OMNISCIA

August 19, 2023

SMART CONTRACT AUDIT REPORT

Arcade Protocol Migration Contracts


omniscia.io


info@omniscia.io


Online report: arcade-protocol-migration

Migration Contracts Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
1342cb264a	August 12th 2023	dbf4262b70
06a5ebc285	August 19th 2023	b2cf60fd83

Audit Overview

We were tasked with performing an audit of the Arcade Protocol codebase and in particular their V2-to-V3 loan rollover systems.

Over the course of the audit, we identified a significant vulnerability in the way flash-loans are consumed by the rollover systems which render the underlying collateral of the loans as well as the principal approvals of the borrowers at risk.

We advise the Arcade Protocol 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 Arcade Protocol 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 Arcade Protocol and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Contracts Assessed

Files in Scope	Repository	Commit(s)
RolloverErrors.sol (RES)	arcade-protocol	1342cb264a, 06a5ebc285
V2ToV3Rollover.sol (VTV)	arcade-protocol	1342cb264a, 06a5ebc285
V2ToV3RolloverBase.sol (VTR)	arcade-protocol	1342cb264a, 06a5ebc285
V2ToV3RolloverWithItems.sol (VTW)	arcade-protocol	1342cb264a, 06a5ebc285

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	2	2	0	0
Informational	7	7	0	0
Minor	1	1	0	0
Medium	3	3	0	0
Major	2	2	0	0

During the audit, we filtered and validated a total of **5 findings utilizing static analysis** tools as well as identified a total of **10 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.

Compilation

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

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 `0.8.18` based on the version specified within the `hardhat.config.ts` file.

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

The `pragma` statements have been locked to `0.8.18` (`=0.8.18`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` 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 **14 potential issues** within the codebase of which **5 were ruled out to be false positives** or negligible findings.

The remaining **9 issues** were validated and grouped and formalized into the **5 exhibits** that follow:

ID	Severity	Addressed	Title
VTV-01S	Medium	Yes	Improper Invocation of EIP-20 `transfer`
VTR-01S	Informational	Yes	Redundant Variable Assignment
VTR-02S	Minor	Yes	Inexistent Sanitization of Input Addresses
VTR-03S	Medium	Yes	Improper Invocation of EIP-20 `transfer`
VTW-01S	Medium	Yes	Improper Invocation of EIP-20 `transfer`

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Arcade Protocol's migration contracts.

As the project at hand implements special-purpose migration systems, intricate care was put into ensuring that the **flow of funds & assets within the system conforms to the specifications and restrictions** laid forth within the protocol's V2 and V3 specifications.

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 race-condition vulnerabilities of the same nature** within the system which could have had **severe ramifications** to its overall operation; we urge the Arcade Protocol team to promptly remediate them.

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 an exemplary extent, containing extensive in-line documentation within the implementations as well as interfaces of the contracts.

A total of **10 findings** were identified over the course of the manual review of which **7 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
VTV-01M	Unknown	Yes	Inexplicable Implementation of Receive Function
VTV-02M	Informational	Yes	Potentially Out-of-Sync Literal
VTV-03M	Major	Yes	Improper Consumption of Flash-Loan Calls
VTR-01M	Informational	Yes	Ill-Advised Toggle Mechanism
VTW-01M	Unknown	Yes	Inexplicable Implementation of Receive Function
VTW-02M	Informational	Yes	Potentially Out-of-Sync Literal

ID	Severity	Addressed	Title
VTW-03M	Major	Yes	Improper Consumption of Flash-Loan Calls

Code Style

During the manual portion of the audit, we identified **3 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
VTR-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
VTR-02C	Informational	Yes	Redundant Contract Member
VTR-03C	Informational	Yes	Unreachable Code

V2ToV3Rollover Static Analysis Findings

VTV-01S: Improper Invocation of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	V2ToV3Rollover.sol:L184

Description:

The linked statement does 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/v2-migration/V2ToV3Rollover.sol

SOL

```sol
184 asset.transfer(address(VAULT), flashAmountDue);
```

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.

Alleviation:

The `SafeERC20::safeTransfer` function is now properly utilized in place of the `ERC20::transfer` function invocation, ensuring that the asset is safely transferred to the `VAULT` and thus alleviating this exhibit.

V2ToV3RolloverBase Static Analysis Findings

VTR-01S: Redundant Variable Assignment

Type	Severity	Location
Gas Optimization	Informational	V2ToV3RolloverBase.sol:L49

Description:

The linked variable is assigned to redundantly to the default value of the relevant data type (i.e. `uint256` assigned to `0`, `address` assigned to `address(0)` etc.).

Example:

contracts/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```
49  bool public paused = false;
```

Recommendation:

We advise the assignment to be safely omitted optimizing the codebase.

Alleviation:

The redundant variable assignment has been safely omitted.

VTR-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	V2ToV3RolloverBase.sol:L51-L60

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/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```sol
51  constructor(IVault _vault, OperationContracts memory _opContracts) {
52      // Set Balancer vault address
53      VAULT = _vault;
54
55      // Set lending protocol contract references
56      feeControllerV3 = IFeeController(_opContracts.feeControllerV3);
57      originationControllerV3 = IOriginationController(_opContracts.originationControlle
58      loanCoreV3 = ILoanCore(_opContracts.loanCoreV3);
59      borrowerNoteV3 = IERC721(_opContracts.borrowerNoteV3);
60  }
```

Recommendation:

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

Alleviation:

All input addresses of the `V2ToV3RolloverBase::constructor` are adequately sanitized as non-zero, preventing the contract from being misconfigured during its deployment and thus addressing this exhibit.

VTR-03S: Improper Invocation of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	V2ToV3RolloverBase.sol:L195

Description:

The linked statement does 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/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```
195 token.transfer(to, balance);
```

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.

Alleviation:

The `SafeERC20::safeTransfer` function is now properly utilized in place of the `ERC20::transfer` function invocation, ensuring that the asset is safely transferred to the `VAULT` and thus alleviating this exhibit.

V2ToV3RolloverWithItems Static Analysis Findings

VTW-01S: Improper Invocation of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	V2ToV3RolloverWithItems.sol:L188

Description:

The linked statement does 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/v2-migration/V2ToV3RolloverWithItems.sol

SOL

```
188 asset.transfer(address(VAULT), flashAmountDue);
```

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.

Alleviation:

The `SafeERC20::safeTransfer` function is now properly utilized in place of the `ERC20::transfer` function invocation, ensuring that the asset is safely transferred to the `VAULT` and thus alleviating this exhibit.

V2ToV3Rollover Manual Review Findings

VTV-01M: Inexplicable Implementation of Receive Function

Type	Severity	Location
Code Style	Unknown	V2ToV3Rollover.sol:L229

Description:

The `V2ToV3Rollover::receive` function's presence is inexplicable as the `V2ToV3Rollover` contract does not deal with native funds.

Impact:

The severity of this exhibit will be adjusted accordingly when the Arcade Protocol team has taken the correct course of action for it.

Example:

contracts/v2-migration/V2ToV3Rollover.sol

SOL

```
229 receive() external payable {}
```

Recommendation:

We advise the Arcade Protocol team to either elaborate its presence or to omit it, depending on the desired functionality of the `V2ToV3Rollover` contract.

Alleviation:

The **`V2ToV3Rollover::receive`** function has been safely omitted as advised, reducing the bytecode size of the contract.

VTV-02M: Potentially Out-of-Sync Literal

Type	Severity	Location
Standard Conformity	Informational	V2ToV3Rollover.sol:L138

Description:

The `"BORROWER_ORIGINATION_FEE"` string literal is meant to act as the key to access the borrower's loan origination fee, however, it may change throughout the lifetime of the `FeeController` contract.

Example:

contracts/v2-migration/V2ToV3Rollover.sol

SOL

```
135 uint256(
136     feeControllerV3.getLendingFee(
137         // FL_01 - borrower origination fee
138         keccak256("BORROWER_ORIGINATION_FEE")
139     )
140 ),
```

Recommendation:

We advise the code to instead import the `FeeLookups` contract and utilize its `FL_01` data entry, ensuring that the code of `V2ToV3Rollover` is properly synchronized with the keys present in and utilized by the `FeeController`.

Alleviation:

The borrower origination fee is adequately fetched using the `FL_01` literal of `FeeLookups` in conjunction with the `FeeController::getLendingFee` function, standardizing the code's style and easing its maintainability.

VTV-03M: Improper Consumption of Flash-Loan Calls

Type	Severity	Location
Logical Fault	Major	V2ToV3Rollover.sol:L101-L111

Description:

The `V2ToV3Rollover::receiveFlashLoan` contract will trust all calls made via the `VAULT` contract which may not have necessarily been initialized by the contract itself via `V2ToV3Rollover::rolloverLoan`. As such, a major vulnerability manifests in the way loans are rolled-over that can be exploited via an on-chain race condition.

To elaborate, the `V2ToV3Rollover::receiveFlashLoan` function will consume the alleged borrower's approvals to repay their old loan and open a new one. In its validation mechanism, it solely ensures that the NFT asset, ID of NFT, and payable currency are the same with no regard to the actual loan terms.

Given that the `V2ToV3Rollover::_initializeNewLoan` function will ultimately validate a lender rather than borrower signature as it creates a loan originating from the roll-over contract, it is possible for a malicious party to detect a loan rollover and "race" it with their own transaction which would cause the V2 loan to be repaid and a significantly unfavourable V3 loan to be opened with a different lender than intended (i.e. w/ a significantly high interest rate).

Impact:

It is presently possible to compromise loans that are being rolled over via on-chain race conditions, causing the underlying collateral of the loan as well as the additional funds the borrower needs to supply for the original loan to be repaid to be compromised.

Example:

contracts/v2-migration/V2ToV3Rollover.sol

SOL

```sol
/**
 * @notice Callback function for flash loan.
 *
 * @dev The caller of this function must be the lending pool.
 *
 * @param assets                The ERC20 address that was borrowed in Flash Loan.
 * @param amounts               The amount that was borrowed in Flash Loan.
 * @param feeAmounts            The fees that are due to the lending pool.
 * @param params                The data to be executed after receiving Flash Loan.
 */
function receiveFlashLoan(
    IERC20[] calldata assets,
    uint256[] calldata amounts,
    uint256[] calldata feeAmounts,
    bytes calldata params
) external nonReentrant {
    if (msg.sender != address(VAULT)) revert R_UnknownCaller(msg.sender, address(VAULT
    OperationData memory opData = abi.decode(params, (OperationData));
    _executeOperation(assets, amounts, feeAmounts, opData);
}
```

Recommendation:

We advise the rollover workflow to be revised, caching the original caller in the `V2ToV3Rollover::rolloverLoan` function and validating them in the `V2ToV3Rollover::receiveFlashLoan` function. Such a security measure would guarantee that loans originate from the `V2ToV3Rollover` contract and thus that the validations performed by `V2ToV3Rollover::_validateRollover` are inherited by the `V2ToV3Rollover::_executeOperation` function.

Alleviation:

The `V2ToV3RolloverBase` dependency was updated to introduce a new modifier called `V2ToV3RolloverBase::whenBorrowerReset` which ensures that the newly declared storage variable `borrower` is zero at the beginning of the function's invocation and is cleared at the end.

This mechanism is utilized to store the `borrower` of the roll-over during a `V2ToV3Rollover::rolloverLoan` invocation that is consequently validated by the `V2ToV3Rollover::receiveFlashLoan` function. As such, the flash-loan callback hook can solely be invoked as part of a flash-loan originating from the contract itself alleviating this exhibit in full.

V2ToV3RolloverBase Manual Review Findings

VTR-01M: Ill-Advised Toggle Mechanism

Type	Severity	Location
Logical Fault	Informational	V2ToV3RolloverBase.sol:L181-L185

Description:

The `V2ToV3RolloverBase::togglePause` function will use a toggle state for the `paused` variable which is ill-advised due to how multiple pending transactions may be submitted in an emergency.

Example:

contracts/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```sol
181 function togglePause() external override onlyOwner {
182     paused = !paused;
183
184     emit PausedStateChanged(paused);
185 }
```

Recommendation:

We advise the code to accept a proper `bool` argument that permits the `paused` state to be set rather than toggled.

Alleviation:

The **`V2ToV3RolloverBase::togglePause`** function has been replaced by the `V2ToV3RolloverBase::pause` implementation that accepts a `bool` argument signifying the pause state that the contract should result in. As such, we consider this exhibit alleviated given that the toggle paradigm has been removed.

V2ToV3RolloverWithItems Manual Review Findings

VTW-01M: Inexplicable Implementation of Receive Function

Type	Severity	Location
Code Style	Unknown	V2ToV3RolloverWithItems.sol:L234

Description:

The `V2ToV3RolloverWithItems::receive` function's presence is inexplicable as the `V2ToV3RolloverWithItems` contract does not deal with native funds.

Impact:

The severity of this exhibit will be adjusted accordingly when the Arcade Protocol team has taken the correct course of action for it.

Example:

contracts/v2-migration/V2ToV3RolloverWithItems.sol

SOL

```
229 borrowerNoteV3.safeTransferFrom(address(this), borrower, newLoanId);
```

Recommendation:

We advise the Arcade Protocol team to either elaborate its presence or to omit it, depending on the desired functionality of the `V2ToV3RolloverWithItems` contract.

Alleviation:

The **`V2ToV3RolloverWithItems::receive`** function has been safely omitted as advised, reducing the bytecode size of the contract.

VTW-02M: Potentially Out-of-Sync Literal

Type	Severity	Location
Standard Conformity	Informational	V2ToV3RolloverWithItems.sol:L142

Description:

The `"BORROWER_ORIGINATION_FEE"` string literal is meant to act as the key to access the borrower's loan origination fee, however, it may change throughout the lifetime of the `FeeController` contract.

Example:

contracts/v2-migration/V2ToV3RolloverWithItems.sol

SOL

```
139 uint256(
140     feeControllerV3.getLendingFee(
141         // FL_01 - borrower origination fee
142         keccak256("BORROWER_ORIGINATION_FEE")
143     )
144 ),
```

Recommendation:

We advise the code to instead import the `FeeLookups` contract and utilize its `FL_01` data entry, ensuring that the code of `V2ToV3RolloverWithItems` is properly synchronized with the keys present in and utilized by the `FeeController`.

Alleviation:

The borrower origination fee is adequately fetched using the `FL_01` literal of `FeeLookups` in conjunction with the `FeeController::getLendingFee` function, standardizing the code's style and easing its maintainability.

VTW-03M: Improper Consumption of Flash-Loan Calls

Type	Severity	Location
Logical Fault	Major	V2ToV3RolloverWithItems.sol:L104-L114

Description:

The `V2ToV3RolloverWithItems::receiveFlashLoan` contract will trust all calls made via the `VAULT` contract which may not have necessarily been initialized by the contract itself via `V2ToV3RolloverWithItems::rolloverLoanWithItems`. As such, a major vulnerability manifests in the way loans are rolled-over that can be exploited via an on-chain race condition.

To elaborate, the `V2ToV3RolloverWithItems::receiveFlashLoan` function will consume the alleged borrower's approvals to repay their old loan and open a new one. In its validation mechanism, it solely ensures that the NFT asset, ID of NFT, and payable currency are the same with no regard to the actual loan terms.

Given that the `V2ToV3RolloverWithItems::_initializeNewLoanWithItems` function will ultimately validate a lender rather than borrower signature as it creates a loan originating from the roll-over contract, it is possible for a malicious party to detect a loan rollover and "race" it with their own transaction which would cause the V2 loan to be repaid and a significantly unfavourable V3 loan to be opened with a different lender than intended (i.e. w/ a significantly high interest rate).

Impact:

It is presently possible to compromise loans that are being rolled over via on-chain race conditions, causing the underlying collateral of the loan as well as the additional funds the borrower needs to supply for the original loan to be repaid to be compromised.

Example:

contracts/v2-migration/V2ToV3RolloverWithItems.sol

SOL

```sol
function receiveFlashLoan(
    IERC20[] calldata assets,
    uint256[] calldata amounts,
    uint256[] calldata feeAmounts,
    bytes calldata params
) external nonReentrant {
    if (msg.sender != address(VAULT)) revert R_UnknownCaller(msg.sender, address(VAULT
    OperationDataWithItems memory opData = abi.decode(params, (OperationDataWithItems)
    _executeOperation(assets, amounts, feeAmounts, opData);
}
```

Recommendation:

We advise the rollover workflow to be revised, caching the original caller in the **`V2ToV3RolloverWithItems::rolloverLoanWithItems`** function and validating them in the **`V2ToV3RolloverWithItems::receiveFlashLoan`** function. Such a security measure would guarantee that loans originate from the `V2ToV3RolloverWithItems` contract and thus that the validations performed by `V2ToV3RolloverWithItems::_validateRollover` are inherited by the **`V2ToV3RolloverWithItems::_executeOperation`** function properly.

Alleviation:

The `V2ToV3RolloverBase` dependency was updated to introduce a new modifier called `V2ToV3RolloverBase::whenBorrowerReset` which ensures that the newly declared storage variable `borrower` is zero at the beginning of the function's invocation and is cleared at the end.

This mechanism is utilized to store the `borrower` of the roll-over during a **`V2ToV3RolloverWithItems::rolloverLoanWithItems`** invocation that is consequently validated by the **`V2ToV3RolloverWithItems::receiveFlashLoan`** function. As such, the flash-loan callback hook can solely be invoked as part of a flash-loan originating from the contract itself alleviating this exhibit in full.

V2ToV3RolloverBase Code Style Findings

VTR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	V2ToV3RolloverBase.sol:L92, L95

Description:

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

Example:

contracts/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```sol
90  if (flashAmountDue > willReceive) {
91      // Not enough - have borrower pay the difference
92      needFromBorrower = flashAmountDue - willReceive;
93  } else if (willReceive > flashAmountDue) {
94      // Too much - will send extra to borrower
95      leftoverPrincipal = willReceive - flashAmountDue;
96  }
```

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:

An `unchecked` code block has been introduced in both referenced subtractions, optimizing their gas cost whilst retaining their security.

VTR-02C: Redundant Contract Member

Type	Severity	Location
Gas Optimization	Informational	V2ToV3RolloverBase.sol:L38

Description:

The `lenderNoteV2` member of the `V2ToV3RolloverBase` contract remains unused throughout the codebase.

Example:

contracts/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```
38  IERC721 public constant lenderNoteV2 = IERC721(0x349A026A43FFA8e2Ab4c4e59FCAa93F87Bd8D
```

Recommendation:

We advise it to be safely omitted, optimizing the gas cost of the contract's deployment.

Alleviation:

The redundant `lenderNoteV2` member has been safely omitted from the `V2ToV3RolloverBase` contract.

VTR-03C: Unreachable Code

Type	Severity	Location
Gas Optimization	Informational	V2ToV3RolloverBase.sol:L99-L101

Description:

The referenced code is logically unreachable as the `V2ToV3RolloverBase::_ensureFunds` function will either assign to `needFromBorrower` or `leftoverPrincipal` based on its `if-else-if` clause.

Example:

contracts/v2-migration/base/V2ToV3RolloverBase.sol

SOL

```sol
90  if (flashAmountDue > willReceive) {
91      // Not enough - have borrower pay the difference
92      needFromBorrower = flashAmountDue - willReceive;
93  } else if (willReceive > flashAmountDue) {
94      // Too much - will send extra to borrower
95      leftoverPrincipal = willReceive - flashAmountDue;
96  }
97
98  // Either leftoverPrincipal or needFromBorrower should be 0
99  if (leftoverPrincipal != 0 && needFromBorrower != 0) {
100     revert R_FundsConflict(leftoverPrincipal, needFromBorrower);
101 }
```

Recommendation:

We advise it to be safely omitted, optimizing the gas cost of the code.

Alleviation:

The unreachable conditional statement has been safely omitted from the codebase.

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.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

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 Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. 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 contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

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.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

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.