OMNISCIA

August 19, 2023

SMART CONTRACT AUDIT REPORT

Arcade Protocol Core Implementation


omniscia.io


info@omniscia.io


Online report: arcade-xyz-protocol-implementation

Protocol Implementation Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
f1eb8ae7b7	July 21st 2023	d8001c12b8
7a4e1dc948	August 16th 2023	1b32f8b17a
45ccaa43fa	August 19th 2023	1e208dfd32

Audit Overview

We were tasked with performing an audit of the Arcade XYZ codebase and in particular their version 3 implementation of a decentralized NFT borrowing and lending market.

Over the course of the audit, we identified multiple misconceptions around the `AssetVault` implementation that could significantly compromise the integrity of "packaged vaults" and loans created around them.

We advise the Arcade XYZ 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 XYZ 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 XYZ and have identified that certain exhibits require a re-evaluation.

We advise the Arcade XYZ team to revisit the following exhibits: CWO-01M, CWA-01M, CWT-01C

Additionally, all acknowledged exhibits should be validated as they may have been missed by the Arcade XYZ team during the first iteration of the report: PVR-01S, PVR-02C, PVR-03C, PVR-01M, CBT-01M, OER-02S, OER-01C, AVT-05C, AVT-01S, AVT-04C, AVT-02C, PNE-01S, LCE-01C, LCE-03C, OCR-01S, OCR-01C, OCR-03M, RCR-01C

Post-Audit Conclusion (45ccaa43fa)

The Arcade XYZ team re-evaluated exhibits CWO-01M, CWA-01M, and CWT-01C providing us with a dedicated response to each as well as a follow-up remediation for CWT-01C.

All remaining exhibits referenced abvoe have additionally been validated as acknowledged by the Arcade XYZ team.

As a result of these updates, we consider all outputs of the report properly consumed by the Arcade XYZ team with no further action required on their end.

Contracts Assessed

Files in Scope	Repository	Commit(s)
AssetVault.sol (AVT)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
ArtBlocksVerifier.sol (ABV)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
ArcadeItemsVerifier.sol (AIV)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
BaseURIDescriptor.sol (BUR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
CallBlacklist.sol (CBT)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
CallWhitelist.sol (CWT)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
CallWhitelistApprovals.sol (CWA)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
CallWhitelistDelegation.sol (CWD)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa

Files in Scope	Repository	Commit(s)
CallWhitelistAllExtensions.sol (CWE)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
CollectionWideOfferVerifier.sol (CWO)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
ERC721Permit.sol (ERC)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
FeeLookups.sol (FLS)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
FeeController.sol (FCR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
InterestCalculator.sol (ICR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
Lending.sol (LGN)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
LoanCore.sol (LCE)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa

Files in Scope	Repository	Commit(s)
LoanLibrary.sol (LLY)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
OwnableERC721.sol (OER)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
OriginationController.sol (OCR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
PunksVerifier.sol (PVR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
PromissoryNote.sol (PNE)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
RepaymentController.sol (RCR)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
UnvaultedItemsVerifier.sol (UIV)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa
Vault.sol (VTL)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa

Files in Scope	Repository	Commit(s)
VaultFactory.sol (VFY)	arcade-protocol	f1eb8ae7b7, 7a4e1dc948, 0a7551b095, 45ccaa43fa

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	4	3	0	1
Informational	32	23	0	9
Minor	13	9	0	4
Medium	4	3	0	1
Major	3	2	0	1

During the audit, we filtered and validated a total of **8 findings utilizing static analysis** tools as well as identified a total of **48 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 located in 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 **46 potential issues** within the codebase of which **37 were ruled out to be false positives** or negligible findings.

The remaining **9 issues** were validated and grouped and formalized into the **8 exhibits** that follow:

ID	Severity	Addressed	Title
AVT-01S	Minor	Acknowledged	Deprecated Approval Methodology
CWD-01S	Minor	Yes	Inexistent Sanitization of Input Address
OCR-01S	Informational	Acknowledged	Illegible Numeric Value Representation
OER-01S	Informational	Yes	Inexistent Event Emission
OER-02S	Minor	Acknowledged	Inexistent Sanitization of Input Address
PNE-01S	Minor	Acknowledged	Inexistent Sanitization of Input Addresses
PVR-01S	Minor	Acknowledged	Inexistent Sanitization of Input Address
VFY-01S	Minor	Yes	Deprecated Native Asset Transfer

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in the P2P NFT-backed borrowing and lending protocol of Arcade XYZ.

As the project at hand implements an NFT collateralized lending and borrowing protocol, 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 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 avenues via which collateral within an `AssetVault` could be compromised** which could have had **severe ramifications** to its overall operation; we urge the Arcade XYZ team to promptly evaluate & remediate findings that relate to the `AssetVault` contract.

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 as well as a comprehensive `README.md` file both of which were utilized when validating the project's specification.

A total of **48 findings** were identified over the course of the manual review of which **21 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
AIV-01M	Minor	Yes	Incorrect Assumption of Function
ABV-01M	Minor	Yes	Improper Enforcement of Check
ABV-02M	Minor	Yes	Incorrect Assumption of Function
AVT-01M	Informational	Yes	Improper Dependency Utilization
CBT-01M	Unknown	Acknowledged	Inexistent Flexibility of Blacklist
CBT-02M	Major	Yes	Potentially Weak Restriction List
CWA-01M	Major	Acknowledged	Insecure Whitelist Potential
CWO-01M	Medium	Yes	Unsafe Casting Operation
ERC-01M	Unknown	Yes	Potential Deviation of Standard
OCR-01M	Unknown	Nullified	On-Chain Race Condition of Loan Consumptions
OCR-02M	Minor	Yes	Restrictive Minimum Loan Principal
OCR-03M	Medium	Acknowledged	Improper Validation of Loan Amounts
OCR-04M	Medium	Nullified	Insufficient Validation of Loan Rollover
OCR-05M	Major	Yes	Invalidation of Predicates
OER-01M	Minor	Yes	Inexistent Validation of Ownership Existence

ID	Severity	Addressed	Title
PNE-01M	Unknown	Yes	Misleading Documentation of Contract
PNE-02M	Informational	Yes	Misleading Token ID Counter
PVR-01M	Informational	Acknowledged	Potentially Unwarranted Predicate Case
PVR-02M	Minor	Yes	Incorrect Assumption of Function
RCR-01M	Minor	Yes	Improper Imposition of Claim Fee
RCR-02M	Medium	Nullified	Inexistent Validation of Loan Expiry

Code Style

During the manual portion of the audit, we identified **27 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
AIV-01C	Informational	Yes	Loop Iterator Optimization
AIV-02C	Informational	Yes	Optimization of `if` Structure
ABV-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
ABV-02C	Informational	Yes	Loop Iterator Optimizations
AVT-01C	Informational	Yes	Loop Iterator Optimization
AVT-02C	Informational	Acknowledged	Misconception of Re-Entrancy Attack Vectors
AVT-03C	Informational	Yes	Optimization of `if-else` Structure
AVT-04C	Informational	Acknowledged	Potential Enhancement of Functionality
AVT-05C	Informational	Acknowledged	Potential Optimization of Project Structure
CWT-01C	Informational	Yes	Inefficient `mapping` Lookups
FCR-01C	Informational	Yes	Redundant Parenthesis Statements
FCR-02C	Informational	Yes	Suboptimal Struct Declaration Styles
LCE-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
LCE-02C	Informational	Yes	Inefficient Loop Limit Evaluation
LCE-03C	Informational	Yes	Inefficient `mapping` Lookups
LCE-04C	Informational	Yes	Loop Iterator Optimizations

ID	Severity	Addressed	Title
LCE-05C	Informational	Yes	Optimization of Assignments
LCE-06C	Informational	Yes	Suboptimal Struct Declaration Style
LLY-01C	Informational	Yes	Inefficient Tight Packing of Structs
OCR-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
OCR-02C	Informational	Yes	Loop Iterator Optimizations
OER-01C	Informational	Acknowledged	Potential Optimization of Project Structure
PNE-01C	Informational	Nullified	Inefficient Renunciation of Role
PVR-01C	Informational	Yes	Loop Iterator Optimization
PVR-02C	Informational	Acknowledged	Redundant Usage of `SafeCast`
PVR-03C	Informational	Acknowledged	Undocumented Predicate Feature
RCR-01C	Informational	Acknowledged	Redundant Conditional Evaluation

AssetVault Static Analysis Findings

AVT-01S: Deprecated Approval Methodology

Type	Severity	Location
Standard Conformity	Minor	AssetVault.sol:L299

Description:

The linked statement invokes the `safeApprove` function which has been **officially deprecated** by the OpenZeppelin standard.

Impact:

The `safeApprove` function indirectly validates that the approval that already exists for the target party has been previously set to zero if being set to a non-zero value. This can cause significant issues in the case of upgrade-able contracts or contracts whose allowance may not be utilized in full as subsequent `safeApprove` invocations will fail rendering it inoperable.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
299 IERC20(token).safeApprove(spender, amount);
```

Recommendation:

We advise the code to utilize a `safeIncreaseAllowance` and / or a `safeDecreaseAllowance` depending on the execution context and desired result.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

CallWhitelistDelegation Static Analysis Findings

CWD-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Minor	CallWhitelistDelegation.sol:L48-L50

Description:

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

Impact:

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

Example:

contracts/vault/CallWhitelistDelegation.sol

SOL

```sol
48  constructor(address _registry) {
49      registry = IDelegationRegistry(_registry);
50  }
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `_registry` input address of the **`CallWhitelistDelegation::constructor`** is now properly sanitized, preventing the contract from being misconfigured during its deployment.

OriginationController Static Analysis Findings

OCR-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	OriginationController.sol:L686

Description:

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

Example:

contracts/OriginationController.sol

SOL

```sol
686 if (terms.durationSecs < 3600 || terms.durationSecs > 94_608_000) revert OC_LoanDurati
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team stated that they do not intend to utilize underscore separators for four-digit numbers and as such, they wish to acknowledge this exhibit.

OwnableERC721 Static Analysis Findings

OER-01S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	OwnableERC721.sol:L44-L46

Description:

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

Example:

contracts/vault/OwnableERC721.sol

SOL

```sol
44  function _setNFT(address _ownershipToken) internal {
45      ownershipToken = _ownershipToken;
46  }
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `SetOwnershipToken` event has been introduced to the `OwnableERC721` contract and is correspondingly emitted in the **`OwnableERC721::_setNFT`** function, addressing this exhibit in full.

OER-02S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Minor	OwnableERC721.sol:L44-L46

Description:

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

Impact:

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

Example:

contracts/vault/OwnableERC721.sol

SOL

```sol
44  function _setNFT(address _ownershipToken) internal {
45      ownershipToken = _ownershipToken;
46  }
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

PromissoryNote Static Analysis Findings

PNE-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	PromissoryNote.sol:L82-L104, L113-L120

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/PromissoryNote.sol

SOL

```sol
constructor(
    string memory name,
    string memory symbol,
    address _descriptor
) ERC721(name, symbol) ERC721Permit(name) {
    if (_descriptor == address(0)) revert PN_ZeroAddress("descriptor");

    descriptor = INFTDescriptor(_descriptor);

    _setupRole(ADMIN_ROLE, msg.sender);
    _setupRole(RESOURCE_MANAGER_ROLE, msg.sender);

    // Allow admin to set mint/burn role, which they will do
    // during initialize. After initialize, admin role is
    // permanently revoked, so mint/burn role becomes immutable
    // and initialize cannot be called again.
    // Do not set role admin for admin role.
    _setRoleAdmin(MINT_BURN_ROLE, ADMIN_ROLE);
    _setRoleAdmin(RESOURCE_MANAGER_ROLE, RESOURCE_MANAGER_ROLE);

    // We don't want token IDs of 0
    _tokenIdTracker.increment();
}
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The exhibit has been amended to only contain one referenced instance instead of two as the first one was incorrect.

The second remains in the codebase unaddressed and as such, we consider this exhibit acknowledged.

PunksVerifier Static Analysis Findings

PVR-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Minor	PunksVerifier.sol:L38-L40

Description:

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

Impact:

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

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```sol
38  constructor(address _punks) {
39      punks = IPunks(_punks);
40  }
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

VaultFactory Static Analysis Findings

VFY-01S: Deprecated Native Asset Transfer

Type	Severity	Location
Language Specific	Minor	VaultFactory.sol:L171

Description:

The linked statement performs a low-level native asset transfer via the `transfer` function exposed by the `address payable` data type.

Impact:

As new EIPs such as **EIP-2930** are introduced to the blockchain, gas costs can change and the `transfer` instruction of Solidity specifies a fixed gas stipend that is prone to failure should such changes be integrated to the blockchain the contract is deployed in. A prime example of this behaviour are legacy versions of Gnosis which were susceptible to this issue and would cause native transfers to fail if sent to a new address.

Example:

contracts/vault/VaultFactory.sol

SOL

```sol
171 if (msg.value > mintFee) payable(msg.sender).transfer(msg.value - mintFee);
```

Recommendation:

We advise alternative ways of transferring assets to be utilized instead, such as OpenZeppelin's `Address.sol` library and in particular the `sendValue` method exposed by it. If re-entrancies are desired to be prevented based on gas costs, we instead advise a mechanism to be put in place that either credits an account with a native balance they can withdraw at a secondary transaction or that performs the native asset transfers at the end of the top-level transaction's execution.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The native fund transfer performed in the referenced statement has been properly replaced by OpenZeppelin's `Address::sendValue` function, ensuring that the transfer will fail irrespective of what blockchain the contract is deployed in.

ArcadeItemsVerifier Manual Review Findings

AIV-01M: Incorrect Assumption of Function

Type	Severity	Location
Logical Fault	Minor	ArcadeItemsVerifier.sol:L81-L82

Description:

The `ArcadeItemsVerifier::verifyPredicates` function assumes that an empty predicates array has been addressed in the `OriginationController::initializeLoanWithItems` and `OriginationController::rolloverLoanWithItems` functions, however, this is **incorrect** as the functions ensure that the **whole predicate array is not empty**, not that the `data` **payload of a predicate call contains non-zero entries**.

Impact:

The `ArcadeItemsVerifier::verifyPredicates` will misbehave if supplied an empty `items` data entry when called via the `OriginationController` despite its documentation specifying that no sanitization is needed as it is incorrect.

Example:

contracts/verifiers/ArcadeItemsVerifier.sol

SOL

```sol
/**
 * @notice Verify that the items specified by the packed SignatureItem array are held
 * @dev    Reverts on a malformed SignatureItem, returns false on missing contents.
 *
 *         Verification for empty predicates array has been addressed in initializeLoa
 *         rolloverLoanWithItems.
 *
 * @param collateralAddress             The address of the loan's collateral.
 * @param collateralId                  The tokenId of the loan's collateral.
 * @param predicates                    The calldata needed for the verifier.
 *
 * @return verified                     Whether the bundle contains the specified item
 */
// solhint-disable-next-line code-complexity
function verifyPredicates(
    address, address,
    address collateralAddress,
    uint256 collateralId,
    bytes calldata predicates
) external view override returns (bool) {
    address vault = IVaultFactory(collateralAddress).instanceAt(collateralId);

    // Unpack items
    SignatureItem[] memory items = abi.decode(predicates, (SignatureItem[]));

    for (uint256 i = 0; i < items.length; i++) {
```

Recommendation:

We advise the code to properly ensure that the `items` decoded contain a non-zero length as otherwise the predicate would succeed without validating anything, signifying a potential scam attempt.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

A proper `if-revert` pattern was introduced that ensures the decoded `SignatureItem` array contains non-zero entries, alleviating this exhibit in full.

ArtBlocksVerifier Manual Review Findings

ABV-01M: Improper Enforcement of Check

Type	Severity	Location
Input Sanitization	Minor	ArtBlocksVerifier.sol:L95-L96

Description:

The referenced statement is meant to validate that the predicate's amount value is non-zero, however, the predicate's amount is solely utilized when `anyIdAllowed` is set to `true`.

Impact:

The code will not assume an `amount` of `1` if `tokenId` has been defined in the current iteration of the codebase for validation purposes, rendering it contradictory to its specification.

Example:

contracts/verifiers/ArtBlocksVerifier.sol

SOL

```sol
// No amount provided
if (item.amount == 0) revert IV_NoAmount(item.asset, item.amount);

if (item.anyIdAllowed) {
    // Iterate through tokens
    uint256 tokenCount = IArtBlocks(item.asset).balanceOf(vault);
    uint256 found;

    for (uint256 j = 0; j < tokenCount; j++) {
        uint256 fullTokenId = IArtBlocks(item.asset).tokenOfOwnerByIndex(vault, j);
        uint256 ownedProjectId = fullTokenId / PROJECT_ID_BASE;

        // If project is owned, increment num found
        // If we've found enough, break
        if (ownedProjectId == item.projectId) {
            found++;

            if (found >= item.amount) break;
        }
    }

    // We looped and didn't find enough, so fail
    if (found < item.amount) return false;
} else {
    // Look for a specific token ID
    uint256 fullTokenId = _getFullTokenId(item.projectId, item.tokenId);

    // Check if the token is owned by the vault
    if (IERC721(item.asset).ownerOf(fullTokenId) != vault) {
        return false;
    }
}
```

Recommendation:

Given that the `struct` documentation explicitly states `amount` is assumed to be `1` if a `tokenId` has been specified, we advise the code to relocate the referenced check within the `if (item.anyIdAllowed)` code branch, ensuring that it is validated only when it is appropriate to do so.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The referenced `if-revert` check has been relocated as advised, ensuring that the code behaves according to its specification.

ABV-02M: Incorrect Assumption of Function

Type	Severity	Location
Logical Fault	Minor	ArtBlocksVerifier.sol:L66-L67

Description:

The `ArtBlocksVerifier::verifyPredicates` function assumes that an empty predicates array has been addressed in the `OriginationController::initializeLoanWithItems` and `OriginationController::rolloverLoanWithItems` functions, however, this is **incorrect** as the functions ensure that the **whole predicate array is not empty**, not that the `data` **payload of a predicate call contains non-zero entries**.

Impact:

The `ArtBlocksVerifier::verifyPredicates` will misbehave if supplied an empty `items` data entry when called via the `OriginationController` despite its documentation specifying that no sanitization is needed as it is incorrect.

Example:

contracts/verifiers/ArtBlocksVerifier.sol

SOL

```sol
/**
 * @notice Verify that the items specified by the packed SignatureItem array are held
 * @dev    Reverts on a malformed SignatureItem, returns false on missing contents.
 *
 *         Verification for empty predicates array has been addressed in initializeLoa
 *         rolloverLoanWithItems.
 *
 * @param collateralAddress             The address of the loan's collateral.
 * @param collateralId                  The tokenId of the loan's collateral.
 * @param predicates                    The SignatureItem[] array of items, packed in
 *
 * @return verified                     Whether the bundle contains the specified item
 */
// solhint-disable-next-line code-complexity
function verifyPredicates(
    address, address,
    address collateralAddress,
    uint256 collateralId,
    bytes calldata predicates
) external view override returns (bool) {
    address vault = IVaultFactory(collateralAddress).instanceAt(collateralId);
    // Unpack items
    SignatureItem[] memory items = abi.decode(predicates, (SignatureItem[]));

    for (uint256 i = 0; i < items.length; ++i) {
```

Recommendation:

We advise the code to properly ensure that the `items` decoded contain a non-zero length as otherwise the predicate would succeed without validating anything, signifying a potential scam attempt.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

A proper `if-revert` pattern was introduced that ensures the decoded `SignatureItem` array contains non-zero entries, alleviating this exhibit in full.

AssetVault Manual Review Findings

AVT-01M: Improper Dependency Utilization

Type	Severity	Location
Language Specific	Informational	AssetVault.sol:L10

Description:

The `AssetVault` is meant to be deployed via a proxy system, however, it inherits a non-upgradeable variant of the `ReentrancyGuard`.

Impact:

While the `ReentrancyGuard::constructor` does contain logic within it, it simply assigns a value to `_status` as a matter of optimization rather than functionality.

In any case, the contract should be standardized and utilize upgradeable variants as it is meant to be deployed as a cloned instance via the `VaultFactory` contract.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";
import "@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol";
import "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/utils/Address.sol";
```

Recommendation:

As the `ReentrancyGuard::constructor` does assign an initial value to `_status`, we advise its upgradeable variant to be utilized from the relevant OpenZeppelin dependency to ensure code consistency. We advise the same dependency to be utilized for all other assets that the `AssetVault` contract imports as well.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

All dependencies of the `AssetVault` contract were updated to be of their upgradeable OpenZeppelin variant and the `ReentrancyGuardUpgradeable` contract is properly initialized via its `ReentrancyGuardUpgradeable::__ReentrancyGuard_init` hook during the contract's **`AssetVault::initialize`** function.

As such, we consider this exhibit fully alleviated.

CallBlacklist Manual Review Findings

CBT-01M: Inexistent Flexibility of Blacklist

Type	Severity	Location
Logical Fault	Unknown	CallBlacklist.sol:L56-L69

Description:

The referenced restriction list imposed by `CallBlacklist::isBlacklisted` is immutable and is applied via a long OR (`||`) conditional chain.

Impact:

While this exhibit relates to potential enhancement of functionality, we consider it imperative given that we have showcased a potential function signature that could lead to the compromise of the `AssetVault` instance's funds.

Example:

contracts/vault/CallBlacklist.sol

SOL

```sol
55  function isBlacklisted(bytes4 selector) public pure returns (bool) {
56      return
57          selector == ERC20_TRANSFER ||
58          selector == ERC20_ERC721_APPROVE ||
59          selector == ERC20_ERC721_TRANSFER_FROM ||
60          selector == ERC20_INCREASE_ALLOWANCE ||
61          selector == ERC721_SAFE_TRANSFER_FROM ||
62          selector == ERC721_SAFE_TRANSFER_FROM_DATA ||
63          selector == ERC721_ERC1155_SET_APPROVAL ||
64          selector == ERC1155_SAFE_TRANSFER_FROM ||
65          selector == ERC1155_SAFE_BATCH_TRANSFER_FROM ||
66          selector == PUNKS_TRANSFER ||
67          selector == PUNKS_OFFER ||
68          selector == PUNKS_OFFER_TO_ADDRESS ||
69          selector == PUNKS_BUY;
70  }
```

Recommendation:

We advise the code to utilize a mutable `mapping` entry instead, permitting the owners of the blacklist to update it to accommodate for future collateral types that an `AssetVault` is expected to support.

As a security measure, the `mapping` entry could be solely written to for blacklisting and to not possess any un-blacklisting functionality as an `AssetVault` can always be marked withdrawable.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team stated that they do not wish to add an administrative capacity to the `CallBlacklist` contract and that it is expected to be a static and `immutable` list of calls that are disallowed. As such, we consider this exhibit acknowledged.

CBT-02M: Potentially Weak Restriction List

Type	Severity	Location
Language Specific	Major	CallBlacklist.sol:L28-L31, L33-L35, L37-L38, L40-L43

Description:

The function signature restriction list defined in `CallBlacklist` appears to lack functions that could still impact assets held by the contract, namely burn operations.

Impact:

As the `CallBlacklist` is a crucial component that is meant to prohibit the compromise of an `AssetVault` instance's collateral, we consider this exhibit to be of "major" severity. It is currently possible to "sabotage" a potential loan by burning the assets held within it after it has been created, harming a lender's position.

Example:

contracts/vault/CallBlacklist.sol

SOL

```
25  /**
26   * @dev Global blacklist for transfer functions.
27   */
28  bytes4 private constant ERC20_TRANSFER = IERC20.transfer.selector;
29  bytes4 private constant ERC20_ERC721_APPROVE = IERC20.approve.selector;
30  bytes4 private constant ERC20_ERC721_TRANSFER_FROM = IERC20.transferFrom.selector;
31  bytes4 private constant ERC20_INCREASE_ALLOWANCE = bytes4(keccak256("increaseAllowance
32
33  bytes4 private constant ERC721_SAFE_TRANSFER_FROM = bytes4(keccak256("safeTransferFrom
34  bytes4 private constant ERC721_SAFE_TRANSFER_FROM_DATA = bytes4(keccak256("safeTransfe
35  bytes4 private constant ERC721_ERC1155_SET_APPROVAL = IERC721.setApprovalForAll.select
36
37  bytes4 private constant ERC1155_SAFE_TRANSFER_FROM = IERC1155.safeTransferFrom.selecto
38  bytes4 private constant ERC1155_SAFE_BATCH_TRANSFER_FROM = IERC1155.safeBatchTransferF
39
40  bytes4 private constant PUNKS_TRANSFER = bytes4(keccak256("transferPunk(address,uint25
41  bytes4 private constant PUNKS_OFFER = bytes4(keccak256("offerPunkForSale(uint256,uint2
42  bytes4 private constant PUNKS_OFFER_TO_ADDRESS = bytes4(keccak256("offerPunkForSaleToA
43  bytes4 private constant PUNKS_BUY = bytes4(keccak256("buyPunk(uint256)"));
```

Recommendation:

Given that burn functionality is permitted in multiple token implementations, we advise the `IERC20::burn` (and its **EIP-721** counterpart) to be blocked as well as the `IERC20::burnFrom` function.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

Multiple function signatures relating to burn functionality have been introduced and are now disallowed by the **`CallBlacklist::isBlacklisted`** function, alleviating this exhibit.

As an addendum, the Arcade XYZ team stated that the `CallBlacklist` is not meant to be a comprehensive blacklist and that extra care should be taken by users when whitelisting functions as they may still affect vault assets.

CallWhitelistApprovals Manual Review Findings

CWA-01M: Insecure Whitelist Potential

Type	Severity	Location
Logical Fault	Major	CallWhitelistApprovals.sol:L26, L51-L54

Description:

The `CallWhitelistApprovals` contract implementation is presently insecure when utilized with an `AssetVault` in the `LoanCore` context as it permits spenders to be set before an `AssetVault` is escrowed, essentially compromising the funds held within the `AssetVault` after the loan has been created.

Impact:

The current implementation of `AssetVault` in conjunction with `CallWhitelistApprovals` is incompatible with the Arcade XYZ lending system.

Example:

contracts/vault/CallWhitelistApprovals.sol

SOL

```sol
/**
 * @notice Sets approval status of a given token for a spender. Note that this is
 *         NOT a token approval - it is permission to create a token approval from
 *         the asset vault.
 *
 * @param token                The token approval to set.
 * @param spender              The token spender.
 * @param _isApproved          Whether the spender should be approved.
 */
function setApproval(address token, address spender, bool _isApproved) external onlyOw
    approvals[token][spender] = _isApproved;
    emit ApprovalSet(msg.sender, token, spender, _isApproved);
}
```

Recommendation:

We advise a new `mapping` to be introduced that marks a particular asset / collateral as "dirty". In essence, whenever **`CallWhitelistApprovals::setApproval`** is called it will mark the specified `token` as "dirty" permitting on-chain code to validate whether the `token` may be compromised within an `AssetVault`.

A new `ISignatureVerifier` can thus be introduced to the codebase that evaluates predicates guaranteeing that the assets are not "dirty" and thus safe to assume that they will remain within the escrowed `AssetVault`.

Alleviation (7a4e1dc948):

The Arcade XYZ team specified that they wish to retain the current functionality of the whitelist in place and that it is up to integrators as well as counterparties to validate the suitability of an `AssetVault` as collateral, potentially by utilizing specialized verifier contracts.

We would like to note that the inclusion of a programmatic check is meant to prevent on-chain race conditions whereby a user manually assesses an `AssetVault` as viable and its status its changed via an on-chain race.

In any case, we consider dedicated verifiers to be a suitable resolution to the vulnerability described and we would like to advise the Arcade XYZ to consider including them as part of the "standard" Arcade XYZ library.

As the Arcade XYZ team stated that they do not wish to pursue any additional immediate action, we consider this exhibit acknowledged albeit to be re-evaluated by the Arcade XYZ team.

Alleviation (45ccaa43fa):

The Arcade XYZ team re-evaluated this exhibit and has opted to retain their acknowledgement as programmatic checks would hinder the flexibility of the protocol and predicates have been introduced to it for this very purpose (i.e. additional validations of a particular loan agreement).

As such, we consider this exhibit acknowledged with no further action pending from the Arcade XYZ team.

CollectionWideOfferVerifier Manual Review Findings

CWO-01M: Unsafe Casting Operation

Type	Severity	Location
Mathematical Operations	Medium	CollectionWideOfferVerifier.sol:L57

Description:

The `CollectionWideOfferVerifier::verifyPredicates` function will improperly cast the input `collateralId` to a `uint160` value without ensuring that the value lies within its bounds, leading to a casting overflow.

Impact:

It is currently possible for a `CollectionWideOfferVerifier::verifyPredicates` operation to succeed incorrectly as it will downcast a `uint256` value to `uint160`, causing multiple different `collateralId` values to result in the same `vaultAddress` and thus bypass the contract's verification.

Example:

contracts/verifiers/CollectionWideOfferVerifier.sol

SOL

```sol
function verifyPredicates(
    address, address,
    address collateralAddress,
    uint256 collateralId,
    bytes calldata data
) external view override returns (bool) {
    // Unpack items
    (address token) = abi.decode(data, (address));

    // Unvaulted case - collateral will be escrowed directly
    if (collateralAddress == token) return true;

    // Do vault check
    address vaultAddress = address(uint160(collateralId));

    if (!IVaultFactory(collateralAddress).isInstance(vaultAddress)) return false;

    return IERC721(token).balanceOf(vaultAddress) > 0;
}
```

Recommendation:

Given that casting overflows **are not automatically detected using checked arithmetic**, we advise the code to utilize a library such as OpenZeppelin's `SafeCast` to perform the operation.

Alternatively, we advise the usage of the **`VaultFactory::instanceAt`** function that is utilized by other similar verifiers as it contains a built-in token existence check.

Alleviation (7a4e1dc948):

The Arcade XYZ team proceeded with utilizing the **`VaultFactory::instanceAt`** function that is safe to utilize with any `collateralId`. However, an `if` check was introduced after all **`VaultFactory::instanceAt`** invocations throughout the codebase that ensures the `vaultAddress` when cast is equal to the `collateralId`.

We would like to note that utilization for the **`VaultFactory::instanceAt`** function directly is safe as it will perform an `ERC721::_exists` call, ensuring that the `tokenId` supplied is not out of bounds. As such, we advise all these `if` statements to be safely omitted.

Given that the original vulnerability described has been addressed, we do consider this exhibit fully alleviated despite the potential for an optimization.

Alleviation (45ccaa43fa):

The Arcade XYZ team assessed that while the additional check introduced is redundant, they wish to retain it so as to decouple the validation logic from the `VaultFactory` to each predicate independently. As such, we consider this exhibit fully alleviated with no follow-up action necessary.

ERC721Permit Manual Review Findings

ERC-01M: Potential Deviation of Standard

Type	Severity	Location
Logical Fault	Unknown	ERC721Permit.sol:L95

Description:

The **EIP-721** standard dictates that an `IERC721::approve` operation should fail if the `msg.sender` is neither the `owner` nor **an authorized operator of the** `owner`. The latter of the two is not evaluated in the `ERC721Permit::permit` function.

Impact:

As the Arcade XYZ team may wish to restrict the `ERC721Permit::permit` functionality solely to the token owner, the current behaviour may be correct.

We advise the Arcade XYZ team to evaluate this exhibit and take appropriate action, either documenting the referenced line or adjusting it to be more flexible and we will adjust the severity accordingly.

Example:

contracts/nft/ERC721Permit.sol

SOL

```sol
85  function permit(
86      address owner,
87      address spender,
88      uint256 tokenId,
89      uint256 deadline,
90      uint8 v,
91      bytes32 r,
92      bytes32 s
93  ) public virtual override {
94      if (block.timestamp > deadline) revert ERC721P_DeadlineExpired(deadline);
95      if (owner != ERC721.ownerOf(tokenId)) revert ERC721P_NotTokenOwner(owner);
```

Recommendation:

We advise the code to potentially evaluate whether the `owner` specified is an authorized member of the token's actual owner via an `ERC721::isApprovedForAll` check.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team adjusted the functionality of the **`ERC721Permit::permit`** function to properly comply with the standard and evaluate whether the payload's signer is approved for all on behalf of the token's owner, alleviating this exhibit in full.

OriginationController Manual Review Findings

OCR-01M: On-Chain Race Condition of Loan Consumptions

Type	Severity	Location
Language Specific	Unknown	OriginationController.sol:L757, L762

Description:

The `OriginationController::_validateCounterparties` function is meant to validate a loan's lender and borrower agreements to the loan's conditions, however, it may perform this validation using ECDSA signatures that are not attached to the caller.

As such, it is possible for a loan agreed by two parties to be "hijacked" by a secondary party that detects the transaction and submits their own with a higher fee in what is known as an on-chain race condition.

Impact:

The severity of this exhibit will be adjusted depending on whether the Arcade XYZ team wishes only direct lender-borrower relations, optional direct lender-borrower relations, or neither of the aforementioned functionalities.

Example:

contracts/OriginationController.sol

SOL

```sol
function _validateCounterparties(
    address borrower,
    address lender,
    address caller,
    address signer,
    Signature calldata sig,
    bytes32 sighash,
    Side neededSide
) internal view {
    address signingCounterparty = neededSide == Side.LEND ? lender : borrower;
    address callingCounterparty = neededSide == Side.LEND ? borrower : lender;

    // Make sure the signer recovered from the loan terms is not the caller,
    // and even if the caller is approved, the caller is not the signing counterparty
    if (caller == signer || caller == signingCounterparty) revert OC_ApprovedOwnLoan(c

    // Check that caller can actually call this function - neededSide assignment
    // defaults to BORROW if the signature is not approved by the borrower, but it cou
    // also not be a participant
    if (!isSelfOrApproved(callingCounterparty, caller) && !isApprovedForContract(calli
        revert OC_CallerNotParticipant(msg.sender);
    }

    // Check signature validity
    if (!isSelfOrApproved(signingCounterparty, signer) && !isApprovedForContract(signi
        revert OC_InvalidSignature(signingCounterparty, signer);
    }

    // Revert if the signer is the calling counterparty
    if (signer == callingCounterparty) revert OC_SideMismatch(signer);
}
```

Recommendation:

As this may be a desirable trait, we advise the Arcade XYZ team to evaluate the impact of this exhibit and either acknowledge it or carry out appropriate action by associating a lender and a borrower on-chain.

We advise that "private" loans between lenders and borrowers directly are facilitated by the contract optionally via an additional signature field as a potential remediation to this exhibit.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team stated that they are aware of the potential to front-run a pending loan, however, they deem it inconsequential as there is no harmful economic impact to either counterparty of a processed loan.

Additionally, private loans can be facilitated between members by utilizing custom verifiers and / or **EIP-1271** smart contracts that validate both counterparty addresses. As such, we consider this exhibit nullified.

OCR-02M: Restrictive Minimum Loan Principal

Type	Severity	Location
Logical Fault	Minor	OriginationController.sol:L683

Description:

The `MIN_LOAN_PRINCIPAL` value is set as `1_000_000` irrespective of the actual decimals of the principal asset, causing the code to be incompatible with tokens such as `WBTC`.

Impact:

The `OriginationController` and thus entry-point of the Arcade XYZ lending and borrowing system is inherently incompatible with low-decimal high-value assets as the minimum enforced would introduce a significantly high bar of entry.

Example:

contracts/OriginationController.sol

SOL

```sol
function _validateLoanTerms(LoanLibrary.LoanTerms memory terms) internal virtual view
    // principal must be greater than or equal to the MIN_LOAN_PRINCIPAL constant
    if (terms.principal < MIN_LOAN_PRINCIPAL) revert OC_PrincipalTooLow(terms.principa

    // loan duration must be greater or equal to 1 hr and less or equal to 3 years
    if (terms.durationSecs < 3600 || terms.durationSecs > 94_608_000) revert OC_LoanDu

    // interest rate must be greater than or equal to 0.01%
    // and less or equal to 10,000% (1e6 basis points)
    if (terms.proratedInterestRate < 1e18 || terms.proratedInterestRate > 1e24) revert

    // signature must not have already expired
    if (terms.deadline < block.timestamp) revert OC_SignatureIsExpired(terms.deadline)

    // validate payable currency
    if (!allowedCurrencies[terms.payableCurrency]) revert OC_InvalidCurrency(terms.pay

    // validate collateral
    if (!allowedCollateral[terms.collateralAddress]) revert OC_InvalidCollateral(terms
}
```

Recommendation:

We advise the code to either enforce a `USD` based minimum loan principal (i.e. by utilizing Chainlink oracles), or to enforce a minimum loan principal per `allowedCurrencies` type, either of which we consider an adequate resolution to this exhibit.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

Minimum loan principals are now dictated by dedicated entries per allowed currency, alleviating this exhibit in full and rendering the system compatible with any-decimal token.

OCR-03M: Improper Validation of Loan Amounts

Type	Severity	Location
Logical Fault	Medium	OriginationController.sol:L508, L549, L863-L864, L867-L868

Description:

The `OriginationController::recoverTokenSignature` and `OriginationController::recoverItemsSignature` functions are meant to allow the borrower and lender of a loan to validate that they agree to the terms of a loan, however, the fees imposed on the loan's `principal` amount are not accounted for in these validations.

As a result, a `borrower` may have agreed to receive i.e. `1_000_000` USDC but would in reality receive less than that, significantly affecting the expectations of users when utilizing the Arcade XYZ platform.

Impact:

Users will sign payloads expecting to receive and provide a specific amount of the payable currency, however, the fees applied by the Arcade XYZ protocol will not be accounted for in these amounts and can additionally change between a transaction's submission and a transaction's execution by the network, further affecting the lender's and the borrower's user experience.

Example:

contracts/OriginationController.sol

SOL

```sol
/**
 * @notice Determine the external signer for a signature specifying only a collateral
 *
 * @param loanTerms                     The terms of the loan.
 * @param sig                           The signature, with v, r, s fields.
 * @param nonce                         The signature nonce.
 * @param side                          The side of the loan being signed.
 *
 * @return sighash                      The hash that was signed.
 * @return signer                       The address of the recovered signer.
 */
function recoverTokenSignature(
    LoanLibrary.LoanTerms calldata loanTerms,
    Signature calldata sig,
    uint160 nonce,
    Side side
) public view override returns (bytes32 sighash, address signer) {
    bytes32 loanHash = keccak256(
        abi.encode(
            _TOKEN_ID_TYPEHASH,
            loanTerms.durationSecs,
            loanTerms.deadline,
            loanTerms.proratedInterestRate,
            loanTerms.principal,
            loanTerms.collateralAddress,
            loanTerms.collateralId,
            loanTerms.payableCurrency,
            loanTerms.affiliateCode,
            nonce,
            uint8(side)
        )
    );

    sighash = _hashTypedDataV4(loanHash);
    signer = ECDSA.recover(sighash, sig.v, sig.r, sig.s);
}

/**
 * @notice Determine the external signer for a signature specifying specific items.
 * @dev    Bundle ID should _not_ be included in this signature, because the loan
 *         can be initiated with any arbitrary bundle - as long as the bundle contains
 *
 * @param loanTerms                     The terms of the loan.
 * @param sig                           The loan terms signature, with v, r, s fields.
 * @param nonce                         The signature nonce.
```

```
530  * @param side                      The side of the loan being signed.
531  * @param itemsHash                 The required items in the specified bundle.
532  *
533  * @return sighash                  The hash that was signed.
534  * @return signer                   The address of the recovered signer.
535  */
536 function recoverItemsSignature(
537     LoanLibrary.LoanTerms calldata loanTerms,
538     Signature calldata sig,
539     uint160 nonce,
540     Side side,
541     bytes32 itemsHash
542 ) public view override returns (bytes32 sighash, address signer) {
543     bytes32 loanHash = keccak256(
544         abi.encode(
545             _ITEMS_TYPEHASH,
546             loanTerms.durationSecs,
547             loanTerms.deadline,
548             loanTerms.proratedInterestRate,
549             loanTerms.principal,
550             loanTerms.collateralAddress,
551             itemsHash,
552             loanTerms.payableCurrency,
553             loanTerms.affiliateCode,
554             nonce,
555             uint8(side)
556         )
557     );
558
559     sighash = _hashTypedDataV4(loanHash);
560     signer = ECDSA.recover(sighash, sig.v, sig.r, sig.s);
561 }
```

Recommendation:

We advise the signature verification code to either utilize the "final" principal amounts for the lender and borrower or the fees the loan will utilize to be included in the signed payload, either of which we consider an adequate resolution to this exhibit.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team weighed the code adjustments required for proper alleviation of this exhibit and concluded that they would significantly overhaul the semantics and logic of the contract in relation to the referenced signature recovery mechanisms.

Based on the perceived seldom occurrence of a fee change in a production environment, the Arcade XYZ team opted to acknowledge the exhibit and advise users to utilize counter-measures such as invalidated nonces as well as expiry timestamps that are in sync with the time required for a fee change to be processed.

OCR-04M: Insufficient Validation of Loan Rollover

Type	Severity	Location
Logical Fault	Medium	OriginationController.sol:L709-L723

Description:

The `OriginationController::_validateRollover` function will not validate whether a loan has expired, permitting expired loans to be rolled over improperly.

Impact:

It is possible for a user to rollover their loan even after it has expired, introducing similar discrepant behaviours to the repayments on expired loans as described in the relevant exhibit of `RepaymentController`.

Example:

contracts/OriginationController.sol

SOL

```sol
function _validateRollover(LoanLibrary.LoanTerms memory oldTerms, LoanLibrary.LoanTerm
    internal
    pure
{
    if (newTerms.payableCurrency != oldTerms.payableCurrency)
        revert OC_RolloverCurrencyMismatch(oldTerms.payableCurrency, newTerms.payableC

    if (newTerms.collateralAddress != oldTerms.collateralAddress || newTerms.collatera
        revert OC_RolloverCollateralMismatch(
            oldTerms.collateralAddress,
            oldTerms.collateralId,
            newTerms.collateralAddress,
            newTerms.collateralId
        );
}
```

Recommendation:

We advise the code to ensure that a loan has not expired by validating the `dueDate` of a loan as calculated within `LoanCore::claim`, disallowing a loan from being possible to rollover when it has defaulted.

Alternatively, the rollover could be permitted solely if the lender remains the same if the loan has defaulted as this would essentially mean that the lender and borrower renegotiated their terms.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and stated that they wish to leave the onus of liquidation / default to the loan provider akin to other lending protocols such as Aave, meaning that a loan can still be rolled over even if it has expired.

As such, we consider this exhibit nullified given that it describes desirable behaviour by the Arcade XYZ team.

OCR-05M: Invalidation of Predicates

Type	Severity	Location
Logical Fault	Major	OriginationController.sol:L225, L406

Description:

The predicates of a loan's initialization are evaluated before the `LoanCore::_mintLoanNotes` function is executed. As the creation flow of `LoanCore` will mint the relevant notes and then proceed to execute the relevant NFT transfers, the predicates can be invalidated by a re-entrant call during the creation of a loan note as a `PromissoryNote::mint` operation relies on `ERC721::_safeMint` which is re-entrant.

Impact:

Any predicate evaluated by the `OriginationController::_runPredicatesCheck` can be trivially invalidated by the `borrower` as their `PromissoryNote` is minted before the `AssetVault` is transferred, exposing even `OwnableERC721::onlyOwner` functionality to them before the asset is transferred.

Example:

contracts/OriginationController.sol

SOL

```sol
function initializeLoanWithItems(
    LoanLibrary.LoanTerms calldata loanTerms,
    address borrower,
    address lender,
    Signature calldata sig,
    uint160 nonce,
    LoanLibrary.Predicate[] calldata itemPredicates
) public override returns (uint256 loanId) {
    _validateLoanTerms(loanTerms);
    if (itemPredicates.length == 0) revert OC_PredicatesArrayEmpty();

    bytes32 encodedPredicates = _encodePredicates(itemPredicates);

    // Determine if signature needs to be on the borrow or lend side
    Side neededSide = isSelfOrApproved(borrower, msg.sender) ? Side.LEND : Side.BORROW

    (bytes32 sighash, address externalSigner) = recoverItemsSignature(
        loanTerms,
        sig,
        nonce,
        neededSide,
        encodedPredicates
    );

    _validateCounterparties(borrower, lender, msg.sender, externalSigner, sig, sighash
    _runPredicatesCheck(borrower, lender, loanTerms, itemPredicates);

    loanCore.consumeNonce(externalSigner, nonce);
    loanId = _initialize(loanTerms, borrower, lender);
}
```

Recommendation:

We advise either the `PromissoryNote::mint` function to be updated to utilize the `ERC721::_mint` operation over the `ERC721::_safeMint` operation, or the code to evaluate the predicates after the `LoanCore` contract has acquired custody of the `AssetVault`.

To note, all predicates would still be invalidated by lingering approval operations, however, this is covered in a separate exhibit within `CallWhitelistApprovals`.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `OriginationController::_runPredicatesCheck` function invocations were relocated after the initialization and rollover of a loan, ensuring that the vault is properly in escrow when the predicates are run and thus preventing re-entrancies from exploiting this attack vector.

OwnableERC721 Manual Review Findings

OER-01M: Inexistent Validation of Ownership Existence

Type	Severity	Location
Input Sanitization	Minor	OwnableERC721.sol:L44-L46

Description:

The `OwnableERC721::_setNFT` function does not validate that the NFT ID that evaluates to ownership (i.e. `uint256(uint160(address(this)))`) is in circulation by the specified token.

Impact:

While a renunciation of ownership rights may be desirable, it is better to expose it via a dedicated function akin to the `Ownable` standards by OpenZeppelin.

Example:

contracts/vault/OwnableERC721.sol

SOL

```sol
/**
 * @notice Specifies the owner of the underlying token ID, derived
 *         from the contract address of the contract implementing.
 *
 * @return ownerAddress         The owner of the underlying token derived from
 *                              the calling address.
 */
function owner() public view virtual returns (address ownerAddress) {
    return IERC721(ownershipToken).ownerOf(uint256(uint160(address(this))));
}

// ========================================== HELPERS ==============================

/**
 * @dev Set the ownership token - the ERC721 that specified who controls
 *      defined addresses.
 *
 * @param _ownershipToken       The address of the ERC721 token that defines ownership
 */
function _setNFT(address _ownershipToken) internal {
    ownershipToken = _ownershipToken;
}
```

Recommendation:

We advise the code to enforce sanity checks as all `OwnableERC721::onlyOwner` functions could fail to execute if the `ownershipToken` is misconfigured.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team has introduced documentation in the `AssetVault` that specifies a deployment of the contract by a non-ERC721 contract would cause ownership functionality to fail. In conjunction with the fact that the `OwnableERC721` contract is solely utilized by the `AssetVault`, we consider this action an adequate resolution to the described exhibit.

PromissoryNote Manual Review Findings

PNE-01M: Misleading Documentation of Contract

Type	Severity	Location
Logical Fault	Unknown	PromissoryNote.sol:L26, L204

Description:

The contract's documentation specifies that it is built off `ERC721PresetMinterPauserAutoId`, however, it solely derives from `ERC721Enumerable` and the base `ERC721` implementation.

Additionally, the **`PromissoryNote::_beforeTokenTransfer`** function `override` specifies that it will not let tokens be transferred when the contract is paused, a state it cannot be in.

Impact:

The severity of this exhibit will be adjusted according to the remediation action taken by the Arcade XYZ team.

Example:

contracts/PromissoryNote.sol

SOL

```sol
/**
 * @dev Hook that is called before any token transfer.
 *      This notifies the promissory note about the ownership transfer.
 *
 * @dev Does not let tokens be transferred when contract is paused.
 *
 * @param from                  The previous owner of the token.
 * @param to                    The owner of the token after transfer.
 * @param tokenId               The token ID.
 */
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual override(ERC721, ERC721Enumerable) {
    super._beforeTokenTransfer(from, to, tokenId);
}
```

Recommendation:

We advise either the documentation or the inheritance structure of the contract to be revised, either of which we consider an adequate remediation to this exhibit.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The documentation was corrected, removing the statement in regard to a paused state and thus addressing this exhibit.

PNE-02M: Misleading Token ID Counter

Type	Severity	Location
Logical Fault	Informational	PromissoryNote.sol:L69, L103

Description:

The `_tokenIdTracker` variable of the contract is specified as `private` and is solely utilized in the `PromissoryNote::constructor` to increment it by one.

Example:

contracts/PromissoryNote.sol

SOL

```sol
82  constructor(
83      string memory name,
84      string memory symbol,
85      address _descriptor
86  ) ERC721(name, symbol) ERC721Permit(name) {
87      if (_descriptor == address(0)) revert PN_ZeroAddress("descriptor");
88
89      descriptor = INFTDescriptor(_descriptor);
90
91      _setupRole(ADMIN_ROLE, msg.sender);
92      _setupRole(RESOURCE_MANAGER_ROLE, msg.sender);
93
94      // Allow admin to set mint/burn role, which they will do
95      // during initialize. After initialize, admin role is
96      // permanently revoked, so mint/burn role becomes immutable
97      // and initialize cannot be called again.
98      // Do not set role admin for admin role.
99      _setRoleAdmin(MINT_BURN_ROLE, ADMIN_ROLE);
100     _setRoleAdmin(RESOURCE_MANAGER_ROLE, RESOURCE_MANAGER_ROLE);
101
102     // We don't want token IDs of 0
103     _tokenIdTracker.increment();
104 }
```

Recommendation:

As `PromissoryNote` NFT IDs are dictated by the caller of **`PromissoryNote::mint`**, we advise the code to omit all code relating to `_tokenIdTracker` as it is misleading.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `_tokenIdTracker` entry of the contract and all instances of it have been properly omitted from the codebase as advised.

PunksVerifier Manual Review Findings

PVR-01M: Potentially Unwarranted Predicate Case

Type	Severity	Location
Code Style	Informational	PunksVerifier.sol:L74

Description:

The `PunksVerifier::verifyPredicates` function contains a special verification mechanism whereby a negative `tokenId` value signifies a simple balance evaluation of the `vault`.

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```sol
function verifyPredicates(
    address, address,
    address collateralAddress,
    uint256 collateralId,
    bytes calldata predicates
) external view override returns (bool) {
    address vault = IVaultFactory(collateralAddress).instanceAt(collateralId);

    // Unpack items
    int256[] memory tokenIds = abi.decode(predicates, (int256[]));

    for (uint256 i = 0; i < tokenIds.length; i++) {
        int256 tokenId = tokenIds[i];

        if (tokenId > 9999) revert IV_InvalidTokenId(tokenId);

        if (tokenId < 0 && punks.balanceOf(vault) == 0) return false;
        // Does not own specifically specified asset
        else if (tokenId >= 0 && punks.punkIndexToAddress(tokenId.toUint256()) != vaul
    }

    // Loop completed - all items found
    return true;
}
```

Recommendation:

Given that the `CollectionWideOfferVerifier::verifyPredicates` function is compatible with the `IPunks` contract, we advise the functionality within `PunksVerifier::verifyPredicates` to be omitted given that it can be confusing and is relatively unrestricted (i.e. any negative `tokenId` value will cause it to execute).

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and stated that they do not plan to deploy the `PunksVerifier` in the immediate future nor whitelist its operation, meaning that they may potentially revamp the contract.

As such, we consider this exhibit safely acknowledged.

PVR-02M: Incorrect Assumption of Function

Type	Severity	Location
Logical Fault	Minor	PunksVerifier.sol:L48-L49

Description:

The `PunksVerifier::verifyPredicates` function assumes that an empty predicates array has been addressed in the `OriginationController::initializeLoanWithItems` and `OriginationController::rolloverLoanWithItems` functions, however, this is **incorrect** as the functions ensure that the **whole predicate array is not empty**, not that the `data` **payload of a predicate call contains non-zero entries**.

Impact:

The `PunksVerifier::verifyPredicates` will misbehave if supplied an empty `tokenIds` data entry when called via the `OriginationController` despite its documentation specifying that no sanitization is needed as it is incorrect.

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```sol
/**
 * @notice Verify that the items specified by the packed int256 array are held by the
 * @dev    Reverts on out of bounds token Ids, returns false on missing contents.
 *
 *         Verification for empty predicates array has been addressed in initializeLoa
 *         rolloverLoanWithItems.
 *
 * @param collateralAddress             The address of the loan's collateral.
 * @param collateralId                  The tokenId of the loan's collateral.
 * @param predicates                    The int256[] array of punk IDs to check for, p
 *
 * @return verified                     Whether the bundle contains the specified item
 */
// solhint-disable-next-line code-complexity
function verifyPredicates(
    address, address,
    address collateralAddress,
    uint256 collateralId,
    bytes calldata predicates
) external view override returns (bool) {
    address vault = IVaultFactory(collateralAddress).instanceAt(collateralId);

    // Unpack items
    int256[] memory tokenIds = abi.decode(predicates, (int256[]));

    for (uint256 i = 0; i < tokenIds.length; i++) {
```

Recommendation:

We advise the code to properly ensure that the `tokenIds` decoded contain a non-zero length as otherwise the predicate would succeed without validating anything, signifying a potential scam attempt.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

A proper `if-revert` pattern was introduced that ensures the decoded `int256` array contains non-zero entries, alleviating this exhibit in full.

RepaymentController Manual Review Findings

RCR-01M: Improper Imposition of Claim Fee

Type	Severity	Location
Logical Fault	Minor	RepaymentController.sol:L115

Description:

The `RepaymentController::claim` function will impose a `claimFee` to the `lender` that needs to be paid immediately before unlocking the NFT asset.

This approach is counter-intuitive as the `lender` may wish to liquidate the asset to cover their other obligations, meaning that they may not have the funds available to repay the `claimFee` before unlocking the NFT.

This would be especially prevalent in high-value loans that use f.e. `CryptoPunks` as the `claimFee` is proportionate.

Impact:

Imposing a claim fee on a loan that has entered default is an ill-advised trait of the protocol as it indicates an edge case whereby the `lender` has potentially already incurred a loss.

Example:

contracts/RepaymentController.sol

SOL

```sol
function claim(uint256 loanId) external override {
    LoanLibrary.LoanData memory data = loanCore.getLoan(loanId);
    if (data.state == LoanLibrary.LoanState.DUMMY_DO_NOT_USE) revert RC_CannotDerefere

    // make sure that caller owns lender note
    // Implicitly checks if loan is active - if inactive, note will not exist
    address lender = lenderNote.ownerOf(loanId);
    if (lender != msg.sender) revert RC_OnlyLender(lender, msg.sender);

    LoanLibrary.LoanTerms memory terms = data.terms;
    uint256 interest = getInterestAmount(terms.principal, terms.proratedInterestRate);
    uint256 totalOwed = terms.principal + interest;

    uint256 claimFee = (totalOwed * data.feeSnapshot.lenderDefaultFee) / BASIS_POINTS_

    loanCore.claim(loanId, claimFee);
}
```

Recommendation:

We advise the `claimFee` to either be removed, be a fixed amount, or to be due after the NFT has been claimed. The latter of the three options is already achievable due to the usage of the re-entrant **EIP-721** `ERC721::safeTransferFrom` function via the **`LoanCore::claim`** function.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team re-ordered the operations performed by the **`LoanCore::claim`** function to collect the claim fee from the lender after the **EIP-721** asset has been transferred to them, alleviating this exhibit in full.

RCR-02M: Inexistent Validation of Loan Expiry

Type	Severity	Location
Logical Fault	Medium	RepaymentController.sol:L72-L77, L88-L93

Description:

The `RepaymentController::repay` and `RepaymentController::forceRepay` functions as well as the `LoanCore::repay` and `LoanCore::forceRepay` functions do not validate whether a loan has expired.

Impact:

The current implementation permits technically-savvy individuals to create bots that would transmit a repayment transaction solely when they detect a `RepaymentController::claim` operation by the lender. As they are able to repay at any time, they can cause all `RepaymentController::claim` operations to fail.

Example:

contracts/RepaymentController.sol

SOL

```sol
/**
 * @notice Repay an active loan, referenced by borrower note ID (equivalent to loan ID
 *         is calculated, and the principal plus interest is withdrawn from the caller
 *         Anyone can repay a loan. Control is passed to LoanCore to complete repaymen
 *
 * @param  loanId               The ID of the loan.
 */
function repay(uint256 loanId) external override {
    (uint256 amountFromBorrower, uint256 amountToLender) = _prepareRepay(loanId);

    // call repay function in loan core -  msg.sender will pay the amountFromBorrower
    loanCore.repay(loanId, msg.sender, amountFromBorrower, amountToLender);
}

/**
 * @notice Repay an active loan, referenced by borrower note ID (equivalent to loan ID
 *         is calculated, and the principal plus interest is withdrawn from the caller
 *         Using forceRepay will not send funds to the lender: instead, those funds wi
 *         available for withdrawal in LoanCore. Can be used in cases where a borrower
 *         but the lender is not able to receive those tokens (e.g. token blacklist).
 *
 * @param  loanId               The ID of the loan.
 */
function forceRepay(uint256 loanId) external override {
    (uint256 amountFromBorrower, uint256 amountToLender) = _prepareRepay(loanId);

    // call repay function in loan core -  msg.sender will pay the amountFromBorrower
    loanCore.forceRepay(loanId, msg.sender, amountFromBorrower, amountToLender);
}
```

Recommendation:

Either the `LoanCore` or the `RepaymentController` system should prohibit repayments from occurring if the loan's `dueDate` as calculated within **`LoanCore::claim`** has surpassed.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team stated that repayments of expired loans are expected to be possible as the onus of liquidating / defaulting a loan is up to the lender akin to other lending protocols such as Aave.

As such, we consider this exhibit nullified given that it represents desirable behaviour by the Arcade XYZ team.

ArcadeItemsVerifier Code Style Findings

AIV-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ArcadeItemsVerifier.sol:L102

Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

Example:

contracts/verifiers/ArcadeItemsVerifier.sol

SOL

```sol
102 for (uint256 i = 0; i < items.length; i++) {
```

Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The loop iterator has been optimized as advised, relocating it to a dedicated `unchecked` code block that optimizes its execution.

AIV-02C: Optimization of `if` Structure

Type	Severity	Location
Gas Optimization	Informational	ArcadeItemsVerifier.sol:L127

Description:

The `if-else-if` structure within the `ArcadeItemsVerifier::verifyPredicates` function's `for` loop can be optimized as the last condition is redundantly evaluated.

Example:

contracts/verifiers/ArcadeItemsVerifier.sol

SOL

```sol
112 if (item.cType == CollateralType.ERC_721) {
113     IERC721 asset = IERC721(item.asset);
114
115     // Wildcard, but vault has no assets or not enough specified
116     if (item.anyIdAllowed && asset.balanceOf(vault) < item.amount) return false;
117     // Does not own specifically specified asset
118     if (!item.anyIdAllowed && asset.ownerOf(item.tokenId) != vault) return false;
119 } else if (item.cType == CollateralType.ERC_1155) {
120     IERC1155 asset = IERC1155(item.asset);
121
122     // Wildcard not allowed, since we can't check overall 1155 balances
123     if (item.anyIdAllowed) revert IV_InvalidWildcard(item.asset);
124
125     // Does not own specifically specified asset
126     if (asset.balanceOf(vault, item.tokenId) < item.amount) return false;
127 } else if (item.cType == CollateralType.ERC_20) {
```

Recommendation:

Given that the `CollateralType` contract member represents an `enum`, the Solidity decoding mechanism will disallow values that cannot be cast to the `enum`.

As such, we advise the last `else if` conditional to be made a simple `else` clause optimizing the gas cost of the contract during **EIP-20** balance validations.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `if-else-if` structure has been optimized as advised, reducing the conditional's worst-case evaluation gas cost.

ArtBlocksVerifier Code Style Findings

ABV-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	ArtBlocksVerifier.sol:L110

Description:

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

Example:

contracts/verifiers/ArtBlocksVerifier.sol

SOL

```
110 found++;
111
112 if (found >= item.amount) break;
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The referenced increment of the `found` variable has been wrapped in an `unchecked` code block safely, optimizing its gas cost.

ABV-02C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	ArtBlocksVerifier.sol:L86, L103

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

Example:

contracts/verifiers/ArtBlocksVerifier.sol

SOL

```sol
86  for (uint256 i = 0; i < items.length; ++i) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

All referenced iterator optimizations have been applied, addressing this exhibit in full.

AssetVault Code Style Findings

AVT-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	AssetVault.sol:L206

Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

Example:

contracts/vault/AssetVault.sol

SOL

```sol
206 for (uint256 i = 0; i < tokensLength; i++) {
```

Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The loop iterator has been optimized as advised, relocating it to a dedicated `unchecked` code block that optimizes its execution.

AVT-02C: Misconception of Re-Entrancy Attack Vectors

Type	Severity	Location
Gas Optimization	Informational	AssetVault.sol:L224, L231

Description:

The `AssetVault::withdrawETH` function specifies that a potential re-entrancy attack vector can be introduced by the transmission of funds to the `to` address and that is why the `ReentrancyGuard::nonReentrant` modifier has been introduced to the function, however, the `ReentrancyGuard::nonReentrant` modifier will solely protect the functions it is specified in.

As the `AssetVault::onlyWithdrawEnabled` modifier is solely applied in this function, a potential re-entrancy would not protect the contract as it solely guards the `AssetVault::withdrawETH` function that will transmit all funds to the `to` address.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
224 function withdrawETH(address to) external override onlyOwner onlyWithdrawEnabled nonRe
225     if (to == address(0)) revert AV_ZeroAddress("to");
226 
227     // perform transfer
228     uint256 balance = address(this).balance;
229     // sendValue() internally uses call() which passes along all of
230     // the remaining gas, potentially introducing an attack vector
231     payable(to).sendValue(balance);
232     emit WithdrawETH(msg.sender, to, balance);
233 }
```

Recommendation:

We advise the code to simply omit the `ReentrancyGuard::nonReentrant` modifier as it is practically ineffectual at the point it has been introduced.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

AVT-03C: Optimization of `if-else` Structure

Type	Severity	Location
Gas Optimization	Informational	AssetVault.sol:L211

Description:

The referenced `else-if` clause is redundant as it will always evaluate to `true` based on the preceding clauses.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
209 if (tokenTypes[i] == TokenType.ERC721) {
210     _withdrawERC721(tokens[i], tokenIds[i], to);
211 } else if (tokenTypes[i] == TokenType.ERC1155) {
212     _withdrawERC1155(tokens[i], tokenIds[i], to);
213 }
```

Recommendation:

We advise it to be omitted, replacing it with a direct `else` clause thus optimizing the code's gas cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The `if-else` structure has been optimized as advised, reducing its gas cost on each iteration of the `for` loop its located in.

AVT-04C: Potential Enhancement of Functionality

Type	Severity	Location
Code Style	Informational	AssetVault.sol:L196, L209-L213

Description:

The `AssetVault::withdrawBatch` function does not presently support **EIP-20** assets when it could, simplifying the extraction process of a vault via a single function.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
195 function withdrawBatch(
196     address[] calldata tokens,
197     uint256[] calldata tokenIds,
198     TokenType[] calldata tokenTypes,
199     address to
200 ) external override onlyOwner onlyWithdrawEnabled {
201     uint256 tokensLength = tokens.length;
202     if (tokensLength > MAX_WITHDRAW_ITEMS) revert AV_TooManyItems(tokensLength);
203     if (tokensLength != tokenIds.length) revert AV_LengthMismatch("tokenId");
204     if (tokensLength != tokenTypes.length) revert AV_LengthMismatch("tokenType");
205
206     for (uint256 i = 0; i < tokensLength; i++) {
207         if (tokens[i] == address(0)) revert AV_ZeroAddress("token");
208
209         if (tokenTypes[i] == TokenType.ERC721) {
210             _withdrawERC721(tokens[i], tokenIds[i], to);
211         } else if (tokenTypes[i] == TokenType.ERC1155) {
212             _withdrawERC1155(tokens[i], tokenIds[i], to);
213         }
214     }
215 }
```

Recommendation:

We advise the `AssetVault::withdrawBatch` to be updated to support a token type of `ERC20` that should also be introduced to the relevant `TokenType` enum.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase as most use cases of the vault will involve **EIP-721** and **EIP-1155** assets.

As such, we consider this exhibit safely acknowledged.

AVT-05C: Potential Optimization of Project Structure

Type	Severity	Location
Gas Optimization	Informational	AssetVault.sol:L79-L82, L93-L100

Description:

The `AssetVault` contract is meant to be deployed once as a `template` that the `VaultFactory` will utilize.

Example:

contracts/vault/AssetVault.sol

SOL

```sol
69  address public override whitelist;
```

Recommendation:

We advise template flow to be revised by having the `VaultFactory` deploy an instance of `AssetVault` during its **`VaultFactory::constructor`** and assign it in its `template` data entry. This will permit the `ownershipToken` of `OwnableERC721` as well as the `whitelist` member of `AssetVault` to be set as `immutable`, greatly optimizing the gas cost of the contracts across the board.

Any `immutable` variable is compatible with proxy systems given that their values are "baked in" the bytecode of the deployed instance. We advise this and the relevant exhibit of `OwnableERC721` to be applied in tandem if the optimizations are ultimately desired.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and assessed that the flexibility of the `VaultFactory` will be greatly diminishing if the `immutable` based optimization is applied, a trait that the Arcade XYZ team does not wish to forfeit.

As such, we consider this exhibit safely acknowledged.

CallWhitelist Code Style Findings

CWT-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	CallWhitelist.sol:L72, L74, L88, L90

Description:

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

Example:

contracts/vault/CallWhitelist.sol

SOL

```sol
72  if (whitelist[callee][selector]) revert CW_AlreadyWhitelisted(callee, selector);
73
74  whitelist[callee][selector] = true;
```

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.

Alleviation (45ccaa43fa72dfe818736e5dc737af4afb2b2af3):

The referenced optimization has been applied properly, caching the interim `whitelist[callee]` lookup to a `storage` variable that is consequently utilized for both access statements.

FeeController Code Style Findings

FCR-01C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	FeeController.sol:L128-L136, L146-L151

Description:

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

Example:

contracts/FeeController.sol

SOL

```sol
return (
    FeesOrigination (
        loanFees[FL_01],
        loanFees[FL_02],
        loanFees[FL_05],
        loanFees[FL_06],
        loanFees[FL_07]
    )
);
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The redundant parenthesis statements have been properly omitted, optimizing the code's legibility.

FCR-02C: Suboptimal Struct Declaration Styles

Type	Severity	Location
Code Style	Informational	FeeController.sol:L129-L135, L147-L150

Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

Example:

contracts/FeeController.sol

SOL

```sol
129 FeesOrigination (
130     loanFees[FL_01],
131     loanFees[FL_02],
132     loanFees[FL_05],
133     loanFees[FL_06],
134     loanFees[FL_07]
135 )
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

Both struct instantiations have been replaced by their key-value declaration style, greatly increasing their legibility.

LoanCore Code Style Findings

LCE-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	LoanCore.sol:L176, L415, L569, L666, L702

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/LoanCore.sol

SOL

```sol
169 // Check that we will not net lose tokens
170 if (_amountToBorrower > _amountFromLender) revert LC_CannotSettle(_amountToBorrower, _
171
172 // Mark collateral as escrowed
173 collateralInUse[collateralKey] = true;
174
175 // Assign fees for withdrawal
176 uint256 feesEarned = _amountFromLender - _amountToBorrower;
```

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 (0a7551b095ee0dc4fb2a56d2267e05472128cf0e):

The `unchecked` code blocks have been introduced to all referenced calculations, addressing this exhibit in full.

LCE-02C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	LoanCore.sol:L521

Description:

The linked `for` loop evaluates its limit inefficiently on each iteration.

Example:

contracts/LoanCore.sol

SOL

```sol
521 for (uint256 i = 0; i < borrowerNote.balanceOf(caller); i++) {
```

Recommendation:

We advise the statements within the `for` loop limit to be relocated outside to a local variable declaration that is consequently utilized for the evaluation to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in such limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The loop's length is now properly calculated in a dedicated variable outside the `for` loop (`noteCount`), ensuring that the loop's limit evaluation code is significantly more optimal and thus addressing this exhibit.

LCE-03C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	LoanCore.sol:L180, L181, L313, L314, L355, L356, L422, L423, L501, L527, L528, L566, L569, L588, L589, L671, L672, L713, L715

Description:

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

Example:

contracts/LoanCore.sol

SOL

```sol
180 feesWithdrawable[terms.payableCurrency][address(this)] += protocolFee;
181 feesWithdrawable[terms.payableCurrency][affiliate] += affiliateFee;
```

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.

Alleviation (7a4e1dc948):

The exhibit has been partially alleviated as multiple `mapping` lookup pairs have not been optimized (`L180 + L181`, `L566 + L569`, `L588 + L589`, `L713 + L715`) and certain optimizations were incorrectly applied by using `memory` instead of `storage` pointers (`L501`, `L527 + L528`).

Alleviation (0a7551b095):

The remaining highlighted exhibits apart from the first one were properly optimized and the `memory` pointers changed to `storage`. Given that the `L180 + L181` instance is conditional, the optimization would work solely in a worst-case scenario and as such we consider it to be up to the Arcade XYZ team's discretion.

As such, we consider this exhibit fully alleviated.

LCE-04C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	LoanCore.sol:L521, L611

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

Example:

contracts/LoanCore.sol

SOL

```sol
521 for (uint256 i = 0; i < borrowerNote.balanceOf(caller); i++) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

All referenced iterator optimizations have been applied, addressing this exhibit in full.

LCE-05C: Optimization of Assignments

Type	Severity	Location
Gas Optimization	Informational	LoanCore.sol:L180-L181, L313-L314, L355-L356, L422-L423, L671-L672

Description:

The referenced assignments are performed unconditionally when they may result in a zero-value increment.

Example:

contracts/LoanCore.sol

SOL

```sol
180 feesWithdrawable[terms.payableCurrency][address(this)] += protocolFee;
181 feesWithdrawable[terms.payableCurrency][affiliate] += affiliateFee;
```

Recommendation:

We advise them to be performed conditionally akin to transfers, optimizing the gas cost of the function significantly under certain cases.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

All referenced assignments are now properly performed solely when the values that are being added to the existing data entries are non-zero, optimizing each assignment's worst-case gas cost significantly.

LCE-06C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	LoanCore.sol:L263

Description:

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

Example:

contracts/LoanCore.sol

SOL

```sol
263 noteReceipts[loanId] = NoteReceipt(data.terms.payableCurrency, _amountToLender);
```

Recommendation:

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

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The structure's instantiation has been replaced by the key-value declaration style, optimizing its legibility greatly.

LoanLibrary Code Style Findings

LLY-01C: Inefficient Tight Packing of Structs

Type	Severity	Location
Gas Optimization	Informational	LoanLibrary.sol:L36, L38, L44, L50, L54

Description:

The `LoanTerms` and `LoanTermsWithItems` structs defined within `LoanLibrary` specify that they are meant to tight-pack their variables, however, they are packed inefficiently. In detail, two `uint32` variables are packed with a `uint160` variable whilst the `struct` contains two `address` types (represented by 160-bits) that occupy a full slot.

Example:

contracts/libraries/LoanLibrary.sol

SOL

```sol
/**
 * @dev The raw terms of a loan.
 */
struct LoanTerms {
    /// @dev Packed variables
    // The number of seconds representing relative due date of the loan.
    /// @dev Max is 94,608,000, fits in 32 bits
    uint32 durationSecs;
    // Timestamp for when signature for terms expires
    uint32 deadline;
    // Interest expressed as a rate, unlike V1 gross value.
    // Input conversion: 0.01% = (1 * 10**18) ,  10.00% = (1000 * 10**18)
    // This represents the rate over the lifetime of the loan, not APR.
    // 0.01% is the minimum interest rate allowed by the protocol.
    /// @dev Max is 10,000%, fits in 160 bits
    uint160 proratedInterestRate;
    /// @dev Full-slot variables
    // The amount of principal in terms of the payableCurrency.
    uint256 principal;
    // The token ID of the address holding the collateral.
    /// @dev Can be an AssetVault, or the NFT contract for unbundled collateral
    address collateralAddress;
    // The token ID of the collateral.
    uint256 collateralId;
    // The payable currency for the loan principal and interest.
    address payableCurrency;
    // Affiliate code used to start the loan.
    bytes32 affiliateCode;
}
```

Recommendation:

We advise the `proratedInterestRate` to be up-cast to 256 bits and the two `uint32` variables to be relocated after each `address` member.

Additionally, we advise the `uint32` variables to be upcast to `uint96` ensuring that they occupy a full slot when paired with an `address` type to prevent data corruption in future updates.

Regardless of the upcast operations, relocating the `uint32` variables will lead to a reduction of one storage slot in each struct.

As a final note, we would like to specify that the `LoanData` could also be refactored to tight-pack the `FeeSnapshot` and `LoanTerms` entries via a new `struct` declaration (i.e. `LoanTermsWithFeeSnapshot`).

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The recommended course of action has been applied properly, ensuring that the `address` members are optimally packed with the relevant sub-256 bit variables and that the `proratedInterestRate` is increased in accuracy whilst reducing the overall storage footprint of the referenced structs.

OriginationController Code Style Findings

OCR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	OriginationController.sol:L977, L983, L1001

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/OriginationController.sol

SOL

```sol
974 if (repayAmount > borrowerOwedForNewLoan) {
975     // amount to collect from borrower
976     // new loan principal is less than old loan repayment amount
977     amounts.needFromBorrower = repayAmount - borrowerOwedForNewLoan;
978 } else {
979     // amount to collect from lender (either old or new)
980     amounts.leftoverPrincipal = amounts.amountFromLender - repayAmount;
981
982     // amount to send to borrower
983     amounts.amountToBorrower = borrowerOwedForNewLoan - repayAmount;
984 }
```

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 (0a7551b095ee0dc4fb2a56d2267e05472128cf0e):

The `unchecked` code blocks have been introduced to all referenced calculations, addressing this exhibit in full.

OCR-02C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	OriginationController.sol:L583, L620, L654, L784, L814

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

Example:

contracts/OriginationController.sol

SOL

```sol
583 for (uint256 i = 0; i < tokens.length; ++i) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

All referenced iterator optimizations have been applied, addressing this exhibit in full.

OwnableERC721 Code Style Findings

OER-01C: Potential Optimization of Project Structure

Type	Severity	Location
Gas Optimization	Informational	OwnableERC721.sol:L44-L46

Description:

The `OwnableERC721` contract is meant to be inherited by the `AssetVault` implementation which in turn is meant to be deployed once as a `template` that the `VaultFactory` will utilize.

Example:

contracts/vault/OwnableERC721.sol

SOL

```
20  /// @dev The ERC721 token that contract owners should have ownership of.
21  address public ownershipToken;
```

Recommendation:

We advise template flow to be revised by having the `VaultFactory` deploy an instance of `AssetVault` during its **`VaultFactory::constructor`** and assign it in its `template` data entry. This will permit the `ownershipToken` of `OwnableERC721` as well as the `whitelist` member of `AssetVault` to be set as `immutable`, greatly optimizing the gas cost of the contracts across the board.

Any `immutable` variable is compatible with proxy systems given that their values are "baked in" the bytecode of the deployed instance. We advise this and the relevant exhibit of `AssetVault` to be applied in tandem if the optimizations are ultimately desired.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and assessed that the flexibility of the `VaultFactory` will be greatly diminishing if the `immutable` based optimization is applied, a trait that the Arcade XYZ team does not wish to forfeit.

As such, we consider this exhibit safely acknowledged.

PromissoryNote Code Style Findings

PNE-01C: Inefficient Renunciation of Role

Type	Severity	Location
Gas Optimization	Informational	PromissoryNote.sol:L119

Description:

The `PromissoryNote::initialize` function will invoke the `AccessControl::renounceRole` function for revoking the `ADMIN_ROLE` from the `msg.sender`, however, this is inefficient as it will perform unnecessary checks.

Example:

contracts/PromissoryNote.sol

SOL

```sol
113 function initialize(address loanCore) external onlyRole(ADMIN_ROLE) {
114     // Grant mint/burn role to loanCore
115     _setupRole(MINT_BURN_ROLE, loanCore);
116
117     // Revoke admin role from msg.sender. Since there is no ROLE_ADMIN,
118     // nobody can ever get ADMIN_ROLE again.
119     renounceRole(ADMIN_ROLE, msg.sender);
120 }
```

Recommendation:

We advise the code to instead invoke the `AccessControl::_revokeRole` function with the same input arguments, optimizing the function's gas cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team stated that the `AccessControl::_revokeRole` function is `private` within its dependency as the codebase utilizes `4.3.2` whereas the latest versions of the library specify the function as `internal`.

As such, we consider this exhibit nullified as the described optimization cannot be applied.

PunksVerifier Code Style Findings

PVR-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	PunksVerifier.sol:L69

Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```sol
69  for (uint256 i = 0; i < tokenIds.length; i++) {
```

Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The loop iterator has been optimized as advised, relocating it to a dedicated `unchecked` code block that optimizes its execution.

PVR-02C: Redundant Usage of `SafeCast`

Type	Severity	Location
Gas Optimization	Informational	PunksVerifier.sol:L23, L76

Description:

The referenced `SafeCast::toUint256` operation is performed on a `tokenId` value which is guaranteed to be non-negative by the preceding conditional. Given that `int256` contains a smaller value range in comparison to `uint256`, any non-negative `int256` value will be successfully cast safely to the `uint256` data type.

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```
76  else if (tokenId >= 0 && punks.punkIndexToAddress(tokenId.toUint256()) != vault) retur
```

Recommendation:

As such, we advise the `SafeCast` library to be omitted and the casting operation to be performed directly, optimizing the gas cost and bytecode size of the contract.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

PVR-03C: Undocumented Predicate Feature

Type	Severity	Location
Code Style	Informational	PunksVerifier.sol:L74

Description:

The referenced statement illustrates a special use-case of the `tokenId` value in the negative range which is utilized as a sanity check of a non-zero `punks` balance for the `vault`.

Example:

contracts/verifiers/PunksVerifier.sol

SOL

```sol
74  if (tokenId < 0 && punks.balanceOf(vault) == 0) return false;
```

Recommendation:

We advise this feature to be documented properly as documentation for it is absent.

Additionally, we advise the consideration of utilizing the `tokenId` value in the negative range as an indicator of the minimum balance of `punks` that the `vault` should hold, further increasing the usability of the function.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of the codebase.

RepaymentController Code Style Findings

RCR-01C: Redundant Conditional Evaluation

Type	Severity	Location
Gas Optimization	Informational	RepaymentController.sol:L153

Description:

The referenced conditional evaluation is redundant as the ensuing `if` conditional restricts the `data.state` value to `LoanLibrary\.LoanState\.Active`.

Example:

contracts/RepaymentController.sol

SOL

```sol
153 if (data.state == LoanLibrary.LoanState.DUMMY_DO_NOT_USE) revert RC_CannotDereference(
154 if (data.state != LoanLibrary.LoanState.Active) revert RC_InvalidState(data.state);
```

Recommendation:

We advise the code to be omitted, optimizing the function's gas cost.

Alleviation (7a4e1dc948e94ded7385dbb74818bcf93ecc207c):

The Arcade XYZ team evaluated this exhibit and opted to retain the current behaviour of 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.