Zodiac Roles Modifier Security Audit

SUB7-ZODIAC-RM-1

Sub7 Security

16-12-2022

Contents

1 Project Overview

1.1 Description

Zodiac is a collection of tools built according to an open standard which proposes a composable design philosophy for DAOs. Built by the Gnosis Guild team, this extension pack for DAOs enables connection between platforms, protocols, and chains, no longer confined to monolithic designs. It not only changes access, but bridges chains, organizations, and networks through DAO-to-DAO (D2D) collaboration mechanisms.

The Roles Modifier belongs to the Zodiac collection of tools, which can be accessed through the Zodiac App available on Gnosis Safe. This modifier allows avatars to enforce granular, role-based, permissions for attached modules.

Modules that have been granted a role are able to unilaterally make calls to any approved addresses, approved functions, and approved variables the role has access to.

The interface mirrors the relevant parts of the Gnosis Safe's interface, so this contract can be placed between Gnosis Safe modules and a Gnosis Safe to enforce role-based permissions.

2 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

Three Security Auditors/Consultants were engaged in this activity.

2.1 Summary of Findings Identified

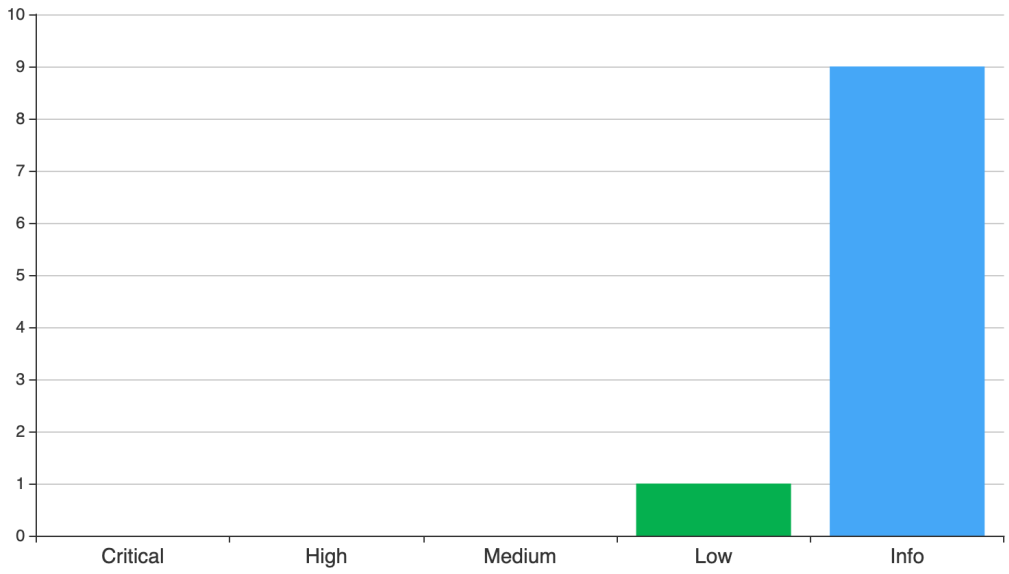

Figure 1: Executive Summary

 # 1 Info Revoking a Target does not remove the previous scopeConfigs – ***Acknowledged***

 # 2 Low Explicit downcast may cause unknown issues – ***Fixed***

 # 3 Info setDefaultRole() can be called on a not yet enabled module – ***Open***

 # 4 Info Missing 0x0 address checks in the constructor – ***Acknowledged***

 # 5 Info Inconsistent use of uint8 for paramIndex – ***Open***

 # 6 Info Custom error is not used, remove dead code – ***Fixed***

 # 7 Info Wrong or misleading comments throughout the code – ***Fixed***

 # 8 Info scopeFunction() can be used to scope a function even when isScoped.length == 0 – ***Open***

 # 9 Info Use underscore prefix for `internal` functions – ***Open***

 # 10 Info Assertion is not needed – ***Open***

2.2 Scope

2.2.1 In Scope

Hash: b96259f97dcfa0c45e814672c022f3df903b8107

- https://github.com/gnosis/zodiac-modifier-roles/blob/main/packages/evm/contracts/Roles.sol
- https://github.com/gnosis/zodiac-modifier-roles/blob/main/packages/evm/contracts/Permissions.sol

Retest on Jan 13, 2023

Hash: f84b65ccf905be623006e434c298357e0a315b4a

- https://github.com/gnosis/zodiac-modifier-roles/blob/static-analyzer-suggestions/packages/evm/contracts/Role
- https://github.com/gnosis/zodiac-modifier-roles/blob/static-analyzer-suggestions/packages/evm/contracts/Perm

2.2.2 Out of Scope

-

2.3 Methodology

Sub7 Security follows a phased assessment approach that includes thorough application profiling, threat analysis, dynamic and manual testing. Security Auditors/Consultants utilize automated security tools, custom scripts, simulation apps, manual testing and validation techniques to scan for, enumerate and uncover findings with potential security risks.

As part of the process of reviewing solidity code, each contract is checked against lists of known smart contract vulnerabilities, which is crafted from various sources like SWC Registry , DeFi threat and previous audit reports.

The assessment included (but was not limited to) reviews on the following attack vectors:

Oracle Attacks | Flash Loan Attacks | Governance Attacks | Access Control Checks on Critical Function | Account Existence Check for low level calls | Arithmetic Over/Under Flows | Assert Violation | Authorization through tx.origin | Bad Source of Randomness | Block Timestamp manipulation | Bypass Contract Size Check | Code With No Effects | Delegatecall | Delegatecall to Untrusted Callee | DoS with (Unexpected) revert | DoS with Block Gas Limit | Logical Issues | Entropy Illusion | Function Selector Abuse | Floating Point and Numerical | Precision | Floating Pragma | Forcibly Sending Ether to a Contract | Function Default Visibility | Hash Collisions With Multiple Variable Length Arguments | Improper Array Deletion | Incorrect interface | Insufficient gas griefing | Unsafe Ownership Transfer | Loop through long arrays | Message call with hardcoded gas amount | Outdated Compiler Version | Precision Loss in Calculations | Price Manipulation | Hiding Malicious Code with External Contract | Public burn() function | Race Conditions / Front Running | Re-entrancy | Requirement Violation | Right-To-Left-Override control character (U+202E) | Shadowing State Variables | Short Address/Parameter Attack | Signature Malleability | Signature Replay Attacks | State Variable Default Visibility | Transaction Order Dependence | Typographical Error | Unchecked Call Return Value | Unencrypted Private Data On-Chain | Unexpected Ether balance | Uninitialized Storage Pointer | Unprotected Ether Withdrawal | Unprotected SELFDESTRUCT Instruction | Unprotected Upgrades | Unused Variable | Use of Deprecated Solidity Functions | Write to Arbitrary Storage Location | Wrong inheritance | Many more…

2.4 Recommendations

Sub7 Security recomends improving the security of these contracts by introducing security into the current CI/CD.

2.4.1 Linters

Solhint, Eslint & Prettier

You can run all of them at once, for example below

```
lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Get node.js
        uses: actions/setup-node@v1
        with:
          node-version: "16.x"
      - run: npm install
      - run: npx solhint "contracts/**/*.sol"
      - run: npx eslint .
      - run: npx prettier -c .
```

2.4.2 Slither

Slither Action

Run slither with `fail-on: medium` to avoid to many false positives

```
  slither:
    name: Slither
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: crytic/slither-action@v0.2.0
        with:
          fail-on: medium
```

2.4.3 Coverage

```
  coverage:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Get node.js
        uses: actions/setup-node@v2
        with:
          node-version: "16.x"
          cache: "npm"
      - run: npm ci
```

```
11        - run: npx hardhat compile
12        - name: solidity-coverage
13          run: npx hardhat coverage
14        - name: coveralls
15          uses: coverallsapp/github-action@1.1.3
16          with:
17            github-token: ${{ secrets.GITHUB_TOKEN }}
```

3 Findings and Risk Analysis

3.1 Revoking a Target does not remove the previous scopeConfigs

Severity: Info
Status: Acknowledged

Description

We note that in `revokeTarget()` and `scopeTarget()`, it overwrites the previous `TargetAddress` with a new one and set `clearance == Clearance.None` and options == `ExecutionOptions.None`. There is an implied assumption here, that if we revoke or reset a target, we should clear all of its `ExecutingOptions` but this is not true.

```
    function revokeTarget(
        Role storage role,
        uint16 roleId,
        address targetAddress
    ) external {
        role.targets[targetAddress] = TargetAddress(
            Clearance.None,
            ExecutionOptions.None
        );
        emit RevokeTarget(roleId, targetAddress);
    }
```

```
    function scopeTarget(
        Role storage role,
        uint16 roleId,
        address targetAddress
    ) external {
        role.targets[targetAddress] = TargetAddress(
            Clearance.Function,
            ExecutionOptions.None
        );
        emit ScopeTarget(roleId, targetAddress);
    }
```

We see that in `checkTransaction()`, when `target.clearance == Clearance.Function`, we only check for the `ExecutionOptions` values which is saved in `role.functions[]`. We are not using the values stored in `roles.target[].options` unlike when `target.clearance == Clearance.Target`.

```
    function checkTransaction(
        Role storage role,
        address targetAddress,
        uint256 value,
        bytes memory data,
        Enum.Operation operation
    ) internal view {
        ...
        if (target.clearance == Clearance.Target){
```

```
10              checkExecutionOptions(value, operation, target.options);
11              return;
12          }
13
14          if (target.clearance == Clearance.Function) {
15              uint256 scopeConfig = role.functions[
16                  keyForFunctions(targetAddress, bytes4(data))
17              ];
18
19              if (scopeConfig == 0) {
20                  revert FunctionNotAllowed();
21              }
22
23              (ExecutionOptions options, bool isWildcarded, ) = unpackFunction(
24                  scopeConfig
25              );
26
27              checkExecutionOptions(value, operation, options);
28
29              if (isWildcarded == false) {
30                  checkParameters(role, scopeConfig, targetAddress, data);
31              }
32              return;
33          }
34
35          assert(false);
36      }
```

In essence, what can happen is, if we `revokeTarget()`, and later on `scopeTarget()` on the same address, we may not be aware of the options that are previously set. Unauthorized and unexpected permissions may be allowed by the target address.

Location

Permissions.sol#L388-L398

Permissions.sol#L400-L410

Permissions.sol#L239-L282

Impact

Impact can be severe here if this vulnerability happens. We may unknowingly grant permissions that we do not want to allow to the target address.

Recommendation

We recommend that in `revokeTarget()` and `scopeTarget()`, we clear the values of `role.functions[]`. There is currently no way in the implementation to check the `functionSig` that was previously allowed. We can store these values when a new function is allowed for each target address, so that it can be used to clear the values when we are revoking permission.

Client Comments

The function level configurations linger and are not pruned by a revokeTarget.

While they won't take effect if the target is revoked (revokeTarget call) they will come back into play if a target is reset to Clearance.Function (scopeTarget call)

With the current storage layout it wasn't viable from a gas cost perspective to perform the flush. This is noted and a well know behaviour for now, and we might improve it in an upcoming version.

3.2 Explicit downcast may cause unknown issues

Severity: Low
Status: Fixed

Description

Explicit downcast in solidity cuts off the higher order bits. We generally do not want to do this as it does not give a warning to us should this happens. For example, if `index` value happens to be `256`, `uint(index)` would silently return us a value of 1, giving us wrong values for the rest of the functions that rely on this value. This can create potential issues if not caught.

Location

Permissions.sol#L972

Impact

The wrong permissions may be granted if we use an unintended `index` value.

Recommendation

While we understand that expected value of `index` is < 48 in its current implementation, we still recommend to use OpenZeppelin's SafeCast here to be on the safe side.

3.3 setDefaultRole() can be called on a not yet enabled module

Severity: Info
Status: Open

Description

The `setDefaultRole` method can be called with a `module` argument that has not yet been enabled. It also doesn't check if the module is set as a member of this role. This is a logic discrepancy with the `assignRoles` functionality.

Location

Roles.sol#L310

Recommendation

Add the following validations to `setDefaultRole`

```
if (!roles[role].members[module]) {
        revert NoMembership();
}
if (!isModuleEnabled(module)) {
    enableModule(module);
}
```

3.4 Missing 0x0 address checks in the constructor

Severity: Info
Status: Acknowledged

Description

It is a best practice to check `address` type arguments in functions/constructors so they are not unintentionally set to the zero address.

Location

Roles.sol#L38

Recommendation

Add `require` or `if` statements for validation of `address` type arguments in the constructor of `Roles.sol`

Client Comments

@dev comment added:

```
1       /// @dev There is no zero address check as solidty will check for
2       /// missing arguments and the space of invalid addresses is too large
3       /// to check. Invalid avatar or target address can be reset by owner.
```

3.5 Inconsistent use of uint8 for paramIndex

Severity: Info
Status: Open

Description

`paramIndex` is declared as `uint256` throughout the contract even though its expected value can be contained in `uint8`. There is however, one instance of it being declared as `uint8` in `unscopeParameter` which is inconsistant with the rest of the code base.

Location

Roles.sol#L271-L284

Recommendation

Use `uint256`instead to align with the rest of the code base.

```
    function unscopeParameter(
        uint16 role,
        address targetAddress,
        bytes4 functionSig,
-        uint8 paramIndex
+        uint256 paramIndex
    ) external onlyOwner {
        Permissions.unscopeParameter(
            roles[role],
            role,
            targetAddress,
            functionSig,
            paramIndex
        );
    }
```

3.6 Custom error is not used, remove dead code

Severity: Info
Status: Fixed

Description

Custom error `NoMembership` in `Roles.sol` is not used and is dead code.

Location

Roles.sol#L30

Recommendation

Remove `NoMembership` custom error from `Roles.sol`

3.7 Wrong or misleading comments throughout the code

Severity: Info
Status: Fixed

Description

The comment in `Permissions.sol` on line 487 says `0 -> length` but actually the value of `length` is the `length` variable in this case.

Also, the comments in `Permissions.sol` on line 501 & 567 say `//set scopeConfig` but actually different variables are set on the next line.

Throughout the codebase there are such `set scopeConfig` comments which are not actually useful.

Location

Permissions.sol#L487

Permissions.sol#L501

Permissions.sol#L567

Recommendation

Fix wrong comments and remove not useful ones.

Client Comments

Lots of comment improvements have been added around the code

3.8 scopeFunction() can be used to scope a function even when isScoped.length == 0

Severity: Info
Status: Open

Description

`scopeFunction()` currently can be called even when empty arrays are given into the function. An event will also be emitted at the end with these empty arrays. There is no reason to allow this possibility. The only use case would be to set the `role.functions[]` to an empty scopeConfig, but we have `scopeRevokeFunction()` to do that for us.

Location

Permissions.sol#L450-L523

Recommendation

We recommend adding the check, `require(length != 0)` in `scopeFunction()`.

3.9 Use underscore prefix for `internal` functions

Severity: Info
Status: Open

Description

Functions in the audited contracts do not follow solidity naming conventions. Internal functions should be prefixed with underscore for better readability. Linters like solhint would give a warning if `internal` functions do not follow this convention.

Recommendation

We recommend adding the underscore prefix for functions that are declared `internal`. For example, instead of `pluckStaticValue`, we use `_pluckStaticValue`.

3.10 Assertion is not needed

Severity: Info
Status: Open

Description

The assertion `assert(paramType != ParameterType.Static);` in `pluckDynamicValue` is not needed because the method is only called when `paramType != ParameterType.Static` anyway

Location

Permissions.sol#L720

Recommendation

Remove not needed `assert` clause

4 Additional Notes