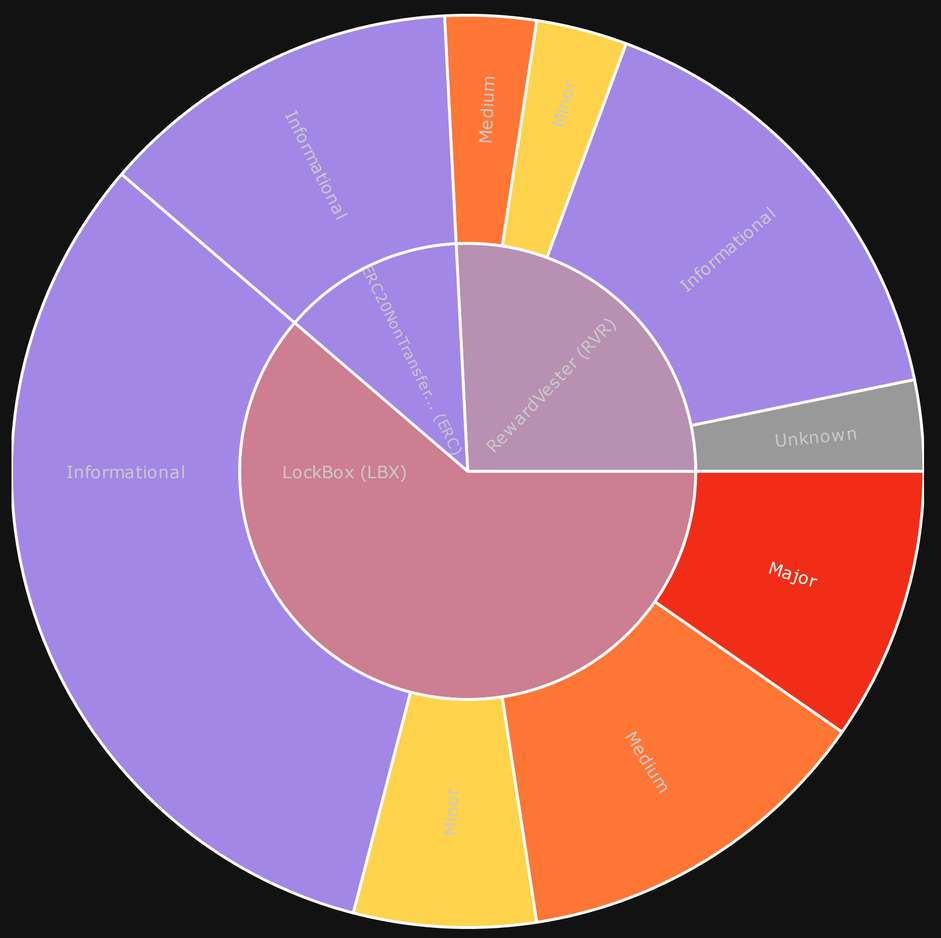
Scope

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/Fantom-Money-Market/MultiRewards-Working-Version**
- Commit: f0531e9dee005d990384257506ca638c50390a1f
- Language: Solidity
- Network: Fantom
- Revisions: **f0531e9dee**, **0dd3c4b420**, **28160b61c6**

Findings Per File



Contracts Assessed

File	Total Finding(s)
contracts/ERC20/Context.sol (CTX)	0
contracts/ERC20/ERC20NonTransferable.sol (ERC)	4
contracts/LockBox.sol (LBX)	19
contracts/RewardVester.sol (RVR)	8
contracts/ERC20/draft-IERC6093.sol (IER)	0

Compilation

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

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.20` based on the version specified within the `hardhat.config.js` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely located in external dependencies and can thus be safely ignored.

In the state the codebase was supplied to us, the code could not be compiled due to an error in the `LockBox` contract and specifically the closing brackets in the `LockBox::_getUserRewardRate` code segment.

Apart from this issue, the `package.json` configuration of the system had conflicting versions that could not be automatically resolved. We ported the codebase to a manual `hardhat` installation to compile it as a result.

The `pragma` statements have been locked to `0.8.20` (`=0.8.20`), the same version utilized for our static analysis as well as optimizational review of the codebase.

Static Analysis

The execution of our static analysis toolkit identified **41 potential issues** within the codebase of which **19 were ruled out to be false positives** or negligible findings.

The remaining **22 issues** were validated and grouped and formalized into the **7 exhibits** that follow:

ID	Severity	Addressed	Title
LBX-01S	Informational	Acknowledged	Illegible Numeric Value Representations
LBX-02S	Informational	Yes	Inexistent Event Emission
LBX-03S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses
LBX-04S	Informational	Yes	Inexistent Visibility Specifier
LBX-05S	Medium	Yes	Improper Invocations of EIP-20 `transfer`
RVR-01S	Informational	Yes	Inexistent Event Emissions
RVR-02S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses

PREV
Compilation

NEXT
Manual Review

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Fantom's locking and vesting systems.

As the project at hand implements Fantom, 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 multiple significant vulnerabilities** within the system which could have had **severe ramifications** to its overall operation; for more details, kindly consult the audit's summary.

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 a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the integration points of the `IVault` system present in the `RewardVester` contract.

A total of **24 findings** were identified over the course of the manual review of which **11 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
LBX-01M	Minor	Yes	Improper Calculations of Penalties
LBX-02M	Minor	Partial	Inefficient Management of Transfer Restrictions
LBX-03M	Medium	Yes	Incorrect Burn Operation
LBX-04M	Medium	Yes	Incorrect Conditional Evaluation
LBX-05M	Medium	Yes	Incorrect Order of Operations
LBX-06M	Major	Yes	Inexistent Prevention of Self Transfer (Locking)
LBX-07M	Major	Yes	Inexistent Update of Recipient Rewards (Locking)
LBX-08M	Major	Yes	Inexistent Update of Treasury Rewards
RVR-01M	Unknown	Acknowledged	Unknown Integration Points
RVR-02M	Minor	Nullified	Inexistent Validation of Preference
RVR-03M	Medium	Partial	Improper Evaluation of USDC Pair

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
ERC-01C	Informational	Yes	Generic Typographic Mistakes
ERC-02C	Informational	Nullified	Inefficient Variable Mutability
ERC-03C	Informational	Yes	Redundant Conditional Logic
ERC-04C	Informational	Nullified	Repetitive Value Literal
LBX-01C	Informational	Acknowledged	Generic Typographic Mistakes
LBX-02C	Informational	Yes	Inefficient Mapping Structures
LBX-03C	Informational	Partial	Inefficient `mapping` Lookups
LBX-04C	Informational	Acknowledged	Inexistent Error Messages
LBX-05C	Informational	Yes	Redundant Calculation of Value Outside Code Block
LBX-06C	Informational	Acknowledged	Repetitive Value Literal
RVR-01C	Informational	Yes	Inefficient Mapping Structures
RVR-02C	Informational	Acknowledged	Inexistent Error Messages
RVR-03C	Informational	Acknowledged	Suboptimal Struct Declaration Style

PREV
Manual Review

NEXT
LockBox.sol (LBX-S)

LockBox Static Analysis Findings

LBX-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	LockBox.sol:L40, L41, L42, L46, L47, L48

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/LockBox.sol

```
SOL
40 uint internal constant MINLOCK = 4 * 7 * 86400;
```

Recommendation:

To properly illustrate each 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` `100` becomes `1_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 (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

LBX-02S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	LockBox.sol:L745-L748

Description:

The linked function adjusts a sensitive contract variable yet does not emit an event for it.

Example:

contracts/LockBox.sol

```
SOL
745 function setSonicMigration(bool state) external onlyRole(DEFAULT_ADMIN_ROLE)
746     if(!paused){revert NotPaused();}
747     sonicMigration = state;
748 }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted to ensure off-chain processes can properly react to this system adjustment.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `SonicMigrated` event was introduced to the codebase and is correspondingly emitted in the `LockBox::setSonicMigration` function, addressing this exhibit in full.

View GitHub Discussion

LBX-03S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	LockBox.sol:L105-L133

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:

contracts/LockBox.sol

```
SOL
105 constructor(
106     address[3] memory _operators,
107     address _admin,
108     address _treasury,
109     address _stakingtoken,
110     address _masterChef,
111     address _rewardVester,
112     address _beets,
113     string memory _name,
114     string memory _symbol
115 ) ERC20(_name, _symbol) ReentrancyGuard() {
116     _grantRole(DEFAULT_ADMIN_ROLE, _admin);
117     _grantRole(OPERATOR_ROLE, _admin);
118     _grantRole(OPERATOR_ROLE, _operators[0]);
119     _grantRole(OPERATOR_ROLE, _operators[1]);
120     _grantRole(OPERATOR_ROLE, _operators[2]);
121
122     multisig = _admin;
123     treasury = _treasury;
124     stakingToken = _stakingtoken;
125     masterChef = _masterChef;
126     rewardVester = _rewardVester;
127     beets = _beets;
128
129     rewards.push(_beets);
130     isReward[_beets] = true;
131
132     IERC20(stakingToken).approve(_masterChef, type(uint).max);
133 }
```

Recommendation:

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

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The Fantom team evaluated this exhibit and opted to not apply input sanitization as the configurational arguments of the `LockBox` are defined by automated scripts in the deployment stack of the Fantom team.

As such, we consider this exhibit safely acknowledged.

View GitHub Discussion

LBX-04S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	LockBox.sol:L31

Description:

The linked variable has no visibility specifier explicitly set.

Example:

contracts/LockBox.sol

```
SOL
31 bytes32 constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");
```

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 (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

LBX-05S: Improper Invocations of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	LockBox.sol:L734, L741

Description:

The linked statement do not properly validate the returned `bool` of the **EIP-20** standard `transfer` function. As the **standard dictates**, callers **must not** assume that `false` is never returned.

Impact:

If the code mandates that the returned `bool` is `true`, this will cause incompatibility with tokens such as USDT / Tether as no such `bool` is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

Example:

contracts/LockBox.sol

```
SOL
734 IERC20(token).transfer(to, amount);
```

Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as `SafeERC20` by OpenZeppelin to opportunistically validate the returned `bool` only if it exists in each instance.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

Both invocations of `ERC20::transfer` have been replaced by the internal utility function `LockBox::_safeTransfer`, ensuring that they are safely performed regardless of the specific underlying **EIP-20** implementation.

View GitHub Discussion

RewardVester Static Analysis Findings

RVR-01S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	RewardVester.sol:L217-L222, L226-L228

Description:

The linked functions adjust sensitive contract variables yet do not emit an event for it.

Example:

contracts/RewardVester.sol

SOL Copy

```sol
226function setPaused(bool state) external onlyRole(DEFAULT_ADMIN_ROLE){
227    isPaused = state;
228}
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation (0dd3c4b420):

An `event` was declared and is correspondingly emitted solely for one of the two referenced functions by the exhibit, rendering it partially addressed.

Alleviation (28160b61c6):

An `event` has been properly declared for the `RewardVester::setLockBox` function, addressing this exhibit in full.

View GitHub Discussion

RVR-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	RewardVester.sol:L57-L80

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:

contracts/RewardVester.sol

SOL Copy

```sol
57 constructor(
58     address _admin,
59     address _fBux,
60     address _usdc,
61     address _lp,
62     address _unitroller,
63     address _chef,
64     address _vault,
65     bytes32 _poolId
66 ) {
67     _grantRole(DEFAULT_ADMIN_ROLE, _admin);
68
69     multisig = _admin;
70     usdc = _usdc;
71     fBux = _fBux;
72     unitroller = _unitroller;
73     chef = _chef;
74     vault = _vault;
75     poolId = _poolId;
76     lp = _lp;
77     lpTokens = [fBux, usdc];
78
79     renewApprovals();
80 }
```

Recommendation:

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

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The Fantom team evaluated this exhibit and opted to not apply input sanitization as the configurational arguments of the `RewardVester` are defined by automated scripts in the deployment stack of the Fantom team.

As such, we consider this exhibit safely acknowledged.

View GitHub Discussion

ON THIS PAGE

LockBox Manual Review Findings

LBX-01M: Improper Calculations of Penalties

Type	Severity	Location
Mathematical Operations	Minor	LockBox.sol:L375-L377, L505-L507

Description:

The `LockBox::earlyUnlock` function will improperly calculate the proportionate penalties to be applied to an early withdrawal leading to truncation issues and thus lost funds.

Impact:

The sum of the retained and received values may not actually match the `amount` due to truncation errors.

Example:

contracts/LockBox.sol

```sol
375uint userReceived = amount - ((amount * EARLYPENALTY) / PENALTYDIV);
376uint lockerRetained = (amount * LOCKRETAINED) / PENALTYDIV;
377uint treasuryRetained = (amount * TREASURYRETAINED) / PENALTYDIV;
```

Recommendation:

We advise the code to calculate the early penalty and store it to a local variable, and to calculate the `lockerRetained` and `treasuryRetained` as a proportion of the early penalty rather than the full amount. Additionally, we advise the `treasuryRetained` to be assigned to the value of the early penalty minus the locker retained amount ensuring that truncation errors are accounted for.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The penalty calculations have all been corrected to fully accommodate for potential truncation issues ensuring the full amount is disbursed by the contract and thus addressing this exhibit.

View GitHub Discussion

LBX-02M: Inefficient Management of Transfer Restrictions

Type	Severity	Location
Logical Fault	Minor	LockBox.sol:L135-L139, L206, L288, L306, L358, L404, L411, L458, L490, L534, L632, L785-L790, L792-L797

Description:

The `LockBox` inherits from the custom `ERC20` implementation of the Fantom team that imposes transfer restrictions based on a contract-level `uint256` variable that is overridden temporarily by the `LockBox` via the `LockBox::allowTransfer` modifier.

This system is significantly inefficient, as the `LockBox` contract can directly invoke the `ERC20::_update` function of its dependency to bypass the `transferBlocked` related restriction.

Impact:

The present system is inefficient, prone to exploitation in a theoretically re-entrant call, and incurs a significant gas overhead.

Example:

contracts/LockBox.sol

```sol
135modifier allowTransfer(){
136    transferBlocked = 1;
137    _;
138    transferBlocked = 2;
139}
```

Recommendation:

We advise the code to be revamped based on our recommendation in the `ERC20NonTransferable` contract and this one, omitting the `transferBlocked` related logic entirely and making use of the `ERC20::_update` function whenever needed.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `LockBox::allowTransfer` modifier has been appropriately removed from the contract in all cases except for the `LockBox::notifyRewardAmount` function which will fail to function as expected for an input `token` equal to the contract itself.

We advise this case to be revisited and special handling to be introduced if deemed necessary.

View GitHub Discussion

LBX-03M: Incorrect Burn Operation

Type	Severity	Location
Mathematical Operations	Medium	LockBox.sol:L379, L509

Description:

The amount of the `user` burned by the `LockBox::earlyUnlock` and `LockBox::earlyUnvest` functions is incorrect as the full `amount` should be burned.

Impact:

A user will retain their `LockBox` balance even though they opted to perform an early unlock.

Example:

contracts/LockBox.sol

```sol
371userLockedAmount[user] -= amount;
372
373if(userLockedAmount[user] == 0){isLocked[user] = false; lockedFor[user] = 0;
374
375uint userReceived = amount - ((amount * EARLYPENALTY) / PENALTYDIV);
376uint lockerRetained = (amount * LOCKRETAINED) / PENALTYDIV;
377uint treasuryRetained = (amount * TREASURYRETAINED) / PENALTYDIV;
378
379_burn(user, userReceived + treasuryRetained);
```

Recommendation:

We advise the full `amount` to be burned, preventing the user from having a non-zero `LockBox` balance without any vest or lock entries.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The full `amount` is adequately burned in both scenarios outlined by the exhibit, alleviating in full.

View GitHub Discussion

LBX-04M: Incorrect Conditional Evaluation

Type	Severity	Location
Logical Fault	Medium	LockBox.sol:L662, L663

Description:

The `LockBox::harvestBeets` function is meant to utilize `unsyncedBeets`, however, its execution is guarded by whether there are unsynced beets **actively available** rather than at rest.

As a result, a user can hijack all beet harvest operations by performing an action that would automatically harvest them (i.e. a withdrawal) in the same block.

Impact:

All `LockBox::harvestBeets` operations can be hijacked to prevent their use in the system.

Example:

contracts/LockBox.sol

```sol
661uint _unsyncedBeets = beetsBalanceAfter - beetsBalance;
662if(_unsyncedBeets == 0){revert ZeroAmount();}
663_unsyncedBeets += unsyncedBeets;
664unsyncedBeets = 0;
```

Recommendation:

We advise the `LockBox::harvestBeets` function to evaluate whether the `_unsyncedBeets` value is `0` after it has been incremented by the `unsyncedBeets` contract level variable.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The referenced `if` conditional has been relocated after the increment of storage-tracked `unsyncedBeets`, rendering this exhibit alleviated.

View GitHub Discussion

LBX-05M: Incorrect Order of Operations

Type	Severity	Location
Logical Fault	Medium	LockBox.sol:L189-L194

Description:

The `LockBox::earned` function contains an incorrect order of operations as it will divide the reward per token delta by `PRECISION` prior to multiplying it with the balance of the user, thereby causing truncation issues to manifest.

Impact:

Any reward per token delta that is not a multiple of the `PRECISION` will improperly truncate the multiplier by one and in most circumstances will result in a `0` multiplier thereby not offering any rewards to the `account` even though some should be awarded to it.

Example:

contracts/LockBox.sol

```sol
186// Returns depositor's accrued rewards
187function earned(address token,address account) public view returns (uint _rew
188    uint userRewardRate = _getUserRewardRate();
189    _reward =
190        ((balanceOf(account) *
191            (((rewardPerToken(token) -
192                userRewardPerTokenStored[account][token])) / PRECISION) *
193                userRewardRate) / 100) +
194                storedRewardsPerUser[account][token];
195}
```

Recommendation:

We advise the code to multiply the reward per token delta with the balance of the `account` prior to dividing by the `PRECISION`, ensuring a higher degree of accuracy is maintained in the reward calculations of the system.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The order of operations has been corrected, ensuring that the delta of the reward per token stored is multiplied by the `account` balance before being divided by the multiplier's precision, thereby maximizing the possible accuracy of the overall mathematical formula.

View GitHub Discussion

LBX-06M: Inexistent Prevention of Self Transfer (Locking)

Type	Severity	Location
Logical Fault	Major	LockBox.sol:L409-L440

Description:

The `LockBox::transferLock` does not prevent a self-transfer from occurring, allowing a user to eliminate their lock status and locked for duration by performing a full amount self transfer.

A user can exploit this path to reduce their maximum lock to the minimum possible, effectively compromising the locking system.

Impact:

It is presently possible to reduce a maximum lock to the minimum possible, effectively capturing rewards as if a user has had the maximum lock duration with the minimum available.

Example:

contracts/LockBox.sol

```sol
409// Partial or complete transfer of a lock to new or existing lock. If receiv
410// If receiver isn't locked, creates one with the same lockedFor as msg.send
411function transferLock(address receiver, uint amount) external nonReentrant a
412    address user = msg.sender;
413
414    if(!isLocked[user]){revert NoLock();}
415    if(_getTimestamp() >= lockedFor[user]){revert Expired();}
416    if(amount == 0){revert NotAllowed();}
417    if(amount > userLockedAmount[user]){amount = userLockedAmount[user];}
418
419    if(isLocked[receiver]){
420    if(lockedFor[receiver] < lockedFor[user]){revert NotAllowed();}} // Can'
421
422    if(!isLocked[receiver]){
423        isLocked[receiver] = true;
424        lockedFor[receiver] = lockedFor[user];
425        emit LockCreated(receiver, amount, lockedFor[receiver]);
426    }
427
428    userLockedAmount[user] -= amount;
429
430    if(userLockedAmount[user] == 0){
431        isLocked[user] = false;
432        lockedFor[user] = 0;
433    }
434
435    _burn(user, amount);
436    _mint(receiver, amount);
437    userLockedAmount[receiver] += amount;
438
439    emit LockTransfered(user, receiver, amount);
440}
```

Recommendation:

We advise the `LockBox::transferLock` function to prevent a self-transfer from occurring, preventing the misbehaviour outlined from manifesting.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

An `if` conditional was introduced that prevents the `receiver` from matching the `user` from which the transfer occurs, preventing the misbehaviour outlined by the exhibit and thus addressing it in full.

View GitHub Discussion

LBX-07M: Inexistent Update of Recipient Rewards (Locking)

Type	Severity	Location
Logical Fault	Major	LockBox.sol:L411

Description:

The `LockBox::transferLock` function does not update the rewards of the recipient, permitting them to exploit the lock transfer system to acquire disproportionate rewards.

Impact:

A user can acquire inflationary rewards by transferring a lock to themselves as their reward index would be updated after the balance change as if they held those units since their last interaction with the contract.

Example:

contracts/LockBox.sol

```sol
409// Partial or complete transfer of a lock to new or existing lock. If receiv
410// If receiver isn't locked, creates one with the same lockedFor as msg.send
411function transferLock(address receiver, uint amount) external nonReentrant a
412    address user = msg.sender;
413
414    if(!isLocked[user]){revert NoLock();}
415    if(_getTimestamp() >= lockedFor[user]){revert Expired();}
416    if(amount == 0){revert NotAllowed();}
417    if(amount > userLockedAmount[user]){amount = userLockedAmount[user];}
418
419    if(isLocked[receiver]){
420    if(lockedFor[receiver] < lockedFor[user]){revert NotAllowed();}} // Can'
421
422    if(!isLocked[receiver]){
423        isLocked[receiver] = true;
424        lockedFor[receiver] = lockedFor[user];
425        emit LockCreated(receiver, amount, lockedFor[receiver]);
426    }
427
428    userLockedAmount[user] -= amount;
429
430    if(userLockedAmount[user] == 0){
431        isLocked[user] = false;
432        lockedFor[user] = 0;
433    }
434
435    _burn(user, amount);
436    _mint(receiver, amount);
437    userLockedAmount[receiver] += amount;
438
439    emit LockTransfered(user, receiver, amount);
440}
```

Recommendation:

We advise the `LockBox::updateReward` modifier to be introduced for the `receiver` as well, preventing improper rewards from being distributed to them.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The recipient's rewards are now properly updated whenever a lock transfer occurs, alleviating this exhibit in full.

View GitHub Discussion

LBX-08M: Inexistent Update of Treasury Rewards

Type	Severity	Location
Logical Fault	Major	LockBox.sol:L364, L382-L389, L512-L519

Description:

The early unvest and early unlock mechanisms apply a penalty to the relevant amount that is effectively transferred to the `treasury` as a vest / lock entry.

As the system does not update the rewards of the treasury before this action, the treasury will acquire inflated rewards on their next interaction with the contract.

Impact:

The rewards the treasury acquires will be significantly higher than expected due to its reward index not being properly maintained.

Example:

contracts/LockBox.sol

```sol
495function earlyUnvest(uint amount) external nonReentrant allowTransfer update
496    address user = msg.sender;
497    if(!isVested[user]){revert NoVest();}
498    if(_getTimestamp() >= vestedFor[user]){revert Expired();}
499    if(amount == 0) {revert ZeroAmount();}
500    if(amount > userVestedAmount[user]){amount = userVestedAmount[user];}
501
502    userVestedAmount[user] -= amount;
503    if(userVestedAmount[user] == 0){isVested[user] = false;}
504
505    uint userReceived = amount - (amount * EARLYPENALTY / PENALTYDIV);
506    uint lockerRetained = amount * LOCKRETAINED / PENALTYDIV;
507    uint treasuryRetained = amount * TREASURYRETAINED / PENALTYDIV;
508
509    _burn(user, userReceived + treasuryRetained);
510    _mint(treasury, treasuryRetained);
511
512    if(!isLocked[treasury]){
513        isLocked[treasury] = true;
514        vestedFor[treasury] = block.timestamp + MAXLOCK;
515        userLockedAmount[treasury] += treasuryRetained;
516
517    } else {
518        userLockedAmount[treasury] += treasuryRetained;
519    }
520
521    _distroEarlyPenalty(lockerRetained);
522
523    (uint chefBal,) = chefBalances();
524
525    if(chefBal >= userReceived){
526    uint beetsBalance = _balanceOf(beets, address(this));
527    IMasterChef(masterChef).withdrawAndHarvest(CHEFID, userReceived, address
528    uint beetsBalanceAfter = _balanceOf(beets, address(this));
529
530    uint _unsyncedBeets = beetsBalanceAfter - beetsBalance;
531    if (_unsyncedBeets > 0) {unsyncedBeets += _unsyncedBeets;}
532    }
533
534    _safeTransfer(stakingToken, user, userReceived);
535
536    emit UnlockedEarly(user, userReceived, lockerRetained + treasuryRetained
537}
```

Recommendation:

We advise the `LockBox::updateReward` modifier to be properly invoked for the treasury, preventing this inflationary action from manifesting.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The rewards of the `treasury` are now updated at both functions in which the `treasury` may receive locked amounts, addressing the misbehaviour outlined by the exhibit.

View GitHub Discussion

RewardVester Manual Review Findings

RVR-01M: Unknown Integration Points

Type	Severity	Location
Standard Conformity	Unknown	RewardVester.sol:L171, L187

Description:

The referenced interactions are performed with an external vault-like contract whose implementation is unknown.

Example:

contracts/RewardVester.sol

```sol
/// @notice Adds liquidity to fMoney's 80/20 pool on BeethovenX.
function _joinPool(uint fBuxAmt, uint usdcAmt) internal {
    uint256[] memory amounts = new uint256[](lpTokens.length);

    amounts[0] = fBuxAmt;
    amounts[1] = usdcAmt;
    bytes memory userData = abi.encode(1, amounts, 1);

    IVault.JoinPoolRequest memory request = IVault.JoinPoolRequest(lpTokens,
    IVault(vault).joinPool(poolId, address(this), address(this), request);
}

/// @notice Returns remaining vest time for a user
function timeLeft(address user) external view returns (uint) {
    if(userVestedAmount[user] == 0){ return 0;}
    if(block.timestamp >= userVestEnd[user]){return 0;}
    return userVestEnd[user] - block.timestamp;
}

// Returns total vested fBux in contract
function vestedfBux() external view returns (uint fBuxBal){
    fBuxBal = IERC20(fBux).balanceOf(address(this));
}
/// @notice Returns user required usdc to earlyUnvest an fBux amount;
function getUsdcPairAmount(uint fBuxAmount) public view returns (uint usdcRe
    (, uint256[] memory balances, ) = IVault(vault).getPoolTokens(poolId);
    usdcRequired = fBuxAmount * balances[1] / balances[0];
}
```

Recommendation:

We advise the Fantom team to ensure that the integration has been securely performed and behaves as expected, for example by ensuring that the full balance of the contract is utilized during a pool join request.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The Fantom team stated that they have re-applied the same logic in a different set of contracts with success and that they have consulted with the Balancer team directly.

As we cannot validate any of these statements directly within the confines of the security audit, we consider this exhibit acknowledged **based on these statements accepted as facts**

View GitHub Discussion

RVR-02M: Inexistent Validation of Preference

Type	Severity	Location
Logical Fault	Minor	RewardVester.sol:L120-L145

Description:

The `RewardVester::earlyClaim` will permit an `fBUX` vesting position to be converted into an `fBUX/USDC` LP position with a reduced vesting window, however, the present mechanism does not evaluate whether the conversion makes sense based on the remaining time of the `RewardVester` vesting entry.

Impact:

A user who wishes to claim early might cause their position to be vested for a longer duration inadvertently due to the fixed duration for which `LockBox::createVest` function executions will create vesting entries for.

Example:

contracts/RewardVester.sol

```sol
/// @notice Allows partial or full exit from 3 month fBux vest into a 1.5M l
/// If vest expired, _forceClaim rewards for user and reset stored info. Req
function earlyClaim(uint fBuxAmount) external nonReentrant {
    address user = msg.sender;

    if(!isVested[user]) {revert NotVested();}
    if(isPaused){revert Paused();}
    if(fBuxAmount == 0){revert ZeroAmount();}

    if(block.timestamp >= userVestEnd[user]){_forceClaim(user);}
    else{
        uint toClaim = userVestedAmount[user];
        if(toClaim == 0){revert ZeroAmount();}
        if(fBuxAmount > toClaim){fBuxAmount = toClaim;}

        userVestedAmount[user] -= fBuxAmount;

        // Create LP on BeethovenX & vest on Lockbox.
        uint usdcAmount = getUsdcPairAmount(fBuxAmount);
        _safeTransferFrom(usdc, user, address(this), usdcAmount);
        _joinPool(fBuxAmount, usdcAmount);

        uint lpAmount = IERC20(lp).balanceOf(address(this));
        lockBox(lockBox).createVest(user, lpAmount);

        emit ClaimedEarly(user, fBuxAmount, usdcAmount, lpAmount);
    }
}
```

Recommendation:

We advise the `RewardVester::earlyClaim` function to prevent invocation if the vest duration as established in the `LockBox` will become higher than the remaining vest duration of the local `RewardVester` entry.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The Fantom team evaluated this submission and stated that there are incentives to convert a user's position even if they end up with a higher lock time due to being able to receive yield via the LP position while their assets are being vested.

As such, we consider this exhibit to be invalid given that there are other incentives beyond time involved in converting a token vest into an LP one.

View GitHub Discussion

RVR-03M: Improper Evaluation of USDC Pair

Type	Severity	Location
External Call Validation	Medium	RewardVester.sol:L187-L188

Description:

The `RewardVester::getUsdcPairAmount` function is not flash-loan resistant, permitting it to be manipulated and users to be harmed by depositing more USDC to pair the `fBUX` with than necessary.

Impact:

A user interacting with the `RewardVester` to claim their vest early as a locked LP position may be manipulated into oversupplying USDC for the said position, thereby significantly reducing the value that the user would benefit from via this conversion mechanism.

Example:

contracts/RewardVester.sol

```sol
/// @notice Allows partial or full exit from 3 month fBux vest into a 1.5M l
/// If vest expired, _forceClaim rewards for user and reset stored info. Req
function earlyClaim(uint fBuxAmount) external nonReentrant {
    address user = msg.sender;

    if(!isVested[user]) {revert NotVested();}
    if(isPaused){revert Paused();}
    if(fBuxAmount == 0){revert ZeroAmount();}

    if(block.timestamp >= userVestEnd[user]){_forceClaim(user);}
    else{
        uint toClaim = userVestedAmount[user];
        if(toClaim == 0){revert ZeroAmount();}
        if(fBuxAmount > toClaim){fBuxAmount = toClaim;}

        userVestedAmount[user] -= fBuxAmount;

        // Create LP on BeethovenX & vest on Lockbox.
        uint usdcAmount = getUsdcPairAmount(fBuxAmount);
        _safeTransferFrom(usdc, user, address(this), usdcAmount);
        _joinPool(fBuxAmount, usdcAmount);

        uint lpAmount = IERC20(lp).balanceOf(address(this));
        lockBox(lockBox).createVest(user, lpAmount);

        emit ClaimedEarly(user, fBuxAmount, usdcAmount, lpAmount);
    }
}
```

Recommendation:

We advise the amount of USDC funds to be paired with the `fBUX` to be defined by the user themselves, permitting them to denote the exact amount they wish to use for creating the LP position.

Alleviation (0dd3c4b420):

The Fantom team evaluated this submission and opted to acknowledge it.

We do not consider this action adequate from a security perspective as the Fantom team's own deliberation indicates a drain-type attack could in theory manifest. As such, we still advise a user-based USDC amount limitation to be introduced to mitigate this attack path.

Alleviation (28160b61c6):

The code was instead adjusted to accept `USDC` as the amount to pair with available `fBUX` that are vested, thereby mitigating the maximum effect of this issue.

The system still permits more `fBUX` to be paired with the `USDC` than necessary, thereby rendering this exhibit partially alleviated. We advise two slippage arguments to be supplied which imitates the behaviour of Uniswap LP provisioning functions themselves, highlighting that this is an issue that should be properly addressed.

View GitHub Discussion

ERC20NonTransferable Code Style Findings

ERC-01C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	ERC20NonTransferable.sol:L36, L51

Description:

The referenced lines contain typographical mistakes (i.e. `private` variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.

Example:

contracts/ERC20/ERC20NonTransferable.sol

SOL Copy

```
36 error transferDenied();
```

Recommendation:

We advise them to be corrected enhancing the legibility of the codebase.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The referenced `error` has been corrected to `TransferDenied` addressing this exhibit.

View GitHub Discussion

ERC-02C: Inefficient Variable Mutability

Type	Severity	Location
Gas Optimization	Informational	ERC20NonTransferable.sol:L43, L57

Description:

The `transferBlocked` variable does not have an optimal mutability specifier yet is assigned to only once during the `ERC20::constructor`.

Example:

contracts/ERC20/ERC20NonTransferable.sol

SOL Copy

```
43 uint256 internal transferBlocked;
44
45 string private _name;
46 string private _symbol;
47
48 /**
49  * @dev Sets the values for {name} and {symbol}.
50  *
51  * All two of these values are immutable: they can only be set once during
52  * construction.
53  */
54 constructor(string memory name_, string memory symbol_) {
55     _name = name_;
56     _symbol = symbol_;
57     transferBlocked = 2;
58 }
```

Recommendation:

We advise the variable's assignment to be relocated to its declaration, and its mutability to be set to `constant` optimizing the code's gas cost significantly.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `transferBlocked` notion has been removed from the codebase due to alleviations meant for `ERC-03C`, rendering this exhibit no longer applicable.

View GitHub Discussion

ERC-03C: Redundant Conditional Logic

Type	Severity	Location
Gas Optimization	Informational	ERC20NonTransferable.sol:L115, L161

Description:

Per the `ERC20` contract's constructor, the `transferBlocked` variable will always equate to `2`. As such, conditional logic in relation to it is unnecessary as it will always be executed.

Example:

contracts/ERC20/ERC20NonTransferable.sol

SOL Copy

```
114function transfer(address to, uint256 value) public virtual returns (bool) {
115    if(transferBlocked == 2){revert transferDenied();}
116    address owner = msg.sender;
117    _transfer(owner, to, value);
118    return true;
119}
```

Recommendation:

We advise the contents of the `ERC20::transfer` and `ERC20::transferFrom` functions to be replaced by direct `revert` statements of the `transferDenied` error, optimizing the code's gas cost.

The `LockBox` implementation can utilize the `ERC20::_mint`, `ERC20::_burn`, and `ERC20::_update` functions if needed which do not impose transfer related restrictions.

Alleviation (0dd3c4b420):

While the instances of conditional logic have been replaced by direct `revert` statements, all statements that follow them are unreachable and should be removed as well.

Alleviation (28160b61c6):

The redundant statements have been properly removed, addressing this exhibit in full.

View GitHub Discussion

ERC-04C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	ERC20NonTransferable.sol:L57, L115, L161

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

contracts/ERC20/ERC20NonTransferable.sol

SOL Copy

```
57 transferBlocked = 2;
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `transferBlocked` notion has been removed from the codebase due to alleviations meant for `ERC-03C`, rendering this exhibit no longer applicable.

View GitHub Discussion

LockBox Code Style Findings

LBX-01C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	LockBox.sol:L216-L220, L254-L264

Description:

The referenced lines contain typographical mistakes (i.e. `private` variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.

Example:

contracts/LockBox.sol

```
SOL
215if (lockedFor[user] >= vestedFor[user]) {
216    userLockedAmount[user] += _reward;
217  } else {
218    userVestedAmount[user] += _reward;
219}
```

Recommendation:

We advise them to be corrected enhancing the legibility of the codebase.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The code segments remain with improper indentation rendering the exhibit to be acknowledged.

View GitHub Discussion

LBX-02C: Inefficient Mapping Structures

Type	Severity	Location
Gas Optimization	Informational	LockBox.sol:L61-L62, L64-L69

Description:

The referenced `mapping` declarations all use the same key yet are located distinctly, incurring extraneous gas when they are simultaneously accessed and/or written to.

Example:

contracts/LockBox.sol

```
SOL
60 mapping(address token => Reward) internal _rewardData;
61 mapping(address user => mapping(address token => uint rewardPerToken)) publi
62 mapping(address user => mapping(address token => uint reward)) public stored
63 mapping(address token => bool) public isReward;
64 mapping(address user => bool) public isLocked;
65 mapping(address user => bool) public isVested;
66 mapping(address user => uint) public lockedFor;
67 mapping(address user => uint) public vestedFor;
68 mapping(address user => uint) public userLockedAmount;
69 mapping(address user => uint) public userVestedAmount;
```

Recommendation:

We advise all members to be combined into a single `struct` that is referenced by a single `mapping` declaration, greatly optimizing the gas cost of the overall `LockBox` system.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

A significant portion of the referenced data entries that utilize a `user` key have been merged to a single `UserInfo` structure, significantly optimizing the code's gas cost.

View GitHub Discussion

LBX-03C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	LockBox.sol:L149, L152, L160-L162, L203, L205, L206, L207, L211, L213, L215, L230, L233, L235, L236, L613, L614, L615, L628, L636, L637, L639, L641, L642, L645, L646, L666, L668, L669, L671, L672, L673, L676, L677

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/LockBox.sol

```
SOL
619function notifyRewardAmount(address token, uint amount) external allowTransfe
620   if (amount == 0) {revert ZeroAmount();}
621
622   if (!isReward[token]) {
623       rewards.push(token);
624       isReward[token] = true;
625   }
626
627   uint timestamp = _getTimestamp();
628   _rewardData[token].rewardPerTokenStored = rewardPerToken(token);
629
630   // Check actual amount transferred for compatibility with fee on transfe
631   uint balanceBefore = _balanceOf(token, address(this));
632   _safeTransferFrom(token, msg.sender, address(this), amount);
633   uint balanceAfter = _balanceOf(token, address(this));
634   amount = balanceAfter - balanceBefore;
635
636   if (timestamp >= _rewardData[token].periodFinish) {
637       _rewardData[token].rewardRate = amount / DURATION;
638   } else {
639       uint remaining = _rewardData[token].periodFinish -
640           timestamp;
641       uint _left = remaining * _rewardData[token].rewardRate;
642       _rewardData[token].rewardRate = (amount + _left) / DURATION;
643   }
644
645   _rewardData[token].lastUpdateTime = timestamp;
646   _rewardData[token].periodFinish = timestamp + DURATION;
647
648   emit NotifyReward(msg.sender, token, amount);
649}
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `mapping` lookup optimization has been partially applied across the codebase.

LBX-04C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	LockBox.sol:L789, L796, L803

Description:

The linked `require` checks have no error messages explicitly defined.

Example:

contracts/LockBox.sol

```
SOL
789require(success && (data.length == 0 || abi.decode(data, (bool))));
```

Recommendation:

We advise each to be set so to increase the legibility of the codebase and aid in validating the `require` checks' conditions.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

LBX-05C: Redundant Calculation of Value Outside Code Block

Type	Severity	Location
Gas Optimization	Informational	LockBox.sol:L448

Description:

The `unlockTime` variable is declared and calculated outside the `if` block it is utilized in.

Example:

contracts/LockBox.sol

```
SOL
448uint unlockTime = timestamp + MAXVEST;
449
450if(!isVested[account]){
451    isVested[account] = true;
452    vestedFor[account] = unlockTime;
453    emit VestCreated(account, amount, unlockTime);
454} else {
455    emit AddedToVest(account, amount);
456}
```

Recommendation:

We advise its declaration to be relocated to the block it is used in, optimizing the code's gas cost.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

The `unlockTime` calculation has been relocated within the `if` block as advised, optimizing the function's `else` clause gas cost.

View GitHub Discussion

LBX-06C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	LockBox.sol:L40, L41, L42

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

contracts/LockBox.sol

```
SOL
40 uint internal constant MINLOCK = 4 * 7 * 86400;
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

RewardVester Code Style Findings

RVR-01C: Inefficient Mapping Structures

Type	Severity	Location
Gas Optimization	Informational	RewardVester.sol:L43-L45

Description:

The referenced `mapping` declarations all utilize the same key and thus can be combined into a single entry that points to a `struct`.

Example:

contracts/RewardVester.sol

SOL Copy

```
43 mapping(address user => uint amount) public userVestedAmount;
44 mapping(address user => uint timestamp) public userVestEnd;
45 mapping(address user => bool state) public isVested;
```

Recommendation:

We advise this optimization to be performed, reducing the gas cost associated with mutating multiple entries in a single function call.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

All referenced `mapping` structures have been combined into one that points to a `UserInfo` structure, optimizing the code's gas cost across the board.

View GitHub Discussion

RVR-02C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	RewardVester.sol:L235, L242, L249, L256

Description:

The linked `require` checks have no error messages explicitly defined.

Example:

contracts/RewardVester.sol

SOL Copy

```
235require(success && (data.length == 0 || abi.decode(data, (bool))));
```

Recommendation:

We advise each to be set so to increase the legibility of the codebase and aid in validating the `require` checks' conditions.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

RVR-03C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	RewardVester.sol:L170

Description:

The linked declaration style of a struct is using index-based argument initialization.

Example:

contracts/RewardVester.sol

SOL Copy

```
170IVault.JoinPoolRequest memory request = IVault.JoinPoolRequest(lpTokens, amo
```

Recommendation:

We advise the key-value declaration format to be utilized instead, greatly increasing the legibility of the codebase.

Alleviation (0dd3c4b420f52d20607da5eef7311ae6067727cc):

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

View GitHub Discussion

PREV
LockBox.sol (LBX-C)

NEXT
Finding Types

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.

PREV
Severity Definition

NEXT
Source Code