OMNISCIA

July 15, 2023

SMART CONTRACT AUDIT REPORT

Arcade XYZ Governance


omniscia.io


info@omniscia.io


Online report: arcade-xyz-governance-system

Governance System Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
35be066e44	June 25th 2023	f52dbf82e8
a8475f2c29	July 15th 2023	1af9b0217b

Audit Overview

We were tasked with performing an audit of the Arcade XYZ codebase and in particular their governance implementation.

Their governance implementation relies on the Council implementation by Element Finance which has been copied over to the `external` subfolder of the system.

As certain contracts pertain to an external module, we considered them secure during our audit process. We validated that the latest public audit performed of the Element Finance codebase applies to the code that is located under the `external` folder.

Over the course of the audit, we identified a significant flaw in the NFT-based boosting vault that could compromise user funds inadvertently.

The `History` dependency in use throughout the new voting power modules of the system does not prohibit the current block from being queried in regard to its voting power.

Given that the current block's voting power is susceptible to flash-loan related attacks, we strongly urge the Arcade XYZ team to properly configure their `ArcadeGSCCoreVoting` system to prohibit the voting power of the current block from being queried.

As a final note, we would like to state that the contracts of the Arcade XYZ codebase utilize an upgrade-able compatible layout (via the `Storage` related dependencies), however, they seem to make use of `constructor` implementations.

Based on this fact, we considered that the contracts are meant to be deployed directly and are not going to be proxied during the audit process.

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 all non-informational exhibits have been adequately dealt with.

We advise the Arcade XYZ team to revisit the following exhibits in case they wish to apply follow-up remediations: ATD-01M, ARC-02C, ARC-01S, ARC-07C, ARC-08C, NFT-04C, NFT-02S, NFT-01C, NFT-02C, ATY-01C, RBE-01C, RBE-02C

Contracts Assessed

Files in Scope	Repository	Commit(s)
Airdrop.sol (APO)	governance	35be066e44, a8475f2c29
ArcadeToken.sol (ATN)	governance	35be066e44, a8475f2c29
ArcadeAirdrop.sol (AAP)	governance	35be066e44, a8475f2c29
ArcadeGSCVault.sol (AGS)	governance	35be066e44, a8475f2c29
ArcadeTreasury.sol (ATY)	governance	35be066e44, a8475f2c29
ARCDVestingVault.sol (ARC)	governance	35be066e44, a8475f2c29
ArcadeGSCCoreVoting.sol (AGC)	governance	35be066e44, a8475f2c29
ArcadeMerkleRewards.sol (AMR)	governance	35be066e44, a8475f2c29
ArcadeTokenDistributor.sol (ATD)	governance	35be066e44, a8475f2c29
ARCDVestingVaultStorage.sol (ARD)	governance	35be066e44, a8475f2c29
Badge.sol (BEG)	governance	35be066e44, a8475f2c29
BadgeDescriptor.sol (BDR)	governance	35be066e44, a8475f2c29
BaseVotingVault.sol (BVV)	governance	35be066e44, a8475f2c29

Files in Scope	Repository	Commit(s)
Governance.sol (GEC)	governance	35be066e44, a8475f2c29
HashedStorageReentrancyBlock.sol (HSR)	governance	35be066e44, a8475f2c29
ImmutableVestingVault.sol (IVV)	governance	35be066e44, a8475f2c29
NFTBoostVault.sol (NFT)	governance	35be066e44, a8475f2c29
NFTBoostVaultStorage.sol (NFB)	governance	35be066e44, a8475f2c29
ReputationBadge.sol (RBE)	governance	35be066e44, a8475f2c29
Token.sol (TNE)	governance	35be066e44, a8475f2c29
Treasury.sol (TYR)	governance	35be066e44, a8475f2c29

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	4	3	0	1
Informational	33	24	3	6
Minor	5	5	0	0
Medium	8	6	2	0
Major	3	3	0	0

During the audit, we filtered and validated a total of **10 findings utilizing static analysis** tools as well as identified a total of **43 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 **293 potential issues** within the codebase of which **267 were ruled out to be false positives** or negligible findings.

The remaining **26 issues** were validated and grouped and formalized into the **10 exhibits** that follow:

ID	Severity	Addressed	Title
ARC-01S	Medium	Partial	Improper Invocations of EIP-20 `transfer` / `transferFrom`
AAP-01S	Informational	Yes	Inexistent Event Emission
AAP-02S	Medium	Yes	Improper Invocation of EIP-20 `transfer`
ATD-01S	Informational	Nullified	Illegible Numeric Value Representations
ATD-02S	Medium	Yes	Improper Invocations of EIP-20 `transfer`
ATY-01S	Informational	Yes	Literal Equality of `bool` Variable
ATY-02S	Medium	Yes	Improper Invocation of EIP-20 `transfer`
BVV-01S	Minor	Yes	Inexistent Sanitization of Input Addresses
NFT-01S	Informational	Yes	Data Location Optimization
NFT-02S	Medium	Partial	Improper Invocations of EIP-20 `transfer` / `transferFrom`

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Arcade XYZ's governance system.

As the project at hand implements novel voting vaults that are meant to be compatible with the Council system of Element Finance, 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 a significant vulnerability** in one of the voting vaults within the system which could have had **severe ramifications** to its overall operation.

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 exhaustive in-line documentation as well as properly annotated `interface` declarations.

A total of **43 findings** were identified over the course of the manual review of which **15 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
ARC-01M	Minor	Yes	Inexistent Validation of Non-Zero Grant Amount
AMR-01M	Minor	Yes	Unsafe Casting Operation
ATD-01M	Unknown	Acknowledged	Inexistent Guarantee of Token Ownership
ATD-02M	Unknown	Yes	Inexplicable Capability of Re-Invocation
ATY-01M	Unknown	Nullified	Potential Failure of Disbursement
ATY-02M	Minor	Yes	Ineffectual Restriction of Approvals

ID	Severity	Addressed	Title
BVV-01M	Informational	Yes	Inexistent Sanitization of History Pruning Threshold
BVV-02M	Major	Yes	Incorrect Storage Integration
HSR-01M	Major	Yes	Incorrect Implementation of Reentrancy Guard
NFT-01M	Minor	Yes	Inexistent Validation of Existing Registration
NFT-02M	Medium	Yes	Inexistent Sanitization of Delegated Member
NFT-03M	Medium	Yes	Inexistent Validation of Delegated Member
NFT-04M	Major	Yes	Improper Validation of Registration Existence
RBE-01M	Unknown	Nullified	Potential Failure of Disbursement
RBE-02M	Medium	Yes	Inexistent Multiplication of Mint Price

Code Style

During the manual portion of the audit, we identified **28 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
ARC-01C	Informational	Yes	Generic Typographic Mistakes
ARC-02C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
ARC-03C	Informational	Yes	Redundant Calculation of Current Voting Power
ARC-04C	Informational	Yes	Redundant Conditional
ARC-05C	Informational	Yes	Redundant Conditional Clause
ARC-06C	Informational	Yes	Redundant Load of Storage
ARC-07C	Informational	Partial	Redundant Parenthesis Statements
ARC-08C	Informational	Partial	Redundant Utilizations of Inverses
ARC-09C	Informational	Nullified	Repetitive Calculations of Storage Offsets
ARC-10C	Informational	Yes	Suboptimal Struct Declaration Style
ARD-01C	Informational	Yes	Inefficient Variable Type

ID	Severity	Addressed	Title
AAP-01C	Informational	Yes	Inefficient Assignment of Owner
AMR-01C	Informational	Nullified	Variable Mutability Specifiers (Immutable)
ATD-01C	Informational	Yes	Generic Typographic Mistakes
ATY-01C	Informational	Acknowledged	Loop Iterator Optimization
BDR-01C	Informational	Yes	Potentially Redundant Indexed Argument
BVV-01C	Informational	Yes	Generic Typographic Mistake
BVV-02C	Informational	Yes	Redundant Parenthesis Statement
HSR-01C	Informational	Yes	Repetitive Value Literals
NFT-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
NFT-02C	Informational	Acknowledged	Loop Iterator Optimization
NFT-03C	Informational	Yes	Redundant Parenthesis Statements
NFT-04C	Informational	Partial	Repetitive Calculations of Storage Offsets
NFT-05C	Informational	Yes	Suboptimal Struct Declaration Style
NFB-01C	Informational	Yes	Dynamic Hash Evaluations
NFB-02C	Informational	Yes	Generic Typographic Mistakes
RBE-01C	Informational	No	Inefficient `mapping` Lookups

ID	Severity	Addressed	Title
RBE-02C	Informational	Acknowledged	Loop Iterator Optimization

ARCDVestingVault Static Analysis Findings

ARC-01S: Improper Invocations of EIP-20 `transfer` / `transferFrom`

Type	Severity	Location
Standard Conformity	Medium	ARCDVestingVault.sol:L164, L169, L194, L210, L245

Description:

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

Impact:

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

Example:

contracts/ARCDVestingVault.sol

SOL

```
164 token.transfer(who, withdrawable);
```

Recommendation:

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

Alleviation:

All but **the EIP-20 `transferFrom` invocation** have been updated, rendering this exhibit partially alleviated.

ArcadeAirdrop Static Analysis Findings

AAP-01S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	ArcadeAirdrop.sol:L67-L69

Description:

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

Example:

contracts/token/ArcadeAirdrop.sol

SOL

```sol
function setMerkleRoot(bytes32 _merkleRoot) external onlyOwner {
    rewardsRoot = _merkleRoot;
}
```

Recommendation:

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

Alleviation:

A `SetMerkleRoot` event has been introduced to the codebase and is correspondingly emitted during an **`ArcadeAirdrop::setMerkleRoot`** operation, addressing this exhibit in full.

AAP-02S: Improper Invocation of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	ArcadeAirdrop.sol:L59

Description:

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

Impact:

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

Example:

contracts/token/ArcadeAirdrop.sol

SOL

```sol
59  token.transfer(destination, unclaimed);
```

Recommendation:

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

Alleviation:

The referenced **EIP-20** `transfer` invocation has been replaced by its `SafeERC20::safeTransfer` counterpart, ensuring the code is compatible with all types of **EIP-20** tokens.

ArcadeTokenDistributor Static Analysis Findings

ATD-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	ArcadeTokenDistributor.sol:L30, L35, L40, L45, L50, L55

Description:

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

Example:

contracts/token/ArcadeTokenDistributor.sol

SOL

```sol
30  uint256 public constant treasuryAmount = 25_500_000 ether;
```

Recommendation:

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

Alleviation:

The Arcade XYZ team has retained the original code style in the codebase. Our original advice was to retain the underscore separator (`_`) solely where the decimal point would be (i.e. `25_000_000` would become `25_000000`), however, the current style is a matter of perspective and as such we consider this exhibit nullified.

ATD-02S: Improper Invocations of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	ArcadeTokenDistributor.sol:L77, L93, L109, L125, L142, L159

Description:

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

Impact:

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

Example:

contracts/token/ArcadeTokenDistributor.sol

SOL

```
77  arcadeToken.transfer(_treasury, treasuryAmount);
```

Recommendation:

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

Alleviation:

All **EIP-20** `transfer` invocations have been replaced by their `SafeERC20::safeTransfer` counterpart, ensuring that the code is compatible with all types of **EIP-20** tokens.

ArcadeTreasury Static Analysis Findings

ATY-01S: Literal Equality of `bool` Variable

Type	Severity	Location
Gas Optimization	Informational	ArcadeTreasury.sol:L335

Description:

The linked `bool` comparison is performed between a variable and a `bool` literal.

Example:

contracts/ArcadeTreasury.sol

SOL

```sol
335 if (success == false) revert T_CallFailed();
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its negated (`!`) or original form.

Alleviation:

The referenced statement now properly utilizes the `bool` variables value directly in its negated format in place of an equality comparison with `false`.

ATY-02S: Improper Invocation of EIP-20 `transfer`

Type	Severity	Location
Standard Conformity	Medium	ArcadeTreasury.sol:L361

Description:

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

Impact:

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

Example:

contracts/ArcadeTreasury.sol

SOL

```sol
361 IERC20(token).transfer(destination, amount);
```

Recommendation:

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

Alleviation:

The referenced **EIP-20** `transfer` invocation has been replaced by its `SafeERC20::safeTransfer` counterpart, ensuring the code is compatible with all types of **EIP-20** tokens.

BaseVotingVault Static Analysis Findings

BVV-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	BaseVotingVault.sol:L67-L69, L77-L79

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/BaseVotingVault.sol

SOL

```sol
67  function setTimelock(address timelock_) external onlyTimelock {
68      Storage.set(Storage.addressPtr("timelock"), timelock_);
69  }
```

Recommendation:

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

Alleviation:

Both referenced functions properly validate their input `address` arguments via a non-zero check, preventing the contract from being misconfigured and alleviating this exhibit.

NFTBoostVault Static Analysis Findings

NFT-01S: Data Location Optimization

Type	Severity	Location
Gas Optimization	Informational	NFTBoostVault.sol:L344

Description:

The linked input argument is set as `memory` in an `external` function.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
344 function updateVotingPower(address[] memory userAddresses) public override {
```

Recommendation:

We advise it to be set as `calldata` optimizing its read-access gas cost.

Alleviation:

The data location of the referenced argument has been converted from `memory` to `calldata`, optimizing its read-access gas cost.

NFT-02S: Improper Invocations of EIP-20 `transfer` / `transferFrom`

Type	Severity	Location
Standard Conformity	Medium	NFTBoostVault.sol:L245, L624

Description:

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

Impact:

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

Example:

contracts/NFTBoostVault.sol

SOL

```sol
245 token.transfer(msg.sender, amount);
```

Recommendation:

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

Alleviation:

While the **EIP-20** `transfer` invocation was corrected, **the EIP-20** `transferFrom` **invocation** remains unaddressed causing this exhibit to be partially alleviated.

ARCDVestingVault Manual Review Findings

ARC-01M: Inexistent Validation of Non-Zero Grant Amount

Type	Severity	Location
Input Sanitization	Minor	ARCDVestingVault.sol:L92

Description:

The `ARCDVestingVault::addGrantAndDelegate` function will not validate that a non-zero amount is being vested.

Impact:

A zero-value grant will create "dummy" entries in the contract and lead to a waste of gas.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
function addGrantAndDelegate(
    address who,
    uint128 amount,
    uint128 cliffAmount,
    uint128 startTime,
    uint128 expiration,
    uint128 cliff,
    address delegatee
) external onlyManager {
```

Recommendation:

We advise the code to ensure that a non-zero `amount` has been specified.

Alleviation:

An `if-revert` pattern was introduced preventing an `amount` of `0` from being specified and thus alleviating this exhibit.

ArcadeMerkleRewards Manual Review Findings

AMR-01M: Unsafe Casting Operation

Type	Severity	Location
Mathematical Operations	Minor	ArcadeMerkleRewards.sol:L88

Description:

The `ArcadeMerkleRewards::claimAndDelegate` function permits values that exceed the `type(uint128).max` boundary to be supplied as the `totalGrant` claimed and validated by the Merkle Proof, however, such values would cause a casting overflow to occur thus yielding a different grant value than expected to the eligible user.

Impact:

While the impact of this exhibit is significant, it would solely arise from misconfiguration of the Merkle Root. As such, we consider this exhibit to be of minor severity.

Example:

contracts/libraries/ArcadeMerkleRewards.sol

SOL

```sol
77  function claimAndDelegate(address delegate, uint256 totalGrant, bytes32[] calldata mer
78      // must be before the expiration time
79      if (block.timestamp > expiration) revert AA_ClaimingExpired();
80      // no delegating to zero address
81      if (delegate == address(0)) revert AA_ZeroAddress();
82      // validate the withdraw
83      _validateWithdraw(totalGrant, merkleProof);
84
85      // approve the voting vault to transfer tokens
86      token.approve(address(votingVault), totalGrant);
87      // deposit tokens in voting vault for this msg.sender and delegate
88      votingVault.airdropReceive(msg.sender, uint128(totalGrant), delegate);
89  }
```

Recommendation:

We advise the code of `ArcadeMerkleRewards::claimAndDelegate` to ensure that `totalGrant` is less-than-or-equal-to the value of `type(uint128).max`, preventing casting overflows from occurring.

Alleviation:

The `totalGrant` argument of the `ArcadeMerkleRewards::claimAndDelegate` function was downgraded to the `uint128` accuracy, causing the code to no longer perform a down-casting operation and thus alleviating this exhibit.

ArcadeTokenDistributor Manual Review Findings

ATD-01M: Inexistent Guarantee of Token Ownership

Type	Severity	Location
Centralization Concern	Unknown	ArcadeTokenDistributor.sol:L171-L175

Description:

The `ArcadeTokenDistributor` contract does not guarantee it possesses the necessary amount of funds to perform its distribution when its `arcadeToken` is set.

Example:

contracts/token/ArcadeTokenDistributor.sol

SOL

```sol
166 /**
167  * @notice Sets the Arcade Token contract address, to be used in token distribution.
168  *
169  * @param _arcadeToken             The Arcade Token contract.
170  */
171 function setToken(IArcadeToken _arcadeToken) external onlyOwner {
172     if (address(_arcadeToken) == address(0)) revert AT_ZeroAddress();
173
174     arcadeToken = _arcadeToken;
175 }
```

Recommendation:

We advise either a balance measurement of a transfer operation from the `owner` to be performed during the `ArcadeTokenDistributor::setToken` call, ensuring that the contract is sufficiently funded to perform its expected distribution.

Alleviation:

The Arcade XYZ team evaluated this exhibit but opted not to apply a remediation for it in the current iteration of the codebase.

ATD-02M: Inexplicable Capability of Re-Invocation

Type	Severity	Location
Centralization Concern	Unknown	ArcadeTokenDistributor.sol:L171-L175

Description:

The `ArcadeTokenDistributor::setToken` function can be invoked an arbitrary number of times, permitting the `arcadeToken` to be re-set arbitrarily.

Example:

contracts/token/ArcadeTokenDistributor.sol

SOL

```sol
166 /**
167  * @notice Sets the Arcade Token contract address, to be used in token distribution.
168  *
169  * @param _arcadeToken             The Arcade Token contract.
170  */
171 function setToken(IArcadeToken _arcadeToken) external onlyOwner {
172     if (address(_arcadeToken) == address(0)) revert AT_ZeroAddress();
173
174     arcadeToken = _arcadeToken;
175 }
```

Recommendation:

We advise the `arcadeToken` entry to be set only once as otherwise the distribution flow can be compromised and distribute a different token than expected.

Alleviation:

The **`ArcadeTokenDistributor::setToken`** function was updated to permit the `arcadeToken` to be updated solely once, alleviating this exhibit.

ArcadeTreasury Manual Review Findings

ATY-01M: Potential Failure of Disbursement

Type	Severity	Location
Language Specific	Unknown	ArcadeTreasury.sol:L359

Description:

The `ArcadeTreasury::_spend` function will attempt to perform a native fund transfer via the low-level `transfer` function exposed by the `address payable` data type.

This function assigns a fixed gas stipend to the transaction, causing native transfers to fail in certain L2 protocols like ZkSync.

Impact:

The severity of this exhibit will be adjusted based on Arcade XYZ's assessment of its applicability.

Example:

contracts/ArcadeTreasury.sol

SOL

```sol
/**
 * @notice helper function to send tokens from the treasury. This function is used by
 *         transfer functions to send tokens to their destinations.
 *
 * @param token             address of the token to spend
 * @param amount            amount of tokens to spend
 * @param destination       recipient of the transfer
 * @param limit             max tokens that can be spent/approved in a single block fo
 */
function _spend(address token, uint256 amount, address destination, uint256 limit) int
    // check that after processing this we will not have spent more than the block lim
    uint256 spentThisBlock = blockExpenditure[block.number];
    if (amount + spentThisBlock > limit) revert T_BlockSpendLimit();
    blockExpenditure[block.number] = amount + spentThisBlock;

    // transfer tokens
    if (address(token) == ETH_CONSTANT) {
        // will out-of-gas revert if recipient is a contract with logic inside receive
        payable(destination).transfer(amount);
    } else {
        IERC20(token).transfer(destination, amount);
    }

    emit TreasuryTransfer(token, destination, amount);
}
```

Recommendation:

We advise the blockchains the contract will be deployed in to be evaluated and a different fund distribution flow to be performed instead, potentially following a "pull" pattern.

Alleviation:

The Arcade XYZ team evaluated this exhibit and has specified that the intended recipients are expected to be able to receive the intended funds via the fixed gas stipend. As such, we consider this exhibit nullified given that Arcade XYZ has assessed that their native transfers will not fail.

ATY-02M: Ineffectual Restriction of Approvals

Type	Severity	Location
Logical Fault	Minor	ArcadeTreasury.sol:L279, L306

Description:

The `ArcadeTreasury::setGSCAllowance` function is meant to apply a limitation on the `newAllowance` that ensures the `small` spend threshold of the `token` is honored, however, this logical constraint can be invalidated by setting an allowance before reducing the `small` threshold of the `token`.

Impact:

The allowance of GSC does not necessarily conform to the spend threshold of the token despite the `ArcadeTreasury::setGSCAllowance` function attempting to enforce this limitation.

Example:

contracts/ArcadeTreasury.sol

SOL

```sol
/**
 * @notice function to set the spend/approve thresholds for a token. This function is
 *         callable by the contract admin.
 *
 * @param token             address of the token to set the thresholds for
 * @param thresholds        struct containing the thresholds to set
 */
function setThreshold(address token, SpendThreshold memory thresholds) external onlyRo
    // verify that the token is not the zero address
    if (token == address(0)) revert T_ZeroAddress();
    // verify small threshold is not zero
    if (thresholds.small == 0) revert T_ZeroAmount();

    // verify thresholds are ascending from small to large
    if (thresholds.large < thresholds.medium || thresholds.medium < thresholds.small)
        revert T_ThresholdsNotAscending();
    }

    // Overwrite the spend limits for specified token
    spendThresholds[token] = thresholds;

    emit SpendThresholdsUpdated(token, thresholds);
}

/**
 * @notice function to set the GSC allowance for a token. This function is only callab
 *         by the contract admin. The new allowance must be less than or equal to the
 *         spend threshold for that specific token. There is a cool down period of 7 d
 *         after this function has been called where it cannot be called again. Once t
 *         period is over the allowance can be updated by the admin again.
 *
 * @param token             address of the token to set the allowance for
 * @param newAllowance      new allowance amount to set
 */
function setGSCAllowance(address token, uint256 newAllowance) external onlyRole(ADMIN_
    if (token == address(0)) revert T_ZeroAddress();
    if (newAllowance == 0) revert T_ZeroAmount();

    // enforce cool down period
    if (uint48(block.timestamp) < lastAllowanceSet[token] + SET_ALLOWANCE_COOL_DOWN) {
        revert T_CoolDownPeriod(block.timestamp, lastAllowanceSet[token] + SET_ALLOWAN
    }

    uint256 spendLimit = spendThresholds[token].small;

    // new limit cannot be more than the small threshold
```

```
305     // new limit cannot be more than the small threshold
306     if (newAllowance > spendLimit) {
307         revert T_InvalidAllowance(newAllowance, spendLimit);
308     }
309
310     // update allowance state
311     lastAllowanceSet[token] = uint48(block.timestamp);
312     gscAllowance[token] = newAllowance;
313
314     emit GSCAllowanceUpdated(token, newAllowance);
315 }
```

Recommendation:

We advise the code of **`ArcadeTreasury::setThreshold`** to either ensure the new `thresholds.small` value is greater-than the value of `gscAllowance[token]`, or the `gscAllowance[token]` to be set to the new `thresholds.small` if it exceeds it.

Based on the expected usage of **`ArcadeTreasury::setThreshold`**, we advise the latter to be performed.

Alleviation:

The Arcade XYZ team applied our latter recommendation, ensuring that if the `gscAllowance` is greater than the new `small` threshold, it is overwritten to it. As such, we consider this exhibit fully alleviated.

BaseVotingVault Manual Review Findings

BVV-01M: Inexistent Sanitization of History Pruning Threshold

Type	Severity	Location
Input Sanitization	Informational	BaseVotingVault.sol:L52-L57

Description:

The `BaseVotingVault::constructor` does not sanitize the `_staleBlockLag` that will be utilized for pruning a user's history beyond a certain block in the past.

Impact:

While a misconfigured contract would lead to subtraction underflows in the relevant pruning functions, the contract can simply be re-deployed. As such, we classify this vulnerability as informational in nature.

Example:

contracts/BaseVotingVault.sol

SOL

```sol
52  constructor(IERC20 _token, uint256 _staleBlockLag) {
53      if (address(_token) == address(0)) revert BVV_ZeroAddress();
54
55      token = _token;
56      staleBlockLag = _staleBlockLag;
57  }
```

Recommendation:

We advise the code to ensure that the value is less than the current `block.number` and preferably within a gas-sensible range of values.

Alleviation:

The `_staleBlockLag` value is now properly validated as being less-than the current `block.number`, alleviating this exhibit and ensuring the contract's instantiation is properly sanitized.

BVV-02M: Incorrect Storage Integration

Type	Severity	Location
Logical Fault	Major	BaseVotingVault.sol:L121-L123, L132-L134, L154-L156, L165-L167

Description:

The `Storage` implementation of the Council implementation is meant to be interfaced by storage-mutating functions. It is marked as `pure` because the calculation of the storage offset itself is "pure", however, accessing its data is not and results in an `SLOAD` operation.

Impact:

The **`BaseVotingVault::_timelock`** and **`BaseVotingVault::_manager`** functions will perform an implicit `SLOAD` operation as they convert the return value of `Storage::addressPtr` from `storage` to `memory`. This is undetected due to the usage of low-level `assembly` and will cause these functions to be invoked by a `STATICCALL` instruction incorrectly, failing in a production environment.

Example:

contracts/BaseVotingVault.sol

SOL

```sol
/**
 * @notice A function to access the storage of the timelock address.
 *
 * @dev The timelock can access all functions with the onlyTimelock modifier.
 *
 * @return timelock                  A struct containing the timelock address.
 */
function _timelock() internal pure returns (Storage.Address memory) {
    return Storage.addressPtr("timelock");
}

/**
 * @notice A function to access the storage of the manager address.
 *
 * @dev The manager can access all functions with the onlyManager modifier.
 *
 * @return manager                   A struct containing the manager address.
 */
function _manager() internal pure returns (Storage.Address memory) {
    return Storage.addressPtr("manager");
}
```

Recommendation:

We advise the functions to remove their `pure` attribute and the data types they yield to be set as `storage`.

Alleviation:

The code was updated per our recommendation, ensuring that the data types and corresponding pointers are correctly loaded and yielded without error.

HashedStorageReentrancyBlock Manual Review Findings

HSR-01M: Incorrect Implementation of Reentrancy Guard

Type	Severity	Location
Logical Fault	Major	HashedStorageReentrancyBlock.sol:L33

Description:

The `HashedStorageReentrancyBlock::nonReentrant` modifier is meant to apply a reentrancy guard, however, the data structure it utilizes for setting the reentrancy flags is specified as `memory` instead of `storage`.

As such, the **current reentrancy guard is ineffectual** and thus compromises the security of the system.

Impact:

The current reentrancy guard is ineffectual, compromising the security assumptions of the system and allowing any form of reentrancy to occur.

Example:

contracts/libraries/HashedStorageReentrancyBlock.sol

SOL

```sol
abstract contract HashedStorageReentrancyBlock {
    // ========================================== HELPERS ==========================

    /**
     * @dev Returns the storage pointer to the entered state variable.
     *
     * @return Storage              pointer to the entered state variable.
     */
    function _entered() internal pure returns (Storage.Uint256 memory) {
        return Storage.uint256Ptr("entered");
    }

    // ======================================== MODIFIERS ==========================

    /**
     * @dev Re-entrancy guard modifier using hashed storage.
     */
    modifier nonReentrant() {
        Storage.Uint256 memory entered = _entered();
        // Check the state variable before the call is entered
        require(entered.data == 1, "REENTRANCY");

        // Store that the function has been entered
        entered.data = 2;

        // Run the function code
        _;

        // Clear the state
        entered.data = 1;
    }
}
```

Recommendation:

We advise the `Storage.Uint256` pointer that is yielded by **`HashedStorageReentrancyBlock::_entered`** and utilized by **`HashedStorageReentrancyBlock::nonReentrant`** to be set as `storage`, ensuring that the values that are set and read at each instance are properly performed at the contract-level rather than the local function context.

Alleviation:

The code was updated to properly affect the `storage` space of the contract it is inherited by, rendering the re-entrancy guard effective and thus alleviating this exhibit.

NFTBoostVault Manual Review Findings

NFT-01M: Inexistent Validation of Existing Registration

Type	Severity	Location
Logical Fault	Minor	NFTBoostVault.sol:L304-L332

Description:

The `NFTBoostVault::updateNft` flow permits an NFT to be associated with a user's registration even if they haven't actually registered yet.

Impact:

If an NFT is associated with a user prior to their registration, they will be unable to register until they have withdrawn it via the `NFTBoostVault::withdrawNft` function.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
/**
 * @notice A function that allows a user's to change the ERC1155 nft they are using fo
 *         accessing a voting power multiplier. Or if the users does not have a NFT
 *         registered, they can register one and their voting power will be updated.
 *
 * @param newTokenAddress           Address of the new ERC1155 token the user wants t
 * @param newTokenId                Id of the new ERC1155 token the user wants to use
 */
function updateNft(uint128 newTokenId, address newTokenAddress) external override nonR
    if (newTokenAddress == address(0) || newTokenId == 0) revert NBV_InvalidNft(newTok

    if (IERC1155(newTokenAddress).balanceOf(msg.sender, newTokenId) == 0) revert NBV_D

    NFTBoostVaultStorage.Registration storage registration = _getRegistrations()[msg.s

    if (registration.tokenAddress != address(0) && registration.tokenId != 0) {
        // withdraw the current ERC1155 from the registration
        withdrawNft();
    }

    // set the new ERC1155 values in the registration
    registration.tokenAddress = newTokenAddress;
    registration.tokenId = newTokenId;

    _lockNft(msg.sender, newTokenAddress, newTokenId, 1);

    // update the delegatee's voting power based on new ERC1155 nft's multiplier
    _syncVotingPower(msg.sender, registration);
}
```

Recommendation:

We advise the code to properly validate that a registration has occurred by ensuring that the `delegatee` of the registration is non-zero akin to the `NFTBoostVault::addTokens` function.

Alleviation:

The `NFTBoostVault::updateNft` function properly validates that a `delegatee` has been specified in the user's registration thus alleviating this exhibit.

NFT-02M: Inexistent Sanitization of Delegated Member

Type	Severity	Location
Input Sanitization	Medium	NFTBoostVault.sol:L175

Description:

The `NFTBoostVault::delegate` function does not sanitize its input address, permitting the zero address to be delegated to. This is incorrect as functions like `NFTBoostVault::addTokens` rely on its presence as an indicator of a registration's existence.

Impact:

It is possible to cause a registration to "cease existing" by setting its `delegatee` to the zero-address, a significantly undesirable trait of the system.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
function delegate(address to) external override {
    NFTBoostVaultStorage.Registration storage registration = _getRegistrations()[msg.s

    // If to address is already the delegate, don't send the tx
    if (to == registration.delegatee) revert NBV_AlreadyDelegated();

    History.HistoricalBalances memory votingPower = _votingPower();
    uint256 oldDelegateeVotes = votingPower.loadTop(registration.delegatee);

    // Remove voting power from old delegatee and emit event
    votingPower.push(registration.delegatee, oldDelegateeVotes - registration.latestVo
    emit VoteChange(msg.sender, registration.delegatee, -1 * int256(uint256(registrati

    // Note - It is important that this is loaded here and not before the previous sta
    // to == registration.delegatee and re-delegation was allowed we could be working
    uint256 newDelegateeVotes = votingPower.loadTop(to);
    // return the current voting power of the Registration. Varies based on the multip
    // user's ERC1155 token at the time of txn
    uint256 addedVotingPower = _currentVotingPower(registration);

    // add voting power to the target delegatee and emit event
    votingPower.push(to, newDelegateeVotes + addedVotingPower);

    // update registration properties
    registration.latestVotingPower = uint128(addedVotingPower);
    registration.delegatee = to;

    emit VoteChange(msg.sender, to, int256(addedVotingPower));
}
```

Recommendation:

We advise the `NFTBoostVault::delegate` function to properly sanitize its input address, disallowing the zero-address from being delegated to.

Alleviation:

The `NFTBoostVault::delegate` function properly ensures that the `to` address being delegated is not the zero-address, protecting the zero-address's "special" role.

NFT-03M: Inexistent Validation of Delegated Member

Type	Severity	Location
Logical Fault	Medium	NFTBoostVault.sol:L139, L151-L161

Description:

The `NFTBoostVault::airdropReceive` workflow will not validate that the `delegatee` supplied is the same as the existing registration of the user, causing a discrepancy whereby the `NFTBoostVault::airdropReceive` function will not actually delegate the voting power to the `delegatee`.

Impact:

An `ArcadeMerkleRewards::claimAndDelegate` transaction may not actually delegate the claimed funds to the `delegatee` supplied under the condition described by this exhibit which is a significant flaw in the function's expected behavior.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
function airdropReceive(
    address user,
    uint128 amount,
    address delegatee
) external override onlyAirdrop nonReentrant {
    if (amount == 0) revert NBV_ZeroAmount();
    if (user == address(0)) revert NBV_ZeroAddress();

    // load the registration
    NFTBoostVaultStorage.Registration storage registration = _getRegistrations()[user]

    // if user is not already registered, register them
    // else just update their registration
    if (registration.delegatee == address(0)) {
        _registerAndDelegate(user, amount, 0, address(0), delegatee);
    } else {
        // get this contract's balance
        Storage.Uint256 storage balance = _balance();
        // update contract balance
        balance.data += amount;

        // update registration amount
        registration.amount += amount;
        // update the delegatee's voting power
        _syncVotingPower(msg.sender, registration);
    }

    // transfer user ERC20 amount only into this contract
    _lockTokens(msg.sender, uint256(amount), address(0), 0, 0);
}
```

Recommendation:

We advise the code to mandate that the `delegatee` of the `user` is equivalent to their original registration's `delegatee`.

Alternatively, we advise the `delegatee` to be changed to the new one by adjusting the `registration` of the existing `user`.

Alleviation:

The code was updated to ensure that the input `delegatee` is equal to an existing registration's `delegatee` should it exist, preventing the code from ignoring this discrepancy and ensuring that the `delegatee` is properly specified and remains the same between airdrop receipts and registration operations.

NFT-04M: Improper Validation of Registration Existence

Type	Severity	Location
Logical Fault	Major	NFTBoostVault.sol:L120, L150, L479

Description:

The `NFTBoostVault::_registerAndDelegate` function will assess whether a user has already registered by validating their `tokenId` and `tokenAddress` entries, however, they are explicitly set to zero during a registration via `ArcadeMerkleRewards` and can also be set to zero during a normal `NFTBoostVault::addNftAndDelegate` registration.

Impact:

It is currently possible for a user to "re-register", erasing their previous registration and thus losing access to the locked `amount` they had supplied.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
function _registerAndDelegate(
    address user,
    uint128 amount,
    uint128 tokenId,
    address tokenAddress,
    address delegatee
) internal {
    uint256 multiplier = 1e18;

    // confirm that the user is a holder of the tokenId and that a multiplier is set f
    if (tokenAddress != address(0) && tokenId != 0) {
        if (IERC1155(tokenAddress).balanceOf(user, tokenId) == 0) revert NBV_DoesNotOw

        multiplier = getMultiplier(tokenAddress, tokenId);

        if (multiplier == 0) revert NBV_NoMultiplierSet();
    }

    // load this contract's balance storage
    Storage.Uint256 storage balance = _balance();

    // load the registration
    NFTBoostVaultStorage.Registration storage registration = _getRegistrations()[user]

    // If the token id and token address is not zero, revert because the Registration
    // is already initialized. Only one Registration per user
    if (registration.tokenId != 0 && registration.tokenAddress != address(0)) revert N

    // load the delegate. Defaults to the registration owner
    delegatee = delegatee == address(0) ? user : delegatee;

    // calculate the voting power provided by this registration
    uint128 newVotingPower = (amount * uint128(multiplier)) / MULTIPLIER_DENOMINATOR;

    // set the new registration
    _getRegistrations()[user] = NFTBoostVaultStorage.Registration(
        amount,
        newVotingPower,
        0,
        tokenId,
        tokenAddress,
        delegatee
    );

    // update this contract's balance
```

```
498     balance.data += amount;
499
500     _grantVotingPower(delegatee, newVotingPower);
501
502     emit VoteChange(user, registration.delegatee, int256(uint256(newVotingPower)));
503 }
```

Recommendation:

We advise the code to validate a registration's presence by assessing whether the `delegatee` is non-zero, similarly to the `NFTBoostVault::addTokens` function.

Alleviation:

The code of `NFTBoostVault::_registerAndDelegate` now properly validates a registration's existence via the `delegatee` member rather than the `tokenId` and `tokenAddress` members, alleviating this exhibit in full.

ReputationBadge Manual Review Findings

RBE-01M: Potential Failure of Disbursement

Type	Severity	Location
Language Specific	Unknown	ReputationBadge.sol:L171

Description:

The `ReputationBadge::withdrawFees` function will attempt to perform a native fund transfer via the low-level `transfer` function exposed by the `address payable` data type.

This function assigns a fixed gas stipend to the transaction, causing native transfers to fail in certain L2 protocols like ZkSync.

Impact:

The severity of this exhibit will be adjusted based on Arcade XYZ's assessment of its applicability.

Example:

contracts/nft/ReputationBadge.sol

SOL

```sol
/**
 * @notice Withdraw all ETH fees from the contract.
 *
 * @param recipient        The address to withdraw the fees to.
 */
function withdrawFees(address recipient) external onlyRole(BADGE_MANAGER_ROLE) {
    if (recipient == address(0)) revert RB_ZeroAddress();

    // get contract balance
    uint256 balance = address(this).balance;

    // transfer balance to recipient
    // will out-of-gas revert if recipient is a contract with logic inside receive()
    payable(recipient).transfer(balance);

    emit FeesWithdrawn(recipient, balance);
}
```

Recommendation:

We advise the blockchains the contract will be deployed in to be evaluated and a different fund distribution flow to be performed instead, potentially following a "pull" pattern.

Alleviation:

The Arcade XYZ team evaluated this exhibit and has specified that the intended recipients are expected to be able to receive the intended funds via the fixed gas stipend. As such, we consider this exhibit nullified given that Arcade XYZ has assessed that their native transfers will not fail.

RBE-02M: Inexistent Multiplication of Mint Price

Type	Severity	Location
Logical Fault	Medium	ReputationBadge.sol:L109

Description:

A `ReputationBadge::mint` operation will incorrectly apply the same `mintPrice` regardless of the `amount` of `tokenId` that will be minted.

Impact:

Depending on the business requirements of Arcade XYZ, the system either unfairly charges users for multiple mints of the same `tokenId` or incorrectly charges a fixed price regardless of the `amount` of tokens minted.

Example:

contracts/nft/ReputationBadge.sol

SOL

```sol
function mint(
    address recipient,
    uint256 tokenId,
    uint256 amount,
    uint256 totalClaimable,
    bytes32[] calldata merkleProof
) external payable {
    uint256 mintPrice = mintPrices[tokenId];
    uint48 claimExpiration = claimExpirations[tokenId];

    if (block.timestamp > claimExpiration) revert RB_ClaimingExpired(claimExpiration,
    if (msg.value < mintPrice) revert RB_InvalidMintFee(mintPrice, msg.value);
    if (!_verifyClaim(recipient, tokenId, totalClaimable, merkleProof)) revert RB_Inva
    if (amountClaimed[recipient][tokenId] + amount > totalClaimable) {
        revert RB_InvalidClaimAmount(amount, totalClaimable);
    }

    // increment amount claimed
    amountClaimed[recipient][tokenId] += amount;

    // mint to recipient
    _mint(recipient, tokenId, amount, "");
}
```

Recommendation:

We advise the function to either apply the payment only once for the `tokenId` or the price to be proportionate to the `amount` that will be minted, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The `mintPrice` assignment now performs a multiplication by the `amount` value, ensuring that the price a mint operation should pay is proportionate to the number of tokens being minted.

ARCDVestingVault Code Style Findings

ARC-01C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	ARCDVestingVault.sol:L85, L86-L87

Description:

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

Example:

contracts/ARCDVestingVault.sol

SOL

```
85  * @param expiration                Timestamp when the grant ends (all tokens count as
```

Recommendation:

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

Alleviation:

The referenced typographic mistakes have been corrected.

ARC-02C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	ARCDVestingVault.sol:L138, L208, L323-L325

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/ARCDVestingVault.sol

SOL

```sol
Storage.Uint256 storage unassigned = _unassigned();
if (unassigned.data < amount) revert AVV_InsufficientBalance(unassigned.data);

// load the grant
ARCDVestingVaultStorage.Grant storage grant = _grants()[who];

// if this address already has a grant, a different address must be provided
// topping up or editing active grants is not supported.
if (grant.allocation != 0) revert AVV_HasGrant();

// load the delegate. Defaults to the grant owner
delegatee = delegatee == address(0) ? who : delegatee;

// calculate the voting power. Assumes all voting power is initially locked.
uint128 newVotingPower = amount;

// set the new grant
_grants()[who] = ARCDVestingVaultStorage.Grant(
    amount,
    cliffAmount,
    0,
    startTime,
    expiration,
    cliff,
    newVotingPower,
    delegatee
);

// update the amount of unassigned tokens
unassigned.data -= amount;
```

Recommendation:

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

Alleviation:

The Arcade XYZ team evaluated this exhibit and while acknowledging the gas optimization it brings, they wish to retain checked arithmetic throughout the codebase in favour of code clarity as well as safety.

As such, we consider this exhibit acknowledged.

ARC-03C: Redundant Calculation of Current Voting Power

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L259

Description:

The result of the `ARCDVestingVault::_currentVotingPower` function will solely be different than the `grant.latestVotingPower` value when the `grant.withdrawn` has increased.

The `withdrawn` value of a grant is solely increased in the `ARCDVestingVault::claim` and `ARCDVestingVault::revokeGrant` functions, both of which sync the value of `latestVotingPower` via the `ARCDVestingVault::_syncVotingPower`. As such, its calculation when delegating is redundant.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
function delegate(address to) external {
    ARCDVestingVaultStorage.Grant storage grant = _grants()[msg.sender];
    if (to == grant.delegatee) revert AVV_AlreadyDelegated();

    History.HistoricalBalances memory votingPower = _votingPower();
    uint256 oldDelegateeVotes = votingPower.loadTop(grant.delegatee);
    uint256 newVotingPower = _currentVotingPower(grant);

    // Remove old delegatee's voting power and emit event
    votingPower.push(grant.delegatee, oldDelegateeVotes - grant.latestVotingPower);
    emit VoteChange(grant.delegatee, msg.sender, -1 * int256(uint256(grant.latestVotin

    // Note - It is important that this is loaded here and not before the previous sta
    // to == grant.delegatee and re-delegation was allowed we could be working with ou
    uint256 newDelegateeVotes = votingPower.loadTop(to);

    // add voting power to the target delegatee and emit event
    votingPower.push(to, newDelegateeVotes + newVotingPower);

    // update grant info
    grant.latestVotingPower = uint128(newVotingPower);
    grant.delegatee = to;

    emit VoteChange(to, msg.sender, int256(newVotingPower));
}
```

Recommendation:

We advise the value of `grant.latestVotingPower` to be utilized for both voting power updates in the delegation workflow, optimizing the function's gas cost.

Alleviation:

Our recommendation has been applied, utilizing the `grant.latestVotingPower` value over calculating and retaining the voting power of the grant in memory.

ARC-04C: Redundant Conditional

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L322

Description:

The referenced conditional is ineffectual as the `if` block at the beginning of the `ARCDVestingVault::_getWithdrawableAmount` guarantees that `block.number >= grant.cliff`.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
312 function _getWithdrawableAmount(ARCDVestingVaultStorage.Grant memory grant) internal v
313     // if before cliff or created date, no tokens have unlocked
314     if (block.number < grant.cliff) {
315         return 0;
316     }
317     // if after expiration, return the full allocation minus what has already been wit
318     if (block.number >= grant.expiration) {
319         return (grant.allocation - grant.withdrawn);
320     }
321     // if after cliff, return vested amount minus what has already been withdrawn
322     if (block.number >= grant.cliff) {
323         uint256 postCliffAmount = grant.allocation - grant.cliffAmount;
324         uint256 blocksElapsedSinceCliff = block.number - grant.cliff;
325         uint256 totalBlocksPostCliff = grant.expiration - grant.cliff;
326         uint256 unlocked = grant.cliffAmount + (postCliffAmount * blocksElapsedSinceCl
327 
328         return unlocked - grant.withdrawn;
329     } else {
330         return 0;
331     }
332 }
```

Recommendation:

We advise it and its `else` clause to be omitted, optimizing the function's gas cost.

Alleviation:

The `if-else` structure has been flattened with the `else` clause entirely omitted per our recommendation.

ARC-05C: Redundant Conditional Clause

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L104

Description:

The `ARCDVestingVault::addGrantAndDelegate` function will evaluate that a valid schedule has been defined by applying three conditional checks whereas only two are required.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
104 if (cliff >= expiration || startTime >= expiration || cliff < startTime) revert AVV_In
```

Recommendation:

We advise the referenced `if` clause to retain the `cliff >= expiration` and `cliff < startTime` evaluations in the OR (`||`) clause.

These conditionals will guarantee that `startTime <= cliff < expiration`, rendering the `startTime >= expiration` check (as a failure case) redundant.

Alleviation:

The `if` conditional has been optimized as advised.

ARC-06C: Redundant Load of Storage

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L169

Description:

The referenced instruction will query the current `BaseVotingVault::_manager` value when applying the `BaseVotingVault::onlyManager` modifier.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
169 token.transfer(_manager().data, remaining);
```

Recommendation:

We advise the `msg.sender` to be utilized instead as it is guaranteed to be equal to the `BaseVotingVault::_manager` data point due to how the modifier operates.

Alleviation:

The `msg.sender` value is utilized as advised, optimizing the statement's gas cost while being functionally identical to the original.

ARC-07C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	ARCDVestingVault.sol:L319, L342, L381

Description:

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

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
319 return (grant.allocation - grant.withdrawn);
```

Recommendation:

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

Alleviation:

The first redundant parenthesis instance has been removed, the second is no longer present in the codebase, and the third remains unaddressed rendering this exhibit partially alleviated.

ARC-08C: Redundant Utilizations of Inverses

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L358, L360, L362

Description:

The referenced statements attempt to calculate the delta between the voting powers using signed arithmetic and then reset the sign of the change to be positive.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
351 function _syncVotingPower(address who, ARCDVestingVaultStorage.Grant storage grant) in
352     History.HistoricalBalances memory votingPower = _votingPower();
353
354     uint256 delegateeVotes = votingPower.loadTop(grant.delegatee);
355
356     uint256 newVotingPower = _currentVotingPower(grant);
357     // get the change in voting power. Negative if the voting power is reduced
358     int256 change = int256(newVotingPower) - int256(uint256(grant.latestVotingPower));
359     // voting power can only go down since only called when tokens are claimed or gran
360     if (change < 0) {
361         // if the change is negative, we multiply by -1 to avoid underflow when castin
362         votingPower.push(grant.delegatee, delegateeVotes - uint256(change * -1));
363         emit VoteChange(grant.delegatee, who, change);
364
365         grant.latestVotingPower = uint128(newVotingPower);
366     }
367 }
```

Recommendation:

We advise the delta to instead be calculated as an "absolute" value by subtracting `newVotingPower` from `grant.latestVotingPower` and pushing a new entry to the voting history if the change is non-zero, optimizing the function's gas cost while minimizing its complexity.

Alleviation:

While the code was updated, it still tracks the voting power `change` as a signed integer.

Given that it is guaranteed to be negative, we advise it to be tracked as a `uint256` by inversing the subtraction (`grant.latestVotingPower - newVotingPower`), permitting the value to be used as is in the ensuing subtraction and thus optimizing both the legibility as well as gas cost of these statements.

ARC-09C: Repetitive Calculations of Storage Offsets

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVault.sol:L113, L126, L156, L178

Description:

The referenced statements contain the dynamic calculation of a `storage` offset whilst it does not change throughout each function's context.

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
112 // load the grant
113 ARCDVestingVaultStorage.Grant storage grant = _grants()[who];
114
115 // if this address already has a grant, a different address must be provided
116 // topping up or editing active grants is not supported.
117 if (grant.allocation != 0) revert AVV_HasGrant();
118
119 // load the delegate. Defaults to the grant owner
120 delegatee = delegatee == address(0) ? who : delegatee;
121
122 // calculate the voting power. Assumes all voting power is initially locked.
123 uint128 newVotingPower = amount;
124
125 // set the new grant
126 _grants()[who] = ARCDVestingVaultStorage.Grant(
127     amount,
128     cliffAmount,
129     0,
130     startTime,
131     expiration,
132     cliff,
133     newVotingPower,
134     delegatee
135 );
```

Recommendation:

We advise the dynamic `storage` offset to be calculated once by invoking the relevant function and to be stored to a local `storage` variable that is consequently utilized in place of repetitive invocations of the calculation function, optimizing the codebase.

Alleviation:

Both `storage` offset instances are solely present once in their function block within the latest iteration of the codebase, rendering this exhibit no longer applicable.

ARC-10C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	ARCDVestingVault.sol:L126-L135

Description:

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

Example:

contracts/ARCDVestingVault.sol

SOL

```sol
126 _grants()[who] = ARCDVestingVaultStorage.Grant(
127     amount,
128     cliffAmount,
129     0,
130     startTime,
131     expiration,
132     cliff,
133     newVotingPower,
134     delegatee
135 );
```

Recommendation:

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

Alleviation:

The Arcade XYZ team has opted to instead utilize an assignment statement per key value, greatly increasing the legibility of the statement albeit using a different approach.

To note, what we recommended was the usage of the key-value declaration style (i.e. `struct Foo { uint256 a; uint256 b; }; Foo memory foo = Foo({b: 32, a: 41});`) and not the currently employed style.

In any case, the legibility of the `Grant` structure's instantiation has been greatly increased rendering this exhibit alleviated.

ARCDVestingVaultStorage Code Style Findings

ARD-01C: Inefficient Variable Type

Type	Severity	Location
Gas Optimization	Informational	ARCDVestingVaultStorage.sol:L36

Description:

The referenced variable of the `Grant` structure is set as a `uint128`, however, it is stored in its own full 256-bit slot.

Example:

contracts/libraries/ARCDVestingVaultStorage.sol

SOL

```sol
29  struct Grant {
30      uint128 allocation;
31      uint128 cliffAmount;
32      uint128 withdrawn;
33      uint128 created;
34      uint128 expiration;
35      uint128 cliff;
36      uint128 latestVotingPower;
37      address delegatee;
38  }
```

Recommendation:

We advise it to be upgraded to 256-bits, optimizing all operations performed on it whilst retaining the same storage footprint for the `Grant` struct.

Alleviation:

The referenced variable has been upcasted to `uint256` per our recommendation, optimizing all statements that operate on it.

ArcadeAirdrop Code Style Findings

AAP-01C: Inefficient Assignment of Owner

Type	Severity	Location
Gas Optimization	Informational	ArcadeAirdrop.sol:L43

Description:

The `ArcadeAirdrop::constructor` will inefficiently invoke the `Authorizable::setOwner` function instead of assigning the `owner` value directly.

Example:

contracts/token/ArcadeAirdrop.sol

SOL

```sol
22  /**
23   * @notice Initiate the contract with a merkle tree root, a token for distribution,
24   *         an expiration time for claims, and the voting vault that tokens will be
25   *         airdropped into. In addition, set a governance parameter for the address th
26   *         can reclaim tokens after expiry.
27   *
28   * @param _governance           The address that can reclaim tokens after expiry
29   * @param _merkleRoot           The merkle root with deposits encoded into it as hash
30   * @param _token                The token to airdrop
31   * @param _expiration           The expiration of the airdrop
32   * @param _votingVault         The voting vault to deposit tokens to
33   */
34  constructor(
35      address _governance,
36      bytes32 _merkleRoot,
37      IERC20 _token,
38      uint256 _expiration,
39      INFTBoostVault _votingVault
40  ) ArcadeMerkleRewards(_merkleRoot, _token, _expiration, _votingVault) {
41      if (_governance == address(0)) revert AA_ZeroAddress();
42
43      setOwner(_governance);
44  }
```

Recommendation:

We advise the `owner` value to be adjusted directly as the `Authorizable::setOwner` function will inefficiently apply access control and incur extraneous gas.

Alleviation:

The `owner` value is adjusted directly in the updated **`ArcadeAirdrop::constructor`** implementation, optimizing its gas cost significantly.

ArcadeMerkleRewards Code Style Findings

AMR-01C: Variable Mutability Specifiers (Immutable)

Type	Severity	Location
Gas Optimization	Informational	ArcadeMerkleRewards.sol:L37, L43

Description:

The linked variables are assigned to only once during the contract's `constructor`.

Example:

contracts/libraries/ArcadeMerkleRewards.sol

SOL

```sol
57  constructor(bytes32 _rewardsRoot, IERC20 _token, uint256 _expiration, INFTBoostVault _
58      if (_expiration <= block.timestamp) revert AA_ClaimingExpired();
59      if (address(_token) == address(0)) revert AA_ZeroAddress();
60      if (address(_votingVault) == address(0)) revert AA_ZeroAddress();
61
62      rewardsRoot = _rewardsRoot;
63      token = _token;
64      expiration = _expiration;
65      votingVault = _votingVault;
66  }
```

Recommendation:

We advise them to be set as `immutable` greatly optimizing their read-access gas cost.

Alleviation:

The `ArcadeMerkleRewards` contract is expected to be inherited by other contracts that mutate those variables and as such, the Arcade XYZ team set the contract to `abstract` thus nullifying this exhibit.

ArcadeTokenDistributor Code Style Findings

ATD-01C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	ArcadeTokenDistributor.sol:L49, L54

Description:

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

Example:

contracts/token/ArcadeTokenDistributor.sol

SOL

```
49  /// @notice 32.7% of initial distribution is for Arcade's launch partners
```

Recommendation:

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

Alleviation:

The numbers specified by the referenced comments have been corrected, addressing this exhibit.

ArcadeTreasury Code Style Findings

ATY-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ArcadeTreasury.sol:L331

Description:

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

Example:

contracts/ArcadeTreasury.sol

SOL

```sol
331 for (uint256 i = 0; i < targets.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:

The Arcade XYZ team evaluated this exhibit and while acknowledging the gas optimization it brings, they wish to retain checked arithmetic throughout the codebase in favour of code clarity as well as safety.

As such, we consider this exhibit acknowledged.

BadgeDescriptor Code Style Findings

BDR-01C: Potentially Redundant Indexed Argument

Type	Severity	Location
Gas Optimization	Informational	BadgeDescriptor.sol:L20

Description:

THe `SetBaseURI` event within `BadgeDescriptor` contains an `indexed` URI argument, however, the URI is expected to be set to a new value on each instance.

Example:

contracts/nft/BadgeDescriptor.sol

SOL

```sol
20  event SetBaseURI(address indexed caller, string indexed baseURI);
```

Recommendation:

If this assumption is true, we advise the `indexed` keyword to be omitted from it, optimizing the event's emission gas cost.

Alleviation:

The `baseURI` argument has had its `indexed` keyword omitted, optimizing the event's emission.

BaseVotingVault Code Style Findings

BVV-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	BaseVotingVault.sol:L95

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/BaseVotingVault.sol

SOL

```
95  // Find the historical data and clear everything more than 'staleBlockLag' into the pa
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation:

Remove.

BVV-02C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	BaseVotingVault.sol:L176

Description:

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

Example:

contracts/BaseVotingVault.sol

SOL

```sol
176 return (History.load("votingPower"));
```

Recommendation:

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

Alleviation:

The redundant parenthesis statement has been safely omitted per our recommendation.

HashedStorageReentrancyBlock Code Style Findings

HSR-01C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	HashedStorageReentrancyBlock.sol:L35, L38, L44

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

contracts/libraries/HashedStorageReentrancyBlock.sol

SOL

```sol
35  require(entered.data == 1, "REENTRANCY");
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead optimizing the legibility of the codebase.

Alleviation:

The referenced literals were relocated to contract-level `constant` declarations akin to OpenZeppelin's `ReentrancyGuard` implementation, optimizing the codebase's legibility.

NFTBoostVault Code Style Findings

NFT-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	NFTBoostVault.sol:L230

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/NFTBoostVault.sol

SOL

```sol
222 Storage.Uint256 storage balance = _balance();
223 if (balance.data < amount) revert NBV_InsufficientBalance();
224
225 // get the withdrawable amount
226 uint256 withdrawable = _getWithdrawableAmount(registration);
227 if (withdrawable < amount) revert NBV_InsufficientWithdrawableBalance(withdrawable);
228
229 // update contract balance
230 balance.data -= amount;
```

Recommendation:

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

Alleviation:

The Arcade XYZ team evaluated this exhibit and while acknowledging the gas optimization it brings, they wish to retain checked arithmetic throughout the codebase in favour of code clarity as well as safety.

As such, we consider this exhibit acknowledged.

NFT-02C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	NFTBoostVault.sol:L347

Description:

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

Example:

contracts/NFTBoostVault.sol

SOL

```sol
347 for (uint256 i = 0; i < userAddresses.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:

The Arcade XYZ team evaluated this exhibit and while acknowledging the gas optimization it brings, they wish to retain checked arithmetic throughout the codebase in favour of code clarity as well as safety.

As such, we consider this exhibit acknowledged.

NFT-03C: Redundant Parenthesis Statements

Type	Severity	Location
Code Style	Informational	NFTBoostVault.sol:L534, L655

Description:

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

Example:

contracts/NFTBoostVault.sol

SOL

```sol
534 return (NFTBoostVaultStorage.mappingAddressToRegistrationPtr("registrations"));
```

Recommendation:

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

Alleviation:

Both referenced redundant parenthesis statements have been safely omitted.

NFT-04C: Repetitive Calculations of Storage Offsets

Type	Severity	Location
Gas Optimization	Informational	NFTBoostVault.sol:L219, L241, L348, L475, L488

Description:

The referenced statements contain the dynamic calculation of a `storage` offset whilst it does not change throughout each function's context.

Example:

contracts/NFTBoostVault.sol

SOL

```sol
219 NFTBoostVaultStorage.Registration storage registration = _getRegistrations()[msg.sende
220
221 // get this contract's balance
222 Storage.Uint256 storage balance = _balance();
223 if (balance.data < amount) revert NBV_InsufficientBalance();
224
225 // get the withdrawable amount
226 uint256 withdrawable = _getWithdrawableAmount(registration);
227 if (withdrawable < amount) revert NBV_InsufficientWithdrawableBalance(withdrawable);
228
229 // update contract balance
230 balance.data -= amount;
231 // update withdrawn amount
232 registration.withdrawn += amount;
233 // update the delegatee's voting power. Varies based on the multiplier associated with
234 // user's ERC1155 token at the time of the call
235 _syncVotingPower(msg.sender, registration);
236
237 if (registration.withdrawn == registration.amount) {
238     if (registration.tokenAddress != address(0) && registration.tokenId != 0) {
239         withdrawNft();
240     }
241     delete _getRegistrations()[msg.sender];
242 }
```

Recommendation:

We advise the dynamic `storage` offset to be calculated once by invoking the relevant function and to be stored to a local `storage` variable that is consequently utilized in place of repetitive invocations of the calculation function, optimizing the codebase.

Alleviation:

All but the interim **potential optimization** remain in the codebase, rendering this exhibit partially addressed.

NFT-05C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	NFTBoostVault.sol:L488-L495

Description:

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

Example:

contracts/NFTBoostVault.sol

SOL

```sol
_getRegistrations()[user] = NFTBoostVaultStorage.Registration(
    amount,
    newVotingPower,
    0,
    tokenId,
    tokenAddress,
    delegatee
);
```

Recommendation:

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

Alleviation:

The Arcade XYZ team has opted to instead utilize an assignment statement per key value, greatly increasing the legibility of the statement albeit using a different approach.

To note, what we recommended was the usage of the key-value declaration style (i.e. `struct Foo { uint256 a; uint256 b; }; Foo memory foo = Foo({b: 32, a: 41});`) and not the currently employed style.

In any case, the legibility of the `Registration` structure's instantiation has been greatly increased rendering this exhibit alleviated.

NFTBoostVaultStorage Code Style Findings

NFB-01C: Dynamic Hash Evaluations

Type	Severity	Location
Gas Optimization	Informational	NFTBoostVaultStorage.sol:L53, L70

Description:

The referenced `keccak256` operations are performed on known-at-compile-time payloads, however, they are performed dynamically within function blocks.

Example:

contracts/libraries/NFTBoostVaultStorage.sol

SOL

```sol
/**
 * @notice Returns the storage pointer for a named mapping of address to registration
 *
 * @param name                      The variable name for the pointer.
 *
 * @return data                     The mapping pointer.
 */
function mappingAddressToRegistrationPtr(
    string memory name
) internal pure returns (mapping(address => Registration) storage data) {
    bytes32 typehash = keccak256("mapping(address => Registration)");
    bytes32 offset = keccak256(abi.encodePacked(typehash, name));
    assembly {
        data.slot := offset
    }
}

/**
 * @notice Returns the storage pointer for a named mapping of address to uint128 pair
 *
 * @param name                      The variable name for the pointer.
 *
 * @return data                     The mapping pointer.
 */
function mappingAddressToPackedUintUint(
    string memory name
) internal pure returns (mapping(address => mapping(uint128 => AddressUintUint)) stora
    bytes32 typehash = keccak256("mapping(address => mapping(uint128 => AddressUintUin
    bytes32 offset = keccak256(abi.encodePacked(typehash, name));
    assembly {
        data.slot := offset
    }
}
```

Recommendation:

We advise them to be relocated to dedicated `constant` declarations, optimizing each function's gas cost.

Alleviation:

Both referenced typehashes have been relocated to dedicated `constant` variable declarations, optimizing their gas cost significantly.

NFB-02C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	NFTBoostVaultStorage.sol:L38, L67

Description:

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

Example:

contracts/libraries/NFTBoostVaultStorage.sol

SOL

```
38  struct AddressUintUint {
```

Recommendation:

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

Alleviation:

The Arcade XYZ team updated the contract's comments to better illustrate what the `UintUint` pair of each referenced name is meant to depict. As such, we consider this exhibit alleviated.

ReputationBadge Code Style Findings

RBE-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	ReputationBadge.sol:L111, L116

Description:

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

Example:

contracts/nft/ReputationBadge.sol

SOL

```sol
111 if (amountClaimed[recipient][tokenId] + amount > totalClaimable) {
112     revert RB_InvalidClaimAmount(amount, totalClaimable);
113 }
114
115 // increment amount claimed
116 amountClaimed[recipient][tokenId] += amount;
```

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:

The Arcade XYZ team has evaluated this exhibit and has opted not to apply a remediation for it as they deem it to not optimize their code.

To note, our recommendation is to cache the interim lookup (`amountClaimed[recipient]`) to a `storage` variable (i.e. `mapping(uint256 => uint256) storage`) which would optimize the codebase.

RBE-02C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ReputationBadge.sol:L144

Description:

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

Example:

contracts/nft/ReputationBadge.sol

SOL

```sol
144 for (uint256 i = 0; i < _claimData.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:

The Arcade XYZ team evaluated this exhibit and while acknowledging the gas optimization it brings, they wish to retain checked arithmetic throughout the codebase in favour of code clarity as well as safety.

As such, we consider this exhibit acknowledged.

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.