OMNISCIA

November 6, 2023

SMART CONTRACT AUDIT REPORT

Gnosis Guild
Zodiac PR206


omniscia.io


info@omniscia.io


Online report: gnosis-guild-zodiac-pr206

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

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

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

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

omniscia.io


info@omniscia.io

Zodiac PR206 Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
6a7fb909a1	September 25th 2023	10747ee4ce
6a7fb909a1	September 25th 2023	880e513565
e6d315f917	October 23rd 2023	89d03a8320
4d851d2a84	November 1st 2023	cde3ac0181
4d851d2a84	November 6th 2023	040261a811

Audit Overview

We were tasked with performing an audit of the Gnosis Guild codebase and in particular a revision of the Zodiac Modifier Roles codebase we have previously audited.

The audit encompassed the changes introduced between **the original audit** and the latest revision of the repository's **PR#206** (`6a7fb909a1`) in relation to the contracts in scope of the original audit.

Changes introduced to contracts that were outside the scope of the original audit have not been evaluated and should not be considered in-scope. These contracts consist of the following:

Over the course of the audit, we evaluated the changes' purposed intentions described in PR#206 and their actual implementation in the code.

We validated that the `AbiEncoded` parameter type was correctly renamed to `Calldata` by updating the relevant references across the codebase's contracts.

To this end, we observed a discrepancy in how the re-purposed `AbiEncoded` parameter type is evaluated as inline within the `Topology` and `Packer` contracts that we advise the Gnosis Guild team to consider.

The relaxed `Integrity` validation flows incurred a significant gas overhead that we detailed and we advise the Gnosis Guild team to evaluate and potentially optimize.

A security check that was removed from the `PermissionBuilder` contract was identified to have an impact on the out-of-scope `AllowanceTracker` contract and should be re-introduced to the contract's code.

The `AvatarIsOwnerOfERC721` custom checker was validated to behave as expected; we recommended certain stylistic adjustments as well as an enhancement of its return data to aid off-chain and on-chain systems in handling/utilizing the checker.

Finally, as part of the adjustments necessary for the `AvatarIsOwnerOfERC721` custom checker our original audit report's exhibit **`TSE-01M`** has been alleviated.

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

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	7	2	0	5
Minor	0	0	0	0
Medium	3	3	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **1 findings utilizing static analysis** tools as well as identified a total of **9 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Scope

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

Target

- Repository: **https://github.com/gnosis/zodiac-modifier-roles**
- Commit: 6a7fb909a1a5dc55d5cfbe759a624ce20c625f46
- Language: Solidity
- Network: Ethereum
- Revisions: 6a7fb909a1, e6d315f917, 4d851d2a84

Contracts Assessed

File	Total Finding(s)
packages/evm/contracts/adapters/AvatarIsOwnerOfERC721.sol (AIO)	3
packages/evm/contracts/Decoder.sol (DRE)	0
packages/evm/contracts/Integrity.sol (IYT)	2
packages/evm/contracts/packers/Packer.sol (PRE)	1
packages/evm/contracts/PermissionBuilder.sol (PBR)	1
packages/evm/contracts/PermissionChecker.sol (PCR)	2
packages/evm/contracts/adapters/Types.sol (TSE)	0
packages/evm/contracts/Types.sol (TSP)	0
packages/evm/contracts/Topology.sol (TYG)	1

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

The project contains discrepancies with regards to the Solidity version used as the pragma statements of the contracts are open-ended (`>=0.8.17 <0.9.0`).

We advise them to be locked to `0.8.21` (`=0.8.21`), 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 **2 potential issues** within the codebase of which **none were ruled out to be false positives** or negligible findings.

The remaining **2 issues** were validated and grouped and formalized into the **1 exhibits** that follow:

ID	Severity	Addressed	Title
AIO-01S	Informational	Acknowledged	Multiple Top-Level Declarations

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Gnosis Guild's Zodiac module revisions.

As the project at hand represents a delta between the original audit and the new changes introduced in its revision, intricate care was put into ensuring that the **the system retains its conformity to the specifications and restrictions** laid forth within the protocol's specification throughout the changes introduced.

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 potential discrepancies** in the way the system handles the `AbiEncoded` parameter type as well as **an allowance-related flaw** within the system which could have had **minor-to-moderate ramifications** to its overall operation; we urge the Gnosis Guild team to promptly evaluate and remediate these exhibits if needed.

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 thorough inline documentation throughout the codebase's functions, data structures, and interface declarations.

A total of **9 findings** were identified over the course of the manual review of which **3 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
PRE-01M	Medium	Yes	Discrepant Support of `AbiEncoded` Inline Evaluation
PBR-01M	Medium	Yes	Removal of Important Security Check
TYG-01M	Medium	Yes	Discrepant Support of `AbiEncoded` Inline Evaluation

Code Style

During the manual portion of the audit, we identified **6 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
AIO-01C	Informational	Yes	Documentation Enhancement
AIO-02C	Informational	Acknowledged	Potential Enhancement of Custom Adapter Data
IYT-01C	Informational	Acknowledged	Inefficient Re-Calculation of Base Type Tree
IYT-02C	Informational	Acknowledged	Inefficient Re-Calculation of Iterated Type Tree
PCR-01C	Informational	Acknowledged	Inefficient Usage of Structure
PCR-02C	Informational	Yes	Regression of Gas Optimization

AvatarIsOwnerOfERC721 Static Analysis Findings

AIO-01S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	AvatarIsOwnerOfERC721.sol:L7, L13

Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

Example:

packages/evm/contracts/adapters/AvatarIsOwnerOfERC721.sol

SOL

```sol
7   interface IModifier {
8       function avatar() external view returns (address);
9
10      function target() external view returns (address);
11  }
12
13  contract AvatarIsOwnerOfERC721 is ICustomCondition {
```

Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

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

Packer Manual Review Findings

PRE-01M: Discrepant Support of `AbiEncoded` Inline Evaluation

Type	Severity	Location
Logical Fault	Medium	Packer.sol:L76

Description:

The `Packer::_isInline` function will discrepantly handle the `AbiEncoded` parameter type. In detail, a `Calldata` parameter type differs from an `AbiEncoded` parameter type solely in that it is prefixed with a 4-byte function signature.

In the latest `Packer::_isInline` implementation, an `AbiEncoded` parameter type is considered inline if all its children are inline, however, the `Calldata` parameter type is always considered to not be inline.

This is discrepant as both types can be considered inline using the same approach (i.e. a `Calldata` parameter type is inline if all its arguments are statically-sized).

Impact:

It is presently possible for the `Packer` contract to treat an `AbiEncoded` type as inline incorrectly if all its elements are inline.

Example:

packages/evm/contracts/packers/Packer.sol

SOL

```sol
function _isInline(
    ConditionFlat[] memory conditions,
    uint256 index
) private pure returns (bool) {
    ParameterType paramType = conditions[index].paramType;
    if (paramType == ParameterType.Static) {
        return true;
    } else if (
```

Example (Cont.):

```sol
            paramType == ParameterType.Dynamic ||
            paramType == ParameterType.Array ||
            paramType == ParameterType.Calldata
        ) {
            return false;
        } else {
            uint256 length = conditions.length;

            for (uint256 j = index + 1; j < length; ++j) {
                uint8 parent = conditions[j].parent;
                if (parent < index) {
                    continue;
                }

                if (parent > index) {
                    break;
                }

                if (!_isInline(conditions, j)) {
                    return false;
                }
            }
            return true;
        }
    }
```

Recommendation:

We advise the code to treat `Calldata` and `AbiEncoded` parameter types identically and to strictly define how they should be evaluated as inline.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The **`Packer::_isInline`** function was updated to properly treat the `AbiEncoded` type identically to the `Calldata` type, addressing this exhibit in full.

PermissionBuilder Manual Review Findings

PBR-01M: Removal of Important Security Check

Type	Severity	Location
Input Sanitization	Medium	PermissionBuilder.sol:L160, L161, L166

Description:

The `PermissionBuilder::setAllowance` function contained an `if-revert` check that ensured the `balance` being established for the `key` is less-than-or-equal-to the `maxBalance` dynamically calculated within the function.

This condition was removed and affects out-of-scope contract `AllowanceTracker` and specifically `AllowanceTracker::_accruedAllowance`.

The function will improperly reset the `balance` of the `key` to the `maxBalance` whenever an interval elapses even if the `balance` is significantly higher than the `maxBalance`, effectively reducing it.

Impact:

An invoker of the `PermissionBuilder::setAllowance` function would expect the `balance` allowed to solely decrease when consumed rather than when updated which can cause significant misbehaviours into how users as well as protocols interact with the `PermissionBuilder`.

Example:

packages/evm/contracts/PermissionBuilder.sol

SOL

```sol
158 function setAllowance(
159     bytes32 key,
160     uint128 balance,
161     uint128 maxBalance,
162     uint128 refill,
163     uint64 period,
164     uint64 timestamp
165 ) external onlyOwner {
```

Example (Cont.):

```sol
    maxBalance = maxBalance > 0 ? maxBalance : type(uint128).max;

    allowances[key] = Allowance({
        refill: refill,
        period: period,
        timestamp: timestamp,
        balance: balance,
        maxBalance: maxBalance
    });
```

Recommendation:

We advise the code to re-introduce the `if-revert` pattern check present in the original code to ensure that the `AllowanceTracker` behaves as expected.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The `AllowanceTracker` and `PermissionBuilder` contracts were slightly refactored to utilize a `maxRefill` notion rather than `maxBalance` notion, with the `AllowanceTracker::_accruedAllowance` function measuring whether the present balance is lower than the `maxRefill` value and increasing it by the elapsed periods.

Should the balance after the refill is executed exceed the `maxRefill` value, it will be set to it to avoid the `balance` exceeding the `maxRefill` value **when it is refilled**.

As the contract no longer improperly resets the balance to the refill amount if it has been set to a greater value, we consider this exhibit fully alleviated.

We would like to note that a side-effect of the current implementation is that refill periods that have elapsed in the past whilst the balance is greater than the refill threshold will be unaccounted for when the balance needs to refill again.

We consider this a desirable trait that the Gnosis Guild team wishes to enforce.

Topology Manual Review Findings

TYG-01M: Discrepant Support of `AbiEncoded` Inline Evaluation

Type	Severity	Location
Logical Fault	Medium	Topology.sol:L55

Description:

The `Topology::isInline` function will discrepantly handle the `AbiEncoded` parameter type. In detail, a `Calldata` parameter type differs from an `AbiEncoded` parameter type solely in that it is prefixed with a 4-byte function signature.

In the latest `Topology::isInline` implementation, an `AbiEncoded` parameter type is considered inline if all its children are inline, however, the `Calldata` parameter type is always considered to not be inline.

This is discrepant as both types can be considered inline using the same approach (i.e. a `Calldata` parameter type is inline if all its arguments are statically-sized).

Impact:

It is presently possible for the `Topology` contract to treat an `AbiEncoded` type as inline incorrectly if all its elements are inline.

Example:

packages/evm/contracts/Topology.sol

SOL

```
48  function isInline(TypeTree memory node) internal pure returns (bool) {
49      ParameterType paramType = node.paramType;
50      if (paramType == ParameterType.Static) {
51          return true;
52      } else if (
53          paramType == ParameterType.Dynamic ||
54          paramType == ParameterType.Array ||
55          paramType == ParameterType.Calldata
```

Example (Cont.):

```sol
    ) {
        return false;
    } else {
        uint256 length = node.children.length;

        for (uint256 i; i < length; ) {
            if (!isInline(node.children[i])) {
                return false;
            }
            unchecked {
                ++i;
            }
        }
        return true;
    }
}
```

Recommendation:

We advise the code to treat `Calldata` and `AbiEncoded` parameter types identically and to strictly define how they should be evaluated as inline.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The **`Topology::isInline`** function was updated to properly treat the `AbiEncoded` type identically to the `Calldata` type, addressing this exhibit in full.

AvatarIsOwnerOfERC721 Code Style Findings

AIO-01C: Documentation Enhancement

Type	Severity	Location
Code Style	Informational	AvatarIsOwnerOfERC721.sol:L20

Description:

Similarly to how the `value` named argument is implied in the **`AvatarIsOwnerOfERC721::check`** function, the `extra` argument should be documented as well.

Example:

packages/evm/contracts/adapters/AvatarIsOwnerOfERC721.sol

SOL

```sol
14  function check(
15      address to,
16      uint256 /* value */,
17      bytes calldata data,
18      uint256 location,
19      uint256 size,
20      bytes12
21  ) public view returns (bool success, bytes32 reason) {
```

Recommendation:

We advise the `extra` name to be denoted in in-line comments after the `bytes12` type specifier.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The `extra` named variable is properly denoted in the commented out section of the **`AvatarIsOwnerOfERC721::check`** function, addressing this exhibit.

AIO-02C: Potential Enhancement of Custom Adapter Data

Type	Severity	Location
Standard Conformity	Informational	AvatarIsOwnerOfERC721.sol:L24

Description:

The `AvatarIsOwnerOfERC721::check` function will yield a value of `0` as the `reason` the check failed.

Example:

packages/evm/contracts/adapters/AvatarIsOwnerOfERC721.sol

SOL

```sol
function check(
    address to,
    uint256 /* value */,
    bytes calldata data,
    uint256 location,
    uint256 size,
    bytes12
) public view returns (bool success, bytes32 reason) {
```

Example (Cont.):

```sol
        address avatar = IModifier(msg.sender).avatar();
        uint256 tokenId = uint256(bytes32(data[location:location + size]));
        return (IERC721(to).ownerOf(tokenId) == avatar, 0);
    }
```

Recommendation:

We advise its usability to be increased by simply yielding the actual `avatar` expected, aiding off-chain as well as on-chain software in handling the custom condition's failure.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The Gnosis Guild team evaluated this exhibit and opted not to apply a remediation for it as the avatar expected value is publicly and easily accessible. As such, we consider this exhibit acknowledged safely.

Integrity Code Style Findings

IYT-01C: Inefficient Re-Calculation of Base Type Tree

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L266, L267, L281, L291-L293

Description:

The `Integrity::_compatibleSiblingTypes` function will inefficiently re-calculate the `Topology::typeTree` of the `start` exactly `end - start - 2` times.

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
256 function _compatibleSiblingTypes(
257     ConditionFlat[] memory conditions,
258     uint256 index,
259     Topology.Bounds[] memory childrenBounds
260 ) private pure {
261     uint256 start = childrenBounds[index].start;
262     uint256 end = childrenBounds[index].end;
263
```

Example (Cont.):

```sol
    for (uint256 j = start + 1; j < end; ++j) {
        if (
            !_isTypeMatch(conditions, start, j, childrenBounds) &&
            !_isTypeEquivalent(conditions, start, j, childrenBounds)
        ) {
            revert UnsuitableChildTypeTree(index);
        }
    }
}

function _isTypeMatch(
    ConditionFlat[] memory conditions,
    uint256 i,
    uint256 j,
    Topology.Bounds[] memory childrenBounds
) private pure returns (bool) {
    return
        typeTreeId(Topology.typeTree(conditions, i, childrenBounds)) ==
        typeTreeId(Topology.typeTree(conditions, j, childrenBounds));
}

function _isTypeEquivalent(
    ConditionFlat[] memory conditions,
    uint256 i,
    uint256 j,
    Topology.Bounds[] memory childrenBounds
) private pure returns (bool) {
    ParameterType leftParamType = Topology
```

Example (Cont.):

```sol
        .typeTree(conditions, i, childrenBounds)
        .paramType;
    return
        (leftParamType == ParameterType.Calldata ||
            leftParamType == ParameterType.AbiEncoded) &&
        Topology.typeTree(conditions, j, childrenBounds).paramType ==
        ParameterType.Dynamic;
}
```

Recommendation:

We advise the `Topology::typeTree` result of the `start` to be calculated outside the `for` loop and passed in on each iteration, significantly reducing the function's execution cost.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The Gnosis Guild team evaluated this exhibit but opted not to apply its optimization as it relates to a configurational function and not a run-time gas cost.

IYT-02C: Inefficient Re-Calculation of Iterated Type Tree

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L266, L267

Description:

The `Integrity::_compatibleSiblingTypes` function will inefficiently re-calculate the `Topology::typeTree` of the `j` iterator twice on each `for` loop execution.

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
256 function _compatibleSiblingTypes(
257     ConditionFlat[] memory conditions,
258     uint256 index,
259     Topology.Bounds[] memory childrenBounds
260 ) private pure {
261     uint256 start = childrenBounds[index].start;
262     uint256 end = childrenBounds[index].end;
263
```

Example (Cont.):

```sol
    for (uint256 j = start + 1; j < end; ++j) {
        if (
            !_isTypeMatch(conditions, start, j, childrenBounds) &&
            !_isTypeEquivalent(conditions, start, j, childrenBounds)
        ) {
            revert UnsuitableChildTypeTree(index);
        }
    }
}

function _isTypeMatch(
    ConditionFlat[] memory conditions,
    uint256 i,
    uint256 j,
    Topology.Bounds[] memory childrenBounds
) private pure returns (bool) {
    return
        typeTreeId(Topology.typeTree(conditions, i, childrenBounds)) ==
        typeTreeId(Topology.typeTree(conditions, j, childrenBounds));
}

function _isTypeEquivalent(
    ConditionFlat[] memory conditions,
    uint256 i,
    uint256 j,
    Topology.Bounds[] memory childrenBounds
) private pure returns (bool) {
    ParameterType leftParamType = Topology
```

Example (Cont.):

```sol
        .typeTree(conditions, i, childrenBounds)
        .paramType;
    return
        (leftParamType == ParameterType.Calldata ||
            leftParamType == ParameterType.AbiEncoded) &&
        Topology.typeTree(conditions, j, childrenBounds).paramType ==
        ParameterType.Dynamic;
}
```

Recommendation:

We advise the code to calculate the `Topology::typeTree` of the `j` iterator once and to pass it in to the `Integrity::_isTypeMatch` and `Integrity::_isTypeEquivalent` functions, optimizing the gas cost of the code significantly.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The Gnosis Guild team evaluated this exhibit but opted not to apply its optimization as it relates to a configurational function and not a run-time gas cost.

PermissionChecker Code Style Findings

PCR-01C: Inefficient Usage of Structure

Type	Severity	Location
Gas Optimization	Informational	PermissionChecker.sol:L626, L643, L646, L704, L705, L706

Description:

The `Context` structure, while significantly standardizing the codebase, incurs a gas overhead throughout all functions it is currently utilized. Namely, it reserves enough memory and copies the `to` argument to all calls redundantly whilst it is solely required by **`PermissionChecker::_custom`**.

Additionally, the **`PermissionChecker::_withinAllowance`** function solely utilizes the `consumptions` meaning that the `value` argument is redundantly passed in as well.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```
612 function _custom(
613     bytes calldata data,
614     Condition memory condition,
615     ParameterPayload memory payload,
616     Context memory context
617 ) private view returns (Status, Result memory) {
618     // 20 bytes on the left
619     ICustomCondition adapter = ICustomCondition(
```

Example (Cont.):

```sol
        address(bytes20(condition.compValue))
    );
    // 12 bytes on the right
    bytes12 extra = bytes12(uint96(uint256(condition.compValue)));

    (bool success, bytes32 info) = adapter.check(
        context.to,
        context.value,
        data,
        payload.location,
        payload.size,
        extra
    );
    return (
        success ? Status.Ok : Status.CustomConditionViolation,
        Result({consumptions: context.consumptions, info: info})
    );
}

function _withinAllowance(
    bytes calldata data,
    Condition memory condition,
    ParameterPayload memory payload,
    Context memory context
) private pure returns (Status, Result memory) {
    uint256 value = uint256(Decoder.word(data, payload.location));
    return __consume(value, condition, context.consumptions);
}
```

Example (Cont.):

```sol

function _etherWithinAllowance(
    Condition memory condition,
    Context memory context
) private pure returns (Status status, Result memory result) {
    (status, result) = __consume(
        context.value,
        condition,
        context.consumptions
    );
    return (
        status == Status.Ok ? Status.Ok : Status.EtherAllowanceExceeded,
        result
    );
}
```

Recommendation:

We advise the `Context` structure to be solely utilized by the `PermissionChecker::_custom` function and all other functions to be reverted to their original implementation thus leading to an overall gas reduction in the `PermissionChecker` contract.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The Gnosis Guild team evaluated this exhibit but has opted not to apply its optimization in favour of code standardization and clarity.

PCR-02C: Regression of Gas Optimization

Type	Severity	Location
Gas Optimization	Informational	PermissionChecker.sol:L126, L128, L131

Description:

The `role.targets[to]` entry was previously cached to a local `storage` pointer, however, this optimization has been regressed and the `role.targets[to]` entry is specified throughout the **`PermissionChecker::_transaction`** function.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```
126 if (role.targets[to].clearance == Clearance.Target) {
127     return (
128         _executionOptions(value, operation, role.targets[to].options),
129         Result({consumptions: consumptions, info: 0})
130     );
131 } else if (role.targets[to].clearance == Clearance.Function) {
```

Recommendation:

We advise the code to revert its changes and properly cache the `storage` pointer as the `role.targets[to]` offset is calculated at minimum twice throughout all possible execution paths of the `PermissionChecker::_transaction` beyond the first `if` conditional.

Alleviation (e6d315f9170dcf4c622d504bd2fb6eafbdac9b75):

The Gnosis Guild team evaluated this optimization and instead opted to re-order the `if-else` chain, setting the most frequent execution path as the first `if` conditional to be evaluated.

As a result of this change, some of the gas cost from the common-case scenario has been shifted to the worst-case scenarios effectively optimizing the median gas cost of the function.

As such, we consider this exhibit alleviated as the relevant code block has been optimized based on the function's usage.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

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.