Security Review Report

NM-0618-FILEVERSE
Identity SDK

October 17, 2025



NETHERMIND
SECURITY

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Issues 3

4 System Overview 3
4.1 Identity package . . . 3

5 Risk Rating Methodology 4

6 Security Approach 5

7 Issues 6
7.1 [Medium] Fields storing secret values should be private . . . 6
7.2 [Low] Incorrectly resetting secret keys . . . 7
7.3 [Best Practice] Should inform user legacy key store will not be deployed . . . 7
7.4 [Info] Debuger statement . . . 7
7.5 [Info] Only sepolia can be deployed . . . 8
7.6 [Info] Legacy Key Store has unneeded field . . . 8

8 Documentation Evaluation 9

9 Test Suite Evaluation 10

10 About Nethermind 11

1 Executive Summary

This document presents the results of the security review conducted by Nethermind Security for Fileverse's **Identity SDK**. The identity package describes the usage and storage of a Fileverse identity and the key store used to deploy permissions.

This audit focused on the end-to-end review of the `Identity` package. **The audit comprises 1552** lines of Typescript code. The audit was performed using (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report 7 points of attention, where one is classified as `Medium`, one is classified as `Low`, and five are classified as `Informational` or `Best Practices` severity. The issues are summarized in Fig. 1. **It is important to note that the codebase does not include a test suite. Implementing a comprehensive suite of tests before deployment is essential to validate the protocol's behavior and ensure its correctness. We recommend that the Fileverse team develop and maintain such a suite, covering both expected workflows and edge-case scenarios.**

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test suite evaluation and automated tools used. Section 9 concludes the document.

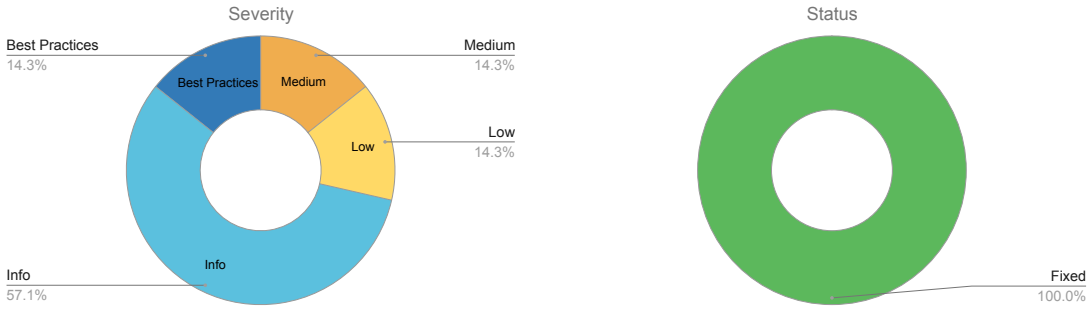


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (1), **Undetermined** (0), **Informational** (4), **Best Practices** (1).
Distribution of status: Fixed (7), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	September 19, 2025
Final Report	October 17, 2025
Initial Commit Hash	`bd95a0771948c9f2e2941a29ce0a61629b2b0b36`
Final Commit Hash	`86f89ee55a53501ac4d0334af9c7d97c1ffd9c85`
Documentation Assessment	High
Test Suite Assessment	No test available

Summary of Engagement

Dates	Methods	Consultants Engaged	Level of Effort
Sept. 10-Sept. 19, 2025	Manual	2	2-person weeks

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/identity/key-store-utils.ts	284	7	2.46%	45	336
2	src/identity/keystore-manager.ts	265	0	0%	37	302
3	src/identity/identity.ts	233	0	0%	45	278
4	src/identity/contract-utils.ts	180	1	0.55%	34	215
5	src/identity/key-store.ts	168	0	0%	32	200
6	src/identity/legacy-key-store.ts	159	2	1.26%	29	190
7	src/identity/types.ts	142	0	0%	21	163
8	src/identity/key-derivation-utils.ts	43	0	0%	8	51
9	src/identity/constants.ts	27	0	0%	11	38
10	src/identity/ucan-utils.ts	26	0	0%	2	28
11	src/identity/viem-client.ts	16	0	0%	2	18
12	src/identity/index.ts	9	0	0%	2	9
Total		**1552**	**10**	**0.64%**	**266**	**1828**

3 Summary of Issues

	Finding	Severity	Update
1	Fields storing secret values should be private	Medium	Fixed
2	Incorrectly resetting secret keys	Low	Fixed
2	Should inform user legacy key store will not be deployed	Best Practice	Fixed
3	Debbuger statement	Info	Fixed
4	Only sepolia can be deployed	Info	Fixed
5	Legacy Key Store has unneeded field	Info	Fixed

4 System Overview

4.1 Identity package

The identity system enables users to establish secure digital identities within Filverse dsheets. Each identity is derived from a password using Argon2id hashing in combination with HKDF key derivation, resulting in the generation of three cryptographic keys: a master key for encryption, a signing key for authentication, and an agent key for blockchain operations. These keys are stored using ECIES encryption, ensuring forward compatibility with new document formats while maintaining support for legacy types.

New identities are initialized through the deployment of an Identity Module smart contract. As part of this process, the system generates portal contracts to facilitate data sharing, deploys permission contracts to enforce access control, and uploads encrypted keystore data to IPFS for persistence. For existing identities, the system reconstructs the user's configuration by querying the BeaconFactory to retrieve the associated Identity Module, while authentication is managed through UCAN tokens that carry a fixed validity period of seven days.

The architecture relies on a coordinated set of smart contracts that manage identity, permissions, and data flow. The BeaconFactory contract is responsible for the creation of Identity Modules, while the IdentityModule contract provides storage and retrieval of user keys. V2Portal and V2Registry contracts are employed to handle data sharing, and dedicated Permissions contracts define and enforce access rights. This framework is designed for deployment on both the Sepolia testnet and the Gnosis mainnet, ensuring scalability and interoperability across environments.

Storage is split between on-chain and off-chain layers. On-chain, the system maintains identity modules, public keys, and encrypted private keys. Off-chain, IPFS is used to store encrypted keystore data, with content hash addressing providing verifiable integrity. Multiple IPFS gateways are supported to ensure redundancy, and content hash types determine the correct decryption methods to be applied.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		Low	**Medium**	**High**
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Security Approach

This section describes the approach Nethermind Security team took to assess the security of the Identity SDK. To do so, the team assessed the following key areas of concern:

Cryptographic Usage and Derivation – Verification that all cryptographic primitives exposed by the SDK are modern, follow NIST standards, and correctly implemented. This included reviewing key generation and that storage is handled securely, and confirming that key derivation functions (KDFs) are applied consistently when deriving multiple keys from shared material. The team also validated that the SDK enforces authenticated encryption and constant-time comparison operations to mitigate common classes of attacks.

Integration Logic and Misuse Resistance – Evaluation of how the SDK prevents or mitigates common integration errors made by developers. This included analyzing the default configuration, error handling, and exposed APIs to determine whether insecure usage patterns are possible. The team assessed whether the SDK constrains unsafe behaviors by design and whether documentation and provided examples reinforce correct usage.

The team discussed internally and with the **Fileverse** team in meetings to correctly understand the goals, architecture, and usage of the SDK in the wider context. Through this, it was possible to correctly and authenticate the codebase.

7 Issues

7.1 [Medium] Fields storing secret values should be private

File(s): identity.ts, key-store.ts

Description: It is best practice to keep private keys and generally sensitive information in private fields rather than public and outline specific getters/setters to restrict unwanted access and modifications to these values and only allow specific pre-determined actions with them. This is because if secrets are public, any consumer of the SDK (or even malicious code in the same runtime) can directly read, and misuse the keys.

Recommendation(s): Set the following to be private fields:

```
 // identity.ts
class Identity {
  private privateKey: Uint8Array | undefined;
  private signingKey: EdKeypair | undefined;
  private agentKey: Uint8Array | undefined;
  ...
}
// key-store.ts
class KeyStore {
  private appKeyPair: EciesKeyPairBytes | undefined;
  private ownerSecret: string | undefined;
  ...
}
```

Additionally, the private keys should not be exposed but should instead expose the signing or decrypting functions only. Specifically in identity:

```
// identity.ts
// @audit Make this function private and should expose only decryptEnvelope function
getPrivateKey() {
    if (!this.privateKey) throw new Error("Private key not initialized");
    return this.privateKey;
}
```

Status: Fixed

Update from the client: Merged in PR 1.

7.2 [Low] Incorrectly resetting secret keys

File(s): legacy-key-store.ts, key-store.ts

Description: When resetting the private key it is insecure to set to undefined as the private key value is still retained in memory until it is rewritten eventually or the garbage collected. This allows side channel vulnerability of a malicious user being able to read a recently reset private key of an identity. Especially as this is running on the client side it is possible to read the memory of an identity once they have reset.

```
// key-store.ts
resetKeyStore() {
    this.appKeyPair = undefined; // @audit should not be undefined but instead should be set to 0's
    this.ownerSecret = undefined; // @audit should not be undefined but instead should be set to 0's
    this.portalAddress = undefined;
    this.permissionAddress = undefined;
    this.ucanToken = undefined;
}
```

This is also the case in 'legacy-key-store.ts' when it is reset:

```
 // legacy-key-store.ts
resetKeyStore() { // @audit should not be resetting the keys like this
    this.ownerKeyPair = undefined;
    this.memberKeyPair = undefined;
    this.portalKeyPair = undefined;
    ...
}
```

Recommendations(s): Instead of setting the secret keys to undefined, set to a 0 value and then set to undefined as this will fill the memory register with 0's and overwrite the existing private key .

Status: Fixed

Update from the client: Removed resets in commit 86f89ee55a53501ac4d0334af9c7d97c1ffd9c85.

7.3 [Best Practice] Should inform user legacy key store will not be deployed

File(s): legacy-key-store.ts

Description: In `legacy-key-store.ts`, there exists the deploy function which returns nothing and performs no action. This is bad practice as a caller of this function does not know that nothing was deployed.

Recommendations(s): We recommend to return an error or to inform the user that the legacy key store will no longer be used.

Status: Fixed.

Update from the client: Fixed in commit 86f89ee55a53501ac4d0334af9c7d97c1ffd9c85.

7.4 [Info] Debuger statement

File(s): key-store-utils.ts

Description: A debugger statement is present in the production codebase within the `getKeyStoreValueByAddress(...)` function.This statement will halt JavaScript execution when developer tools are open.

Recommendations(s): Remove the debugger statement before any production deployment

Status: Fixed.

Update from the client: Removed debugger in commit 86f89ee55a53501ac4d0334af9c7d97c1ffd9c85.

7.5 [Info] Only sepolia can be deployed

File(s): contract-utils.ts, constants.ts

Description: When creating a new identity in `contract-utils.ts/createNewIdentity(...)` only deploying on Sepolia is allowed even though Gnosis is also an option.

```
const trxRecipt = await agentInstance.executeUserOperationRequest(
    {
      contractAddress: BEACON_FACTORY_ADDRESS, // @audit only allows for sepolia
      data: encodedCallData,
    },
    DEFAULT_TIMEOUT
);
...
const publicClient = getPublicClient({
    chain: AllowedChain.Sepolia,
    rpcUrl: "",
});
```

Additionally, this problem also exists in `constants.ts`:

```
export const BEACON_FACTORY_ADDRESS =
  BEACON_FACTORY_ADDRESS_MAP[AllowedChain.Sepolia];
```

Recommendations(s): Add as input parameter which chain to deploy on and add ability to deal with both.

Status: Fixed.

Update from the client: Added Gnosis support in commit 86f89ee55a53501ac4d0334af9c7d97c1ffd9c85.

7.6 [Info] Legacy Key Store has unneeded field

File(s): legacy-key-store.ts

Description: The legacy key store class has a field called `isLegacyUser: boolean`. This is redundant since for this key store to exist it must be a legacy.

Recommendations(s): Remove this field.

Status: Fixed

Update from the client: Removed field in commit 86f89ee55a53501ac4d0334af9c7d97c1ffd9c85.

8 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. SDK's can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the SDK's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the SDK. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the SDK. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the SDK. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the SDK was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the SDK. This type of documentation is critical in ensuring that the SDK is secure and free from vulnerabilities.

These types of documentation are essential for SDK development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Fileverse's documentation

`Fileverse's` team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

9 Test Suite Evaluation

Remarks about Fileverse's tests

The project currently lacks a test suite to validate the protocol's functionality. **A comprehensive test suite** is a critical step in ensuring the reliability and security of any protocol. Tests serve as the first line of defense against implementation errors, catching issues early before they can impact production.

They provide confidence that core functionalities behave as intended, help verify the handling of edge cases, and make future upgrades safer by preventing regressions. The absence of tests, as observed in this codebase, significantly increases the risk of undetected bugs, hinders maintainability, and reduces the overall assurance in the protocol's correctness.

10 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE 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 OTHER ADVICE.