3767bb08e1**, **715e11a37e**

Contracts Assessed

File	Total Finding(s)
src/ambient-lp/AmbientUser.sol (AUR)	2
src/ambient-lp/AmbientCommon.sol (ACN)	0
src/ambient-lp/AmbientOperator.sol (AOR)	1
src/ambient-lp/AmbientInterface.sol (AIE)	1
src/ambient-lp/AmbientLiquidity.sol (ALY)	0
src/ambient-lp/AmbientStorageLayout.sol (ASL)	2
src/utils/BridgeOracle.sol (BOE)	3
src/ambient-lp/BaseAmbientOperator.sol (BAO)	0
src/BaseAmbientStrategy.sol (BAS)	2
src/ambient-lp/BaseAmbientStorageLayout.sol (BAL)	1

src/utils/Constants.sol (CST)	1
src/utils/DepositWithdrawPausable.sol (DWP)	1
src/utils/Errors.sol (ESR)	0
src/lst/LstAmbientUser.sol (LAU)	0
src/lst/LstAmbientCommon.sol (LAC)	1
src/lst/LstAmbientOperator.sol (LAO)	0
src/lst/LstAmbientInterface.sol (LAI)	2
src/lst/LstAmbientLiquidity.sol (LAL)	3
src/lst/LstAmbientStorageLayout.sol (LAS)	2
src/utils/RswETHOracle.sol (RET)	0
src/RswEthStrategy.sol (RES)	1
src/lst/rsweth/RswEthAmbientUser.sol (REA)	0
src/lst/rsweth/RswEthAmbientCommon.sol (REC)	0
src/lst/rsweth/RswEthAmbientOperator.sol (REO)	0

src/lst/rsweth/RswEthAmbientStorageLayout.sol (REL)	2
src/ambient-lp/SymetricAmbientOperator.sol (SAO)	0
src/SymetricAmbientStrategy.sol (SAS)	2
src/ambient-lp/SymetricAmbientStorageLayout.sol (SAL)	2
src/utils/Variables.sol (VSE)	0
src/libraries/VaultLibrary.sol (VLY)	3
src/utils/WstETHOracle.sol (WET)	0
src/WstEthStrategy.sol (WES)	1
src/lst/wsteth/WstEthAmbientUser.sol (WEA)	0
src/lst/wsteth/WstEthAmbientCommon.sol (WEC)	0
src/lst/wsteth/WstEthAmbientOperator.sol (WEO)	0
src/lst/wsteth/WstEthAmbientStorageLayout.sol (WEL)	2

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.23` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`>=0.8.23`).

We advise them to be locked to `0.8.23` (`=0.8.23`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **53 potential issues** within the codebase of which **31 were ruled out to be false positives** or negligible findings.

The remaining **22 issues** were validated and grouped and formalized into the **14 exhibits** that follow:

ID	Severity	Addressed	Title
ASL-01S	Informational	Yes	Inexistent Visibility Specifier
BAL-01S	Informational	Yes	Inexistent Visibility Specifier
BAS-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
BOE-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
BOE-02S	Informational	Yes	Inexistent Visibility Specifier
CST-01S	Informational	Acknowledged	Illegible Numeric Value Representation
LAI-01S	Informational	Yes	Inexistent Sanitization of Input Address
LAS-01S	Informational	Yes	Inexistent Visibility Specifier
REL-01S	Informational	Yes	Inexistent Visibility Specifier
RES-01S	Informational	Yes	Inexistent Sanitization of Input Addresses

SAL-01S	Informational	Yes	Inexistent Visibility Specifier
SAS-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
WEL-01S	Informational	Yes	Inexistent Visibility Specifier
WES-01S	Informational	Yes	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Tempest Finance's Ambient Strategy follow-up round.

As the project at hand implements multiple DeFi strategies, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed two important vulnerabilities** within the system which could have had **moderate ramifications** to its overall operation; we urge the Tempest Finance team to promptly evaluate the medium-severity items within the audit report.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **21 findings** were identified over the course of the manual review of which **8 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
ASL-01M	Informational	Acknowledged	Non-Standard Gap Specification
AUR-01M	Minor	Yes	Inexistent Usage of Arbitrary Precision Arithmetic
BOE-01M	Medium	Yes	Inexistent Validation of Sequencer Uptime
DWP-01M	Minor	Yes	Upgradeability Incompatibility
LAS-01M	Informational	Acknowledged	Non-Standard Gap Specification

REL-01M	Informational	Acknowledged	Non-Standard Gap Specification
SAL-01M	Informational	Acknowledged	Non-Standard Gap Specification
WEL-01M	Informational	Acknowledged	Non-Standard Gap Specification

Code Style

During the manual portion of the audit, we identified **13 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
AIE-01C	Informational	Acknowledged	Improper Specification of Function Signatures
AOR-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
AUR-01C	Informational	Acknowledged	Redundant Duplication of Code
BAS-01C	Informational	Acknowledged	Improper Specification of Function Signature
LAC-01C	Informational	Yes	Redundant Parenthesis Statements
LAI-01C	Informational	Acknowledged	Improper Specification of Function Signatures
LAL-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
LAL-02C	Informational	Acknowledged	Potential Misuse of Profit Calculation Mechanism
LAL-03C	Informational	Yes	Redundant Parenthesis Statement
SAS-01C	Informational	Acknowledged	Improper Specification of Function Signatures

VLY-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
VLY-02C	Informational	Acknowledged	Redundant Bit Clearing
VLY-03C	Informational	Yes	Redundant Parenthesis Statements

AmbientStorageLayout Static Analysis Findings

ASL-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	AmbientStorageLayout.sol:L27

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/ambient-lp/AmbientStorageLayout.sol

```sol
27  uint256[50] __gap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

BaseAmbientStorageLayout Static Analysis Findings

BAL-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	BaseAmbientStorageLayout.sol:L7

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/ambient-lp/BaseAmbientStorageLayout.sol

SOL

```sol
7   uint256[50] __baseAmbientGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

BaseAmbientStrategy Static Analysis Findings

BAS-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	BaseAmbientStrategy.sol:L20-L37

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/BaseAmbientStrategy.sol

SOL

```sol
function initialize(
  int24[] memory _upperTicks,
  int24[] memory _lowerTicks,
  AmbientStrategyParameters memory aParams
) public initializer {
  _initialize(aParams, _upperTicks, _lowerTicks);

  crocsQuery = CrocsQuery(aParams.cQuery);
  crocsSwapDex = CrocsSwapDex(aParams.cSwapDex);

```

Example (Cont.):

```sol
    int24 tickSize = int24(int16(crocsQuery.queryPoolParams(aParams.token0, aParams.token1, POOL_IDX).tickSize_));

    VaultLibrary._checkLiqParams(_upperTicks, _lowerTicks, tickSize);

    tokenAddresses[0] = aParams.token0;
    tokenAddresses[1] = aParams.token1;
    oracle = aParams.oracle;
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

Input sanitization has been introduced within the upstream `AmbientInterface` dependency that adequately restricts the input addresses outlined by this exhibit rendering it alleviated.

BridgeOracle Static Analysis Findings

BOE-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	BridgeOracle.sol:L28-L49, L54-L57, L62-L65, L70-L73

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/utils/BridgeOracle.sol

SOL

```sol
28  constructor(
29    address _baseUsdOracle,
30    address _quoteUsdOracle,
31    address _sequencerUptimeOracle,
32    uint64 _baseOracleTimeLimit,
33    uint64 _quoteOracleTimeLimit,
34    uint64 _sequencerDowntimeLimit,
35    bool _isL2,
36    string memory _name,
37    address governor
```

Example (Cont.):

```sol
) {
    baseUsdOracle = _baseUsdOracle;
    quoteUsdOracle = _quoteUsdOracle;
    sequencerUptimeOracle = _sequencerUptimeOracle;
    baseOracleTimeLimit = _baseOracleTimeLimit;
    quoteOracleTimeLimit = _quoteOracleTimeLimit;
    sequencerDowntimeLimit = _sequencerDowntimeLimit;
    isL2 = _isL2;
    name = _name;
    _setRoleAdmin(GOVERNANCE_ROLE, GOVERNANCE_ROLE);
    _grantRole(GOVERNANCE_ROLE, governor);
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

All input arguments of the `BridgeOracle::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

BOE-02S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	BridgeOracle.sol:L18

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/utils/BridgeOracle.sol

SOL

```sol
18  bool isL2;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

Constants Static Analysis Findings

CST-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	Constants.sol:L4

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

src/utils/Constants.sol

```sol
4   uint16 constant BASE = 10_000;
```

Recommendation:

To properly illustrate the value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation:

The Tempest Finance team evaluated this exhibit and opted to retain their `10_000` representation over the `100_00` decimal representation we advised.

LstAmbientInterface Static Analysis Findings

LAI-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	LstAmbientInterface.sol:L369-L428

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/lst/LstAmbientInterface.sol

SOL

```sol
369 function _initialize(LstParameters memory lParams) internal {
370   if (
371     lParams.assetIdx != 0 ||
372     lParams.token0 != address(0) ||
373     lParams.token0 >= lParams.token1 ||
374     lParams.fee > BASE ||
375     lParams.investedPercentage > BASE ||
376     lParams.padding >= lParams.minimumDeposit
377   ) {
378     revert(BAD_SETUP);
```

Example (Cont.):

```sol
    }

    __ReentrancyGuard_init();
    __ERC20_init(lParams.name, lParams.symbol);
    __AccessControl_init();

    if (lParams.lpWeights.length != lParams.upperTicks.length) revert(BAD_SETUP);
    VaultLibrary._checkWeights(lParams.lpWeights);

    crocsQuery = CrocsQuery(lParams.cQuery);
    crocsSwapDex = CrocsSwapDex(lParams.cSwapDex);
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

The input `lParams.cSwapDex` address argument of the **`LstAmbientInterface::_initialize`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

LstAmbientStorageLayout Static Analysis Findings

LAS-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	LstAmbientStorageLayout.sol:L29

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/lst/LstAmbientStorageLayout.sol

SOL

```sol
29  uint256[50] __gap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

RswEthAmbientStorageLayout Static Analysis Findings

REL-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	RswEthAmbientStorageLayout.sol:L17

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/lst/rsweth/RswEthAmbientStorageLayout.sol

SOL

```sol
17  uint256[50] __rswEthGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

RswEthStrategy Static Analysis Findings

RES-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	RswEthStrategy.sol:L23-L27

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/RswEthStrategy.sol

SOL

```sol
function initialize(LstParameters memory lParams) public initializer {
  _initialize(lParams);
  unWrappedToken = lParams.unWrappedToken;
  withdrawalQueue = IrswEXIT(lParams.withdrawalQueue);
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

Input sanitization has been introduced at the `LstAmbientInterface` level alleviating this exhibit as a result.

SymetricAmbientStorageLayout Static Analysis Findings

SAL-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	SymetricAmbientStorageLayout.sol:L9

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/ambient-lp/SymetricAmbientStorageLayout.sol

SOL

```sol
9   uint256[50] __symetricAmbientGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

SymetricAmbientStrategy Static Analysis Findings

SAS-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	SymetricAmbientStrategy.sol:L23-L53

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/SymetricAmbientStrategy.sol

SOL

```sol
function initialize(
  AmbientStrategyParameters memory aParams,
  int24 _baseWidth,
  int24 _limitWidth
) external initializer {
  crocsQuery = CrocsQuery(aParams.cQuery);
  crocsSwapDex = CrocsSwapDex(aParams.cSwapDex);
  oracle = aParams.oracle;
  tokenAddresses[0] = aParams.token0;
  tokenAddresses[1] = aParams.token1;
```

Example (Cont.):

```sol

    int24 tickSize = int24(int16(crocsQuery.queryPoolParams(aParams.token0, aParams.token1, POOL_IDX).tickSize_));
    if (_baseWidth % tickSize != 0 || _limitWidth % tickSize != 0) {
      revert(BAD_RANGE);
    }
    (int24[] memory _upperTicks, int24[] memory _lowerTicks) = VaultLibrary.setTicks(
      TickParameters({
        sqrtOraclePrice: _calOraclePrice(tokenAddresses).sqrtPrice,
        baseWidth: _baseWidth,
        limitWidth: _limitWidth,
        tickSize: tickSize,
        currentTick: currentTick(),
        isLimitRight: true
      })
    );

    _initialize(aParams, _upperTicks, _lowerTicks);

    baseWidth = _baseWidth;
    limitWidth = _limitWidth;
  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

Input sanitization has been introduced within the upstream `AmbientInterface` dependency that adequately restricts the input addresses outlined by this exhibit rendering it alleviated.

WstEthAmbientStorageLayout Static Analysis Findings

WEL-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	WstEthAmbientStorageLayout.sol:L16

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/lst/wsteth/WstEthAmbientStorageLayout.sol

SOL

```sol
16  uint256[50] __wstEthGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

WstEthStrategy Static Analysis Findings

WES-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	WstEthStrategy.sol:L17-L21

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/WstEthStrategy.sol

SOL

```sol
function initialize(LstParameters memory lParams) public initializer {
  _initialize(lParams);
  unWrappedToken = lParams.unWrappedToken;
  withdrawalQueue = IWithdrawalQueueERC721(lParams.withdrawalQueue);
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

Input sanitization has been introduced at the `LstAmbientInterface` level alleviating this exhibit as a result.

AmbientStorageLayout Manual Review Findings

ASL-01M: Non-Standard Gap Specification

Type	Severity	Location
Standard Conformity	Informational	AmbientStorageLayout.sol:L27

Description:

The referenced gap specification is meant to be in compliance with the OpenZeppelin standard, however, it does not adhere to its length specification pattern.

Example:

src/ambient-lp/AmbientStorageLayout.sol

```sol
abstract contract AmbientStorageLayout {
  uint8 internal assetIdx;
  uint8 internal _decimals;
  uint8 public padding;
  uint8 internal cmdId;
  uint16 public fee;
  uint16 public investedPercentage;
  uint16 public swapSlippage;
  uint16 public liqSlippage;
  address public oracle;
```

Example (Cont.):

```sol
    address public feeRecipient;
    address public operatorPath;
    address public userPath;
    address[2] internal tokenAddresses;
    LpParam[] internal lpParams;
    CrocsSwapDex public crocsSwapDex;
    CrocsQuery public crocsQuery;
    uint256 public minimumDeposit;
    uint256[50] __gap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
}
```

Recommendation:

We advise the length to be configured to `50 - numberOfSlotsReserved` (`40`), ensuring that the array can be properly maintained per new storage entry introduced in a consistent manner.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

AmbientUser Manual Review Findings

AUR-01M: Inexistent Usage of Arbitrary Precision Arithmetic

Type	Severity	Location
Mathematical Operations	Minor	AmbientUser.sol:L659

Description:

The referenced arithmetic calculation utilizes fixed-precision arithmetic whereas the `token1` execution path utilizes arbitrary precision arithmetic.

Impact:

The `token0` calculation path of the **`AmbientUser::_checkReceivedAmount`** function will revert for values that the `token1` execution path might not which we consider invalid.

Example:

src/ambient-lp/AmbientUser.sol

SOL

```sol
657 if (_isToken0) {
658   // received is token1
659   amountInOtherToken = (receivedAmount * oraclePrice) / 10 ** token1Decimals;
660 } else {
661   amountInOtherToken = receivedAmount.mulDiv(10 ** token1Decimals, oraclePrice);
662 }
```

Recommendation:

We advise both branches to employ arbitrary precision arithmetic, ensuring that the calculations can be safely performed with a greater range of input values.

Alleviation:

The referenced calculations will now employ arbitrary precision arithmetic, addressing this exhibit in full.

BridgeOracle Manual Review Findings

BOE-01M: Inexistent Validation of Sequencer Uptime

Type	Severity	Location
Logical Fault	Medium	BridgeOracle.sol:L132-L138

Description:

The `BridgeOracle::numeraireLatestAnswer` implementation does not validate the L2 sequencer's uptime in contrast to the `BridgeOracle::latestAnswer` implementation.

Impact:

The `BridgeOracle::numeraireLatestAnswer` function will yield a price measurement when the L2 sequencer might be offline which contradicts the contract's `BridgeOracle::latestAnswer` implementation.

Example:

src/utils/BridgeOracle.sol

SOL

```sol
132 function numeraireLatestAnswer() external view returns (int256) {
133   IOracle _baseUsdOracle = IOracle(baseUsdOracle);
134   (, int256 baseOraclePrice, , uint256 updatedAt, ) =
_baseUsdOracle.latestRoundData();
135   if (baseOraclePrice <= 0 || updatedAt + baseOracleTimeLimit <= block.timestamp)
revert(PRICE_FEED);
136
137   return (baseOraclePrice * (10 ** decimals).toInt256()) / (10 **
_baseUsdOracle.decimals()).toInt256();
138 }
```

Recommendation:

We advise similar provisions to be set in the `BridgeOracle::numeraireLatestAnswer` implementation, ensuring oracle measurements are performed solely when the sequencer is live.

Alleviation:

Proper validation of the sequencer's downtime has been introduced for the `BridgeOracle::numeraireLatestAnswer` function as advised, alleviating this exhibit in full.

DepositWithdrawPausable Manual Review Findings

DWP-01M: Upgradeability Incompatibility

Type	Severity	Location
Standard Conformity	Minor	DepositWithdrawPausable.sol:L10, L11

Description:

The `DepositWithdrawPausable` contract will employ fixed storage slots for its data entries without any form of gap or upgradeability-related accommodation.

Impact:

The `DepositWithdrawPausable` storage state is presently not compatible with the Tempest Finance code's upgradeability pattern.

Example:

src/utils/DepositWithdrawPausable.sol

```sol
abstract contract DepositWithdrawPausable is AccessControlUpgradeable {
  bool public depositPaused;
  bool public withdrawPaused;
```

Recommendation:

We advise the code to either employ **EIP-7201** storage slot name-spacing, or to implement a `gap` that would permit upgrade operations to be carried out securely.

Alleviation:

A gap entry has been introduced albeit with an incorrect `length`, permitting the contract to be upgraded properly.

LstAmbientStorageLayout Manual Review Findings

LAS-01M: Non-Standard Gap Specification

Type	Severity	Location
Standard Conformity	Informational	LstAmbientStorageLayout.sol:L29

Description:

The referenced gap specification is meant to be in compliance with the OpenZeppelin standard, however, it does not adhere to its length specification pattern.

Example:

src/lst/LstAmbientStorageLayout.sol

```sol
abstract contract LstAmbientStorageLayout {
  uint8 public assetIdx;
  uint8 public padding;
  uint8 internal _decimals;
  uint16 public fee;
  uint16 public investedPercentage;
  address public unWrappedToken;
  address public oracle;
  address public feeRecipient;
  address public operatorPath;
```

Example (Cont.):

```sol
    address public userPath;
    address[2] internal tokenAddresses;
    uint16[] internal lpWeights;
    LstParam[] internal lstParams;
    CrocsSwapDex public crocsSwapDex;
    CrocsQuery public crocsQuery;
    uint256 public invested;
    uint256 public minimumDeposit;
    uint256 public lastUnclaimedIndex;
    uint256 public lastWithdrawalRequestIndex;
    uint256[50] __gap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
}
```

Recommendation:

We advise the length to be configured to `50 - numberOfSlotsReserved` (`35`), ensuring that the array can be properly maintained per new storage entry introduced in a consistent manner.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RswEthAmbientStorageLayout Manual Review Findings

REL-01M: Non-Standard Gap Specification

Type	Severity	Location
Standard Conformity	Informational	RswEthAmbientStorageLayout.sol:L17

Description:

The referenced gap specification is meant to be in compliance with the OpenZeppelin standard, however, it does not adhere to its length specification pattern.

Example:

src/lst/rsweth/RswEthAmbientStorageLayout.sol

SOL

```sol
8   abstract contract RswEthAmbientStorageLayout is LstAmbientStorageLayout {
9     struct WithdrawalData {
10      uint256 requestId;
11      uint256 amount;
12      uint256 rate;
13    }
14
15    mapping(uint256 => WithdrawalData) internal withdrawalDatas;
16    IrswEXIT internal withdrawalQueue;
17    uint256[50] __rswEthGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
```

Example (Cont.):

```sol
}
```

Recommendation:

We advise the length to be configured to `50 - numberOfSlotsReserved` (`48`), ensuring that the array can be properly maintained per new storage entry introduced in a consistent manner.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

SymetricAmbientStorageLayout Manual Review Findings

SAL-01M: Non-Standard Gap Specification

Type	Severity	Location
Standard Conformity	Informational	SymetricAmbientStorageLayout.sol:L9

Description:

The referenced gap specification is meant to be in compliance with the OpenZeppelin standard, however, it does not adhere to its length specification pattern.

Example:

src/ambient-lp/SymetricAmbientStorageLayout.sol

SOL

```sol
6   contract SymetricAmbientStorageLayout is AmbientStorageLayout {
7     int24 public baseWidth;
8     int24 public limitWidth;
9     uint256[50] __symetricAmbientGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
10  }
```

Recommendation:

We advise the length to be configured to `50 - numberOfSlotsReserved` (`49`), ensuring that the array can be properly maintained per new storage entry introduced in a consistent manner.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

WstEthAmbientStorageLayout Manual Review Findings

WEL-01M: Non-Standard Gap Specification

Type	Severity	Location
Standard Conformity	Informational	WstEthAmbientStorageLayout.sol:L16

Description:

The referenced gap specification is meant to be in compliance with the OpenZeppelin standard, however, it does not adhere to its length specification pattern.

Example:

src/lst/wsteth/WstEthAmbientStorageLayout.sol

SOL

```sol
8   abstract contract WstEthAmbientStorageLayout is LstAmbientStorageLayout {
9     struct WithdrawalData {
10      uint256 requestId;
11      uint256 amount;
12    }
13
14    mapping(uint256 => WithdrawalData) internal withdrawalDatas;
15    IWithdrawalQueueERC721 internal withdrawalQueue;
16    uint256[50] __wstEthGap; // storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments
17  }
```

Recommendation:

We advise the length to be configured to `50 - numberOfSlotsReserved` (`48`), ensuring that the array can be properly maintained per new storage entry introduced in a consistent manner.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

AmbientInterface Code Style Findings

AIE-01C: Improper Specification of Function Signatures

Type	Severity	Location
Code Style	Informational	AmbientInterface.sol:L70, L88, L112, L143, L171, L184

Description:

The referenced statements will declare function signatures of the Tempest Finance codebase using string literals.

Example:

src/ambient-lp/AmbientInterface.sol

SOL

```sol
(bool success, bytes memory output) = operatorPath.delegatecall(
  abi.encodeWithSignature(
    "provideAllLiquidities(bool,uint128)",
    checkSlippage,
    _calOraclePrice(tokenAddresses).sqrtPrice
  )
);
```

Recommendation:

We advise the `abi.encodeWithSelector` mechanism to be employed for payload construction and the code to import function signatures via `interface` declarations and specifically the `selector` syntax (i.e. `IFoo.deposit.selector`).

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

AmbientOperator Code Style Findings

AOR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	AmbientOperator.sol:L96, L97

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/ambient-lp/AmbientOperator.sol

SOL

```
96  balances[1] > padding ? balances[1] - padding : 0,
97  balances[0] > padding ? balances[0] - padding : 0
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

AmbientUser Code Style Findings

AUR-01C: Redundant Duplication of Code

Type	Severity	Location
Gas Optimization	Informational	AmbientUser.sol:L547-L564, L586-L610

Description:

The referenced statements are replicated across the `AmbientUser::_withdrawLiquidity` and `AmbientUser::_withdrawLiquidityWithoutSwap` functions.

Example:

src/ambient-lp/AmbientUser.sol

SOL

```sol
function _withdrawLiquidity(
  LpParam[] memory _lpParams,
  uint256 _shares,
  address[2] memory _tokenAddresses,
  bool checkSlippage,
  uint128 sqrtOraclePrice
) internal returns (uint256 totalAmountToSend) {
  // Get the current token balances
  uint256[2] memory balBefore = VaultLibrary._getBalances(_tokenAddresses);
  (uint256 bal0Before, uint256 bal1Before) = (balBefore[0], balBefore[1]);
```

Example (Cont.):

```sol

    // Calculate the token balances for the given shares
    uint256 token0FromBal = bal0Before.mulDiv(_shares, totalSupply());
    uint256 token1FromBal = bal1Before.mulDiv(_shares, totalSupply());

    // Burn all liquidities for the given shares
    _burnAllLiquidities(_shares, _lpParams, _tokenAddresses, false, checkSlippage, sqrtOraclePrice);

    // Get the new token balances
    uint256[2] memory balAfter = VaultLibrary._getBalances(_tokenAddresses);
    (uint256 bal0After, uint256 bal1After) = (balAfter[0], balAfter[1]);

    // Calculate the balances to swap
    uint256 balance0ToSwap = bal0After - bal0Before + token0FromBal;
    uint256 balance1ToSwap = bal1After - bal1Before + token1FromBal;

    // Perform token swaps and calculate the total amount to send
    if (assetIdx == 0) {
      (uint256 receivedAmount, ) = _swapTokens(
        SwapParams({ tokenIn: tokenAddresses[1], tokenOut: tokenAddresses[0], amountIn: balance1ToSwap })
      );
      totalAmountToSend = receivedAmount + balance0ToSwap;
    } else {
      (uint256 receivedAmount, ) = _swapTokens(
        SwapParams({ tokenIn: tokenAddresses[0], tokenOut: tokenAddresses[1], amountIn: balance0ToSwap })
      );
      totalAmountToSend = receivedAmount + balance1ToSwap;
    }
```

Example (Cont.):

```sol
}
```

Recommendation:

We advise the `AmbientUser::_withdrawLiquidity` function to invoke the `AmbientUser::_withdrawLiquidityWithoutSwap` function, optimizing the code's generated bytecode.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

BaseAmbientStrategy Code Style Findings

BAS-01C: Improper Specification of Function Signature

Type	Severity	Location
Code Style	Informational	BaseAmbientStrategy.sol:L49

Description:

The referenced statement will declare a function signature of the Tempest Finance codebase using a string literal.

Example:

src/BaseAmbientStrategy.sol

SOL

```sol
48  (bool success, bytes memory output) = operatorPath.delegatecall(
49    abi.encodeWithSignature("rebalance(int24[],int24[],bool)", _newUpperTicks, _newLowerTicks, checkSlippage)
50  );
```

Recommendation:

We advise the `abi.encodeWithSelector` mechanism to be employed for payload construction and the code to import the relevant function signature via an `interface` declaration and specifically the `selector` syntax (i.e. `IFoo.deposit.selector`).

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

LstAmbientCommon Code Style Findings

LAC-01C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	LstAmbientCommon.sol:L77, L78

Description:

The referenced statements are redundantly wrapped in parenthesis' (`()`).

Example:

src/lst/LstAmbientCommon.sol

SOL

```
77  ? lstInUnderlying.mulDiv((uint256(amount1) + curValBal), 10 ** token1Decimals)
```

Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

The redundant parenthesis in the referenced statements have been safely omitted.

LstAmbientInterface Code Style Findings

LAI-01C: Improper Specification of Function Signatures

Type	Severity	Location
Code Style	Informational	LstAmbientInterface.sol:L69, L89, L115, L152, L170, L177, L191, L200, L338

Description:

The referenced statements will declare function signatures of the Tempest Finance codebase using string literals.

Example:

src/lst/LstAmbientInterface.sol

SOL

```sol
337 (bool success, bytes memory output) = operatorPath.delegatecall(
338   abi.encodeWithSignature("setInvestedPercentage(uint16,bytes)",
_investedPercentage, merkleProofs)
339 );
```

Recommendation:

We advise the `abi.encodeWithSelector` mechanism to be employed for payload construction and the code to import function signatures via `interface` declarations and specifically the `selector` syntax (i.e. `IFoo.deposit.selector`).

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

LstAmbientLiquidity Code Style Findings

LAL-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	LstAmbientLiquidity.sol:L193, L306

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/lst/LstAmbientLiquidity.sol

SOL

```sol
182 if (burnLiquidity.quantity <= padding) return;
183 cmd = abi.encode(
184   92, //Burn open knockout liquidity
185   burnLiquidity.tokenAddresses[0],
186   burnLiquidity.tokenAddresses[1],
187   POOL_IDX,
188   burnLiquidity.lstParam.lowerTick,
189   burnLiquidity.lstParam.upperTick,
190   burnLiquidity.isBid,
191   0,
```

Example (Cont.):

```sol
    abi.encode(
      burnLiquidity.quantity - padding, // uint128
      false, // bool
      true // bool
    )
);
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation:

The referenced arithmetic operation has been properly wrapped in an `unchecked` code block optimizing its gas cost.

LAL-02C: Potential Misuse of Profit Calculation Mechanism

Type	Severity	Location
Standard Conformity	Informational	LstAmbientLiquidity.sol:L305, L306, L311

Description:

The `LstAmbientLiquidity::_updateStratPnLAndFees` function will consider that the system has already incremented the `invested` amount by the input `depositAmount` which may not be true in all cases.

Example:

src/lst/LstAmbientLiquidity.sol

SOL

```sol
299 /// @notice Updates the strategy's profit and loss and calculates the fees to
claim.
300 /// @param totalAssets The current total assets in vault
301 /// @param depositAmount The user's deposit amount if this function is called
from deposit()
302 function _updateStratPnLAndFees(uint256 totalAssets, uint256 depositAmount) internal
{
303   uint256 _invested = invested;
304
305   if (totalAssets > _invested) {
306     uint256 assetForFees = ((totalAssets - _invested) * fee) / BASE;
307     uint256 _totalSupply = totalSupply();
308     if (assetForFees != 0 && _totalSupply != 0) {
```

Example (Cont.):

```sol
309         _mint(
310           feeRecipient,
311           VaultLibrary._convertToShares(assetForFees, totalAssets - assetForFees - depositAmount, _totalSupply)
312         );
313       }
314     }
315     invested = totalAssets;
316 }
```

Recommendation:

We advise the `invested` amount to be locally updated by the `LstAmbientLiquidity::_updateStratPnLAndFees` function rather than being updated at the `LstAmbientUser::deposit` function, ensuring that the function will properly accommodate for the `depositAmount` within the `invested` value in all its invocations.

Alleviation:

The Tempest Finance team evaluated this exhibit and opted to not apply the standardization advised.

LAL-03C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	LstAmbientLiquidity.sol:L63

Description:

The referenced statement is redundantly wrapped in parenthesis (`()`).

Example:

src/lst/LstAmbientLiquidity.sol

SOL

```sol
63  bool _isBid = (assetIdx == 0);
```

Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

The redundant parenthesis in the referenced statement have been safely omitted.

SymetricAmbientStrategy Code Style Findings

SAS-01C: Improper Specification of Function Signatures

Type	Severity	Location
Code Style	Informational	SymetricAmbientStrategy.sol:L59, L80

Description:

The referenced statements will declare function signatures of the Tempest Finance codebase using string literals.

Example:

src/SymetricAmbientStrategy.sol

SOL

```sol
58  (bool success, bytes memory output) = operatorPath.delegatecall(
59    abi.encodeWithSignature("rebalance(int24,int24,bool)", baseWidth, limitWidth, checkSlippage)
60  );
```

Recommendation:

We advise the `abi.encodeWithSelector` mechanism to be employed for payload construction and the code to import function signatures via `interface` declarations and specifically the `selector` syntax (i.e. `IFoo.deposit.selector`).

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

VaultLibrary Code Style Findings

VLY-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	VaultLibrary.sol:L136

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/libraries/VaultLibrary.sol

SOL

```sol
135 if (amount <= DEAD_SHARES) revert(SMALL_AMOUNT);
136 return amount - DEAD_SHARES;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation:

The referenced arithmetic operation has been properly wrapped in an `unchecked` code block optimizing its gas cost.

VLY-02C: Redundant Bit Clearing

Type	Severity	Location
Gas Optimization	Informational	VaultLibrary.sol:L95

Description:

The referenced operation is meant to clear the upper bits of `valOne` to ensure that only 128 bits are utilized, however, the operation is redundant as the `uint128` type cast will perform the same.

Example:

src/libraries/VaultLibrary.sol

SOL

```sol
95  uint128 curvePrice = uint128((valOne << 128) >> 128);
```

Recommendation:

We advise the casting operation to remain as-is and the shift operations to be omitted, optimizing the code's gas cost.

Alleviation:

The Tempest Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

VLY-03C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	VaultLibrary.sol:L45, L47, L50, L54

Description:

The referenced statements are redundantly wrapped in parenthesis' (`()`).

Example:

src/libraries/VaultLibrary.sol

SOL

```
45  newUpperTicks[0] = _floorDivide((tickAtOraclePrice + params.baseWidth),
params.tickSize) * params.tickSize;
```

Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

Alleviation:

The redundant parenthesis in the referenced statements have been safely omitted.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.