Ledger Fresh Wallet v1.0

Product Overview

Ledger Fresh

Not Your Keys, Not Your Coins

Content

Document Information

Product version

Release Date

Revision

Initial Release

Disclaimer

(copyrights, sub licenses)

This document provides an overview of the Ledger Fresh Wallet system. It contains the following chapters:

List Of Images

List Of Tables

The Ledger Fresh Wallet

This chapter provides an overview of the Ledger Fresh product.

Introduction to Ledger Fresh

Ledger Fresh Wallet is a secure web browser application designed to protect crypto assets and transactions

Wallet Basics

Ownership

The main property of a crypto wallet is the ownership of accounts. With a wallet, a user is in control of its assets, opposed to centralized exchanges managing the security of the assets as done by the traditional banking system.

Transactions

Transactions is the basic command for users to interact with a blockchain. A classic example is a payment from a user account to another, but can in fact be any action allowed by the concerned blockchain syntax (swap, NFT selling/acquisition, advanced smart contracts). A crypto wallet secures the emitting of those transactions between accounts.

Protect keys

Transactions are protected by the wallet containing the secret key linked to the user account. The assets are not 'stored' in the wallet, which are rather the equivalent of the smartcard of a credit card. Wallets protect the issuing of transactions the same way by **signing** transactions**.** The compromission of a wallet is more severe than a credit card thus, as no warranty protects the assets in case of compromission.

Security vs User Experience

Ledger Values Statement

Ledger values are **security** (resilience by design), **transparency** (don't trust, verify) and **sovereignty** (not your keys, not your coin). By combining improval of smartcards for the next internet era with the will to provide full control to the users, Ledger produced the most secure and agile hardware wallet, the **Nano**. Ledger has done so in three distinct ways:

1. By removing the need to have a card reader to use a smartcard. It can now be directly connected to the USB port of a computer or used over Bluetooth.

2. By providing an open Operating System, BOLOS (Blockchain Open Ledger Operating System) lets developers load their native code into the device and users pick applications from a thriving ecosystem.

3. By adding a screen and buttons connected directly to the smartcard to prevent malware from changing the information displayed to the user or faking user consent, thus ensuring that *What You See Is What You Sign*.

Ledger Fresh Vision

A look back on ourselves

While security will always be our North Star, we collected feedback about Ledger devices and found a few user pain points:

1. Onboarding your Ledger Nano devices requires writing down a list of 12/24 words (the mnemonic) to back up the private cryptographic keys. This backup is critical to recovering the assets if the device is lost and for interoperability with multiple wallets. As those words unlock a lot of power and value, attackers are getting more creative in tricking users into giving them.

2. When using a Ledger device, the user always performs the same actions regardless of the transaction amount. This process can lead to fatigue and a lack of attention to detail for more important transactions.

3. As users spend more time transacting cryptocurrencies, NFTs, and navigating Web3, they may move to different governance solutions (such as multi-signature) or plan for an inheritance. With current solutions, this is complex and costly as this requires moving all assets to a new wallet.

A Smarter Wallet. A Better Web3 Experience.

How to solve these pain points? A smart wallet that can run its independent logic lets users customize part of this logic. With *Ledger Fresh*, you can interact with DApps seamlessly without the need to approve all your transactions on your hardware wallet and without compromising on your digital security. If you're a new user, *Ledger Fresh* will open the possibility to enter the

Ledger ecosystem without a Ledger device while gradually increasing your safety as your usage grows. Here are a few scenarios on how Ledger Fresh could address the pain points we've just examined:

1. **A revamped onboarding process.** The new onboarding phase would let users register different devices to the smart wallet. The recovery phrase would only be used once if all devices are lost, after giving notifications on all possible media and a grace period to cancel the operation to the user.

2. **Personalized security profiles.** Smart wallets could set distinct security profiles. For instance, one profile for daily use cases and another for specific use cases. This process could be done through allowlists or more complex policies linked to sources of truth operated off-chain by the user.

3. **Updatable standards.** The smart wallet logic would be updatable/upgradable on the fly without moving assets to a new wallet implementing new features.

Ledger Fresh Wallet - Overview

Ledger Fresh features

Flexibility through Account abstraction

Account Abstraction is a long-term goal of the Ethereum ecosystem to use the execution capabilities of a blockchain to implement the account verification logic instead of just implementing the logic of applications. Account Abstraction replaces the traditional concept of accounts associated with keys by **account smart contracts**, making it possible to use custom verification logic (for example, different signature schemes) and to perform additional checks such as using an allowlist of contracts and functions.

It has been partially implemented on Ethereum, with Argent pioneering its use since 2018. There are currently several competing specifications (EIP 4337, EIP 3074 and EIP 5806) being reviewed to move to full implementation, enabling, for instance, to pay for network fees using different assets (e.g any ERC-20 token), or define transactions orders.

Unfortunately, even partial implementations of Account Abstraction didn't get a lot of traction because of the related network costs, notably when gas prices surged during Defi Summer in 2020 and then during the NFT era in 2021. This will change with a growing range of efficient L2 solutions aiming to reduce gas costs.

Layer 2 efficiency: Starknet

Starknet is a scalability second layer released in 2021 by Starkware. It allows to compute more expensive transactions on that layer and only submit a brief proof of proper execution to be verified by the settlement layer of the rollup, Ethereum.Given the substantial savings, Starknet uses an Account Abstraction model for all users. Interestingly, as Starknet execution model is not based on the Ethereum Virtual Machine, complex mathematical operations are also less expensive, and it is thus possible to implement a validation of WebAuthn signatures at the application level. Cartridge has pioneered this.

Thanks to Starknet, we can run an acceptable cost Account Abstraction model with several devices that can authenticate to the account smart contract, including smartphones implementing WebAuthn in the browser.

Phishing resilience through FIDO2:WebAuthn

In 2012, FIDO defined a web-native protocol using cryptographic signatures to replace standard Second Factor Authentication (2FA) and One Time Password algorithms using a shared secret. While not protecting users against malware, FIDO helps solve phishing attacks and attacks targeting the server credentials database.

FIDO support was initially only provided by dedicated devices such as Yubikeys, but things got more interesting in 2019 when Apple and Google started including FIDO 2 (aka WebAuthn) support directly in their devices, with keys, signatures, and authorizations backed by the hardware security features of the platform - the Secure Enclave and Face ID for Apple, StrongBox and biometric authentication for Android.

We now have a solid and reliable cryptographic authentication mechanism available on most smartphones that can be used from the browser.

It would be great to use it as an everyday signer with additional restrictions in the user account contract. Currently, verifying WebAuthn signatures on-chain is too costly since the only curve supported by the FIDO2 compliant hardware solutions is secp256r1, the neighbor of Ethereum secp256k1; most chains don't natively support it.

Total User control: Account Plugins system

Account plugins are a flexible mechanism to extend an account smart contract logic with pluggable logic (plugin). The reference implementation is currently being worked on jointly by Argent, Cartridge and Ledger.

Account plugins allow updating mechanisms to validate transactions on the fly. Different signing mechanisms (such as WebAuthn) or security schemes (such as reduced signature friction for games or social applications with session keys) can be added to an account smart contract

supporting the plugin architecture after it has been instantiated, without having to move assets or create a new wallet.

Malware resilience through NANO DEVICE

Ledger Services Assisted: FreshCheck

FreshCheck is a backend ledger service which aims at verifying the properties of transactions to assist the user. The aim is to protect users from missigning hastily dangerous transactions related to a user predefined security policy. The service is completely configurable and users are given the possibility to install its own FreshCheck server. For instance it may be to block transactions with high amount with low security signers, or transactions leading to a bad/instable configuration of the wallet (remove all signers, install a dangerous plugin).

Services Overview

Ledger Fresh builds a security-oriented web wallet interface interacting with an account smart contract on Starknet with different security validation mechanisms (such as creating an allowlist or relying on an external source of truth) depending on the device interacting with the account. This can be, for instance, either a WebAuthn device or a Ledger device. Moreover, this wallet can be enhanced/extended by external plugins.

Without a Ledger device

When starting his journey without a Ledger device, the user can register different WebAuthn devices (mobile phone, laptop, etc) to *Ledger Fresh*.

Ledger Fresh will provide additional guidance when an operation is risky, or the user account value is significant enough to get protected against malware with a Ledger device.

Switching between security profiles or modifying the current security profile will be performed with a customizable delay and notifications to allow users to cancel the operation if it was initiated by malware.

Adding a Ledger device

Adding a Ledger device provides full protection against malware - thus security operations can be confirmed instantly on device (What You See Is What You Sign). The user will also be able to bypass security restrictions set to the account (for example a spent amount per day) if and only if using a Ledger device.

Future *Ledger Fresh* services

In the future, we plan to extend the support of *Ledger Fresh* - both by connecting other chains to Starknet and by supporting the same concept on different chains if it is economically viable. Additional services will also be implemented as plugins, such as a multi-signature service or a trustless legacy service allowing to send crypto to a predefined address if the account has been idle for some time.

System Overview

The following figure provides an overview of the Fresh Wallet System and Subsystems.

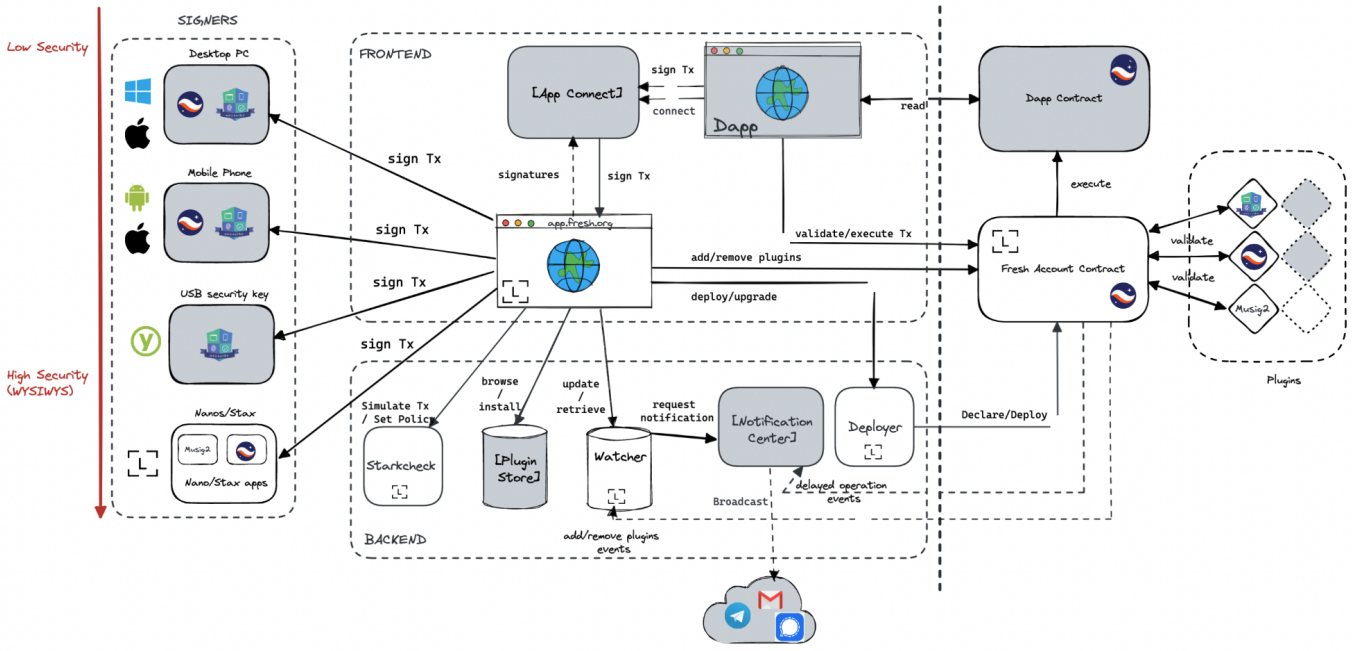


Figure 1. Fresh System and subsystems

The Ledger Fresh System is made of the following subsystems:

- The ledger Fresh Application, which includes the UI, the user's policy (locally stored) and the mechanisms to enable interactions between the user and all other subsystems
- Signers subsystem with their own front end (Android/IoS App, Bolos), hardware (IoS/Android enclave, Nano Secure Element) and display (Unsafe/Trusted)
- External Dapps
- Fresh Account : the Fresh On Chain Contract, deployed over Starknet with an instance for each user
- Back End services:
    - Fresh Deployer: deploy initial Fresh Account contract on Starknet

- Plugin Store : display the list of available plugins
- FreshCheck : control the security policy of received transaction according to user defined policy
- Notification System : independent notification system to warn users of actions.

Dapp and Signers specifications are out of the scope of this document. All Nano specifications and source code is available through ledger developer portal [LEDGER-DEV]. Each other subsystem is described in a related chapter in the given order.

Ledger Fresh Wallet - User actions

This section describes the main actions an user can perform to interact with the wallet, and the related processing by the Fresh subsystems.

Installation : account deployment

The first action performed at first connection to AppFresh by the user is the deployment of a new account as described in the following figure. The user will create a new keypair for AppFresh using WebAuthn application. It is assumed that a WebAuthn compatible signer is used for this enrolment. The corresponding **FIDO2_PUBLIC_KEY** is given to the Deployer, which is in charge of writing the user account contract over Starknet.

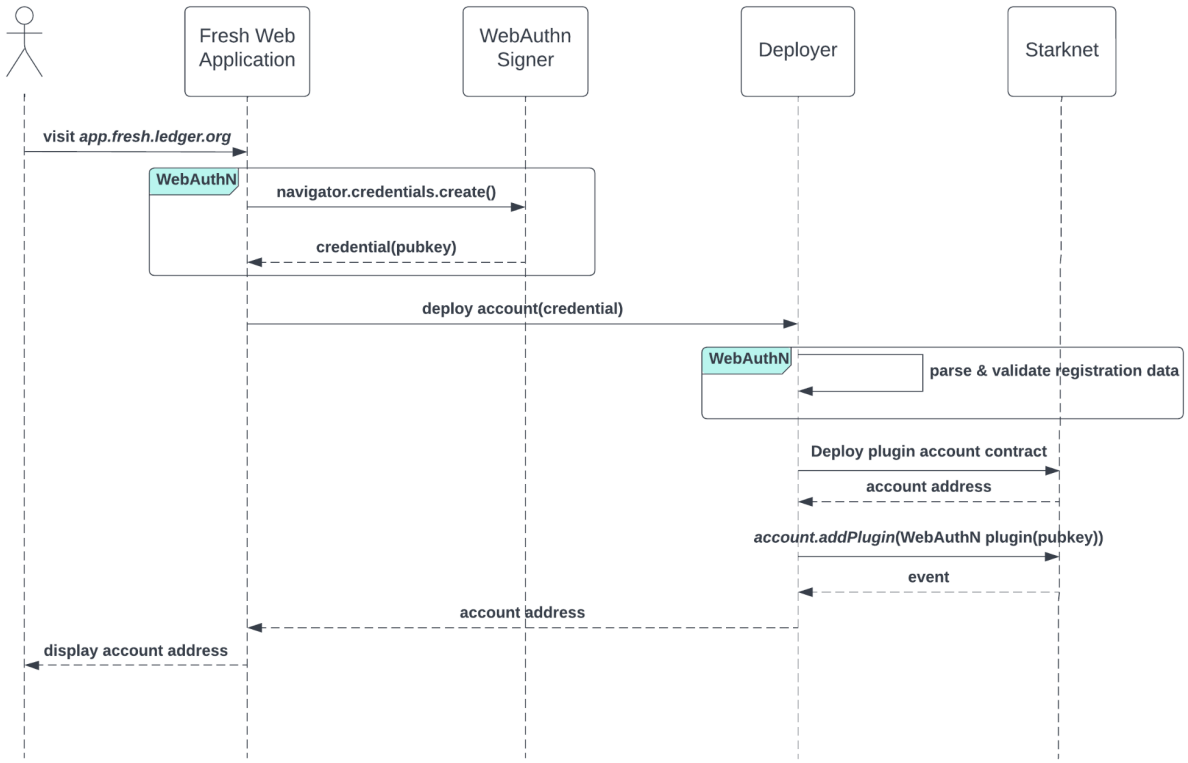


Figure 2. account deployment overview

Configuration : Policy

A policy is proposed by default to the user as described in the next chapter. The policy is displayed in one of the tab of the application. The policy can be configured by checking the options of the tab, and via 'a new policy definition' button. Each time the policy is modified, the modification content is displayed on the configuration signer for user validation. The policy is saved locally on the device.

Installation : restore account

When a user connects to AppFresh, the state is restored through local files.

Configuration : Signer

Configuration : plugins

Signing a basic transaction

How a transaction is processed.

(diagram interaction with Signer, Contract, User Policy, FreshCheck)

Signing with Session Key

Create Session Key

Ledger AppFresh

This chapter describes the Ledger Fresh WebApp:

- The first section describes the architecture and lifecycle of related elements (account, plugins, transactions).
- The second section describes the user APIs and possible interactions with the Wallet.
- The third section describes the external APIs of each connector with an external subsystem

AppFresh Overview

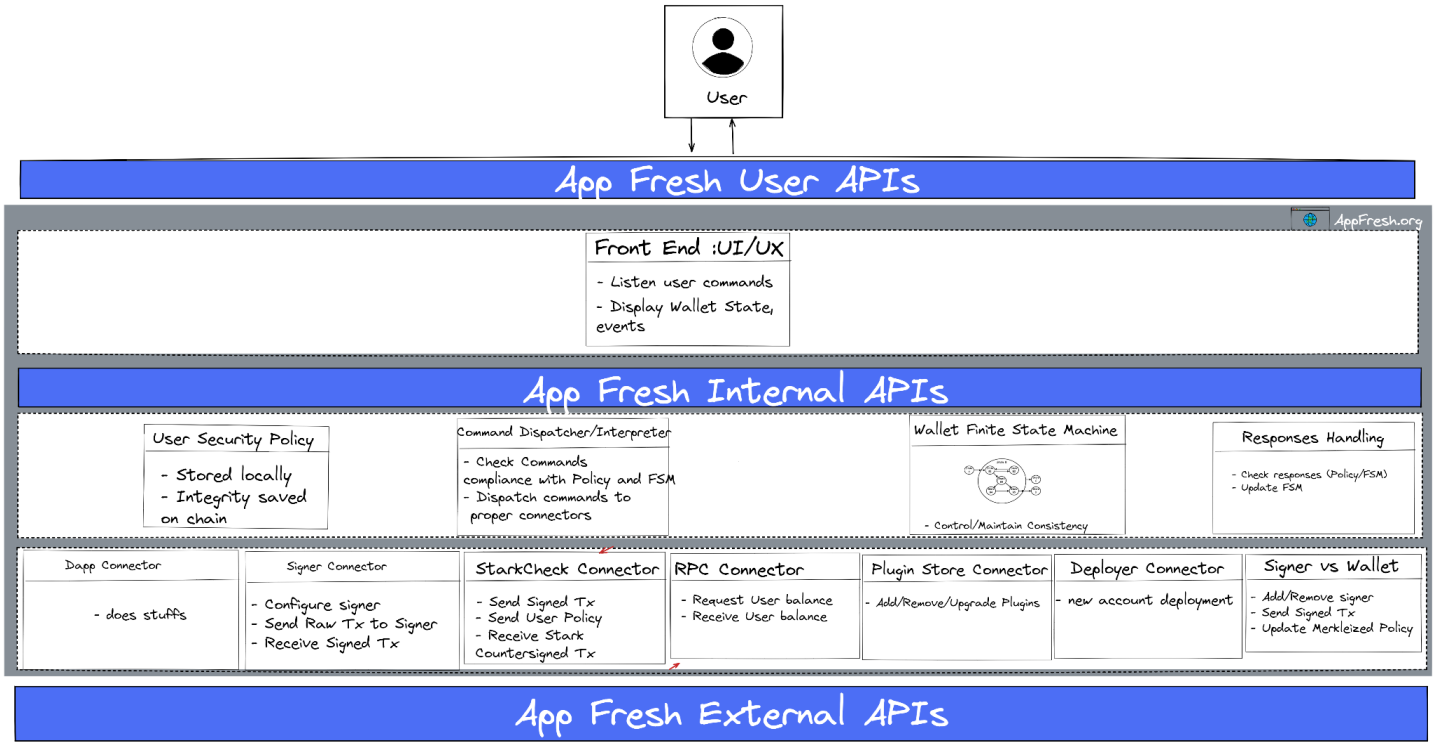


Figure 3. AppFresh Components and APIs

FreshApp works in interaction with the users by displaying information and events (via the FrontEnd) and processing users **commands** provided via the FreshApp IHM. All commands are translated to **Transactions** which are then signed by one of the user's signers. Transitions are then sent to FreshCheck for **policy** compliance verification.

Commands

The lists of all users available commands is defined in the following table:

Command	Parameters	Action
[CMD_USER_ADDPLUGIN]		Add plugin with given classhash
[CMD_USER_RMPLUGIN]		Remove plugin with given classhash
[CMD_USER_SESSIONKEY]		Create a session key for the given duration and classhash
[CMD_USER_ADDSIGNER]		
[CMD_USER_RMSIGNER]		

Table 1. [TAB_USER_COMMANDS]: list of user's available commands

Once a command is given:

- the command is checked to be compatible with the current Fresh State,
- the command is checked to be compatible with the internally stored policy,
- the command is expressed as a transaction to be signed by one of the external signers (WebAuthn or StarkSigner),
- the command is dispatched to the proper connector to be processed by the external subsystem.

Policy

A policy is a set of rules selected by the users.

Initial Policy

Rule	Description	Motivation	

[REJECT_PLUGIN_CRL]	Reject Transaction if(Plugin_ClasHash in PLUGIN_CRL_LIST)	Reject Plugin identified as unsafe in PLUGIN_CRL list	
[REJECT_PUBKEY_CRL]	Reject Transaction if(Plugin_ClasHash in PUBKEY_CRL)	Reject Tx identified as malware/ransomware destination PUBKEY_CRLin	
[DELAY_HERITAGE]		Allow signing of configuration transactions using HEIR_PRIVATE_KEY	

Table 2. Initial FreshCheck Policy

StarkSigner enforced Policy

The rules defined in this section only once a StarkSigner has been added through the ADD_STARKSIGNER transaction (TBD). It is assumed that the StarkSigner uses a hardware device, or a signer with a high level of trust. As such it has more power than the initial WebAuthn signer once installed.

Rule	Description	Motivation
[DELAY_AMOUNT_OVERFLOW]	Delay transaction if signing_key!=**FreshCheck_PRIVATE_KEY and TX_AMOUNT>CST_AMOUNT_OVERFLOW**	Limit the amount of WebAuthn signer to daily use.
[DELAY_ADMIN]	TimeGate a configuration request	

Table 3. StarkSigner Enforced Policy

FSM and Lifecycles

This section describes the management of the state of AppFresh.

Power-On

When launching (opening APPFRESH_URL)

Transactions

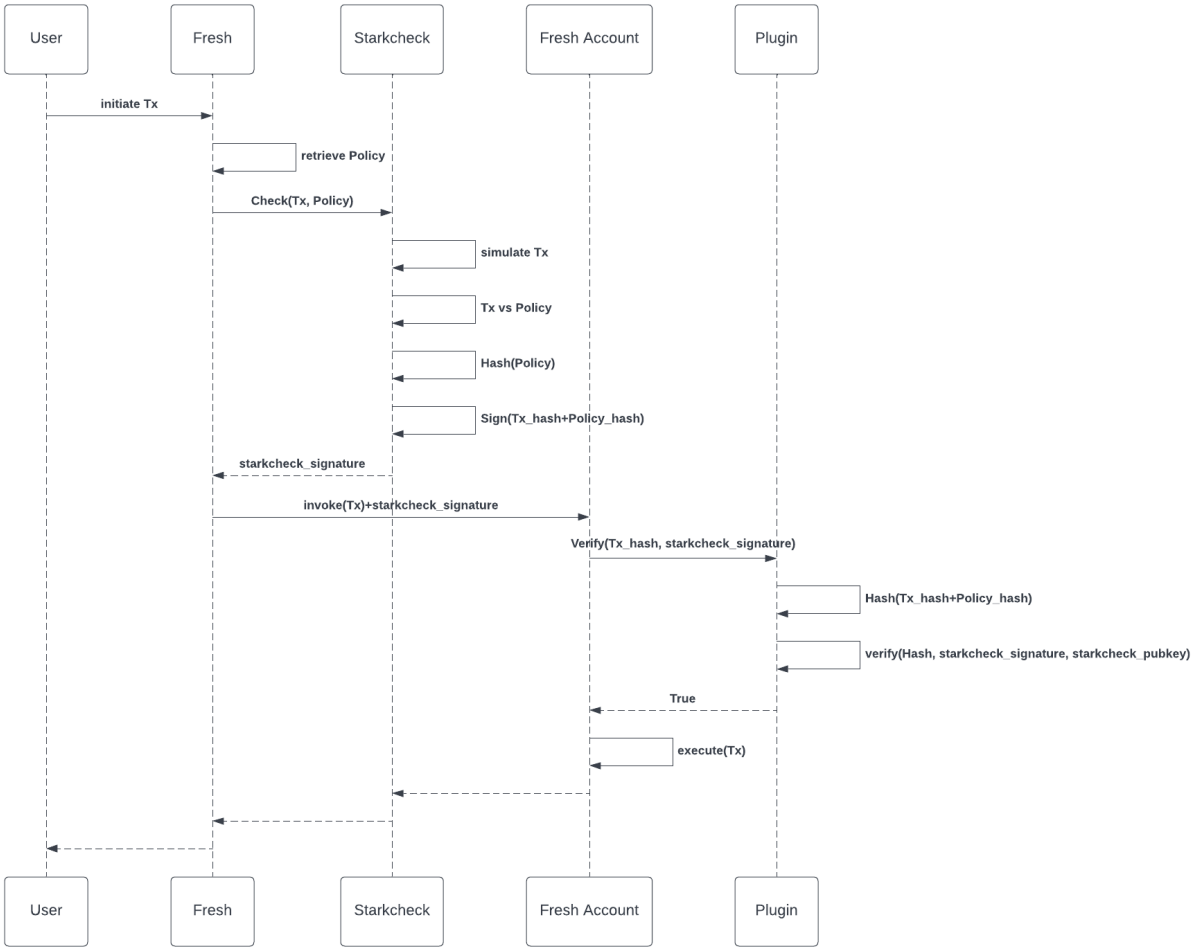


Figure 10. Tx lifecycle

Policy

A policy is a set of rules selected by the users.

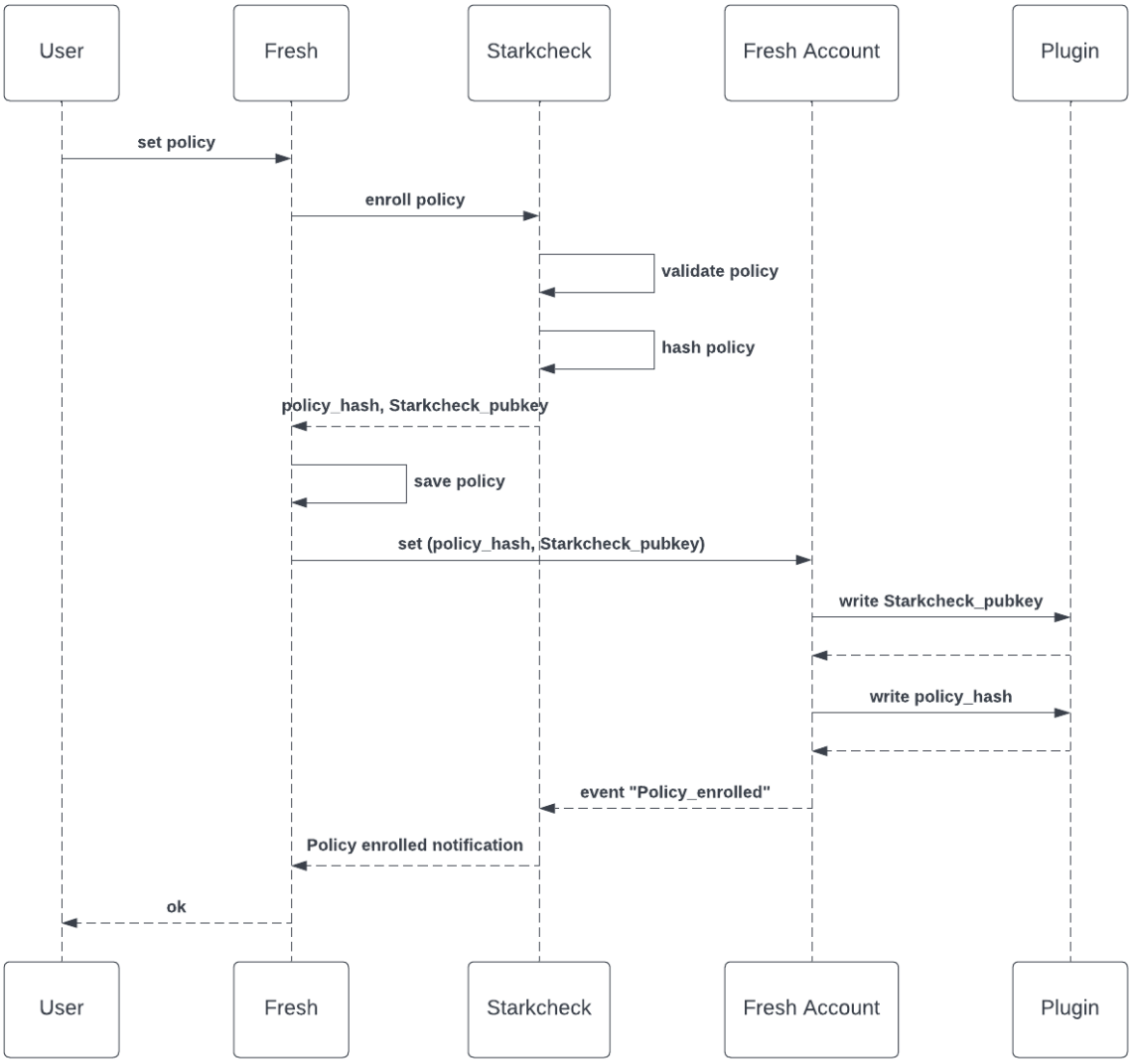


Figure 4. setting up user policy

AppFresh Requirements

Front End- Requirements

[REQ_FRONT_BALANCE]

The Front End shall display the user's ERC20/721/1155 balance.

When the app is properly configured with a functional account, the application makes a call to the RPC via the RPC connector and displays the resulting balance.

[REQ_FRONT_MODES]

The Front End shall implement two display modes.

The two display modes intention is to warn the user about the execution security contex:

- The normal mode - the user interacts with his smart-contract wallet with a WebAuthn authenticator
- The safe mode -- the user interacts with his smart-contract wallet with a starknet authenticator

Both of them would have their own theme, which means the UI would be slightly different.

One of the big changes between the two modes would be the presence of advertising and educational content. When the user would use the app in normal mode, some content that educates the users about the importance of a Ledger device would be displayed (why a hardware wallet is important; what is the difference with a non-ledger authenticator; what is a seed phrase, etc...). In addition to that, it isn't impossible that some raw ads for our products would be displayed too.

[REQ_FRONT_ONGOING]

The Front End shall display the ongoing transactions.

Ongoing Tx are stored locally on the host. When AppFresh is launched, all ongoing transactions

[REQ_FRONT_POLICY]

The Front End shall display all policy rules selected by the user.

The user policy is stored locally. While launching AppFresh, all rules are read to be displayable with their details (actions enabled, contract address) in the rule tab.

[REQ_FRONT_COMMANDS]

The Front End shall display a command button for all commands defined in [TAB_USER_COMMANDS].

[REQ_FRONT_OPT]

The Front End shall display an option button for all options defined in [TAB_USER_OPT].

Command	Parameters	Action
[USER_OPT_]		

Table 4 [TAB_USER_OPT]: list of user's available options.

[REQ_FRONT_EDUCATIVE]

The Front End shall display an educative message when an event described in [TAB_USER_WARNING_EDU] is triggered.

Internal Requirements

Restore state

[REQ_RESTORE]

The Wallet shall be able to restore the state of the wallet, including:

- Ongoing transactions
- User's balance
- Policy

Policy

[REQ_POLICY_STORED]

The Policy shall be stored on the host of the application.

[REQ_POLICY_UPDATE]

The policy shall be updatable through user's command

[REQ_POLICY_APPEND]

The Policy shall be appended to all transactions before being sent to the signer.

[REQ_POLICY_SAVE]

The Policy shall be easily transferred to another device.

[REQ_POLICY_RECOVER]

[REQ_POLICY_MERKLE]
The Merkle root of the policy is stored on chain.

RPC connector- Requirements

[REQ_RPC_COMPATIBILITY]
The RPC Connector shall be compatible with the selected RPC solution.

Choice is still to be made between:

- Infura already supports Starknet (docs)
- Alchemy plans to support Staknet (docs)

[REQ_RPC_REQUEST_BALANCE]

The RPC Connector shall display the user's ERC20/721/1155 balance.

Dapp connector (App connect)- Requirements

TO BE DONE

Since Ledger Fresh is a web wallet (not a browser extension), we need to find an innovative way to enable the connection between the web wallet and any dApps. he solution must be focused on the user experience (keeping in mind the mobile experience) and must be as smooth as possible.

The dApp connection module will allow any dApp to request a wallet interaction (like signing a message). The choice of the signer is done wallet-side and should not be communicated to the dApp. The dApp should only interact with the web wallet without considering the signer choice.

`get-starknet` could be an option to explore. Instead of developing our own in-house solution, that would be great to leverage on top of this project in order to release an open-source that will benefit the community.

FreshCheck connector- Requirements

```
[REQ_FreshCheck_SEND]
All transactions signed by a signer shall be sent to FreshCheck for
countersigning (double signature), unless [OPT_DISABLE_FreshCheck] is
selected.
```

```
[REQ_FreshCheck_CONFIRMED]
All transactions countersigned by FreshCheck shall be send to the
Fresh Account Connector
```

```
[REQ_FreshCheck_REFUSED]
All transactions refused by FreshCheck shall be displayed as policy
violating to the user.
```

The user is then given the choice to send the Tx anyway. The transaction will be delayed before being processed.

Signer connector- Requirements

```
[REQ_SIGNER_SUPPORT]
The following signers shall be supported by Appfresh:
- Webauthn signers : Desktop PC, Mobile phone (IoS and Android), USB
Key (Yubikey)
- Stark signers
```

Deployer connector- Requirements

[REQ_DEPCON_CHECKPUBK_EXISTS]
A RPC call shall be made to check the availability of the target public key on Starknet before call to the Deployer.

AppFresh User API

Describes the User API here

AppFresh Internal API

Describes the internal API here

AppFresh External API

Describes the API with External Subsystems here.

RPC connector

Pluggin Connector API

A plugin is defined by the following object:

```
type Plugin = {
  id: string,
  classhash: string,
  signature: string,
  name: string,
  description: string,
}
```

The template for the id string is `<author_github_handle>:<plugin_name>.` The value `<author_github_handle>` is appended to the developper submitted plugin name by the Plugin store.

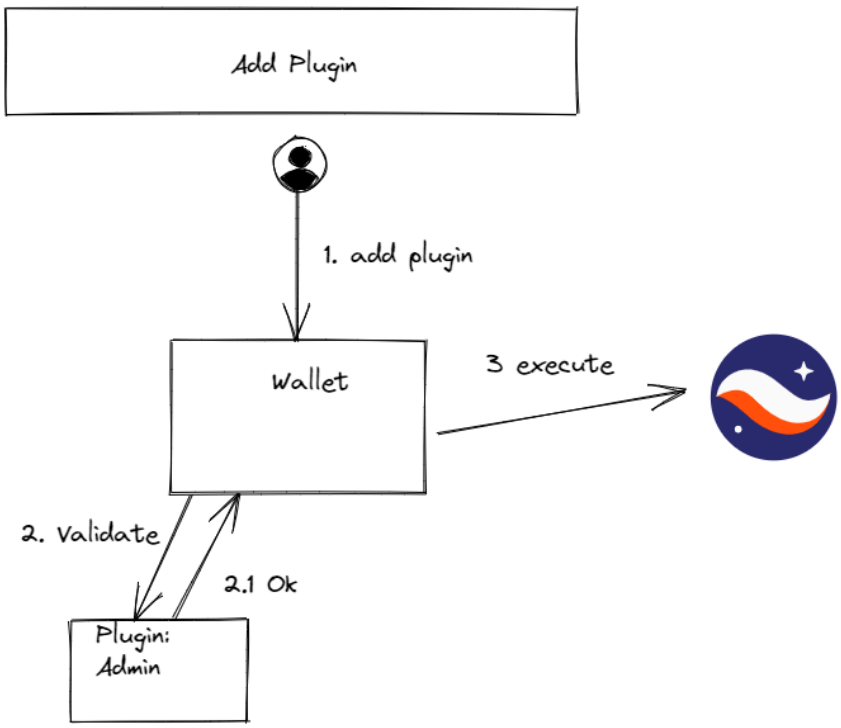
Update Plugin

Configure Plugin

Change FreshCheck

Change FreshCheck Policy (OffChain)

Execution transactions



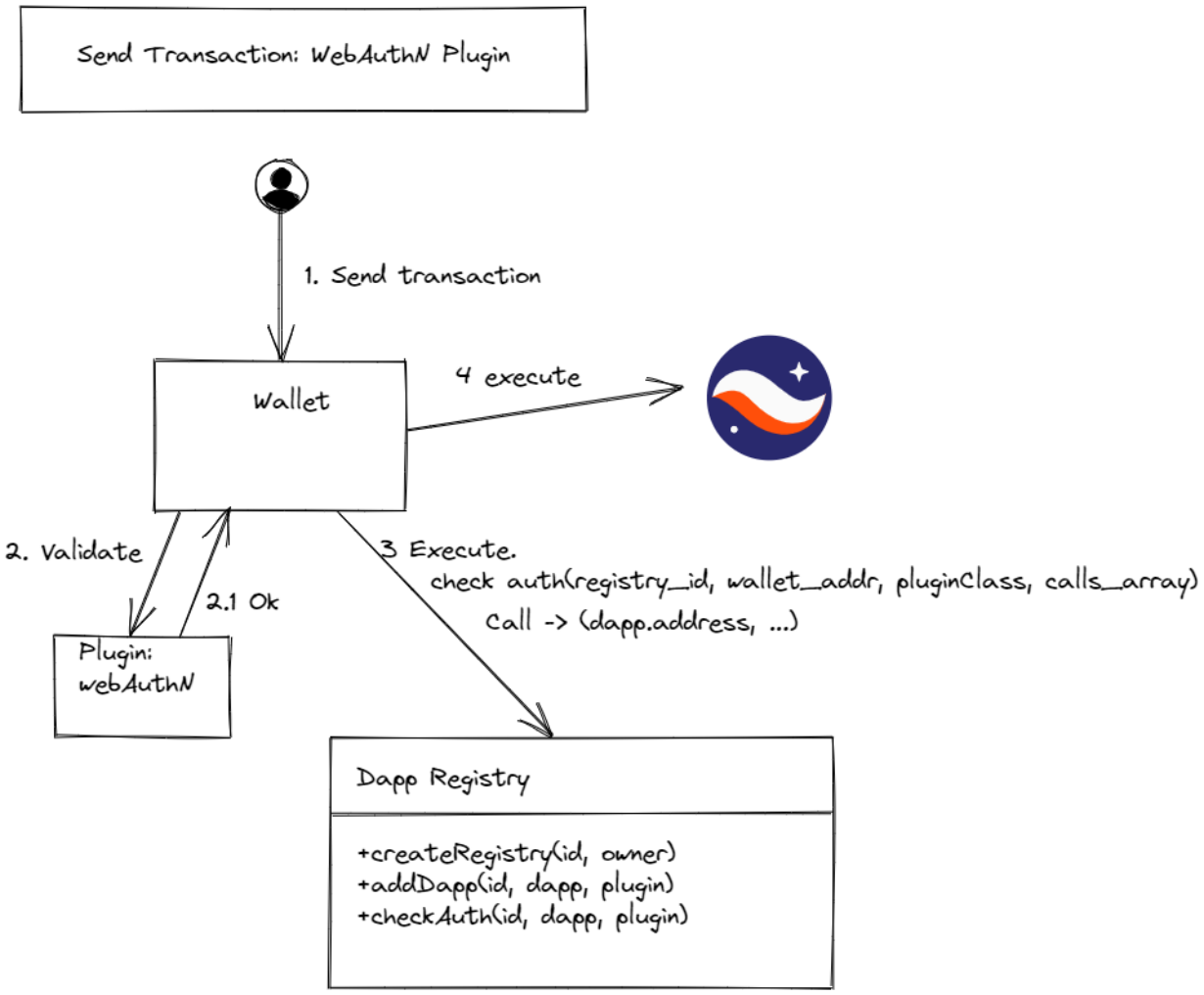
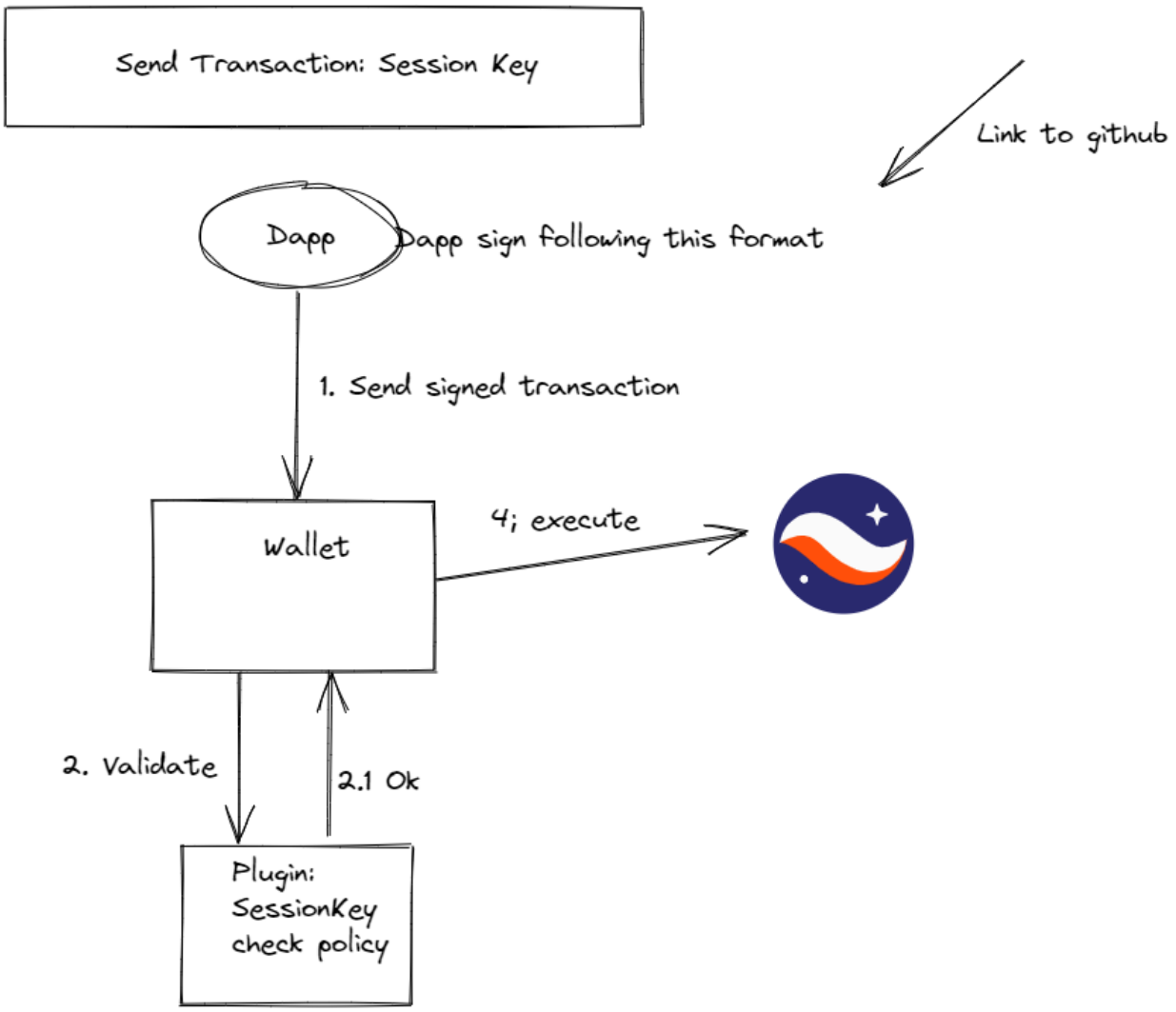

Send Transaction: WebAuthN Plugin
1. Send transaction
Wallet
4 execute
2. Validate
2.1 Ok
3 Execute.
check auth(registry_id, wallet_addr, pluginClass, calls_array)
Call -> (dapp.address, ...)
Plugin:
webAuthN
Dapp Registry
+createRegistry(id, owner)
+addDapp(id, dapp, plugin)
+checkAuth(id, dapp, plugin)

Figure 5.



(selection of the signer, policies)

Figure 6.

Policy

List of Plugins

Heritage
Session Keys

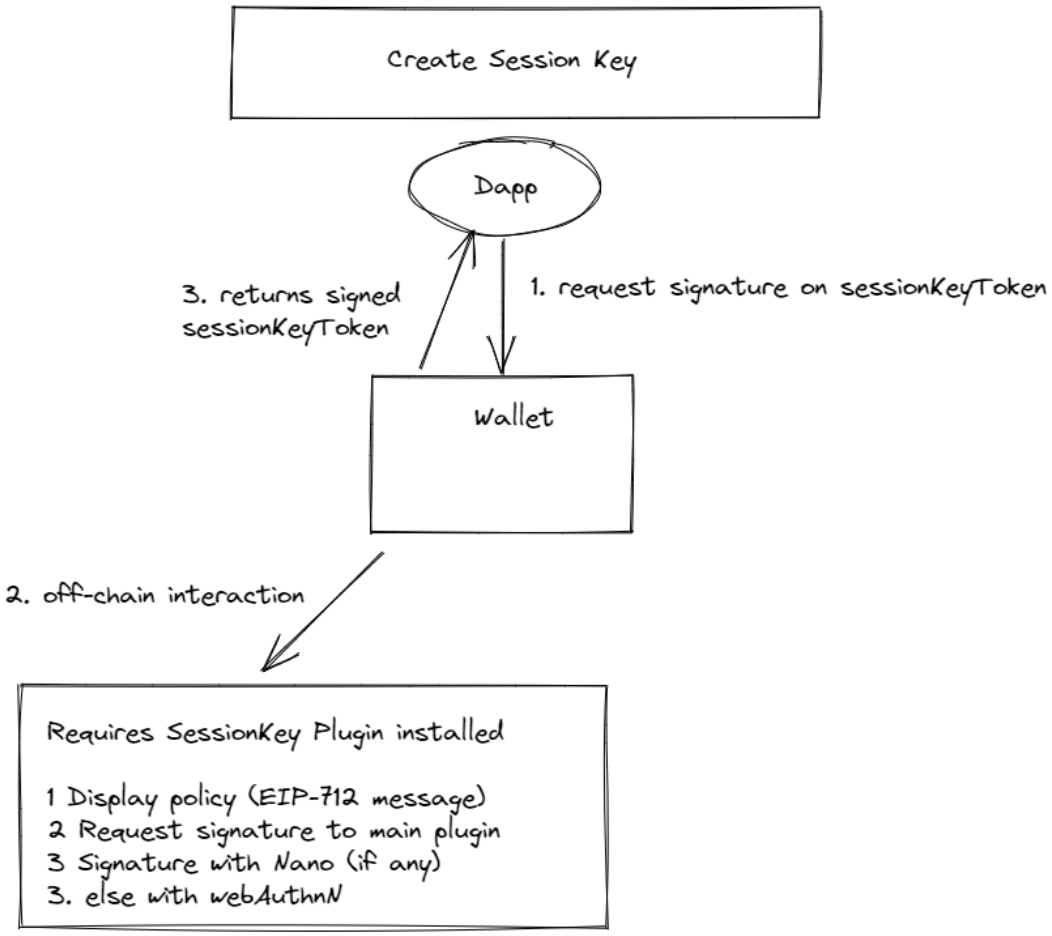

Create Session Key
Dapp
3. returns signed sessionKeyToken
1. request signature on sessionKeyToken
Wallet
2. off-chain interaction
Requires SessionKey Plugin installed
1 Display policy (EIP-712 message)
2 Request signature to main plugin
3 Signature with Nano (if any)
3. else with webAuthnN

Figure 7.

Fresh Account Contract

This chapter describes the initial Ledger Fresh on chain Smart Contract. All contracts are developed over Starknet [STARKNET-DOC].

Fresh On Chain Contracts- Overview

Fresh On Chain Contracts- Requirements

[REQ_FRESHACC_WEBAUTHN]

Fresh Account shall implement an on chain WebAuthn verifier using ECDSA256 with SHA256.

[REQ_FRESHACC_STARKSIGNER]

Fresh Account shall implements an on chain WebAuthn verifier

Fresh On Chain Contracts - APIs

Deployer

Deployer - Overview

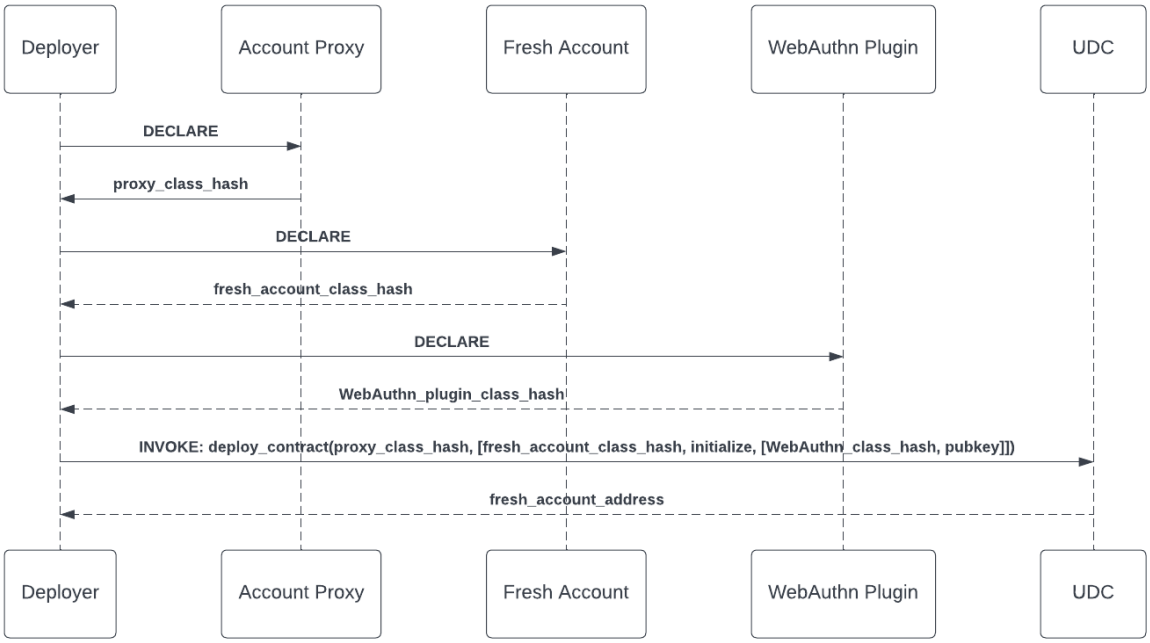


Figure 8. account deployment (on chain)

Deployer - Requirements

The deployer is the back end service intended to deploy users' wallets as such its requirements are

`[REQ_DEPLOYER_MAPPING]`
`The deployer shall build in real-time an off-chain mapping of signer_public_key -> [wallet_address, ...] that will allow finding customer wallets once they reveal their public keys.`

- The deployer shall listen to a specific event triggered by the indexer (the deployment one), parse it, add a new entry `signer_public_key -> wallet_address` using the pub key of the signer who asks for deployment and the address of the freshly deployed account, and finally add to the watcher follow list the address of the account deployed.
- The deployer shall listen to specific events (add_signer and remove_signer) of all wallets deployed using the on-chain deployer in order to update the mapping accordingly.

`[REQ_DEPLOYER_CREATE]`
`The deployer shall deploy a contract given a tuple(signer_classhash, pubkey) over Starknet mainnet.`

An error is returned if the signer_classhash is unknown, or if the pubkey already exists.

Deployer - APIs

FreshCheck Auditor

FreshCheck - Overview

FreshCheck is a Ledger backend service which ensures the security of transactions validated by the user. The service is intended to protect the user against dangerous actions, which could lead the wallet in a bad configuration, or perform unattended transactions.

FreshCheck is totally configurable in the rules applied, and can be replaced by a user's owned server, FreshCheck main features are:

- Easily deployable outside Ledger,
- Open source,
- Stateless.

Considering all the points above this back-end service will have one endpoint /check which takes a raw transaction and a policy as parameters and returns a message with a signed transaction hash.

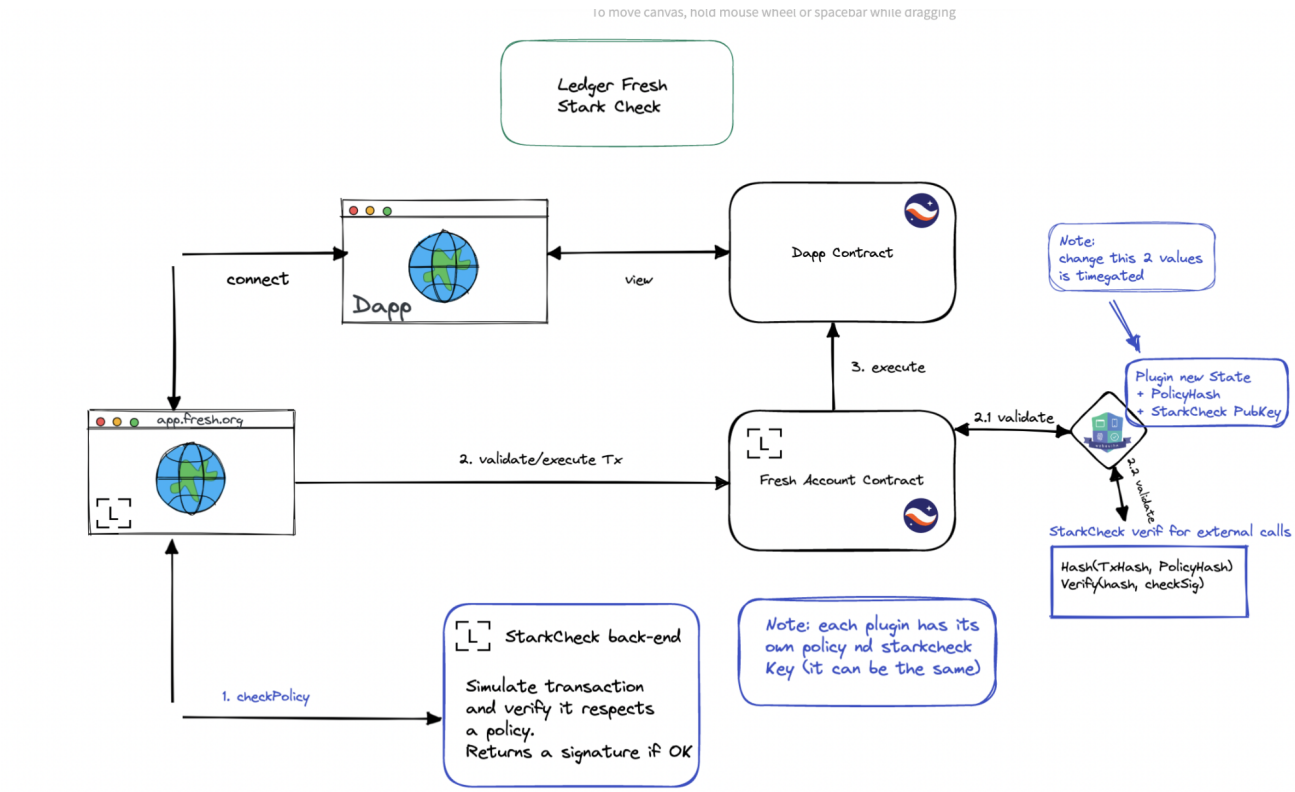


Figure 9. FreshCheck Back end trace analyzer

FreshCheck - Requirements

Revocation lists

[REQ_FreshCheck_PLUGIN_CRL]
FreshCheck shall maintain a CRL of denied plugins. A request to install a revoked plugin is rejected

[REQ_FreshCheck_PUBKEY_CRL]
FreshCheck shall maintain a CRL of denied public keys. Any transaction containing a revoked public key is rejected.

Policy customization

User's own FreshCheck server

[REQ_FreshCheck_SOVEREIGN]
All source code and necessary material shall be available for the user to install its own FreshCheck server

FreshCheck Revocation Tx

Inputs:

Transaction as defined here:
https://docs.starknet.io/documentation/architecture_and_concepts/Blocks/transactions/

Policy is a list of

- ContractAssets -> myContract -> Amount (for ERC20)
- ContractAssets -> myContract -> NFTsIDs (for ERC721)
- Blacklist or allowlist of contract addresses

The back run simulate transaction on a node and analyzes the trace to determine the risks.
The back should also check for approvals (and returns warning if an address in the policy is spotted in the trace).

With an access to node, FreshCheck should watch for state modification of allowed contracts and block it until reallowed.

FreshStore

This chapter describes the Ledger Fresh Plugin Store.

Plugin Store - Overview

A plugin is an extension of the initial Fresh main Contract.

Developper : Submitting a Plugin

The developer develops a plugin, starting from the provided template (To be delivered)). All PRs are done on the `develop` branch of the plugin github.

```
[REQ_PDEV_NAME]

The name of the application shall include its version number.
```

Ledger : Reviewing/Signing a Plugin

Ledger is responsible for reviewing plugins of the `develop` branch . The validation process requires No PR to be merged without these 3 positive reviews (in addition to other rules as defined in other tasks for the same module). Once validated, they are signed using a Ledger attestation key, and stored with version number on the `prod` branch.

In the first version, Plugin signer is implemented through Github actions to run the on-demand script and Github secrets to host the private key that would be used to sign the class hash. Ledger remains the only entity allowed to modify the Github secrets. In addition to the plugin list being hosted and updated on GitHub, it must also exist on IPFS. With each new merge on the prod branch, the latest version of the JSON file must automatically be hosted on IPFS.

Plugin Store : Update Listing/Display

The plugin store continuously weil the Github for a new plugin on the prod branch. It then retrieves the plugin name and author handle and appends it on a Plugin object type as specified in Plugin Connector API subsection.

User : Install/Update the plugin

Plugin Store - Requirements

[REQ_PSTORE_API]

The plugin store’s plugin object is compatible with the

[REQ_PSTORE_LIST]

The Plugin store should display the list of available plugins

[REQ_PSTORE_JSON]

The Plugin list is maintained through a json file.

[REQ_PSTORE_CITESTS]

The Plugin Store shall implement Continuous Integration Tests to automatically run a comprehensive test suite to verify that the proposed plugin meets the necessary specifications and standards.

[REQ_PSTORE_UPDATELIST]

The Plugin store shall listen to the Github to update the list of plugins.

[REQ_PSTORE_SIGNPLUGIN]

All plugins listed in the plugin store are signed by Ledger Fresh CA (**LEDGER-FRESH_CA_PRIVATE_KEY)** key using ECDSA256-SHA256 over Stark Curve.

FreshWatch

Watcher System - Overview

Describe watcher here

Watcher Requirements

[REQ_WATCHER_APIBARA]

The watcher system is implemented using listening of events with [APIBARA] with account address and id filters.

Watcher APIs

Notification System

This chapter describes the functioning of the notification subsystem.

Notification System - Overview

Ledger Fresh wallet owners have the ability to specify the communication channels they want to use to inform them that a challenge period has begun. The communication channels will be stored in a noSQL database users will set up, mapped to their wallet address.

The aim of the Notification System is to monitor the events of a user on-chain address and send notifications to a given destination. This service should be easily self deployed by any party that does not want any other party to know the owner of an address. The link to map an account address and communication channel is made on a DB.

Notification System - Requirements

[REQ_NOTIF_DEST]

The user shall be able to provide the following destinations:

- Telegram
- Mail address
- IFTTT

[REQ_NOTIF_DOCKER]
The solution must be dockerised.

[REQ_NOTIF_MULTIADD]
The database shall be configured to accept multiple email addresses or multiple Telegram IDs for the same wallet address.

Events monitored

[REQ_NOTIF_FUNCT]

The notification system shall listen to a give address for ingoing/outgoing Tx and Starknet ERC20/721/1155 tokens transfers.

An inspiration is the Etherscan tool, as displayed on the following screen capture.
This monitoring system should also have an option to watch and notify “self-transaction”. A self-transaction is a transaction from the account to itself, usually in order to modify its plugins, configuration or other security related operations. By design these operations are timegated, to protect the user in case or hack/malware (signing an unwanted transaction).

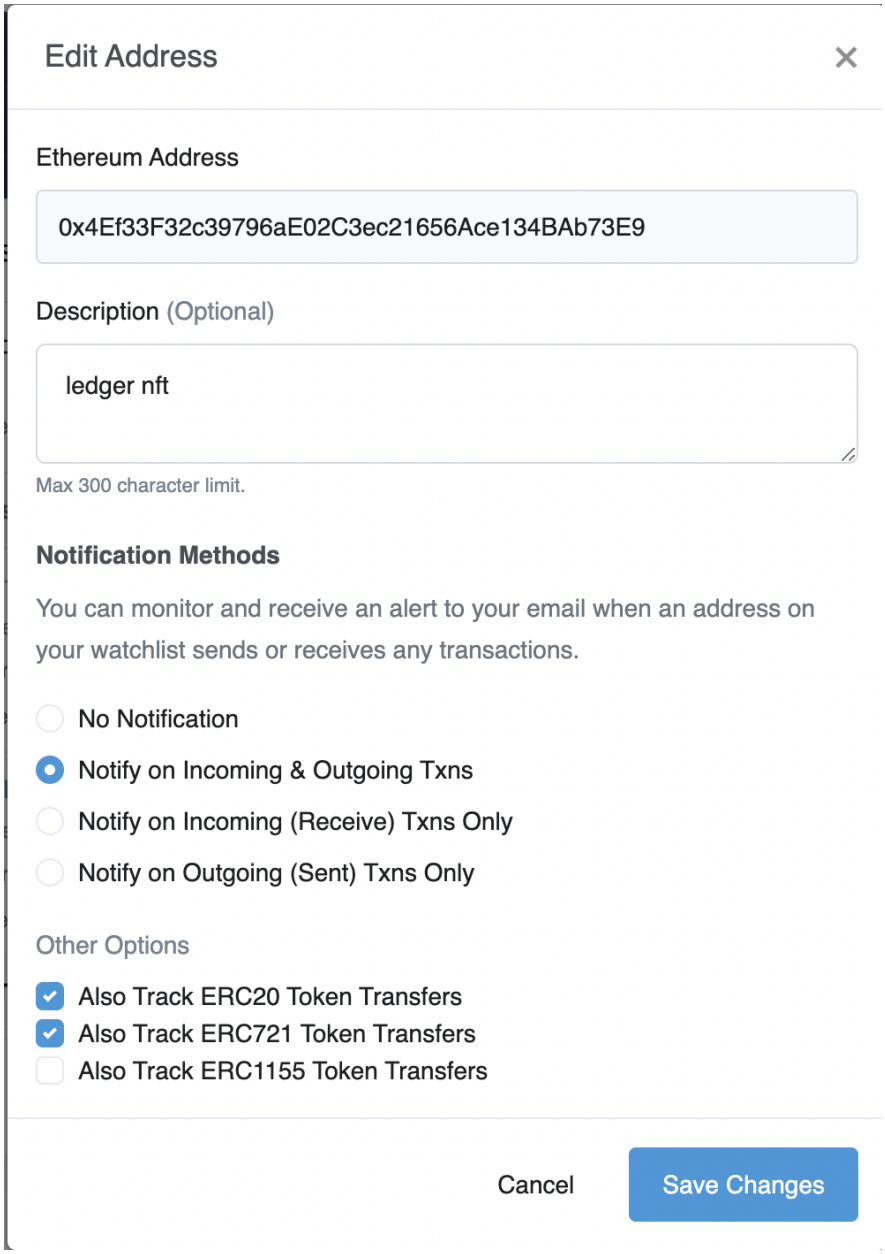


Figure 11. Etherscan screenshot

Notification System - APIs

[REQ_NOTIF_SCHEMAFIELDS_API]
The Schema used for the Database shall at least include the fields below:

```
type Schema = {
    id: string, // wallet address
    telegrams: string[], // handles
    emails: string[],
 }
```

[REQ_NOTIF_UPDATE_API]

The `/updateNotificationPreference` shall receive a payload containing at least the following fields:

```
type Values = {
    telegrams?: string[],
    emails?: string[],
};
type payload = {
  walletAddress: string,
  values: Values,
  timestamp: number,
  signature: string,
  publicKey: string,
}
```

Cryptographic Keys and Services

Identified Keys and Services

Service	Cryptographic Key	Label	Algorithm	Description
FreshCheck	Public Verification Key	**FreshCheck_PUBLIC_KEY**	ECDSA256, StarkCurve+ HashPedersen	Ensure input transaction has been StarChecked by the FreshCheck auditor
WebAuthn	Public FIDO2 Verification Key	**FIDO2_PUBLIC_KEY**	ECDSA256, P256+SHA256	Ensure input transaction has been signed with FIDO2 compliant device
StarkSigner	Public Stark Verification Key	**STARKSIGNER_PUBLIC_KEY**	ECDSA256, StarkCurve+ HashPedersen	Ensure input transaction has been signed with Starknet signing compliant device
Heritage	Public Verification Key	**HEIR_PUBLIC_KEY**	ECDSA256, P256+SHA256	After inactivity of the account of CST_HEIR_DELAY allow transaction signing using HEIR_PRIVATE_KEY
Attestation	Public Verification key	**LEDGER-FRESH_CA_PUBLIC_KEY**	ECDSA256, StarkCurve+ HashPedersen	Proof of Ledger Attestation key

Table 5. Counteract Account Verification Keys

Service	Cryptographic Key	Label	Location	Usage
FreshCheck	Private FreshCheck Key	**FreshCheck_PRIVATE_KEY**	FreshCheck Server	Sign transactions compliant to security policy.
WebAuthn Signer	Private FIDO2 Signing Key	**FIDO2_PRIVATE_KEY**	User Fresh Host (Iphone, Android Phone, PC)	Default signer, quick but malware prone
StarkSigner	Private Signing Key	**STARKSIGNER_PRIVATE_KEY**	Hardware device (recommended)	HW Stark Signer, highest security
Heritage	Heritage Signing Key	**HEIR_PRIVATE_KEY**	Back Up seed Paper, Nano Backup Device, MPC, trusted parent	Give Heir access to configuration of the wallet after DELAY_HERITAGE idle time of the account.
Attestation	Private attestation key	**LEDGER-FRESH_CA_PRIVATE_KEY**	GitHub secrets for v1	Sign plugin to provide proof of origin.

Table 6.

Table: Off-Chain Keys

Identified Critical Security System Parameters

Constant	Location	Description
DELAY_HERITAGE	Heritage Contract	Block delay between last transaction and allowing **HEIR_PRIVATE_KEY** as configuration signer
CST_AMOUNT_OVERFLOW	Initial Contract	

IS_STARKSIGNER_SET	Initial Contract	
APPFRESH_URL	https://www.yettobedefined.org	The AppFresh WebApp URL

Table 7. system parameters

Identified Local Storage

Object name	Location	Description
[LOCAL_ONGOING_TX_STORE]	Host	An object locally stored on host containing all ongoing transactions
[LOCAL_POLICY_STORE]	Host	An object locally stored on host containing all policy rules with its FreshCheck signature and hash

Physical Security

Cryptographic Keys Management

External requirements

This chapter describes the requirements an external subsystem shall meet to be compatible with the Ledger Fresh Wallet.

Signers Requirements

A WebAuthn signer shall implement the FIDO2 ECDSA on P-256 [FIDO2-ALGOSUITE].

A Starknet signer shall implement a starknet signer [STARKNET-DOC].

Dapp Requirements

Roadmap

This chapter describes the roadmap of Ledger Fresh Development.

Version	Content	Estimated completion	Description
v1	Initial smartcontract Plugins session keys, heir	03/2023	
v2	Threshold signature plugin		
	External Ciphered Chest		Cipher the content of local storage on a service like FILECOIN or IPFS.

Table 8. Roadmap

Ressources

(link all ressources here)

Object	Content	Source
[STARKNET-DOC]	Starknet Architecture, Concept and Smart Contracts	https://docs.starknet.io/documentation/

[FIDO2-ALGOSUITE]	FIDO2 cryptographic suite	https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/fido-authenticator-allowed-cryptography-list_20170524.html
[FreshCheck-SOURCE]	FreshCheck PoC	https://github.com/LedgerHQ/FreshCheck
[LEDGER-DEV]	Ledger Developper Portal for Nano Applications	https://developers.ledger.com/docs/embedded-app/start-here/
[APIBARA]		https://www.apibara.com/docs/tutorials/indexing-starknet-nft-events https://www.apibara.com/docs/python-sdk/dynamic-filter
[GETSTRAKNET]	Allow Starknet dApps and wallets to seamlessly connect	https://github.com/starknet-io/get-starknet

Table 9. Ressources

Appendix

Requirements overview

This table lists the requirements with the link to the related github issue.

Requirement	#Github	Attributed(Y/N)	Status (Open/Closed)

Table 10. Table of Requirements

Residual Vulnerabilities

This section addresses the list of identified residual vulnerabilities.

Label	Attacker scenario	Criticity	Description
[VULN_PLUGIN_DOWNGRADE]	- Corrupted plugin Store - Vulnerable anterior plugin version	Low	Attacker lures user into installing old version of plugin.

Required

Identification of missing elements:

1. First time connection and Wallet deployment
2. Ledger plugin deployment for an account (WebAuthN, Starknet signer)
3. Non Ledger plugin deployment for an account (?)
4. Plugin removal (Ledger vs non-Ledger if any difference)
5. Plugin upgrade (Ledger vs non-Ledger if any difference)
6. Account upgrade
7. Wallet recovery

The workflows shall include all the involved actors/components i.e:

- Fresh Front End
- Fresh back ends: Plugin store, Web3 check, ...
- Starknet node
- Starknet Account contract
- Straknet Plugin contract
- Starknet Sequencer/Verifier (if needed)
- Ethereum node/contract (if needed)
- Fresh devices: Ledger Nanos/Stax, Mobile, Desktop, etc...