KINTO CORE SECURITY AUDIT REPORT

February 29, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Kinto is an L2 focused on providing safe and insured access to financial services. It's built on top of the Ethereum network using the Arbitrum Nitro Stack, and it's non-custodial, transparent, permissionless, and governed by the community.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Kinto
Project name	Kinto Core
Timeline	November 27 2023 - February 22 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
29.11.2023	f7dd98f66b9dfba1f73758703b808051196e740b	Commit for the audit
26.12.2023	11d2848864325ab3cacb0e770b95e19963dfbf23	Commit for the re-audit
15.01.2024	296b98d13b5e3fbb6d2e4c6883c0fab174b5fc22	Commit for the re-audit 2
02.02.2024	7155c001ce8b8e9ddddfc7b02273c1760d59ecc5	Commit for the re-audit 3
22.02.2024	a422f33dee7d837546d482e1b8bb28107852f359	Commit for the deployment

Diff audits:

- https://github.com/eth-infinitism/account-abstraction/compare/558e024161cf355e5a0fa59b1b5c0c5cb5aeac74…KintoXYZ:account-abstraction:f9c0a502c3553215a04d0195ebf65737fbc4438d
- PR-1

Project Scope

The audit covered the following files:

File name	Link
src/Faucet.sol	Faucet.sol
src/KintoID.sol	KintoID.sol
src/libraries/ByteSignature.sol	ByteSignature.sol
src/paymasters/SponsorPaymaster.sol	SponsorPaymaster.sol
src/proxy/SafeBeaconProxy.sol	SafeBeaconProxy.sol
src/wallet/KintoWalletFactory.sol	KintoWalletFactory.sol

File name	Link
src/wallet/KintoWallet.sol	KintoWallet.sol

Deployments

File name	Contract deployed on mainnet	Comment
KintoIDV7	0x074e5ECc285b90781f74e491F33fF37849F97220	Impl for KintoID
FaucetV3	0xa0BB7432357634e66b9F56AED03e46c4abfFea49	Impl for Faucet
KintoWalletFactoryV8	0x30D26e75D542Ba2A7e3B35BcC78FDC064B935D8B	Impl for KintoWalletFactory
SponsorPaymasterV7	0x8Fbd7D4e90C442172C2fD6bCB6d78C51D22C1557	Impl for SponsorPaymaster
KintoWalletV5	0x9dd64eA97fFFB2CbCd9ea87b9082250Be50FC820	Impl for KintoWallet

1.5 Summary of findings

Severity	# of Findings
Critical	2
High	6
Medium	8
Low	10

ID	Name	Severity	Status
C-1	Bypassing KYC in KintoWallet	Critical	Fixed
C-2	Malicious bundler can steal funds from SponsorPaymaster	Critical	Fixed
H-1	The SponsorPaymaster owner should be restricted to withdraw	High	Fixed
H-2	Draining SponsorPaymaster.balances with spam operations	High	Acknowledged
H-3	DoS of an account using frontrun	High	Fixed
H-4	DoS `KintoWallet` contract	High	Fixed
H-5	Incorrect signature validation for different signer policies	High	Fixed
H-6	Signature issues in KintoID and Faucet	High	Fixed
M-1	SponsorPaymaster is not compatible with KintoWallet.executeBatch()	Medium	Fixed
M-2	KintoWallet signatures underflow	Medium	Fixed

M-3	KintoID.lastMonitoredAt fails the isKYC() check for all accounts if no new monitors appear	Medium	Fixed
M-4	KintoWalletFactory.deployContract() wrong payable deploy	Medium	Fixed
M-5	Prohibit `_recoverer` from being a EOA	Medium	Fixed
M-6	`UUPSUpgradeable` was not removed from `KintoWallet`.	Medium	Fixed
M-7	Anonymous call	Medium	Acknowledged
M-8	No mechanism for changing the recoverer in KintoWallet	Medium	Fixed
L-1	executeBatch() does not check values.length	Low	Fixed
L-2	Unused methods	Low	Fixed
L-3	SponsorPaymaster unlocks for the deployer, not necessarily the owner	Low	Fixed
L-4	KintoID, Factory and SponsorPaymaster initialize frontrun	Low	Acknowledged
L-5	KintoID.monitor() checks KYC but other functions do not	Low	Fixed
L-6	upgradeAllWalletImplementations() does not check new != old	Low	Fixed
L-7	SponsorPaymaster.deposit() does not deposit for any user	Low	Acknowledged
L-8	Faucet.withdrawAll() does not change active status	Low	Fixed
L-9	Read-only reentrancy is possible in SponsorPaymaster	Low	Fixed
L-10	A user with KintoID can create multiple KintoWallets	Low	Acknowledged

1.6 Conclusion

The scope comprises three main components: account abstraction override, KYC management, and node client updates.

1. We identified several issues with the account abstraction logic. We expect the suggested solutions will improve the overall logic and reduce risk.

2. Aside from the identified vulnerabilities, the overall approach to KYC management is good: KintoID is non-transferable, stores various sanction settings, can be invalidated, and, by design, cannot be burned without user involvement.

3. Diff from the Arbitrum node client was checked only for the ability to restrict users' calls only to the trusted contracts (AA_ENTRY_POINT, KINTO_ID_PROXY, WALLET_FACTORY, PAY_MASTER).

It must be noted that there are strong assumptions about Kinto working correctly with the given KYCs and that any malicious behavior will be immediately noticed and sanctioned. One such assumption is that there will be no fake or incorrect KYCs.

During the security assessment, we paid special attention to these areas:

- The deployment process, proxies used, and transaction flow for updating implementations.
- The logic of accrued balances on EntryPoint and Paymaster, how gas is calculated and repaid, and the potential for DoS due to incorrect balances.
- Integration with Account Abstraction contracts, adherence to Account Abstraction recommendations and known vulnerabilities, and potential conflicts with overridden functions.
- The entire KYC logic, including possible ways to circumvent KYC checks.
- The use of signatures, the application, and behavior of necessary libraries, nonce logic, and correct checks.
- Scenarios involving different sets of owners and policies on KintoWallet, potential conflicts, and transitions between various owners and policies.
- Traits and sanctions, monitoring, and possibly missing sanctions if imposed once.
- Risky situations related to the target account balance on Paymaster, its decoding, and usage in KintoWallet functions.
- Access control and role management.
- Reentrancy attacks.
- Whether it is possible to create a functioning KintoWallet bypassing the KintoWalletFactory.
- Whether it is possible to deploy a contract after the 1000th block without going through the KintoWalletFactory.

2. FINDINGS REPORT

2.1 Critical

C-1	Bypassing KYC in KintoWallet
Severity	Critical
Status	Fixed in 11d28488

Description

- KintoWallet.sol#L204

A KintoWallet can have from one to three owners and have different signer policies. `_validateSignature()` and `resetSigners()` check KintoID only for the `owner[0]`, which allows setting up a KintoWallet in such a way that it can be used by a user without a KintoID.

To do this, a hacker first obtains a KintoID and KintoWallet. This is done either through a fake or stolen ID, or by stealing someone else's private key.

Next, the hacker sets a 2/3 signature scheme in the KintoWallet and sets `owner[0]` to a person who has passed KYC and is not connected to the hacker. For example, they set Vitalik Buterin's account as `owner[0]`. The `owner[1]` and `owner[2]` accounts are the hacker's regular EOAs (Externally Owned Accounts) without KintoID.

Now, the KintoWallet is in no way tied to the original account that created it, and the revocation or invalidation of the original owner's KintoID does not affect it.

The hacker can use the KintoWallet, as they own 2/3 of the signatures, and the `_validateSignature()` function checks the KYC of only the first owner, who, in our example, is Vitalik Buterin.

Recommendation

We recommend that in the `_validateSignature()` function, instead of checking the KYC of the first owner, each signature be simply counted valid only if its owner had a KintoID.

Client's commentary

> Fixed in 12870cad4f532508c69a5223c6547974fd530639.
> It is important to understand that the first owner of an account is always a Turnkey cloud non-custodial account that only the user can access. This is the account that receives the KYC. Made a fix so that this account can only be overriden through a successful recovery. In that case, the recoverer is always a trusted account from Kinto that will set a new Turnkey based account for the user.

C-2	Malicious bundler can steal funds from SponsorPaymaster
Severity	Critical
Status	Fixed in 296b98d1

Description

- SponsorPaymaster.sol#L156

A hacker can drain the SponsorPaymaster in just one transaction. This can be achieved by executing a userOp with an extremely high gas parameters.

As an example, we provided a test where a hacker sets `preVerificationGas` value to 99 ETH. This value is deducted from the SponsorPaymaster and sent to the hacker who runs `handleOps()`.

Recommendation

Almost all gas parameters in userOp allow a hacker to set an arbitrarily high reward for themselves when calling `entryPoint.handleOps()`.

We recommend implementing one of the following solutions:

Method 1: Restrict all gas parameters in userOp so that running `entryPoint.handleOps()` becomes unprofitable. Doing this reliably is challenging.

Method 2: A better solution is for Kinto to be the sole userOp bundler. Add `require(msg.sender == KINTO_BUNDLER)` and `require(beneficiary == KINTO_WALLET)` to the functions `handleOps()` and `handleAggregatedOps()`.

Method 3: Alter the architecture to charge compensation for executing userOp not from sponsors, but directly from the users. This implies deducting other types of cryptocurrencies that users hold in their wallets, such as USDT, for instance.

It's important to note that in the first and second methods, there is a risk of DOS for some users. If users cannot arbitrarily increase gas rewards, they cannot influence the priority of their transactions. Depending on how the bundler algorithm is implemented, this could result in some users' transactions constantly being stuck in the mempool and not getting executed.

Client's commentary

> Client:
> Added three things:
>
> - Modified sponsor paymaster and added a global rate limit and a contract specific limit.
> - Verification of gas settings in validateUserOp to prevent malicious bundlers from draining the Paymaster
> - Made a change in the EntryPoint so only whitelisted addresses can be beneficiaries. Made only this change so everyone can still call it but only whitelisted beneficiaries can be compensated.
>
> New changes vs aa https://github.com/eth-infinitism/account-abstraction/compare/develop…KintoXYZ:account-abstraction:develop
>
> New commit on our repo 698daa4e395e8d3c727b1dd66b4057a4a2d7453a
>
> MixBytes: Fixed, the damage can be limited if the introduced rate limits are used properly.

2.2 High

H-1	The SponsorPaymaster owner should be restricted to withdraw
Severity	High
Status	Fixed in 11d28488

Description

`SponsorPaymaster._postOp()` has these lines:

```
balances[account] -= ethCost;
contractSpent[account] += ethCost;
balances[owner()] += ethCost;
```

- SponsorPaymaster.sol#L160-L162

So, the sum of balances is not changed during UserOp execution.
The same sum of user balances is duplicated in EntryPoint.deposits[], as every SponsorPaymaster deposit and withdrawal is duplicated the same way towards the EntryPoint.
But every UserOp execution decreases the sum of balances stored in the EntryPoint (written ETH balance stored in `Entrypoint.deposits[paymaster]` is transferred to a `beneficiary`). However, the sum of balances written in Paymaster for all accounts has not changed.

It means that some of the `SponsorPaymaster` accounts will not be able to withdraw.
The owner accumulates `ethCost` on every `SponsorPaymaster` usage. If the owner withdraws using `SponsorPaymaster.withdrawTokensTo()`, some other of the `SponsorPaymaster` depositors will have less funds actually stored at `Entrypoint`.
As a result, operations will fail - having a zero balance on EntryPoint, but a non-zero balance on `SponsorPaymaster`.

Recommendation

We recommend removing `balances[owner()] += ethCost;` from `SponsorPaymaster._postOp()`.

Client's commentary

> Removed. Fixed in 12870cad4f532508c69a5223c6547974fd530639

H-2	Draining SponsorPaymaster.balances with spam operations
Severity	High
Status	Acknowledged

Description

Currently, users are not charged for their operations.
Gas compensations are taken from `SponsorPaymaster.balances[account]`, where `account` is the destination address in `KintoWallet.execute()` or `KintoWallet.executeBatch()`.
These accounts are charged with their `SponsorPaymaster.balances[]`.
As a result, users can spam multiple `UserOps`, and gas costs will be compensated from these account balances. This attack has no cost, but results in losses for `account` if actively used.

Moreover, these `accounts` can be other `KintoWallets` if they have some balance on `SponsorPaymaster.balances[]`.
An attacker should build a `calldata` where the target address is another KintoWallet and the function signature is some read-only function on the target `KintoWallet`.

Recommendation

The key problem here is that the attack has no cost, and the `UserOp.account` does not pay for operations.
We recommend charging `UserOp.account` and introducing some cost for potential attackers.

Client's commentary

> Client: Added a rate limit. 12870cad4f532508c69a5223c6547974fd530639 to limit the damage.
> MixBytes: Acknowledged, as users are still not charged, rate limits only allow managing the amount of spam.

H-3	DoS of an account using frontrun
Severity	High
Status	Fixed in 296b98d1

Description

- KintoWalletFactory.sol#L130

In `KintoWalletFactory`, a hacker can make a `deployContract` frontrun before calling `createAccount`. Thus, `deployContract` will create a valid `KintoWallet` without configuring the `walletTs` value.

The following is an example:

```
vm.prank(_hackerWithKYC);
bytes memory a = abi.encodeWithSelector(
    KintoWallet.initialize.selector,
    _owner,
    _owner
);
_walletFactory.deployContract(
    0,
    abi.encodePacked(
        type(SafeBeaconProxy).creationCode,
        abi.encode(address(_beacon), a)
    ),
    bytes32(someSalt)
);
vm.startPrank(_owner);

// create2 will not be called
_kintoWalletv1 =
    _walletFactory.createAccount(
        _owner, _owner, someSalt);

// _walletFactory.getWalletTimestamp(
// address(_kintoWalletv1)
// ) == 0
```

If the user sends tokens to it by mistake, they will be permanently blocked.

Such a contract cannot be recovered in any way, and also the `Recovery` mechanism will not work, since there is no way to call `KintoWallet` via `EntryPoint` on any `owners`.

Recommendation

We recommend not allowing `KintoWallet` accounts to be created via the `deployContract` method. A single fake-KYC or leaked account can create potential problems for all members of the Kinto network.

Client's commentary

> Client: Fixed

H-4	DoS `KintoWallet` contract
Severity	High
Status	Fixed in 11d28488

Description

- KintoWallet.sol#L235

`_resetSigners` does not take into account the current `SignerPolicy` in any way. A user can accidentally call `resetSigners` with an array length of 1 (with current `policy > 1`). A sample code is below:

```
abi.encodeWithSignature(
    'resetSigners(address[])',
    [address1, address2])

abi.encodeWithSignature(
    'setSignerPolicy(uint8)',
    2)

abi.encodeWithSignature(
    'resetSigners(address[])',
    [address1])

// _kintoWalletv1.signerPolicy() == 2
// _kintoWalletv1.getOwnersCount() == 1
```

Thus, the following code (KintoWallet.sol#L220) will be called:

```
...
else {
    (signatures[0], signatures[1], signatures[2]) =
        ByteSignature.extractThreeSignatures(
            userOp.signature);
}
for (uint i = 0; i < owners.length; i++) {
    if (
        owners[i] == hash.recover(signatures[i])
        ) {
        requiredSigners--;
    }
}
return requiredSigners;
```

and since `owners.length == 1`, the `_validateSignature` method will always return an error. You will need to wait for 7 days and restore the account (`finishRecovery`).

Recommendation

We recommend that when calling `resetSigners` you also check the `policy` variable and adjust it if necessary.

Client's commentary

> Fixed 12870cad4f532508c69a5223c6547974fd530639. Policy and signers are now set at the same time.

H-5	Incorrect signature validation for different signer policies
Severity	High
Status	Fixed in 296b98d1

Description

- KintoWallet.sol#L200

A KintoWallet can have from one to three owners and have different signer policies. Incorrect verification of signers in `_validateSignature()` in KintoWallet can lead to DOS (Denial of Service) of wallets or loss of funds.

Problem 1. Using the `MINUS_ONE_SIGNER` can lead to a loss of funds at the moment when the owner sets a single owner, as `_resetSigners()` does not check the current `signerPolicy`, and `_validateSignature()` in such a situation will check **zero** valid signatures, making the wallet accessible for withdrawal by any actors:

```
uint requiredSigners =
    signerPolicy == 3 ?
    owners.length :
    owners.length - 1;
```

Problem 2. The use of the `SINGLE_SIGNER` policy is inconsistent and can lead to DOS.

If it is assumed that the `SINGLE_SIGNER` policy is always 1/1 scheme, then the question arises as to how to organize a 1/3 scheme and why, when changing the signer policy via `setSignerPolicy()`, it is possible to set the `SINGLE_SIGNER` policy with more than one owner:

```
function setSignerPolicy(uint8 policy) external override onlySelf {
    require(
        policy > 0 &&
        policy < 4 &&
        policy != signerPolicy,
        'invalid policy'
        );

    require(
        policy == 1 ||
        owners.length > 1,
        'invalid policy'
        );

    emit WalletPolicyChanged(policy, signerPolicy);
    signerPolicy = policy;
}
```

If it is assumed that `SINGLE_SIGNER` can be 1/2 or 1/3 scheme, then in `_validateSignature()` there is an incorrect check, because only the `owner[0]` is checked and otherwise the transaction is reverted:

```
if (signerPolicy == 1) {
    if (owners[0] != hash.recover(userOp.signature))
        return SIG_VALIDATION_FAILED;
    return 0;
}
```

Thus, if the first owner loses their key, the other owners lose access to the KintoWallet.

Problem 3. If a KintoWallet has more than one owner and the first owner burns their KYC, the rest lose access to the wallet because `_validateSignature()` reverts if the first owner does not have KYC:

```
if (!kintoID.isKYC(owners[0])) {
    return SIG_VALIDATION_FAILED;
}
```

Problem 4. Using `MINUS_ONE_SIGNER` or `ALL_SIGNERS` policies currently lead to `_validateSignature()` possibly returning value of `2` or `3` as a result. Values above 1 have a different meaning in account abstraction and may be interpreted by the entryPoint in an unexpected way:

```
for (uint i = 0; i < owners.length; i++) {
    if (owners[i] == hash.recover(signatures[i])) {
        requiredSigners--;
    }
}
return requiredSigners;
```

Especially if `_validateSignature` returns greater than 1, this is the wrong architecture for ERC-4337. Since EntryPoint considers that the Aggregator is returned (this is considered a valid signature).

Recommendation

We recommend:

1. Limit the result of `_validateSignature()` to only 0 and 1.
2. Check the signer policy and owners to avoid a situation where the wallet checks zero signatures.
3. Standardize the operation of `_validateSignature()` in such a way that for the SINGLE_SIGNER policy, one signature from the list of owners is checked (not necessarily the very first owner).
4. Instead of checking the KYC of the first owner, simply consider any owner's signature as valid only if its owner passed KYC.

Client's commentary

> All the problems should be fixed by this commit 7105be6cd1cdee55c7ab94fba9663106bb160dd4.

H-6	Signature issues in KintoID and Faucet
Severity	High
Status	Fixed in 296b98d1

Description

- KintoID.sol#L371-L399
- Faucet.sol#L90-L114

In the current implementation, it is assumed that a KYC node can issue a user a KintoID and cannot burn it unless the user signs such a transaction. Incorrect handling of nonces and non-standard construction of the hash for signature lead to the situation where a node with the KYC_PROVIDER_ROLE can at any time burn a user's KintoID.

The `SignatureData` struct has fields `signer` and `account`. Signer is the one who makes the signature. Account is the one to whom the KintoID is issued. The minting and burning functions check the signature of the following structure:

```
keccak256(
    abi.encodePacked(
    '\x19\x01',    // EIP-191 header
    keccak256(abi.encode(
        _signature.signer,
        address(this),
        _signature.account,
        _id,
        _signature.expiresAt,
        nonces[_signature.signer],
        bytes32(block.chainid)
    ))
    )
)
```

It can be noticed that `nonces[_signature.signer]` is hashed. However, after the signature check, `nonces[_signature.account]` is increased, which is incorrect.

Furthermore, in building the hash, the name of the function for which the signature is given is missing. This allows the same signature to be used to call any of the functions, such as `mintIndividualKyc()`, as well as `burnKYC()`.

As the nonce for the signature effectively does not change, the KYC node can subsequently use the same signature that was applied for the `mintIndividualKyc()` function to burn the KintoID through

`burnKYC()`.

It should be noted that similar problems are present in the Faucet contract.

Recommendation

We recommend increasing the nonce of the signer; revisiting the logic when signer and account does not match; and also adding the function signature into the hash for which the signature is given.

Client's commentary

> Fixed by 698daa4e395e8d3c727b1dd66b4057a4a2d7453a

2.3 Medium

M-1	SponsorPaymaster is not compatible with KintoWallet.executeBatch()
Severity	Medium
Status	Fixed in 11d28488

Description

`KintoWallet.executeBatch()` makes multiple calls to a list of `address[] calldata dest`. But SponsorPaymaster strictly chooses the first input address as a sponsor for gas costs, and it is only one address.

```
address targetAccount =  address(bytes20(userOp.callData[16:]));
```

- SponsorPaymaster.sol#L142

This logic does not work for KintoWallet.executeBatch() which accepts an array as the first argument:

```
function executeBatch(
    address[] calldata dest,
    uint256[] calldata values,
    bytes[] calldata func
) external override {
    _requireFromEntryPoint();
    require(
        dest.length == func.length,
        'wrong array lengths'
    );

    for (uint256 i = 0; i < dest.length; i++) {
        dest[i].functionCallWithValue(
            func[i],
            values[i]
            );
    }
}
```

So, the UserOp.calldata cannot be properly decoded to extract the dest addresses, as the calldata for array arguments do not start with the address.

- Calldata of one address element:
  `0xc79fb9f500000000000000000000000a0b86991c6218b36c1d19d4a2e9eb0ce3606eb48`

- Calldata of one address element as an array:
  `0x4ab285f400000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000001000000000000000000000000a0b86991c6218b36c1d19d4a2e9eb0ce3606eb48`

Moreover, the array is designed to have a few addresses, usually different ones. It means it is not correct to choose only one address in the array.

Recommendation

The current logic of choosing the `targetAccount` cannot work with `executeBatch()`. We recommend either removing `executeBatch()` or reworking SponsorPaymaster to separately measure the gas cost for every `dest` in the inputted array.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

M-2	KintoWallet signatures underflow
Severity	Medium
Status	Fixed in 11d28488

Description

KintoWallet._validateSignatures has the following lines:

```
...
if (owners.length == 2) {
    (signatures[0], signatures[1]) =
        ByteSignature.extractTwoSignatures(userOp.signature);
} else {
    (signatures[0], signatures[1], signatures[2]) =
        ByteSignature.extractThreeSignatures(userOp.signature);
}
for (uint i = 0; i < owners.length; i++) {
    if (owners[i] == hash.recover(signatures[i])) {
        requiredSigners--;
    }
}
return requiredSigners;
```

- KintoWallet.sol#L220-L230

Imagine we have 3 owners, and signerPolicy=2.
It means that at least N-1 correct signatures are required = at least 2 sigantures.
Every correct signature found in the loop will decrease `requiredSigners`.
If `requiredSigners` is equal to 0 in the end, it means the validation went correctly.
But the third signature will make the underflow, and the whole validation will fail.

Recommendation

Consider exiting the loop when `requiredSigners` reaches 0.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

M-3	KintoID.lastMonitoredAt fails the isKYC() check for all accounts if no new monitors appear
Severity	Medium
Status	Fixed in 11d28488

Description

`KintoID.lastMonitoredAt` is a very risky parameter. It is reset to the current `block.timestamp` every time `monitor()` is called.
If `monitor()` is not called for 7 days - all `isKYC()` checks for all accounts will return `false`.

- KintoID.sol#L282
- KintoID.sol#L300
- KintoID.sol#L291

Being KYCed is extremely important to make transactions. So, `lastMonitoredAt` can stop the whole chain activity.
But `lastMonitoredAt` can become outdated if new treats and sanctions are set via other public functions: `addTrait()`, `removeTrait()`, `addSanction()`, `removeSanction()`.

Recommendation

We recommend updating `lastMonitoredAt` in `addTrait()`, `removeTrait()`, `addSanction()`, `removeSanction()`.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

M-4	KintoWalletFactory.deployContract() wrong payable deploy
Severity	Medium
Status	Fixed in 11d28488

Description

`KintoWalletFactory` has the `deployContract()` function:

```
    function deployContract(
        uint amount,
        bytes memory bytecode,
        bytes32 salt
    ) external override returns (address) {
        require(kintoID.isKYC(msg.sender), 'KYC required');
        return Create2.deploy(amount, salt, bytecode);
    }
```

- KintoWalletFactory.sol#L130-L137

This function accepts the `amount` parameter which will be used as a `msg.value` in the payable deployment. But the whole deployContract() is not payable and does not receive the native token.
It is commented as:

```
- the factory must have a balance of at least `amount`.
```

So, someone should top up the Factory to have the necessary native balance.
There are multiple problems with this approach:

1. `KintoWalletFactory` does not have functions to receive native balance
2. Even if native balance is deposited, it can be stolen by anyone who is faster to deploy their own smart contracts

Recommendation

We recommend having a `payable` modifier in `deployContract()` and checking that the `msg.value==amount`.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

M-5	Prohibit `_recoverer` from being a EOA
Severity	Medium
Status	Fixed in 296b98d1

Description

- KintoWallet.sol#L93

In the current architecture it is not allowed to call `KintoWallet` directly. Thus, the `startRecovery` and `finishRecovery` methods are not available to EOA.

Users may mistakenly specify an EOA as `recoverer`, resulting in an inability to recover it.

Recommendation

We recommend adding a check that `_recoverer` is a contract.

Client's commentary

> Recoverer will always be a Turnkey wallet with dual custody between user and Kinto so it can be recovered in case the user loses everything, after waiting a week to prevent malicious takeovers. Added methods to the factory to handle recovery on 698daa4e395e8d3c727b1dd66b4057a4a2d7453a

M-6	`UUPSUpgradeable` was not removed from `KintoWallet`.
Severity	Medium
Status	Fixed in 11d28488

Description

- KintoWallet.sol#L29

Since `KintoWallet` is currently `BeaconProxy`(KintoWalletFactory.sol#L98), the value `_IMPLEMENTATION_SLOT` does not exist.

So, the `upgradeTo` and `upgradeToAndCall` methods are not available.

Recommendation

We recommend removing inheritance from `UUPSUpgradeable`.

Client's commentary

> Fixed 7105be6cd1cdee55c7ab94fba9663106bb160dd4

M-7	Anonymous call
Severity	Medium
Status	Acknowledged

Description

- EntryPoint.sol#L66

Any anonymous user on the Kinto network can call an empty `handleOps` or `handleAggregatedOps`.

This is not a threat, but it does create an exploitable contract (e.g. in the first 1000 blocks after the network starts) that can be called anonymously from any account.

Recommendation

We recommend considering this when deploying contracts.

Client's commentary

> Noted.

M-8	No mechanism for changing the recoverer in KintoWallet
Severity	Medium
Status	Fixed in 296b98d1

Description

- KintoWallet.sol#L48

In the current implementation of KintoWallet, it is impossible to change the address of the recoverer. If the recoverer loses access to their account, or goes rogue, the wallet owners have no way to change it, creating a risk for their funds.

Recommendation

We recommend implementing a mechanism to change the recoverer in KintoWallet.

Client's commentary

> Client: Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4
> MixBytes: It is assumed that the KintoWallet wallet owner has the option to change the recoverer himself. Now it can be done only by the recoverer. If this solution is not acceptable the issue should be acknowledged and risk should be accepted.

2.4 Low

L-1	executeBatch() does not check values.length
Severity	Low
Status	Fixed in 11d28488

Description

KintoWallet executeBatch() accepts arrays as inputs but checks only that `dest.length == func.length`.

```
    function executeBatch(
        address[] calldata dest,
        uint256[] calldata values,
        bytes[] calldata func
    ) external override {
        _requireFromEntryPoint();
        require(
            dest.length == func.length,
            'wrong array lengths'
            );

        for (uint256 i = 0; i < dest.length; i++) {
            dest[i].functionCallWithValue(
            func[i],
            values[i]);
        }
    }
```

- KintoWallet.sol#L112-L118

In fact, checking `values.length` is also necessary, because it should have exactly the same length as `dest` and `func`. It can be an array of zero values. Otherwise, it can be reverted, if e.g. `values[i]` tries to take the index above the length.

Recommendation

We recommend additionally checking that `dest.length == values.length`.

Client's commentary

> Fixed 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-2	Unused methods
Severity	Low
Status	Fixed in 11d28488

Description

KintoWallet has the following modifier:

```
modifier onlyFactory() {
    _onlyFactory();
    _;
}
```

But it is never used.
Moreover, the `factory` address is never set, so it remains `address(0)`.

It was also noticed that the following import is not used:

- SponsorPaymaster.sol#L14.

Recommendation

We recommend removing import and setting the `factory` address, implementing new logic using `onlyFactory`. If it is not required, it can be deleted.

Client's commentary

> Fixed 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-3	SponsorPaymaster unlocks for the deployer, not necessarily the owner
Severity	Low
Status	Fixed in 11d28488

Description

`SponsorPaymaster.initialize()` calls internally `unlockTokenDeposit()`, which sets:

```
unlockBlock[msg.sender] = block.number;
```

It likely aims to unlock withdrawals for the owner.
There is the comment proving this intent.

```
...
// unlocks owner
...
```

- SponsorPaymaster.sol#L48-L49

But it unlocks not the `owner`, but the `msg.sender` - the deployer.
So, if `initialize()` indicates the owner which is different from the deployer, only the deployer will be unlocked, not the owner.

Recommendation

We recommend replacing `unlockTokenDeposit();` with the direct `unlockBlock[_owner] = block.number;` line if it was designed to unlock the owner and not the deployer.

Client's commentary

> Fixed 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-4	KintoID, Factory and SponsorPaymaster initialize frontrun
Severity	Low
Status	Acknowledged

Description

KintoId, Factory and SponsorPaymaster contracts require calling `initialize()` during the deployment. These functions are public. So, they can be frontrun by not authorized addresses.
In the Factory contract, it will allow to set own KintoId address, in KintoID it will allow to set malicious roles, and in SponsorPaymaster it will allow to set the owner which will receive gas compensation balance.

- KintoID.sol#L63-L73
- KintoWalletFactory.sol#L50-L56
- SponsorPaymaster.sol#L45-L50

The severity of this issue is high enough but it must be noted that the likelihood is quite low as the deployment process happens in the controlled chain.

Recommendation

We recommend having either all deployment steps in one transaction or setting the allowed `initialize()` caller in the `constructor()`.

Client's commentary

> Noted.

L-5	KintoID.monitor() checks KYC but other functions do not
Severity	Low
Status	Fixed in 11d28488

Description

KintoID.monitor() function checks KYC for every address inputted:

```
require(balanceOf(_accounts[i], 1) > 0, 'Invalid account address');
```

- KintoID.sol#L195

But other functions that perform the same operations do not check KYC:

- addTrait()
- removeTrait()
- addSanction()
- removeSanction()

Moreover, KYC can be revoked (NFT burn) and given again (NFT mint).
All traits and sanctions remain unchanged in the meta.
So ,in fact traits and sanctions can exist even for accounts with no KYC.

Recommendation

The simplest solution is removing the requirement for an address KYC in `KintoID.monitor()`.
Otherwise, consider adding this check to other trait and sanction functions and deleting the meta storage for burned NFTs.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-6	upgradeAllWalletImplementations() does not check new != old
Severity	Low
Status	Fixed in 11d28488

Description

`KintoWalletFactory` has `upgradeAllWalletImplementations()`.

```
    function upgradeAllWalletImplementations(
        IKintoWallet newImplementationWallet
    ) external override {
        require(msg.sender == factoryOwner, 'only owner');
        require(
            address(newImplementationWallet) != address(0),
            'invalid address');
        factoryWalletVersion++;
        emit KintoWalletFactoryUpgraded(address(newImplementationWallet),
            address(newImplementationWallet));
        beacon.upgradeTo(address(newImplementationWallet));
    }
```

- KintoWalletFactory.sol#L62-L71

But it does not check that `newImplementationWallet` is not equal to the old implementation (`UpgradableBeacon._implementation`).

Recommendation

Consider implementing this check.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-7	SponsorPaymaster.deposit() does not deposit for any user
Severity	Low
Status	Acknowledged

Description

`SponsorPaymaster` inherits from the `BasePaymaster` which has the `deposit()` function. It just uses the provided `msg.value` and deposits it to the `EntryPoint`.

- BasePaymaster.sol#L81-L83
  Balance for the `msg.sender` is not increased.

It is misleading, given that `SponsorPaymaster` has `addDepositFor()` which correctly increases the balance for an address.

`deposit()` is used only internally as `this.deposit()` to transfer funds to the `Entrypoint` during the `addDepositFor()`.

It is better to remove this function. It can be an argument that this function can be used to donate native tokens. But there are better ways - using `addDepositFor()`.

Recommendation

Consider overriding the `deposit()` function requiring that `msg.sender==address(this)`.
A good reference for the same situation is the EntryPoint:
EntryPoint.sol#L265.

Client's commentary

> Client: Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4
> MixBytes: `deposit()` still remains public and can be called by anyone. Fixing the issue would require changing an AA lib contract, which is also a risky solution given that the finding severity is Low. As a result, we mark the finding as Acknowledged.

L-8	Faucet.withdrawAll() does not change active status
Severity	Low
Status	Fixed in 11d28488

Description

`Faucet.active` is `false` if the native balance is below `CLAIM_AMOUNT`.

- Faucet.sol#L78-L79

But when `withdrawAll()` is called, it does not change the status:

- Faucet.sol#L54-L56.

Recommendation

We recommend setting `active` as `false` in `withdrawAll()`.

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-9	Read-only reentrancy is possible in SponsorPaymaster
Severity	Low
Status	Fixed in 11d28488

Description

`SponsorPaymaster.withdrawTokensTo()` does not follow CEI pattern:

1. sends native tokens to the withdrawal target
2. then updates balances

- SponsorPaymaster.sol#L107-L108

When native tokens are withdrawn, it uses a raw call without gas limits, so the reentrancy is possible:

```
(bool success, ) = withdrawAddress.call{value: withdrawAmount}("");
```

- StakeManager.sol#L145

In this case, the withdrawal target is able to receive tokens and run additional attack logic.

At least, in this reentrancy, some read-only functions will return wrong balances:

- SponsorPaymaster.sol#L119-L120
- SponsorPaymaster.sol#L32

Recommendation

We recommend replacing lines from:

```
        withdrawTo(payable(target), amount);
        balances[msg.sender] -= amount;
```

…to:

```
        balances[msg.sender] -= amount;
        withdrawTo(payable(target), amount);
```

Client's commentary

> Fixed by 7105be6cd1cdee55c7ab94fba9663106bb160dd4

L-10	A user with KintoID can create multiple KintoWallets
Severity	Low
Status	Acknowledged

Description

- KintoWalletFactory.sol#L86

A user with KintoID can create multiple KintoWallets by calling `KintoWalletFactory.createAccount()` with different values for `salt` and `recoverer`.

This is not a vulnerability, but developers need to decide whether this is appropriate behavior within the Kinto network.

Recommendation

We recommend coming to a decision on whether a single user with KintoID should be allowed to create multiple KintoWallets or not.

Client's commentary

> Noted. We plan to allow more than one wallet per user (tied to same KYC).

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes