HALBORN

Biconomy - Smart Wallet Contracts V2

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 1st, 2022 - October 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/18/2022	Ataberk Yavuzer
0.2	Document Edits	10/18/2022	Ataberk Yavuzer
0.3	Draft Review	10/19/2022	Gabi Urrutia
1.0	Remediation Plan	10/29/2022	Ataberk Yavuzer
1.1	Remediation Plan Edits	11/01/2022	Ataberk Yavuzer
1.2	Remediation Plan Review	11/02/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Biconomy engaged Halborn to conduct a security audit on their smart contracts beginning on October 1st, 2022 and ending on October 17th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided nearly three weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Biconomy team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Dynamic Analysis (foundry)
- Static Analysis(slither)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Biconomy Smart Wallet Security Audit Test Scope

    (a) Repository: Smart Wallet Contracts

    (b) Commit ID: 827839c1920e104637dc19060570984e43056633

2. Out-of-Scope

    (a) contracts/Greeter.sol

    (b) contracts/test/*

    (c) contracts/modules/test/*

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	5	7

LIKELIHOOD

IMPACT

			(HAL-01)	
(HAL-03)				
	(HAL-04) (HAL-05) (HAL-06)	(HAL-02)		
(HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12) (HAL-13)	(HAL-07)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) VULNERABLE ECDSA LIBRARY	High	SOLVED - 10/28/2022
(HAL-02) MISSING REENTRANCY GUARD	Low	PARTIALLY SOLVED
(HAL-03) POSSIBLE INTEGER OVERFLOWS DUE TO IMPROPER USE OF UNCHECKED KEYWORD	Low	SOLVED - 10/28/2022
(HAL-04) SINGLE-STEP OWNERSHIP CHANGE	Low	RISK ACCEPTED
(HAL-05) MISSING ZERO ADDRESS CHECKS	Low	SOLVED - 10/28/2022
(HAL-06) IGNORED RETURN VALUES	Low	SOLVED - 11/01/2022
(HAL-07) FLOATING PRAGMA	Informational	SOLVED - 10/28/2022
(HAL-08) FOR LOOPS CAN BE OPTIMIZED	Informational	SOLVED - 11/01/2022
(HAL-09) USE DECLARED FUNCTION INSTEAD OF MSG.SENDER	Informational	SOLVED - 11/02/2022
(HAL-10) IMMUTABLE KEYWORD COSTS LESS GAS FOR CONSTANT VARIABLES	Informational	SOLVED - 10/28/2022
(HAL-11) LONG REVERT MESSAGES USE ADDITIONAL PUSH32 INSTRUCTION	Informational	ACKNOWLEDGED
(HAL-12) UNUSED VARIABLES, COMMENTS AND IMPORTS	Informational	ACKNOWLEDGED
(HAL-13) OPEN TODOS	Informational	SOLVED - 10/28/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) VULNERABLE ECDSA LIBRARY - HIGH

Description:

A vulnerability found in the ECDSA library developed by the OpenZeppelin team. OpenZeppelin team published a security advisory on GitHub on August 22nd, 2022. According to the vulnerability, recover and tryRecover functions are vulnerable, as those functions accept the standard signature format and compact signature format. (EIP-2098)

> The functions ECDSA.recover and ECDSA.tryRecover are vulnerable to some sort of signature malleability because they accept compact EIP-2098 signatures in addition to the traditional 65-byte signature format.
> This is only an issue for the functions that take a single byte argument, and not the functions that take r, v, s or r, vs as separate arguments.

> Potentially affected contracts are those that implement signature reuse or replay protection by marking the signature itself as used, rather than the signed message or a nonce included in it. A user can take a signature that has already been submitted, submit it again in a different form, and bypass this protection.

Affected Versions
>= 4.1.0 < 4.7.3

It was observed that Biconomy contracts were using the vulnerable ECDSA library to recover signatures.

Reference:

ECDSA Signature Malleability

Code Location:

Listing 1: package.json (Line 58)

```
58 "@openzeppelin/contracts": "^4.2.0",
59 "@openzeppelin/contracts-upgradeable": "^4.7.3",
```

Proof of Concept:

The PoC code below proofs that both signatures addresses the same address, as they are identical but in different formats.

Listing 2: ECDSA vulnerability test case - PoC

```
1  function testValidateUserOpPaymaster() public {
2
3          UserOperation memory userOp = UserOperation({
4              sender: user1,
5              nonce: 0,
6              initCode:  new bytes(0x0),
7              callData:  new bytes(0x0),
8              callGasLimit: 0,
9              verificationGasLimit: 0,
10             preVerificationGas: 0,
11             maxFeePerGas: 0,
12             maxPriorityFeePerGas: 0,
13             paymasterAndData: new bytes(0x0),
14             signature: new bytes(0x0)
15         });
16
17         bytes32 hash = paymentMasterNew.getHash(userOp);
18
19         (uint8 v, bytes32 r, bytes32 s) = vm.sign(101, hash.
↳ toEthSignedMessageHash());
20         bytes memory signature = abi.encodePacked(r, s, v);
21         bytes memory paymasterAndData = abi.encodePacked(user1,
↳ signature);
```

```
22
23          userOp.paymasterAndData = paymasterAndData;
24
25          paymentMasterNew.validatePaymasterUserOp(userOp, bytes32(
↳ uint256(1)), 0); // standart signature
26
27          // standart sig: 0xe34...b2a0a094...6c1c
28          // short sig(to2098format): 0xe34...baa0a094...6c
29
30          bytes memory shortSig2098 = hex"
↳ e3402d57abf6818387638870 91f7d4fbb52cd03073a375dd010a1dd0e251f87baa0a0948e6a3579
↳ ";
31
32          paymasterAndData = abi.encodePacked(user1, shortSig2098);
33
34          userOp.paymasterAndData = paymasterAndData;
35
36          paymentMasterNew.validatePaymasterUserOp(userOp, bytes32(
↳ uint256(1)), 0); // same signature(eip2098 format)
37      }
```

```
[40916] ECDSATest::testValidateUserOpPaymaster()
    ├─ [6462] VerifyingPaymasterProxy::getHash((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0x, 0x)) [staticcall]
    │   ├─ [1564] VerifyingPaymaster::getHash((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0x, 0x)) [delegatecall]
    │   │   └─ ← 0xe7e7cc55ccd57359955ada660916971b2b141bd090147eba511d74eacf81dc32
    │   └─ ← 0xe7e7cc55ccd57359955ada660916971b2b141bd090147eba511d74eacf81dc32
    ├─ [0] VM::sign(101, 0xf0f4cb84a5917c49e231c7ee757d9ced588a22460e2800dfdc416ae017493d90)
    │   └─ ← 28, 0xe3402d57abf6818387638870 91f7d4fbb52cd03073a375dd010a1dd0e251f87b, 0x2a0a0948e6a3579f7b88876286d372aaaf46f2af8e4db9a2a59845130d174a6c
    ├─ [9036] VerifyingPaymasterProxy::validatePaymasterUserOp((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0xe6b3367318c5e11a6eed3cd0d850ec06a02e9b90e3402d57abf6818387638870 91f7d4fbb52cd0
3073a375dd010a1dd0e251f87b2a0a0948e6a3579f7b88876286d372aaaf46f2af8e4db9a2a59845130d174a6c1c, 0x), 0x0000000000000000000000000000000000000000000000000000000000000001, 0) [staticcall]
    │   ├─ [8605] VerifyingPaymaster::validatePaymasterUserOp((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0xe6b3367318c5e11a6eed3cd0d850ec06a02e9b90e3402d57abf6818387638870 91f7d4fbb52cd03
073a375dd010a1dd0e251f87b2a0a0948e6a3579f7b88876286d372aaaf46f2af8e4db9a2a59845130d174a6c1c, 0x), 0x0000000000000000000000000000000000000000000000000000000000000001, 0) [delegatecall]
    │   │   ├─ [3000] PRECOMPILE::ecrecover(0xf0f4cb84a5917c49e231c7ee757d9ced588a22460e2800dfdc416ae017493d90, 28, 102788407782758060901540305678193327742982367690542355194987917623816441017 55, 190148721922672755
040726464672132069403913040373436884814423082455678587562 04) [staticcall]
    │   │   │   └─ ← 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90
    │   │   └─ ← 0x
    │   └─ ← 0x
    ├─ [7132] VerifyingPaymasterProxy::validatePaymasterUserOp((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0xe6b3367318c5e11a6eed3cd0d850ec06a02e9b90e3402d57abf6818387638870 91f7d4fbb52cd0
3073a375dd010a1dd0e251f87baa0a0948e6a3579f7b88876286d372aaaf46f2af8e4db9a2a59845130d174a6c, 0x), 0x0000000000000000000000000000000000000000000000000000000000000001, 0) [staticcall]
    │   ├─ [6701] VerifyingPaymaster::validatePaymasterUserOp((0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 0, 0x, 0x, 0, 0, 0, 0, 0, 0xe6b3367318c5e11a6eed3cd0d850ec06a02e9b90e3402d57abf6818387638870 91f7d4fbb52cd03
073a375dd010a1dd0e251f87baa0a0948e6a3579f7b88876286d372aaaf46f2af8e4db9a2a59845130d174a6c, 0x), 0x0000000000000000000000000000000000000000000000000000000000000001, 0) [delegatecall]
    │   │   ├─ [3000] PRECOMPILE::ecrecover(0xf0f4cb84a5917c49e231c7ee757d9ced588a22460e2800dfdc416ae017493d90, 28, 102788407782758060901540305678193327742982367690542355194987917623816441017 55, 190148721922672755
040726464672132069403913040373436884814423082455678587562 04) [staticcall]
    │   │   │   └─ ← 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90
    │   │   └─ ← 0x
    │   └─ ← 0x
    └─ ← ()

Test result: ok. 1 passed; 0 failed; finished in 4.29ms
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is suggested to update the version of the @openzeppelin/contracts package version to 4.7.3 to fix this finding.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by upgrading the @openzeppelin/contracts package version to 4.7.3.

Commit ID: b813a60a8475672c2a2d00b7ef5d1839461eaa3e

3.2 (HAL-02) MISSING REENTRANCY GUARD - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

It is also suggested to use call.value method with any Reentrancy protection.

Code Location:

Listing 3: Functions with missing Reentrancy Guard

```
SmartWallet:handlePayment()
SmartWallet:transfer()
SmartWalletNoAuth:handlePayment()
SmartWalletNoAuth:transfer()
EntryPoint:innerHandleOp()
StakeManager:withdrawStake()
StakeManager:withdrawTo()
SimpleWallet:addDeposit()
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

The functions in the code location section are missing nonReentrant modifiers. It is recommended to add the necessary nonReentrant modifier to prevent the introduction of future re-entrancy vulnerabilities.

Remediation Plan:

PARTIALLY SOLVED: The Biconomy team solved this finding by implementing nonReentrant modifiers to functions in the **Code Location** section. This finding will not be fixed for Account Abstraction Core contracts, as they are templates.

Commit ID: 8624fcf9a92a006a2fc6bad7bb9f834ab3641ffe

3.3 (HAL-03) POSSIBLE INTEGER OVERFLOWS DUE TO IMPROPER USE OF UNCHECKED KEYWORD - LOW

Description:

As of version **0.8.0** in Solidity, a control mechanism was implemented to prevent overflow and underflow issues for mathematical operations. Since this costs extra gas, the unchecked keyword was also implemented to remove this extra control mechanism for specific code locations.

This keyword should be used with caution to benefit from gas consumption. Otherwise, overflows/underflows for mathematical operations can occur.

Code Location:

Listing 4: EntryPoint.sol (Line 264)

```
259 function _getRequiredPrefund(MemoryUserOp memory mUserOp) internal
 ↳  view returns (uint256 requiredPrefund) {
260     unchecked {
261         //when using a Paymaster, the verificationGasLimit is used
 ↳  also to as a limit for the postOp call.
262         // our security model might call postOp eventually twice
263         uint256 mul = mUserOp.paymaster != address(0) ? 3 : 1;
264         uint256 requiredGas = mUserOp.callGasLimit + mUserOp.
 ↳ verificationGasLimit * mul + mUserOp.preVerificationGas;
265
266         // TODO: copy logic of gasPrice?
267         requiredPrefund = requiredGas * getUserOpGasPrice(mUserOp)
 ↳ ;
268     }
```

Listing 5: EntryPoint.sol (Line 455)

```
442 function _handlePostOp(uint256 opIndex, IPaymaster.PostOpMode mode
 ↳ , UserOpInfo memory opInfo, bytes memory context, uint256
 ↳ actualGas) private returns (uint256 actualGasCost) {
```

```
443         uint256 preGas = gasleft();
444     unchecked {
445         address refundAddress;
446         MemoryUserOp memory mUserOp = opInfo.mUserOp;
447         uint256 gasPrice = getUserOpGasPrice(mUserOp);
448 
449         address paymaster = mUserOp.paymaster;
450         if (paymaster == address(0)) {
451             refundAddress = mUserOp.sender;
452         } else {
453             refundAddress = paymaster;
454             if (context.length > 0) {
455                 actualGasCost = actualGas * gasPrice;
456                 if (mode != IPaymaster.PostOpMode.postOpReverted)
 ↳ {
457                     IPaymaster(paymaster).postOp{gas : mUserOp.
 ↳ verificationGasLimit}(mode, context, actualGasCost);
458                 } else {
459                     // solhint-disable-next-line no-empty-blocks
460                     try IPaymaster(paymaster).postOp{gas : mUserOp
 ↳ .verificationGasLimit}(mode, context, actualGasCost) {}
461                     catch Error(string memory reason) {
462                         revert FailedOp(opIndex, paymaster, reason
 ↳ );
463                     }
464                     catch {
465                         revert FailedOp(opIndex, paymaster, "
 ↳ postOp revert");
466                     }
467                 }
468             }
469         }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider removing unchecked keyword for the **multiply** operations above.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by removing the unchecked keywords directly from the above functions. That eliminates the risk of overflow.

Commit ID: e3080c259a580b3ddc205776a2aeca999850b0c0

3.4 (HAL-04) SINGLE-STEP OWNERSHIP CHANGE - LOW

Description:

A two-step procedure for changing roles in the contract is missing. If the address is incorrect, the new address will assume the functionality of the new role directly. There is no way to recover these roles back after these types of errors.

Code Location:

Listing 6: SmartWallet.sol (Line 103)

```
100 function setOwner(address _newOwner) external mixedAuth {
101         require(_newOwner != address(0), "Smart Account:: new
 ↳ Signatory address cannot be zero");
102         address oldOwner = owner;
103         owner = _newOwner;
104         emit EOAChanged(address(this), oldOwner, _newOwner);
105     }
```

Listing 7: SmartWalletNoAuth.sol (Line 103)

```
100 function setOwner(address _newOwner) external mixedAuth {
101         require(_newOwner != address(0), "Smart Account:: new
 ↳ Signatory address cannot be zero");
102         address oldOwner = owner;
103         owner = _newOwner;
104         emit EOAChanged(address(this), oldOwner, _newOwner);
105     }
```

Listing 8: BasePaymaster.sol (Line 63)

```
 61 function _transferOwnership(address newOwner) internal virtual {
 62         address oldOwner = owner;
 63         owner = newOwner;
 64         emit OwnershipTransferred(oldOwner, newOwner);
```

```
65     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is suggested to add two-step ownership change for the specified contracts above.

Remediation Plan:

RISK ACCEPTED: The Biconomy team accepted the risk of this finding. It has been decided to continue with the one-step ownership change pattern.

3.5 (HAL-05) MISSING ZERO ADDRESS CHECKS - LOW

Description:

Biconomy Smart Wallet contracts have address fields in multiple functions. These functions are missing address validations. Each address should be validated and checked to be non-zero. This is also considered a best practice.

During testing, it has been found that some of these inputs are not protected against using address(0) as the destination address.

Code Location:

Listing 9: Missing zero address checks - Variables

```
StakeManager.withdrawStake(address).withdrawAddress
StakeManager.withdrawTo(address,uint256).withdrawAddress
VerifyingPaymasterFactory.deployVerifyingPaymaster(address,address
↳ ,IEntryPoint).owner
VerifyingPaymasterFactory.deployVerifyingPaymaster(address,address
↳ ,IEntryPoint).verifyingSigner
VerifyingPaymaster.init(IEntryPoint,address,address).
↳ _verifyingSigner
VerifyingPaymaster.init(IEntryPoint,address,address)._owner
VerifyingPaymaster.setSigner(address)._newVerifyingSigner
VerifyingPaymaster.init(IEntryPoint,address,address).
↳ _verifyingSigner
VerifyingPaymaster.init(IEntryPoint,address,address)._owner
VerifyingPaymaster.setSigner(address)._newVerifyingSigner
StakeManager.withdrawStake(address).withdrawAddress
StakeManager.withdrawTo(address,uint256).withdrawAddress
BasePaymaster.setEntryPoint(IEntryPoint)._entryPoint
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to validate that each address input is non-zero.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by adding require checks to prevent the use of zero addresses.

Commit ID: 10d3933d5b9a461b85aebb9343e92ed94ac74970

3.6 (HAL-06) IGNORED RETURN VALUES - LOW

Description:

The requiredTxGas function was declared to return uint256 as a return variable after a successful call. That function calls execute() function, and that function was designed to return bool, not uint256. Therefore, it does not return any variables at all. It is important to validate these return variables. In this case, calling these functions can break any integrations or composability.

Code Location:

Listing 10: SmartWalletNoAuth.sol (Line 362)

```
357 function requiredTxGas(
358         address to,
359         uint256 value,
360         bytes calldata data,
361         Enum.Operation operation
362     ) external returns (uint256) {
363         // We don't provide an error message here, as we use it to
 ↳  return the estimate
364         require(execute(to, value, data, operation, gasleft()));
365     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider returning an uint256 value for the function above.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by removing the return variable from the requiredTxGas function.

Commit ID: 508c44ed749f7c4b06e704d6f6b4e13db9099634

3.7 (HAL-07) FLOATING PRAGMA - INFORMATIONAL

Description:

The project contains many instances of floating pragma. 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, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too recent which has not been extensively tested.

During the audit, it has seen all contracts use `^0.8.0` as pragma version.

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Consider locking the pragma version with known bugs for the compiler version by removing the caret (^) symbol. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: This finding was solved by updating the Solidity version from `^0.8.0` to `0.8.12`. All caret symbols have been removed from these contracts.

Commit ID: 3b882439d3981db3b1c992fefa18e4b0f80746d4

3.8 (HAL-08) FOR LOOPS CAN BE OPTIMIZED - INFORMATIONAL

Description:

It has been observed all `for` loops in the protocol were not optimized properly. Having not optimized for loops can cost too much gas usage.

These for loops can be optimized with suggestions above:

1. In Solidity (pragma 0.8.0 and later), adding `unchecked` keyword for arithmetical operations can reduce gas usage on contracts where underflow/underflow is unrealistic. It is possible to save gas by using this keyword on multiple code locations.
2. In all for loops, the `index` variable is incremented using `+=`. It is known that, in loops, using `++i` costs less gas per iteration than `+=`. This also affects incremented variables within the loop code block.
3. Do not initialize `index` variables with `0`, Solidity already initializes these `uint` variables as zero.

Check the `Recommendation` section for further details.

Code Location:

Listing 11: Optimizable For Loops - LoC list

```
core/EntryPoint.sol:#L81
core/EntryPoint.sol:#L87
core/EntryPoint.sol:#L107
core/EntryPoint.sol:#L119
core/EntryPoint.sol:#L140
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to apply the following pattern for Solidity pragma version 0.8.0 and later.

Listing 12: Possible Suggestion

```
for (uint256 i; i < arrayLength; ) {
    . . .
    unchecked {
     ++i
    }
```

Remediation Plan:

SOLVED: The Biconomy team solved the issue by optimizing these for loops as per the suggestion above.

Commit ID: 0dc8ee23591fd54318abdd8d892ab84958847cd0

3.9 (HAL-09) USE DECLARED FUNCTION INSTEAD OF MSG.SENDER - INFORMATIONAL

Description:

The BasePaymaster contract uses a modifier to control functions are called by EntryPoint address. The other onlyOwner modifier uses _msgSender() function to return msg.sender address. However, the _requireFromEntryPoint modifier uses msg.sender directly. Only one of these pattern should be used to increase readability.

Code Location:

Listing 13: BasePaymaster.sol (Line 143)

```
142 function _requireFromEntryPoint() internal virtual {
143         require(msg.sender == address(entryPoint));
144     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing the _msgSender() function or change the _requireFromEntryPoint modifier as below.

Listing 14: BasePaymaster.sol - Possible Fix

```
1 function _requireFromEntryPoint() internal virtual {
2         require(_msgSender() == address(entryPoint));
3     }
```

Remediation Plan:

SOLVED: The Biconomy team solved this finding by replacing the msg.sender to _msgSender() function.

Commit ID: f44e1592bbaa90569fd10f4418a4534fa7ff2acd

3.10 (HAL-10) IMMUTABLE KEYWORD COSTS LESS GAS FOR CONSTANT VARIABLES - INFORMATIONAL

Description:

Some variables in Smart Wallet contracts are only set for once. The `immutable` keyword consumes less gas usage. It might be useful to declare these variables as `immutable` to optimize gas usage in the protocol.

Code Location:

Listing 15: WalletFactory.sol

```
9 address internal _defaultImpl;
```

Listing 16: VerifyingPaymasterFactory.sol

```
9 address internal payMasterImp;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is suggested to `immutable` keyword for these constant variables to optimize gas usage.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by adding immutable keyword for constant variables.

Commit ID: d7501353283b9a98295d58e49e051f14f4b4be1b

3.11 (HAL-11) LONG REVERT MESSAGES USE ADDITIONAL PUSH32 INSTRUCTION - INFORMATIONAL

Description:

Revert messages longer than 32 bytes use extra PUSH32 instruction to push strings. Since this instruction use extra gas cost, it is not recommended to use revert messages longer than 32 bytes.

Shortening revert strings to fit in 32 bytes will decrease deployment time gas and will decrease runtime gas usage when the revert condition has been met.

Code Location:

Listing 17: Long revert messages - LoC list

```
SmartWallet.sol:#L72
SmartWallet.sol:#L101
SmartWallet.sol:#L119
WalletFactory.sol:#L18
SmartWalletNoAuth.sol:#L72
SmartWalletNoAuth.sol:#L101
SmartWalletNoAuth.sol:#L119
MultiSend.sol:#L27
VerifyingPaymasterFactory.sol:#L14
VerifyingPaymaster.sol:#L81
BasePaymaster.sol:#L53
WhitelistModule.sol:#L20
WhitelistModule.sol:#L26
WhitelistModule.sol:#L46
WhitelistModule.sol:#L49
EntryPoint.sol:#L256
EntryPoint.sol:#L276
EntryPoint.sol:#L277
EntryPoint.sol:#L304
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using revert messages shorter than 32 bytes.

Remediation Plan:

ACKNOWLEDGED: The Biconomy team acknowledged this finding.

3.12 (HAL-12) UNUSED VARIABLES, COMMENTS AND IMPORTS - INFORMATIONAL

Description:

There are numerous unused variables, events, and imports on the code base. These variables should be cleaned up from the code if they have no purpose. Clearing these variables can reduce gas usage during deployment of contracts. Also, clearing unneeded comments will increase readability of the code.

Code Location:

Listing 18: Unused comments, contracts and imports - LoC list

```
1 Greeter.sol:#L0 - Unused Contract
2 BaseSmartWallet.sol:#L18 - Unneeded comment line
3 WalletFactory.sol:#L4 - Unneeded comment line
4 IAggregatedWallet.sol:#L4,L6 - Unused import
5 BasePaymaster.sol:#L88 - Unused variables
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing unused variables, events, and imports from the code.

Remediation Plan:

ACKNOWLEDGED: The Biconomy team acknowledged this finding.

3.13 (HAL-13) OPEN TODOS - INFORMATIONAL

Description:

Open TODOs can hint at programming or architectural errors that still need to be fixed.

Code Location:

Listing 19: Open TODOs - LoC list

```
1 SmartWallet.sol:#L140
2 SmartWalletNoAuth.sol:#L140
3 EntryPoint.sol:#L266
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to resolve these TODOs.

Remediation Plan:

SOLVED: The Biconomy team solved this finding by removing open TODOs from their contracts.

Commit ID: 5c22c474c90737611cd99d280eb69464cb235d2e

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Slither, a Solidity static analysis framework, was used for static analysis. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
VerifyingPaymaster.init(IEntryPoint,address,address)._verifyingSigner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#30) lacks a zero-check on :
                - verifyingSigner = _verifyingSigner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#34)
VerifyingPaymaster.init(IEntryPoint,address,address)._owner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#30) lacks a zero-check on :
                - owner = _owner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#36)
VerifyingPaymaster.setSigner(address)._newVerifyingSigner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#41) lacks a zero-check on :
                - verifyingSigner = _newVerifyingSigner (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

UserOperationLib.getSender(UserOperation) (contracts/smart-contract-wallet/aa-4337/interfaces/UserOperation.sol#36-41) uses assembly
        - INLINE ASM (contracts/smart-contract-wallet/aa-4337/interfaces/UserOperation.sol#39)
UserOperationLib.pack(UserOperation) (contracts/smart-contract-wallet/aa-4337/interfaces/UserOperation.sol#57-71) uses assembly
        - INLINE ASM (contracts/smart-contract-wallet/aa-4337/interfaces/UserOperation.sol#63-70)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#65-69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2']
        - ^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/IAggregator.sol#2)
        - ^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/IEntryPoint.sol#6)
        - ^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/IPaymaster.sol#2)
        - ^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/IStakeManager.sol#2)
        - ^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/UserOperation.sol#2)
        - ^0.8.0 (contracts/smart-contract-wallet/paymasters/BasePaymaster.sol#2)
        - ^0.8.0 (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymaster.sol#2)
        - ^0.8.2 (node_modules/@openzeppelin/contracts/proxy/utils/Initializable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#95-101) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#118-126) is never used and should be removed
ECDSA.recover(bytes32,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#169-178) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#202-204) is never used and should be removed
ECDSA.toTypedDataHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#215-217) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#103-111) is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts/proxy/utils/Initializable.sol#131-137) is never used and should be removed
Strings.toHexString(address) (node_modules/@openzeppelin/contracts/utils/Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#41-52) is never used and should be removed
```

```
EntryPoint._compensate(address,uint256) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#42-46) sends eth to arbitrary user
        Dangerous calls:
        - (success) = beneficiary.call{value: amount}() (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Reentrancy in EntryPoint._validatePrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,address) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#391-429):
        External calls:
        - (gasUsedByValidateWalletPrepayment,actualAggregator) = _validateWalletPrepayment(opIndex,userOp,outOpInfo,aggregator,requiredPreFund) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#407)
                - IWallet(sender).validateUserOp{gas: mUserOp.verificationGasLimit}(op,opInfo.requestId,aggregator,missingWalletFunds) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#334-339)
        - context = _validatePaymasterPrepayment(opIndex,userOp,outOpInfo,requiredPreFund,gasUsedByValidateWalletPrepayment) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#415)
                - IPaymaster(paymaster).validatePaymasterUserOp{gas: gas}(op,opInfo.requestId,requiredPreFund) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#374-380)
        State variables written after the call(s):
        - context = _validatePaymasterPrepayment(opIndex,userOp,outOpInfo,requiredPreFund,gasUsedByValidateWalletPrepayment) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#415)
                - paymasterInfo.deposit = uint112(deposit - requiredPreFund) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#372)
Reentrancy in EntryPoint._validateWalletPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,address,uint256) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#313-350):
        External calls:
        - IWallet(sender).validateUserOp{gas: mUserOp.verificationGasLimit}(op,opInfo.requestId,aggregator,missingWalletFunds) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#334-339)
        State variables written after the call(s):
        - senderInfo.deposit = uint112(deposit - requiredPreFund) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#346)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

EntryPoint._executeUserOp(uint256,UserOperation,EntryPoint.UserOpInfo)._actualGasCost (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#59) is a local variable never initialized
EntryPoint._handlePostOp(uint256,IPaymaster.PostOpMode,EntryPoint.UserOpInfo,bytes,uint256).reason (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#461) is a local variable never initialized
EntryPoint._validateWalletPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,address,uint256).revertReason (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#335) is a local variable never initialized
EntryPoint.simulateValidation(UserOperation,bool).outOpInfo (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#246) is a local variable never initialized
EntryPoint._validatePaymasterPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,uint256,uint256)._context (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#374) is a local variable never initialized
EntryPoint._validatePaymasterPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,uint256,uint256).revertReason (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#376) is a local variable never initialized
EntryPoint._validateWalletPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,address,uint256).userOpAggregator (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#320) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

EntryPoint._executeUserOp(uint256,UserOperation,EntryPoint.UserOpInfo) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#55-65) ignores return value by this.innerHandleOp(userOp.callData,opInfo,context) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#59-64)
EntryPoint._validateWalletPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,address,uint256) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#313-350) ignores return value by IAggregatedWallet(mUserOp.sender).getAggregator() (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#320-324)
EntryPoint._validatePaymasterPrepayment(uint256,UserOperation,EntryPoint.UserOpInfo,uint256,uint256) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#359-382) ignores return value by IPaymaster(paymaster).validatePaymasterUserOp{gas: gas}(op,opInfo.requestId,requiredPreFund) (contracts/smart-contract-wallet/aa-4337/core/EntryPoint.sol#374-380)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

StakeManager.withdrawStake(address).withdrawAddress (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#104) lacks a zero-check on :
                - (success) = withdrawAddress.call{value: stake}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#114)
StakeManager.withdrawTo(address,uint256).withdrawAddress (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#123) lacks a zero-check on :
                - (success) = withdrawAddress.call{value: withdrawAmount}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#128)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
StakeManager.withdrawStake(address).withdrawAddress (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#104) lacks a zero-check on :
                - (success) = withdrawAddress.call{value: stake}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#114)
StakeManager.withdrawTo(address,uint256).withdrawAddress (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#123) lacks a zero-check on :
                - (success) = withdrawAddress.call{value: withdrawAmount}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#128)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

StakeManager.unlockStake() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#88-96) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.unstakeDelaySec != 0,not staked) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#90)
        - require(bool,string)(info.staked,already unstaking) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#91)
StakeManager.withdrawStake(address) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#104-116) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.withdrawTime <= block.timestamp,Stake withdrawal is not due) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#109)
StakeManager.withdrawTo(address,uint256) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#123-130) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(withdrawAmount <= info.deposit,Withdraw amount too large) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Pragma version^0.8.0 (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#2) allows old versions
Pragma version^0.8.0 (contracts/smart-contract-wallet/aa-4337/interfaces/IStakeManager.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in StakeManager.withdrawStake(address) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#104-116):
        - (success) = withdrawAddress.call{value: stake}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#114)
Low level call in StakeManager.withdrawTo(address,uint256) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#123-130):
        - (success) = withdrawAddress.call{value: withdrawAmount}() (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#128)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter StakeManager.addStake(uint32)._unstakeDelaySec (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#67) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

getDepositInfo(address) should be declared external:
        - StakeManager.getDepositInfo(address) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#33-35)
balanceOf(address) should be declared external:
        - StakeManager.balanceOf(address) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#38-40)
addStake(uint32) should be declared external:
        - StakeManager.addStake(uint32) (contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol#67-82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
contracts/smart-contract-wallet/aa-4337/core/StakeManager.sol analyzed (2 contracts with 78 detectors), 14 result(s) found
```

```
Contract locking ether found:
        Contract VerifyingPaymasterProxy (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#4-37) has payable functions:
         - VerifyingPaymasterProxy.fallback() (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#18-30)
         - VerifyingPaymasterProxy.receive() (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#32-34)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

VerifyingPaymasterProxy.constructor(address) (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#11-16) uses assembly
        - INLINE ASM (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#13-15)
VerifyingPaymasterProxy.fallback() (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#18-30) uses assembly
        - INLINE ASM (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#21-29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Pragma version^0.8.0 (contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
contracts/smart-contract-wallet/paymasters/verifying/VerifyingPaymasterProxy.sol analyzed (1 contracts with 78 detectors), 5 result(s) found
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

THANK YOU FOR CHOOSING

// HALBORN