CHAINSECURITY

PUBLIC

Code Assessment
of the Identity Network Smart Contracts

November 13, 2026

Produced for

Sky



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	6
5	System Considerations	7
6	Terminology	9
7	Findings	10
8	Limitations and use of report	12

1 Executive Summary

Dear Sky Team,

Thank you for trusting ChainSecurity with the review of the Identity Network. As a globally trusted security partner for tier-one DeFi applications, ChainSecurity provides specialized, independent technical analysis for complex smart contract systems. We greatly appreciate the opportunity to collaborate with Sky on this project.

Sky implements the Identity Network, an on-chain registry of eligible addresses that integrating contracts, such as the NFAT facility, query instead of maintaining their own allowlists. Networks can include other networks as subnets, which allows networks managed by different parties to be composed.

We have conducted a thorough review of the scope. The most important areas covered in our review are

- access control, as the networks decide which addresses are eligible in integrating contracts. We reviewed the ward and bud permissions and the handover of ownership during deployment, and only found that a deployment library does not verify the deployer, see `IdentityNetworkDeploy` **Does Not Verify the Deployer Is a Ward**, resolved in **Version 2**.
- the recursive membership resolution, as a single misconfigured subnet affects all networks including it and the contracts integrating them. We documented the assumptions on the subnet graph in **Subnet Graph Must Be Well-Formed**.
- the revocation of membership, as integrating contracts rely on it to exclude addresses, see **Removing a Direct Member Does Not Revoke Membership**.

We provide this private audit report to assist you in forming an opinion on the risk profile of the Identity Network. In summary, we find that the codebase provides a high level of security, if operated under the assumptions documented in **System Overview**, **Trust Model** and **System Considerations**.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Identity Network repository based on the documentation files. The code versions relevant to this report and when they were received are indicated below.

Version	Date	Commit Hash	Note
1	2 November 2026	c17b4547cc06022ff80e48ffa8d93cc8820851bf	Initial review
2	13 November 2026	ba181fede820f2273c3e21e0b0d99de6d2c09f5a	Deploy Fix

For the Solidity smart contracts, the compiler version `0.8.24` and the EVM version `cancun` were chosen.

The following files were in scope:

```
deploy/
├── IdentityNetworkDeploy.sol
├── IdentityNetworkDeployer.sol
└── IdentityNetworkInit.sol
src/
├── IIdentityNetwork.sol
└── IdentityNetwork.sol
```

2.1.1 Excluded from scope

All other files, including tests and external libraries, are not part of this assessment. External libraries are assumed to behave correctly and according to their specification.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Sky offers the Identity Network, an on-chain registry of eligible addresses. Integrating contracts, such as the NFAT facility, query `isMember()` instead of maintaining their own allowlists.

3.1 Identity Network

An `IdentityNetwork` holds a set of direct members and a set of subnets. Subnets are other contracts implementing `isMember()`, typically other networks with their own wards and buds. An address is a member of a network if it is a direct member or a member of any of its subnets. Hence, `isMember()` checks the direct members first and then queries the subnets recursively. This allows networks managed by different parties to be composed into larger ones. The shape of the resulting graph is not enforced on-chain, see Subnet Graph Must Be Well-Formed.

Following the usual Sky conventions, two roles manage a network:

- **Wards** add and remove wards (`rely()`, `deny()`) and buds (`kiss()`, `diss()`).
- **Buds** add and remove direct members and subnets, individually or in batches (e.g., `addMember()`, `removeSubnetBatch()`).

Both sets are stored in `EnumerableSet` instances and can be enumerated through view functions, also in ranges.

3.2 Deployment

Networks can be deployed and configured in two ways:

- In deployment scripts, `IdentityNetworkDeploy` deploys a network and transfers ownership from the deployer to a given owner. `IdentityNetworkInit` is then executed by a ward, e.g., by the pause proxy within a spell, to kiss the initial buds and add the initial members and subnets.
- The one-time `IdentityNetworkDeployer` contract does both in its constructor in a single transaction. It deploys and initializes a network, relies the given admin and denies itself, keeping no permissions.

4 Trust Model

4.1 Roles

Wards

- Trust level: Fully trusted. Expected to be Sky governance, i.e., the pause proxy.
- Permissions: Add and remove wards and buds.
- Worst-case impact: Grant bud permissions to arbitrary addresses and thereby control membership, or remove all wards and buds and leave the network without any administration.

Buds

- Trust level: Fully trusted.
- Permissions: Add and remove direct members and subnets.
- Assumptions: Buds keep the subnet graph well-formed, i.e., free of cycles, with each subnet reachable through a single path, with a small number of subnets and only containing working implementations of `isMember()`. Buds of different networks coordinate with each other.
- Worst-case impact: Make arbitrary addresses members or remove legitimate members. Add a subnet that reverts or consumes excessive gas, which makes `isMember()` revert for all addresses that are not found before reaching that subnet (DoS of integrating contracts until the subnet is removed).

Subnets

- Trust level: Fully trusted, together with their own wards and buds. Consequently, the integrity of a network depends on the access control of all subnets reachable from it.
- Permissions: Decide the membership of their own members in all networks that include them, directly or through other subnets.
- Worst-case impact: Return `true` for arbitrary addresses, or revert or consume excessive gas, with the same impact as described for buds.

Users

- Trust level: Untrusted.
- Permissions: None besides calling view functions.

4.2 External Dependencies

We assume that networks are deployed and configured correctly. In particular, after deployment, only the intended owner is a ward and only the intended addresses are buds.

Integrating contracts, such as the NFAT facility, are assumed to account for the fact that `isMember()` performs external calls, its gas cost depends on the subnet graph, and it can revert if a subnet misbehaves.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 Deployment Verification

System Consideration | Version 1

`IdentityNetworkInit.init()` only checks that the executing contract is a ward of the network. It does not verify that the network has been deployed as intended. For example, a deployment through `IdentityNetworkDeploy` may leave additional wards, buds, members or subnets, e.g., if the deployer misconfigured or deliberately prepared the network, see also `IdentityNetworkDeploy` Does Not Verify the Deployer Is a Ward.

Hence, before a network is initialized or integrated, its bytecode, its wards and buds (e.g., through the `Rely` and `Kiss` events), and its direct members and subnets should be verified.

SC2 Large Configurations Exceed the Transaction Gas Limit

System Consideration | Version 1

Adding a new member or subnet costs roughly 47k gas. Since EIP-7825 limits transactions to 2^{24} gas, a single transaction can therefore add only around 330 entries.

This affects all functions that configure a network in a single transaction:

- `IdentityNetworkDeployer` reverts if its configuration contains more than roughly 330 members and subnets in total.
- A spell executing `IdentityNetworkInit.init()` with such a configuration cannot be cast.
- `addMemberBatch()` and `addSubnetBatch()` revert for such batches.

Large configurations must therefore be split across multiple transactions, e.g., by adding the remaining entries through the buds after the deployment.

SC3 Removing a Direct Member Does Not Revoke Membership

System Consideration | Version 1

`removeMember()` and `removeMemberBatch()` only remove an address from the direct members of a network. If the address is also a member of any subnet reachable from the network, `isMember()` still returns `true` for it. Note that `RemoveMember` is emitted nevertheless, so the event does not imply that the address lost its membership. Similarly, adding or removing a subnet changes the membership of all its members without emitting an event per address.

Hence, to fully revoke the membership of an address, it must be removed from the network and from every reachable subnet. This requires coordination between the buds of all networks involved, which may be different parties. Until then, integrating contracts such as the NFAT facility keep treating the address as eligible.

SC4 Set Order Changes on Removal

System Consideration | Version 1

The direct members and subnets are stored in `EnumerableSet` instances. Removing an element moves the last element of the set into the freed position, so the order of a set may change with every removal. This has two consequences:

- Off-chain consumers enumerating a set across multiple blocks, e.g., through `getDirectMembers(start, end)` or `directMemberAt()`, may skip or double-count elements if removals happen in between.
- `isMember()` queries the subnets in set order. A removal may move an expensive subnet to the front, increasing the gas cost of lookups that were previously cheap.

The result of `isMember()` does not depend on the order.

SC5 Subnet Graph Must Be Well-Formed

System Consideration | Version 1

`isMember()` queries all subnets of a network recursively until one of them returns `true`. If any queried subnet reverts or runs out of gas, the whole lookup reverts. Hence, every subnet reachable from a network must implement `isMember()`, must not revert, and must be cheap to query.

Furthermore, the subnets must form a tree:

- A cycle, including a network being its own subnet, makes `isMember()` recurse until it runs out of gas or reaches the call depth limit.
- A subnet reachable through multiple paths is queried once per path, so the gas cost may grow quickly with the depth of the graph.

None of these properties are checked on-chain. `addSubnet()` accepts addresses without code, the network itself, and any graph shape. While some of these cases could be checked cheaply (e.g., `subnet != address(this)`), cycles spanning multiple networks cannot.

If a subnet violates these properties, `isMember()` reverts for all addresses that are not found before the subnet is reached. This blocks integrating contracts such as the NFAT facility until a bud removes the subnet.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

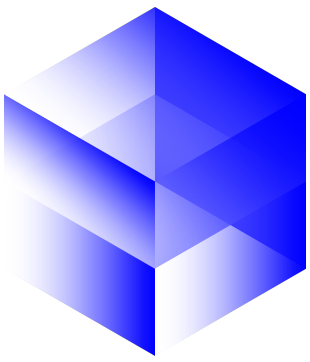
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



No Open Issues

Total number of findings

Critical	High	Medium	Low	Informational
0	0	0	0	1

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Info	#001	`IdentityNetworkDeploy` Does Not Verify the Deployer Is a Ward	Code Corrected

7.2 Descriptions

#001 `IdentityNetworkDeploy` Does Not Verify the Deployer Is a Ward

Informational **Version 1** **Code Corrected**

`IdentityNetworkDeploy.deploy(deployer, owner)` transfers ownership by relying `owner` and denying `deployer`. According to the code comment, this has the "Same behaviour as dss-test ScriptTools.switchOwner". However, `switchOwner()` additionally requires `deployer` to be a ward:

```
require(WardsAbstract(base).wards(deployer) == 1, "deployer-not-authed");
```

In `IdentityNetworkDeploy`, `deployer` is a parameter supplied by the caller, while the actual ward is the account that deployed the network. If the two differ, the actual deployer silently remains a ward: either `deny(deployer)` succeeds without effect, or, if `deployer == owner`, the transfer is skipped and `owner` is never relied.

Code Corrected

In **Version 2**, `deploy()` requires `deployer` to be a ward of the network. This covers the case where `deployer` and `owner` differ: the call now reverts instead of leaving the actual deployer as a ward.

If `deployer == owner`, the handover is still skipped as a whole, so the check does not run, `owner` is never relied, and the actual deployer stays a ward. In the actual deployment script, callers are expected to pass the deploying account as `deployer` and the correct owner.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.