Kinto Security Analysis

by Pessimistic

This report is public

January 23, 2024

Abstract

In this report, we consider the security of smart contracts of Kinto project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Kinto smart contracts. We described the audit process in the section below.

The audit showed several critical issues: Confusing SignerPolicy logic, Incorrect funds withdrawal, Invalid nonce is changed, Possible signature check bypassing, Risk of fund loss through gas payer manipulation. The audit also revealed several issues of medium severity: EIP-4337 violation, EIP-712 violation, Insufficient documentation, Indication can be not valid for all accounts, KYC is checked incorrectly, Overpowered role, Recovery delay is ignored. Moreover, several low-severity issues were found.

The overall code quality of the project is mediocre. The project contains complex architecture and highly relies on the existing codebase of the Account Abstraction repository. We do not recommend using audited code in the production environment prior to fixing all critical and medium severity issues.

After the initial audit, the codebase was updated. We made the following updates to the report:

- In the critical issues part, we marked:
    - Invalid nonce is changed, Risk of fund loss through gas payer manipulation as fixed;
    - Confusing SignerPolicy logic and Invalid nonce is changed as partially fixed.

- In the medium severity issues:
  - The EIP-4337 violation was divided into two issues, and the second part was added as Aggregation functionality is not supported;
  - The EIP-4337 violation, EIP-712 violation and KYC is checked incorrectly issues were fixed;
  - The developers provided the comments for the Indication can be not valid for all accounts and Aggregation functionality is not supported issues;
  - We commented out the Insufficient documentation and Recovery delay is ignored issues.
- The developers also fixed most of the low-severity issues, but we found a few new ones of the same severity.

The number of tests increased, and the code coverage decreased.

After the recheck #1, the codebase was updated again.

The developers fixed the critical issues: the two points of the Confusing SignerPolicy logic and Incorrect funds withdrawal. They also fixed several issues of low severity and the following issues of medium severity: Insufficient documentation and Recovery delay is ignored. For the Possible signature check bypassing, they provided the comment.

However, we found two issues of low severity and two issues of medium severity: EIP-4337 violation - part 2 and Improper signature verification. We also updated the description of the critical issue (Possible signature check bypassing) and removed one note as false positive.

The number of tests increased, but the code coverage decreased (as the new functionality was added) and was insufficient.

After the recheck #2, the critical Confusing SignerPolicy logic issue was fully fixed by the developers on the commit `296b98d13b5e3fbb6d2e4c6883c0fab174b5fc22`.

The EIP-4337 violation - part 2 issue of medium severity was fixed on the commit `9c134248e49d745a206b1f45cab62f0c95344f40`. And the Improper signature verification issue of medium severity was fixed on the commit `6f73fbcc33fcb1085364eaee87f75945dd27aa7b`.

Although we have marked three last issues as fixed, we do not think that new issues cannot arise in the same functions or contracts. Since we looked only at those three issues in the mentioned commits.

General recommendations

We recommend fixing the mentioned issues and improving code coverage. We also recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Kinto project on a public GitHub repository, commit be90054f0a52326c1dfc5d02eda755958b7813e9.

The scope of the audit included:

- **paymaster/SponsorPaymaster.sol**;
- **libraries/ByteSignature.sol**;
- **wallet/KintoWallet.sol**;
- **wallet/KintoWalletFactory.sol**;
- **KintoID.sol**;
- changes in the part of Account Abstraction repository (**Entrypoint.sol**).

The documentation for the project included this link.

All 70 tests pass successfully. The code coverage is 78.78%.

699 lines from the Kinto repository and 16 lines from code changes in the **Entrypoint.sol** were audited, resulting in a total LOC of 715 lines.

Codebase update #1

After the initial audit, the codebase was updated, and we were provided with commit 60c0fdbec87190ebb169dc39b7a24c229d915206.

The developers fixed a few critical and medium-severity issues. Also, they provided comments on some issues. And they fixed most of the low-severity issues. The number of tests increased to 75. All of them passed. The code coverage was 76.08%.

Codebase update #2

After the recheck #1, the codebase was updated, and we were provided with commit 9d2b5de4ea99d69196b4cbc642d5e7b9af7e07f6.

The developers fixed or commented most of the critical and medium severity issues, but we found two new issues of medium severity.

All 107 tests passed and the code coverage was 61.73%.

Audit process

We started the audit on November 16 and finished on November 28, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and started the review.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among others, we verified the following properties of the contracts:

- It is not possible to bypass validation and use wallets or funds of other owners;
- Whether contracts correctly implements EIP-4337;
- Whether only **KintoWallets** from **KintoWalletFactory** are supported by the entrypoint;
- Whether upgrading functionality is implemented correctly;
- Whether signer policies behave correctly with different setups of owners;
- Whether KYC issuing is implemented correctly;
- Whether **SponsorPaymaster** correctly handles deposits, withdrawals and gas payments.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

We made the first recheck from December 18 to December 22, 2023. We checked all fixes of the issues from the audit. We also checked the new functions that were added. We reran the tests and calculated the code coverage.

Finally, we updated the report.

We made the second recheck from January 11 to January 15, 2024. We checked all provided fixes and new funcitonality.

We also reran the tests, calculated the code coverage and updated the report.

During January 19-23, 2024, we checked only issues C01, M09, M10 and marked them as fixed each on their own commit.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Confusing SignerPolicy logic (fixed)

KintoWallet has confusing code logic that is related to `signerPolicy` variable and signature verification. This issue leads to various mistakes, some of which are listed below:

- **(fixed)** The `resetSigners` function allows setting owners without changing the `signerPolicy`.

  Assume that `owners.length = N` and `signerPolicy = MINUS_ONE_SIGNER`. With the help of the `resetSigners` function it is possible to set the number of owners to `1` and not change policy. Consequently, in this scenario, the `requiredSigners` variable in `_validateSignature` will be equal to `0`.

  If a valid signature is then provided, an underflow will occur. However, passing the wrong signature will allow anyone to perform any action from the name of this wallet;

  This point has been fixed on the commit `296b98d13b5e3fbb6d2e4c6883c0fab174b5fc22`
- **(fixed)** Line 217 of the `_validateSignature` function does not align with the comment at line 44, which states "`1 = single signer, 2 = n-1 required, 3 = all required`". It seems to be a typo and should likely be `signerPolicy == 3` instead;
- **(fixed)** In case when `signerPolicy = 2` and `requiredSigners = 2` in the `_validateSignature` function, three signatures will be recovered instead of two. If all three are correct, an underflow might occur at line 227;
- **(fixed)** When `signerPolicy` is set to `1`, and there are multiple `owners`, the requirement is to provide all the signatures instead of just one.
- **(fixed)** When `signerPolicy == 1`, only `owner[0]`'s signature is checked. However, in the case of many owners, the signature of any owner can fit.

The issues have been fixed and are not present in the latest version of the code.

C02. Incorrect funds withdrawal (fixed)

During withdrawing funds, `ether` is transferred from the balance of the **SponsorPayment** contract. However, this contract does not store `ether` and sends the received `ether` to the **EntryPoint** contract when depositing. It seems that it should first receive `ether` from the **EntryPoint** at line 108 in the `withdrawTokensTo` function.

The issue has been fixed and is not present in the latest version of the code.

C03. Invalid nonce is changed (fixed)

The `onlySignerVerified` modifier compares the `nonce` from the `IKintoID.SignatureData` to `nonces[_signature.signer]`. However, the nonce of the `_signature.account` is incremented afterward. This situation allows the creation of a replay attack, enabling the supply of the initial minting signature after the burning event for the same account at lines 155 and 179 in **KintoID** contract.

The issue has been fixed and is not present in the latest version of the code.

C04. Possible signature check bypassing (commented)

In the scenario where the KYC provider acts as an autonomous contract called by a user with a signed message and subsequently forwards the call to the **KintoID** contract, there exists a vulnerability. The main problem occurs when EOA can manipulate both parameters: signer and signature.

If `KYC_PROVIDER` is a contract that just forwards messages, then it will allow the user to submit a valid message that was signed by them (signer). And it is not only possible when the signer is the contract but EOA either. We think that there are some possible ways to resolve the issue:

- Prohibit `KYC_PROVIDER` from being a contract;
- Do not use signer as a parameter for verification. Maybe `KYC_PROVIDER` address should be used (it will allow `KYC_PROVIDER` to perform needed checks).

> *The issue description has been updated.*

> *Comment from the developers: The EOA is a privileged account so this cannot happen.* `KYC_PROVIDER` *is an EOA that we control.*

C05. Risk of fund loss through gas payer manipulation (fixed)

The purpose behind the `_validatePaymasterUserOp` function in **SponsorPaymaster** contract is not immediately evident. This function checks the address balance (for gas payment) recovered from the operation's calldata. It appears that it should relate to `userOp.sender`, as this is the address of the executable wallet. Currently, it is possible to leverage any address for gas payment by manipulating the calldata.

The issue has been fixed and is not present in the latest version of the code.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. EIP-4337 violation (fixed)

The function `validateUserOp` returns `validationData` that is expected to contain specific information. However, the values for `validAfter` and `validUntil` are not encoded properly. Consequently, the first `20` bytes of the value might contain not only expected values like `0`, `1`, or an `aggregator` address, but also unexpected literals such as `2` and `3`.The issue persists at lines 215 and 230 in **KintoWallet** contract.

The returned values follow the EIP-4337. However, the deadlines (`validAfter` and `validUntil`) are still unused.

M02. EIP-712 violation (fixed)

At lines 379–392 in `onlySignerVerified` function of **KintoID** contract, the second byte of the message indicates that the data will be formatted in accordance with `EIP-712`. However, the subsequent data does not adhere to the `EIP-712` standard, as it lacks a proper domain separator and does not encode the struct correctly. Implementing `EIP-712` encoding as specified in the standard is recommended to ensure proper structuring and formatting of the data.

The issue has been fixed and is not present in the latest version of the code.

M03. Indication can be not valid for all accounts (commented)

The `lastMonitoredAt` timestamp in `isSanctionsMonitored` function of **KintoID** contract is set when the `monitor` function is called, yet the `monitor` function only allows for `200` accounts to be processed at a time. Consequently, even with a fresh `lastMonitoredAt` timestamp, it does not guarantee that all accounts were updated. This scenario means that the data for specific accounts might still be outdated, even after the `lastMonitoredAt` timestamp is updated.

> *The developers added the following NatSpec comment above the `monitor` function:*
> *Updates the accounts that have flags or sanctions. Only by the KYC provider role. This method will be called with empty accounts if there are not traits/sanctions to add. Realistically only 1% of the accounts will ever be flagged and a small % of this will happen in the same day. As a consequence, 200 accounts should be enough even when we have 100k users. 200 accounts should fit in the 8M gas limit.*

M04. Insufficient documentation (fixed)

The codebase of the project is well-commented. However, the comments provide only a brief description of functionality and sometimes contain discrepancy with the actual code in the project:

- The `KintoID.isSanctionsMonitored` does not provide monitoring information for a specific account. Instead, it indicates whether any account was monitored within the last `_days` days;
- There is a discrepancy at line 140 of `SponsorPaymaster._validatePaymasterUserOp`, as it conflicts with the comment at line 22. The comment indicates: `"paymasterAndData holds the paymaster address followed by the token address to use,` yet the validation checks reveal that only one address is stored;
- The `execute` function of the **KintoWallet** contract can only be called by the **EntryPoint** contract. This behavior contradicts the NatSpec comment `"execute a transaction (called directly from owner, or by entryPoint)"`.

The issue has been fixed and is not present in the latest version of the code.The developers updated the documentation.

M05. KYC is checked incorrectly (fixed)

The function `deployContract` of **KintoWalletFactory** contract currently verifies that the sender has KYC. However, this function is executed by the **KintoWallet**, which does not possess KYC itself, but its owner does. Therefore, the KYC status of `owners[0]` should be checked instead of the wallet's KYC status.

The developers added another `deployContractByWallet` method, which just checks the KYC of the wallet owner. However, the appearance of the `deployContractByWallet` function has led to the L23 issue.

M06. Overpowered role

There are several roles that we consider overpowered as they have direct access to some parts of the code:

- `KintoID.KYC_PROVIDER_ROLE` is responsible for `mint`, `burn`, add/remove `traits` and `sanctions`;
- This contract is inherited by **BasePaymaster** contract. Within the **BasePaymaster** contract, the `withdrawTo` function can only be accessed by the `owner`. However, this function grants the `owner` the capability to withdraw all user deposits from **SponsorPaymaster**;
- The **SponsorPaymaster**, **KintoWalletFactory**, **KintoWallet** and **KintoID** contracts are upgradeable, meaning their functionalities can be altered at a later stage.

Some scenarios can lead to undesirable consequences for the project and its users, e.g. if the owner's or KYC provider's private key becomes compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., obligatory setting up a multisig in the project for KYC providers.

M07. Recovery delay is ignored (fixed)

At line 167 in `finishRecovery` function of **KintoWallet** contract, when there is no active recovery procedure, `inRecovery` equals to zero. `finishRecovery` does not check that `inRecovery` has special value. It allows to bypass delay check and change owners as soon as `owners[0]` has become sanctioned or lost their KYC. Moreover, `block.timestamp > 0` expression is always true.

The issue has been fixed and is not present in the latest version of the code.

M08. Aggregation functionality is not supported (commented)

The current implementation of the `SponsorPaymaster._validatePaymasterUserOp` function never returns the `aggregator` address, which indicates that the wallet does not support aggregation.

> *Comment from the developers: We don't support aggregation yet.*

M09. EIP-4337 violation - part 2 (fixed)

According to the EIP-4337 description, it is forbidden to use some opcodes (including `block.timestamp`) during verification in the `SponsorPaymaster._validatePaymasterUserOp` function as they can vary between simulation and the actual execution.

The issue has been fixed on the commit `9c134248e49d745a206b1f45cab62f0c95344f40`.

M10. Improper signature verification (fixed)

The for-loop in the `KintoWallet._validateSignature` function validates that `i`-th signature is valid and was signed by `i`-th owner. However, the number of signatures is restricted by the required signers at line 232. Then, such a case is possible:

1. Assume that `requiredSingers` is equal to 2 and `owners.length` is equal to 3;
2. Then `owners[1]` and `owners[2]` submit 2 signatures (`owners[0]` does not supply the signature);
3. However, during validation, the signatures array looks like `[sig_owner_1, sig_owner_2, 0x00...]`. The line 252 compares `owners[0] == hash.recover(sig_owner_1)`, `owners[1] == hash.recover(sig_owner_2)`, `owners[2] == hash.recover(0x000)`. And the last check will not pass.

The issue has been fixed on the commit `6f73fbcc33fcb1085364eaee87f75945dd27aa7b`.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. _beforeTokenTransfer must be virtual (fixed)

According to the OpenZeppelin documentation, the `_beforeTokenTransfer` function has to be declared as `virtual` at line 419 of **KintoID** contract.

The issue has been fixed and is not present in the latest version of the code.

L02. _setupRole is deprecated (fixed)

According to the OpenZeppelin comments, the function `_setupRole` from `AccessControlUpgradeable`, which is used at lines 69–71 in **KintoID** contract, has been deprecated in favor of the `_grantRole` function and is not used in the latest version.

The issues have been fixed and are not present in the latest version of the code.

L03. Dependency management

The project contains no lock files with frozen versions. Dependencies are installed via git submodules without specified versions. We recommend using specific versions of the libraries, as their behaviour can be changed from version to version.

L04. Direct gas handling

It's not considered best practice to directly specify the amount of gas, as opcode costs have the potential to change. Consequently, the cost of the `_postOp` function may fluctuate. As a result, the value of constant `COST_OF_POST` may become misleading, leading to incorrect calculations for `ethCost` at line 159. Additionally, our gas consumption estimation for the function exceeds `35,000` and is calculated to be `47,382` at line 30 in **SponsorPaymaster** contract.

L05. Duplicate check (fixed)

The check ensuring that `newSigners[0]` has KYC status is already conducted within the `_resetSigners` function. To enhance efficiency and avoid duplication, consider removing the duplicate check at line 169 in `finishRecovery` function of **KintoWallet** contract.

The issue has been fixed and is not present in the latest version of the code.

L06. EIP-4337 violation

At line 146 in `_validatePaymasterUserOp` function of **SponsorPaymaster** contract, returned `validationData` is always zero. However, it should contain specific information, including `validAfter` and `validUntil` values. Moreover, `validationData` never contains the address of the `aggregator`, therefore aggregation is not supported by the paymaster.

L07. Incorrect event arguments (fixed)

The event emits the same address (`newImplementationWallet`) twice. However, the first parameter should be equal to the old wallet implementation at lines 68–69 in **KintoWalletFactory** contract.

The issue has been fixed and is not present in the latest version of the code.

L08. Memory to calldata (fixed)

Several arguments are not mutated but are marked as `memory`. Consider changing their location to `calldata` as this adjustment will reduce gas costs at lines 115, 125, 140, 189, 190, and 199 in **KintoID** contract.

The issues have been fixed and are not present in the latest version of the code.

L09. No validation of the length of values array (fixed)

The function currently validates that the length of the `dest` is equal to the length of the `func`, but it does not validate the length of the `values` array. It would be advisable to include a check to ensure that the length of the `values` array matches the lengths of the `dest` and `func` arrays for consistency and to prevent potential issues at line 114 in `executeBatch` function of **KintoWallet** contract.

The issue has been fixed and is not present in the latest version of the code.

L10. Public to external (fixed)

There are several places where `public` can be changed in favor of `external`:

- `SponsorPaymaster.depositInfo`;
- `KintoID.initialize`;
- `KintoWalletFactory.getContractAddress`;
- `KintoWallet.entryPoint` and `KintoWallet.getOwnersCount`.

The issues have been fixed and are not present in the latest version of the code.

L11. Setter does not emit event (fixed)

`setURI` is setter and should emit an event at line 104 of **KintoID** contract.

The issue has been fixed and is not present in the latest version of the code.

L12. Typo in the error message (fixed)

It seems that it should be `"paymasterAndData must contain only paymaster"` error of the `require` at line 140 in `_validatePaymasterUserOp` function of **SponsorPaymaster** contract.

The issue has been fixed and is not present in the latest version of the code.

L13. Unnecessary external call (fixed)

`deposit` function is `public` and there is no need to call it externally. Consider calling `deposit` internally at line 77 in `addDepositFor` function of **SponsorPaymaster** contract.

The issue has been fixed and is not present in the latest version of the code.

L14. Unused imports (fixed)

There are several occurrences where `forge-std/console2.sol` is imported but not utilized. It is advisable to remove these import statements to declutter the code and improve readability in the project.

The issues have been fixed and are not present in the latest version of the code.

L15. Use constants (fixed)

Consider replacing literal `1` to `KYC_TOKEN_ID` for improved readability at line 195 in **KintoID** contract.

The issues have been fixed and are not present in the latest version of the code.

L16. Use encodeCall instead of encodeWithSelector (fixed)

The function `encodeWithSelector` lacks type-checking functionality. For improved code safety, consider utilizing `encodeCall` as it provides better type-checking capabilities at line 100 in `createAccount` function of the **KintoWalletFactory** contract.

The issue has been fixed and is not present in the latest version of the code.

L17. UUPS pattern is not used (fixed)

In the deployment process, wallets are generated using the **KintoWalletFactory** contract using **SafeBeaconProxy**. The **SafeBeaconProxy** obtains the implementation address from the beacon. Upgrading **KintoWallet** through the UUPS pattern will not have any effect because the slot of the implementation is not employed in the proxy at line 29 in **KintoWallet** contract.

The issue has been fixed and is not present in the latest version of the code.

L18. Wrong amount of traits returned

The current implementation allows to store up to `2 ** 16` traits. However, the method currently returns only `2 ** 8 = 256` traits at lines 357–358 in `traits` function of **KintoID** contract.

L19. Wrong pointers' types (fixed)

Small `uints` such as `uint8` and `uint16` are utilized for various pointers throughout the code. However, these small types are compared with the lengths of arrays, which are of type `uint256`. Consider using `uint256` for pointers as shorter types might be efficient for storage usage but are not useful when compared with `uint256` types, which represent the lengths of arrays at lines 150, 194, and 198 in `_mintTo` function of **KintoID** contract.

The issues have been fixed and are not present in the latest version of the code.

L20. Unused parameter

The `_id` parameter is not used in the `KintoID.onlySignerVerified` modifier.

L21. Checks-Effects-Interactions (fixed)

The CEI (checks-effects-interactions) pattern is violated at lines 106-107 in the `withdrawTokensTo` function of the **SponsorPaymaster** contract. Consider swapping out the external call and storage update.

If the user calls the `addDepositFor` function inside the external call of the `withdrawTo` function, it may cause the balance to be wrong. For instance, this could be used to create a read-only reentrancy.

We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.

The issue has been fixed and is not present in the latest version of the code.

L22. Multiple uses of the same contracts in calldata (commented)

It seems that in the `executeBatch` case of the `SponsorPaymaster._getFirstTargetContract` function, it is not necessary to pass the same target contracts multiple times. Consider replacing them with one address to reduce gas consumption and improve code readability.

> *Comment from the developers: For execute batch, all the contracts need to be the same, but want to keep the interface consistent with the EIP. Through `handleAggregatedOps` in **EntryPoint** you can send different `userOps` that go to different contracts.*

L23. Possible KYC bypass in contract deployment (fixed)

The **KintoWalletFactory** contract does not check in the `deployContractByWallet` function that a calling contract is wallet, which this factory created. It only checks the `owners(0)`, but it is possible to create a contract and specify any owner with KYC. As a result, any contract can call the `deployContractByWallet` function.

The issue has been fixed and is not present in the latest version of the code.

L24. The misleading implementation address variable (fixed)

The `immutable implAddress` variable of the **KintoWalletFactory** contract is only used during beacon initialization. The beacon's implementation address can be further updated via the `upgradeAllWalletImplementations` function. In that case, the `implAddress` variable does not change. This may only affect the fact that if someone reads this variable from the storage, it will return an incorrect implementation address since it is `public`.

The issue has been fixed and is not present in the latest version of the code.

L25. The redundant check of balance (fixed)

The `KintoID.monitor` function has the unnecessary check of balance at line 204 since the subsequent `addTrait`, `removeTrait`, `addSanction` and `removeSanction` functions already check it.

The issue has been fixed and is not present in the latest version of the code.

L26. Unordered token identifiers (new)

The `_nextTokenId` variable of the **KintoID** contract is incremented twice. The first time is at lines 115,126 and the second time is at line 156. As a result, token ids are not going sequentially.

L27. Incorrect array's length (new)

The length of an array at line 242 in the `KintoWallet._validateSignature` function is equal to `owners.length`, but actually the number of signers is equal to `requiredSigners` due to the check at line 232.

Notes

N01. ERC1155 is beyond necessity (fixed)

The current implementation of `ERC1155` functionality is underutilized as only tokens of the `KYC_TOKEN_ID` type are being used. Consequently, considering this restricted token usage, it might be more suitable to extend `ERC721` instead of utilizing the `ERC1155` standard at line 23 in **KintoID** contract.

The issue has been fixed and is not present in the latest version of the code.

N02. Frontrunning wallet creation

There exists a potential vulnerability where a malicious account could execute `KintoWalletFactory.deployContract` with the same parameters used in `createAccount`. This possibility could front-run the account creation process of another user. While the resulting contract would be identical to the one created via `createAccount`, it may not be valid for the **Entrypoint** contract because the `walletTs` will not contain the address of the wallet created via `deployContract` in **KintoWalletFactory** contract.

N03. Metadata is not deleted after burn

The metadata is not updated after the burning event. This situation can mislead external providers using methods like `mintedAt` to verify the existence of KYC for a specific account at line 178 in `_burnp` function of **KintoID** contract.

The issue has been partially fixed, since `updatedAt` is not reset in the `_burnp` function.

N04. No validation for KYC existence (fixed)

Several setters and getters like `addTrait`, `removeTrait`, `addSanction`, `removeSanction` and etc. do not validate that supplied `account` is KYCed. It allows to change and validate properties on not existing `account` in **KintoID** contract.

The issues have been fixed and are not present in the latest version of the code.

N05. Recoverer cannot be changed (fixed)

The current implementation of the wallet does not contain functionality for `recoverer` changing at line 93 in **KintoWallet** contract.

The issue has been fixed and is not present in the latest version of the code.

N06. Unused variable (fixed)

In the current implementation, the `withdrawalWhitelist` is set up but is not utilized beyond the setting of its values at line 49 in **KintoWallet** contract.

The code has been removed.

N07. UpdatedAt field is not initialized (fixed)

Minting a new token does not currently set the `updatedAt` field of the corresponding token metadata at line 148 in `_mintTo` function of **KintoID** contract.

The issue has been fixed and is not present in the latest version of the code.

N09. Unsafe finish recovery (commented)

It seems that it is not safe to reset signers through the `KintoWallet.finishRecovery` function since the parameter for signer policy is constant and equal to `1`. It means that any owner can replace others without their consent. It is protected by the recoverer role, but it is not a guarantee.

> *Comment from the developers: It is by design, after a recovery process, the wallet is reset through a turnkey cloud based signer only and then the user needs to add more signers again. Recoverer is always a privileged account (different by the user).*

This analysis was performed by Pessimistic:

Oleg Bobrov, Junior Security Engineer
Daria Korepanova, Senior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager

January 23, 2024