$ ▌

Competitive Security Assessment

IoTex

Jul 27th, 2023

Secure3

secure3.io

Summary

IoTeX account abstraction implement with P256 validation account.

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	IoTex
Platform & Language	Solidity
Codebase	• https://github.com/iotexproject/account-abstraction-contracts • audit commit - 5c3bb748177715ed18cb9937aa497438c1ea7d9f • final commit - a9434741eb635b601c61fa395e05b03c8a0fc496
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

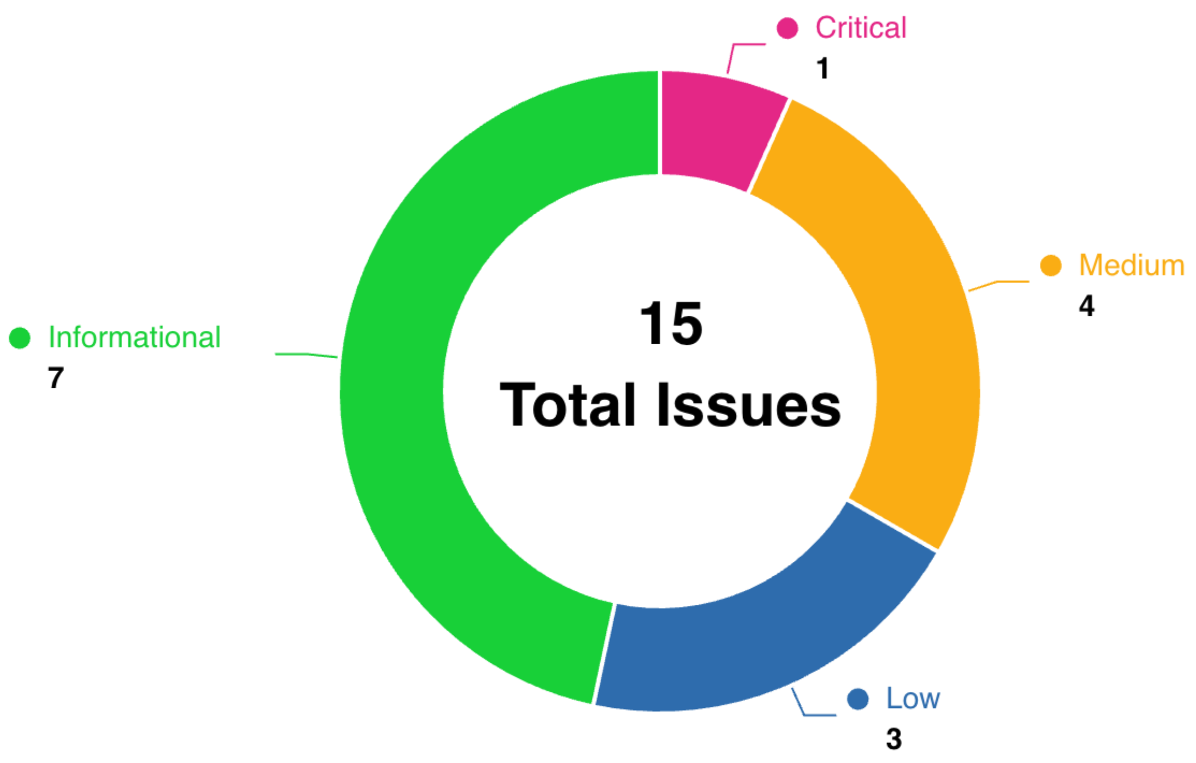
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**1**	**0**	**0**	**1**	**0**	**0**
Medium	**4**	**0**	**0**	**2**	**1**	**1**
Low	**3**	**0**	**2**	**0**	**1**	**0**
Informational	**7**	**0**	**3**	**4**	**0**	**0**

Audit Scope

File	SHA256 Hash
contracts/accounts/secp256r1/P256Account.sol	939931ddcc75cc50257c5bff8d74b6f0917d959105c5f20c66f0d287e552bd6c
contracts/paymaster/VerifyingPaymaster.sol	a9cb6fd89bcb4d2f325ca73646c21d995497315364bab1ea7826362ebb6dc676
contracts/accounts/secp256r1/P256AccountFactory.sol	56e6f15c630b6748286f15fb91d5019cc463fe7df47f5ea05828bd99f031f72a
contracts/accounts/secp256r1/Secp256r1.sol	953daaadc3bdef93ffc4bcf7b1eb97998d31e6516ffbdfe8beafe1c68cc86b02
contracts/accounts/secp256r1/SimpleEmailGuardian.sol	2ef4130d199e95ad7d6ea88bf67992d9380f69796ae075fe324f939b4bd6ede2
contracts/stub/import.sol	37feaaa358f932f916818014a8cd1274ff13f1fe6a6c3f2734cc178da76b7937

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
IOT-1	Replay attack on the signature	Signature Forgery or Replay	Critical	Fixed	grep-er, danielt
IOT-2	The `executeBatch()` will not involve any transfer of ether.	Logical	Medium	Fixed	grep-er, 0xzoobi
IOT-3	Outdated OpenZeppelin dependencies	Logical	Medium	Fixed	0xzoobi
IOT-4	No Storage Gap for Upgradeable Contract in `P256Account.sol`	Logical	Medium	Mitigated	0xzoobi

IOT-5	**`staticall` uses a hardcoded address to verify the signature in `Secp256r1::validateSignature()`**	**Logical**	**Medium**	**Declined**	**0xzoobi**
IOT-6	**Avoid the use of reserved `value` keyword inside _call() function as a variable**	**Code Style**	**Low**	**Acknowledged**	**grep-er**
IOT-7	**Lack of zero address check**	**Code Style**	**Low**	**Acknowledged**	**0xzoobi, danielt**
IOT-8	**Centralized Roles**	**Privilege Related**	**Low**	**Mitigated**	**danielt**
IOT-9	**Missing event record**	**Code Style**	**Informational**	**Fixed**	**danielt, 0xzoobi**
IOT-10	**Redundant `payable` keyword**	**Code Style**	**Informational**	**Acknowledged**	**danielt**
IOT-11	**Missing 2-Step-Process pattern for transferring ownership**	**Privilege Related**	**Informational**	**Acknowledged**	**0xzoobi**
IOT-12	**Use the latest solidity version**	**Language Specific**	**Informational**	**Fixed**	**0xzoobi**
IOT-13	**Gas optimization:Use calldata instead of memory**	**Gas Optimization**	**Informational**	**Fixed**	**danielt**
IOT-14	**Missing error message**	**Code Style**	**Informational**	**Fixed**	**danielt**
IOT-15	**`immutable` variables used in upgradable contract `P256Account` may cause DoS**	**DOS**	**Informational**	**Acknowledged**	**0xzoobi**

IOT-1:Replay attack on the signature

Category	Severity	Client Response	Contributor
Signature Forgery or Replay	**Critical**	**Fixed**	**grep-er, danielt**

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L160
- code/contracts/accounts/secp256r1/P256Account.sol#L165

```
160:    function withdrawDepositTo(

165:        bytes32 hash = keccak256(abi.encode(withdrawAddress, amount, getNonce(), block.chaini
d));
```

Description

grep-er : Summary: In `withdrawDepositTo()` function on P256Account attacker can reuse signature and if the `withdrawAddress` is not controlled by user it may lead to permanent loss of funds.

Impact: The `withdrawDepositTo()` function just reads the current nonce with getNonce() from BaseAccount.sol of account-abstraction dependency, but after validating the signature it doesn't increment it.

Code Snippet:

```
    function withdrawDepositTo(
        address payable withdrawAddress,
        uint256 amount,
        bytes calldata signature
    ) public {
        bytes32 hash = keccak256(abi.encode(withdrawAddress, amount, getNonce(), block.chainid));
        require(
            _validator.validateSignature(sha256(abi.encode(hash)), signature, publicKey),
            "signature invalid"
        );
//@audit just validates the signature
        entryPoint().withdrawTo(withdrawAddress, amount);
    }
```

danielt : Per the description of the EIP-4337, to prevent replay attacks (both cross-chain and multiple EntryPoint implementations), the signature should depend on chainid and the EntryPoint address.

However, the hash in the `withdrawDepositTo` function lacks checking the EntryPoint address, which would probably result in the replay attack.

```
    function withdrawDepositTo(
        address payable withdrawAddress,
        uint256 amount,
        bytes calldata signature
    ) public {
        bytes32 hash = keccak256(abi.encode(withdrawAddress, amount, getNonce(), block.chainid));
        require(
            _validator.validateSignature(sha256(abi.encode(hash)), signature, publicKey),
            "signature invalid"
        );

        entryPoint().withdrawTo(withdrawAddress, amount);
    }
```

Recommendation

grep-er : With the help of incrementNonce() function inside NonceMaster.sol update the Nonce after `validateSignature`

```
    function withdrawDepositTo(
        address payable withdrawAddress,
        uint256 amount,
        bytes calldata signature
    ) public {
        bytes32 hash = keccak256(abi.encode(withdrawAddress, amount, getNonce(), block.chainid));
        require(
            _validator.validateSignature(sha256(abi.encode(hash)), signature, publicKey),
            "signature invalid"
        );
  ++    incrementNonce(0);// 0 is for key of  nonceSequenceNumber[msg.sender][key]++;
        entryPoint().withdrawTo(withdrawAddress, amount);
    }
```

danielt : Adding the EntryPoint address into the hash in the `withdrawDepositTo` function.

Client Response

Fixed. We have updated the function to limit withdrawal methods only invoked by the account contract self.

IOT-2:The `executeBatch()` will not involve any transfer of ether.

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	grep-er, 0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L84
- code/contracts/accounts/secp256r1/P256Account.sol#L87

```
84:    function executeBatch(address[] calldata dest, bytes[] calldata func) external onlyEntryPoint
{

87:            _call(dest[i], 0, func[i]);
```

Description

grep-er : Summary: In executeBatch() function in P256Account.sol value parameter is hard-coded to `0` which makes it impossible to transfer any ether to destination address even after function is executed.

Impact: `executeBatch()` executes without any revert will give false sense that transfer of ether is been made. **Code Snippet:**

```
    function execute(
           address dest,
           uint256 value,
           bytes calldata func
       ) external onlyEntryPoint {
           _call(dest, value, func);
       }

   function executeBatch(address[] calldata dest, bytes[] calldata func) external onlyEntryPoint {
   ...
   _call(dest[i], 0, func[i]);// @audit impossible to transfer eth as value parameter is hard coded to 
   0.
   ...
   }
   function _call(
       address target,
       uint256 value,
       bytes memory data
       ) internal {
   (bool success, bytes memory result) = target.call{value: value}(data);
   ...
   }
```

0xzoobi : The `executeBatch` in turn calls the the internal function `_call` to make a low level assembly call as shown below.

```
   (bool success, bytes memory result) = target.call{value: value}(data);
```

The `{value: value}` is where we set any `msg.value` as part of the function. But in case, of the `executeBatch`, the value is set to zero.

The impact of this would be none of the `payable` functions can be called.

Recommendation

grep-er : Pass a `uint` array in parameters as `uint[] calldata _values`. And instead of hard-coded `0` use `_values[i]` when calling internal function `_call`.
0xzoobi : Update the `executeBatch` input to accept the array of values as well.

Client Response

Fixed. Added uint256[] calldata values parameter for executeBatch method.

IOT-3:Outdated OpenZeppelin dependencies

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	0xzoobi

Code Reference

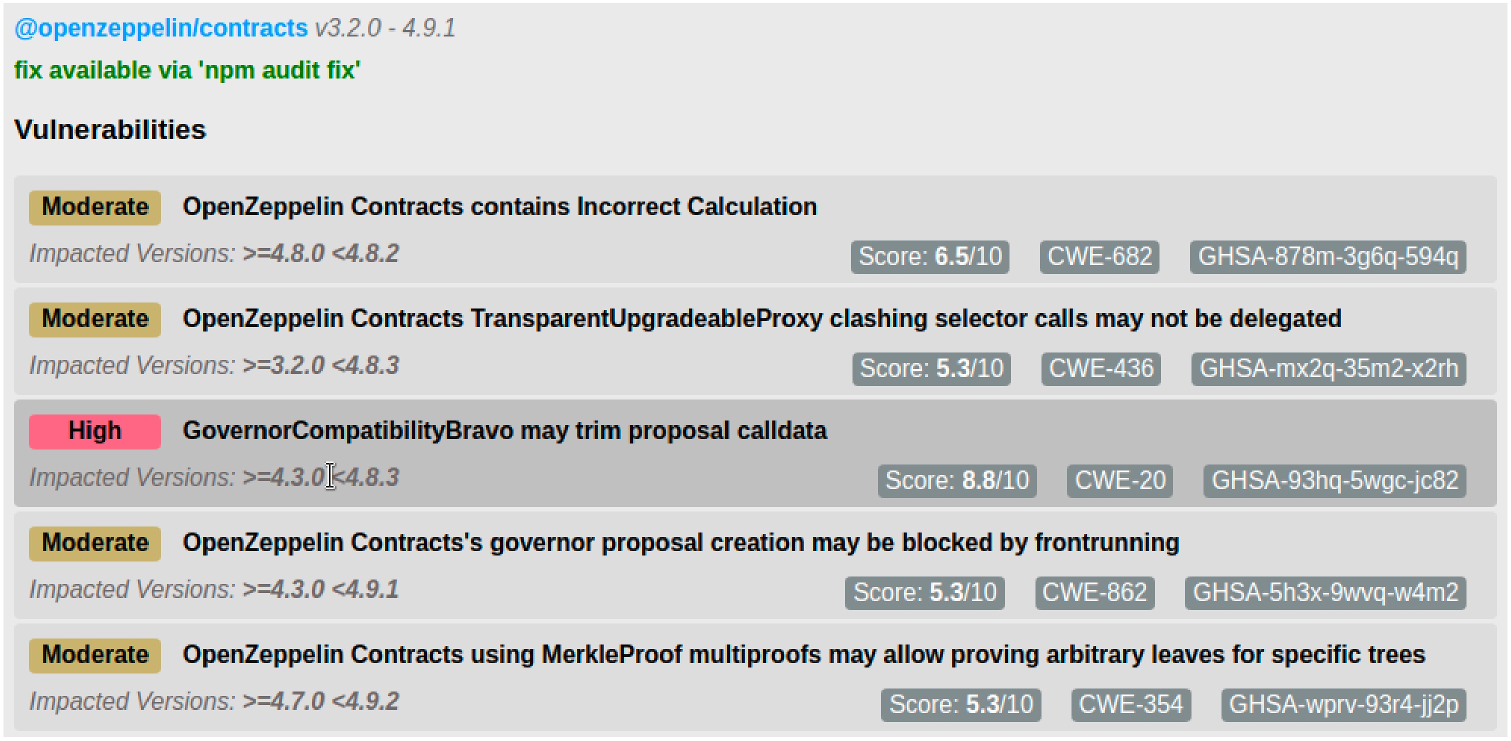
- code/package.json#L46

```
46:        "@openzeppelin/contracts": "^4.2.0"
```

Description

0xzoobi : The `4.2.0` version of openzeppelin-contracts is used in the smart contracts. The latest version is `4.9.2`. It is a good security practice to keep all libraries up-to-date

Reference to Security Advisories - https://github.com/OpenZeppelin/openzeppelin-contracts/security



@openzeppelin/contracts *v3.2.0 - 4.9.1*

fix available via 'npm audit fix'

Vulnerabilities

Moderate **OpenZeppelin Contracts contains Incorrect Calculation**
Impacted Versions: ***>=4.8.0 <4.8.2*** Score: **6.5**/10 CWE-682 GHSA-878m-3g6q-594q

Moderate **OpenZeppelin Contracts TransparentUpgradeableProxy clashing selector calls may not be delegated**
Impacted Versions: ***>=3.2.0 <4.8.3*** Score: **5.3**/10 CWE-436 GHSA-mx2q-35m2-x2rh

High **GovernorCompatibilityBravo may trim proposal calldata**
Impacted Versions: ***>=4.3.0 <4.8.3*** Score: **8.8**/10 CWE-20 GHSA-93hq-5wgc-jc82

Moderate **OpenZeppelin Contracts's governor proposal creation may be blocked by frontrunning**
Impacted Versions: ***>=4.3.0 <4.9.1*** Score: **5.3**/10 CWE-862 GHSA-5h3x-9wvq-w4m2

Moderate **OpenZeppelin Contracts using MerkleProof multiproofs may allow proving arbitrary leaves for specific trees**
Impacted Versions: ***>=4.7.0 <4.9.2*** Score: **5.3**/10 CWE-354 GHSA-wprv-93r4-jj2p

Recommendation

0xzoobi : Use the latest version of openzeppelin-contracts

Client Response

Fixed. Updated with the latest openzeppelin dependency.

IOT-4:No Storage Gap for Upgradeable Contract in `P256Account.sol`

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L20

```
20:contract P256Account is BaseAccount, TokenCallbackHandler, UUPSUpgradeable, Initializable {
```

Description

0xzoobi : For upgradeable contracts, there must be storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments". Otherwise it may be very difficult to write new implementation code. Without storage gap, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts. For Example : Storage Collision and over-write the state variables.

Refer to the storage gaps section of this article: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Recommendation

0xzoobi : Recommend adding storage gap at the end of upgradeable contracts such as the below. Please reference OpenZeppelin upgradeable contract templates.

```
uint256[50] private __gap;
```

Client Response

Mitigated. Added uint256[50] private __gap for P256Account.sol.

IOT-5: `staticall` uses a hardcoded address to verify the signature in `Secp256r1::validateSignature()`

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/Secp256r1.sol#L31

```
31:              let success := staticcall(gas(), 0x8001, input, 0xa1, out, 0x1)
```

Description

0xzoobi : The `staticcall` follows the below format as part of the EVM assembly.

```
    let success := staticcall(gas, target, input, inputSize, output, outputSize)
```

The argument#2 i.e., target is the contract address to which the static call needs to be made. In the current implementation, the target is hardcoded to `0x8001` which is non-existent according to the Iotex Block Explorer - https://iotexscout.io/address/0x8001000000000000000000000000000000000000

As a result, the `staticcall` will always revert and even though the user inputs are correct

Recommendation

0xzoobi : Pass the contract address to verify as part of the inputs to the `validateSignature` function.

Client Response

Declined. 0x8001 is the precompiled contract for secp256r1 only on the IoTeX network.

IOT-6:Avoid the use of reserved `value` keyword inside _call() function as a variable

Category	Severity	Client Response	Contributor
Code Style	Low	Acknowledged	grep-er

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L131

```
131:        (bool success, bytes memory result) = target.call{value: value}(data);
```

Description

grep-er : Summary: The `value` keyword is not interchangeable. It is a reserved keyword specifically used for specifying the amount of ether to be sent. So using this as variable name is not recommended. **Code Snippet:**

```
    function _call(
        address target,
        uint256 value,// @audit change the variable name to _value
        bytes memory data
    ) internal {
        (bool success, bytes memory result) = target.call{value: value}(data);//@audit confusing sit
uation `value:value`
        if (!success) {
            assembly {
                revert(add(result, 32), mload(result))
            }
        }
    }
```

Recommendation

grep-er : Change variable `value` to `_value`.

Client Response

Acknowledged. For code consistency, we will still use value.

IOT-7:Lack of zero address check

Category	Severity	Client Response	Contributor
Code Style	Low	Acknowledged	0xzoobi, danielt

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L58-L68
- code/contracts/accounts/secp256r1/P256AccountFactory.sol#L15-L22
- code/contracts/paymaster/VerifyingPaymaster.sol#L17-L19
- code/contracts/paymaster/VerifyingPaymaster.sol#L93-L95
- code/contracts/accounts/secp256r1/P256Account.sol#L64-L67

```
    constructor(
        IEntryPoint _entryPoint,
        ISecp256r1 _validator,
        IDkimVerifier _verifier,
        IEmailGuardian _emailGauddian
    ) {
        accountImplementation = new P256Account(_entryPoint, _validator, _verifier, _emailGauddian);
    }

    constructor(IEntryPoint _entryPoint, address _verifyingSigner) BasePaymaster(_entryPoint) {
        verifyingSigner = _verifyingSigner;
    }

    constructor(
        IEntryPoint anEntryPoint,
        ISecp256r1 aSecp256r1,
        IDkimVerifier aDkimVerifier,
        IEmailGuardian _aEmailGuardian
    ) {
        _entryPoint = anEntryPoint;
        _validator = aSecp256r1;
        _dkimVerifier = aDkimVerifier;
        _emailGuardian = _aEmailGuardian;
    }

        _entryPoint = anEntryPoint;
        _validator = aSecp256r1;
        _dkimVerifier = aDkimVerifier;
        _emailGuardian = _aEmailGuardian;

    function changeSigner(address _verifyingSigner) external onlyOwner {
        verifyingSigner = _verifyingSigner;
    }
```

Description

0xzoobi : when setting the dependencies for the contract through the constructor, the 0 address is not being checked in any of the contracts.
danielt : Zero addresses assigned to contract addresses by mistake will incur unexpected results.

Example:

```
    constructor(
        IEntryPoint anEntryPoint,
        ISecp256r1 aSecp256r1,
        IDkimVerifier aDkimVerifier,
        IEmailGuardian _aEmailGuardian
    ) {
        _entryPoint = anEntryPoint;
        _validator = aSecp256r1;
        _dkimVerifier = aDkimVerifier;
        _emailGuardian = _aEmailGuardian;
    }
```

Recommendation

0xzoobi : When setting an address variable, always make sure the value is not zero.
danielt : Consider adding zero address checks on contract addresses to prevent zero value assigning to them by mistake.

Client Response

Acknowledged. These addresses are only in the constructor and just use them once, so checking address(0) is optional.

IOT-8:Centralized Roles

Category	Severity	Client Response	Contributor
Privilege Related	Low	Mitigated	danielt

Code Reference

- code/contracts/paymaster/VerifyingPaymaster.sol#L93-L95
- code/contracts/accounts/secp256r1/SimpleEmailGuardian.sol#L21-L25

```
21:    function clean(bytes32 email) external onlyOwner {
22:        emails[users[email]] = 0;
23:        users[email] = address(0);
24:    }
25:

93:    function changeSigner(address _verifyingSigner) external onlyOwner {
94:        verifyingSigner = _verifyingSigner;
95:    }
```

Description

danielt : In the `VerifyingPaymaster` contract, the centralized role owner can update the `verifyingSigner` with the `changeSigner`, which may result in the signature from the old `verifyingSigner` failing to pass the validation.

```
    function _validatePaymasterUserOp(
        UserOperation calldata userOp,
        bytes32, /*userOpHash*/
        uint256 requiredPreFund
    ) internal override returns (bytes memory context, uint256 validationData) {
    ...
        if (verifyingSigner != ECDSA.recover(hash, signature)) {
            ...
        }
    ...
```

In the `SimpleEmailGuardian` contract, the centralized role owner can clean a user's email with the `clean` function, which may bring side-effect for the user if the email is a key property for the user.

```
    function clean(bytes32 email) external onlyOwner {
        emails[users[email]] = 0;
        users[email] = address(0);
    }
```

Recommendation

danielt : Applying the multi-signature and timelock to the centralized role owner to mitigate the centralized issue.

Client Response

Mitigated. SimpleEmailGuardian.sol is only for recording relations for email and account address. We removed this code, and switched to using subgraph to query email binding.VerifyingPaymaster.sol is a centralized service contract, so the owners role make sense.We plan to implement a new service contract without owner.

IOT-9:Missing event record

Category	Severity	Client Response	Contributor
Code Style	**Informational**	**Fixed**	**danielt, 0xzoobi**

Code Reference

- code/contracts/paymaster/VerifyingPaymaster.sol#L93-L95
- code/contracts/accounts/secp256r1/SimpleEmailGuardian.sol#L11-L24
- code/contracts/accounts/secp256r1/P256Account.sol#L36-L44

```
11:    function register(bytes32 email) external override {
12:        require(users[email] == address(0), "already bind");
13:        bytes32 preEmail = emails[msg.sender];
14:        if (preEmail != 0) {
15:            users[preEmail] = address(0);
16:        }
17:        users[email] = msg.sender;
18:        emails[msg.sender] = email;
19:    }
20:
21:    function clean(bytes32 email) external onlyOwner {
22:        emails[users[email]] = 0;
23:        users[email] = address(0);
24:    }

36:    event P256AccountInitialized(
37:        IEntryPoint indexed entryPoint,
38:        ISecp256r1 validator,
39:        IDkimVerifier verifier,
40:        IEmailGuardian emailGuardian,
41:        bytes publicKey
42:    );
43:    event EmailGuardianAdded(bytes32 email);
44:    event AccountRecovered(bytes publicKey);

93:    function changeSigner(address _verifyingSigner) external onlyOwner {
94:        verifyingSigner = _verifyingSigner;
95:    }
```

Description

danielt : The functions update key state variables are needed to emit events for logging the updates of the key state variables, which is good for on-chain tracking.

Example:

```
    function changeSigner(address _verifyingSigner) external onlyOwner {
        verifyingSigner = _verifyingSigner;
    }
```

0xzoobi : When changing state variables events are not emitted.

Events are an important feature in smart contracts and are primarily used to facilitate the communication and interaction between smart contracts and external entities, such as user interfaces and off-chain applications

Recommendation

danielt : Emitting events for the functions that update state variables.
0xzoobi : Emit an event when setting or updating an important state variable. Please make sure they are indexed if needed.

Client Response

Fixed. Added the SignerChanged event for VerifyingPaymaster.sol.

IOT-10:Redundant `payable` keyword

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	danielt

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L160-L172

```
160:    function withdrawDepositTo(
161:        address payable withdrawAddress,
162:        uint256 amount,
163:        bytes calldata signature
164:    ) public {
165:        bytes32 hash = keccak256(abi.encode(withdrawAddress, amount, getNonce(), block.chainid));
166:        require(
167:            _validator.validateSignature(sha256(abi.encode(hash)), signature, publicKey),
168:            "signature invalid"
169:        );
170:
171:        entryPoint().withdrawTo(withdrawAddress, amount);
172:    }
```

Description

danielt : In the `withdrawDepositTo` function, the parameter `withdrawAddress` is declared as a type `address payable`.
There is no necessary to add the `payable` keywords to the parameter `withdrawAddress` since there is no native tokens will be sent to the `withdrawAddress`.

Recommendation

danielt : Removing the redundant `payable` keyword for the parameter `withdrawAddress`.

Client Response

Acknowledged. This is a code style suggestion. We keep the current style.

IOT-11:Missing 2-Step-Process pattern for transferring ownership

Category	Severity	Client Response	Contributor
Privilege Related	Informational	Acknowledged	0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/SimpleEmailGuardian.sol#L4

```
4:import "@openzeppelin/contracts/access/Ownable.sol";
```

Description

0xzoobi : The contracts at path `code/contracts/accounts/secp256r1/SimpleEmailGuardian.sol` does not implement a 2-Step-Process for transferring ownership. So ownership of the contract can easily be lost when making a mistake when transferring ownership.

Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous. So Consider using the `Ownable2Step` contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol) instead.

The way it works is there is a `transferOwnership` to transfer the ownership and `acceptOwnership` to accept the ownership. Refer the above `Ownable2Step.sol` for more details.

Recommendation

0xzoobi : Implement 2-Step-Process for transferring ownership.

Client Response

Acknowledged. We plan to upgrade it in the next version. (Because this version inherits BasePaymaster , it is not straightforward to use 2-step-process pattern for transferring ownership).

IOT-12:Use the latest solidity version

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L2
- code/contracts/paymaster/VerifyingPaymaster.sol#L2
- code/contracts/accounts/secp256r1/P256AccountFactory.sol#L2
- code/contracts/accounts/secp256r1/Secp256r1.sol#L2
- code/contracts/accounts/secp256r1/SimpleEmailGuardian.sol#L2
- code/contracts/stub/import.sol#L2

```
2                       :pragma solidity ^0.8.0;

2                         :pragma solidity ^0.8.0;

2                                           :pragma solidity ^0.8.0;

2                             :pragma solidity ^0.8.0;

2               :pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;
```

Description

0xzoobi : The project is using solidity version `^0.8.0`
Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
Reference - https://swcregistry.io/docs/SWC-103

Recommendation

0xzoobi : Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Client Response

Fixed. Updated to 0.8.19

IOT-13:Gas optimization:Use calldata instead of memory

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	danielt

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L103

```
103:            publicKey
```

Description

danielt : In the `_initialize` function, the storage variable `publicKey` is read twice with the same value.
We can use the calldata variable `_publicKey` as more as we can to reduce the read of storage variable `publicKey` to save gas.

```
    function _initialize(bytes calldata _publicKey) internal virtual {
        publicKey = _publicKey;

        emit P256AccountInitialized(
            _entryPoint,
            _validator,
            _dkimVerifier,
            _emailGuardian,
            publicKey
        );
    }
```

Recommendation

danielt : Consider using the calldata variable `_publicKey` as more as we can to reduce the read of storage variable `publicKey` to save gas.

Client Response

Fixed. Updated.

IOT-14:Missing error message

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	danielt

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L151

```
151:        require(req);
```

Description

danielt : In the `addDeposit` function, there is a `require` check on the result of a call:

```
    function addDeposit() public payable {
        (bool req, ) = address(entryPoint()).call{value: msg.value}("");
        require(req);
    }
```

However, there is no error message for the `require` check, which is not a good practice.

Recommendation

danielt : Consider adding error message for the `require` check.

Client Response

Fixed. Changed `addDeposit` to use entrypoint method.

IOT-15: `immutable` variables used in upgradable contract `P256Account` may cause DoS

Category	Severity	Client Response	Contributor
DOS	Informational	Acknowledged	0xzoobi

Code Reference

- code/contracts/accounts/secp256r1/P256Account.sol#L31-L34

```
31:     IEntryPoint private immutable _entryPoint;
32:     ISecp256r1 private immutable _validator;
33:     IDkimVerifier private immutable _dkimVerifier;
34:     IEmailGuardian private immutable _emailGuardian;
```

Description

0xzoobi : The UUPS upgradable contract `P256Account` has some variables defined as `immutable` which are set during the constructor call.

This may probably work during the first deployment, but once the contract needs to be updated, the `initialize` function needs to be called, and since the `immutable` variables are stored in the contract bytecode, the new upgraded contracts will not retain these changes.

As a result, making the upgraded contract useless causing DoS

References by OpenZeppelin for the Issue and FIx

1. https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#avoid-initial-values-in-field-declarations
2. https://docs.openzeppelin.com/upgrades-plugins/1.x/faq#why-cant-i-use-immutable-variables

Recommendation

0xzoobi : Remove the `immutable` keyword and move the constructor logic to `initialize` function.

Client Response

Acknowledged. The immutable variables will not be changed.

Disclaimer