// HALBORN

Immutable - Wallet

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **June 28th, 2023 - July 25th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/19/2023	Alejandro Taibo
0.2	Document Update	07/25/2023	Alejandro Taibo
0.3	Draft Review	07/25/2023	Gokberk Gulgun
0.4	Draft Review	07/25/2023	Gabi Urrutia
1.0	Remediation Plan	08/18/2023	Alejandro Taibo
1.1	Remediation Plan Review	08/18/2023	Gokberk Gulgun
1.2	Remediation Plan Review	08/18/2023	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Alejandro Taibo	Halborn	Alejandro.Taibo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Immutable engaged Halborn to conduct a security assessment on their smart contracts beginning on June 28th, 2023 and ending on July 25th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to verify 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 assessment 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 Immutable team.

1.3 TEST APPROACH & METHODOLOGY

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

- 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.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Brownie, Anvil, Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

IN-SCOPE CODE & COMMIT:

- Repository: wallet-contracts
  - Commit ID: 84dd5e1b6c2f0c2e78e6162a57bf82aa42e154dd

Out-of-scope: external libraries and financial related attacks.

REMEDIATION COMMITS:

- Repository: wallet-contracts
  - Pull request ID: 31
  - Commit IDs:
    - 6b91e4013b768254de3ad5fd8b4d85423f7204c5
    - fdd8cd20d93f134f58396812214e02932d006f6f
    - bbaddf196d02cc9f727b647fd3da210ef9881277
    - b82bf46b8e93d7c1ace6d668b1442d1458cdb3bb
    - 99819fa06d8677c6ce6a5319b93fe9e4a72403a6

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CURRENT IMAGEHASH VERIFICATION DOES NOT FOLLOW THE SPECIFICATION	Medium (5.6)	SOLVED - 08/18/2023
(HAL-02) MODULECREATOR DOES NOT VERIFY WHETHER A CONTRACT HAS BEEN DEPLOYED	Low (3.1)	SOLVED - 08/18/2023
(HAL-03) USING KECCAK256 RESULTS DIRECTLY FROM A STRING AS STORAGE KEYS COULD LEAD TO STORAGE COLLISIONS	Low (3.1)	SOLVED - 08/18/2023
(HAL-04) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational (0.0)	ACKNOWLEDGED
(HAL-05) UNNECESSARY BOOLEAN EXPRESSION	Informational (0.0)	ACKNOWLEDGED
(HAL-06) FLOATING PRAGMA	Informational (0.0)	SOLVED - 08/18/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) CURRENT IMAGEHASH VERIFICATION DOES NOT FOLLOW THE SPECIFICATION - MEDIUM (5.6)

Description:

Following the ImageHash specification, there are several levels of nested hashes, in this case three nested hashes in order to validate the threshold and both signers implied in the wallet.

Listing 1

```
keccak256(abi.encode(1, address1, keccak256(abi.encode(1, address2
↳ , keccak256(abi.encode(2))))))
```

However, the threshold value is not hashed during verification in the _signatureValidationWithUpdateCheck function. Instead, its value is converted to bytes32 and appended to the second nested keccak256 hash.

Moreover, the ImageHash verification does not follow the defined order in its specification. Therefore, in order to make a valid ImageHash, it should be generated following the logic defined below.

Listing 2

```
bytes32 imageHash = keccak256(abi.encode(
            keccak256(abi.encode(
                bytes32(uint256(2)),
                uint8(1), _signerA)),
            uint8(1), _signerB
        ));
```

As it can be seen, this logic is completely different compared to the specification.

Code Location:

Listing 3: src/contracts/modules/commons/ModuleAuth.sol (Lines 85,127)

```
84 // Start image hash generation
85 bytes32 imageHash = bytes32(uint256(threshold));
86
87 // Acumulated weight of signatures
88 uint256 totalWeight;
89
90 // Iterate until the image is completed
91 while (rindex < _signature.length) {
92   // Read next item type and addrWeight
93   uint256 flag; uint256 addrWeight; address addr;
94   (flag, addrWeight, rindex) = _signature.readUint8Uint8(rindex);
95
96   if (flag == FLAG_ADDRESS) {
97     // Read plain address
98     (addr, rindex) = _signature.readAddress(rindex);
99   } else if (flag == FLAG_SIGNATURE) {
100    // Read single signature and recover signer
101    bytes memory signature;
102    (signature, rindex) = _signature.readBytes66(rindex);
103    addr = recoverSigner(_hash, signature);
104
105    // Acumulate total weight of the signature
106    totalWeight += addrWeight;
107  } else if (flag == FLAG_DYNAMIC_SIGNATURE) {
108    // Read signer
109    (addr, rindex) = _signature.readAddress(rindex);
110
111    // Read signature size
112    uint256 size;
113    (size, rindex) = _signature.readUint16(rindex);
114
115    // Read dynamic size signature
116    bytes memory signature;
117    (signature, rindex) = _signature.readBytes(rindex, size);
118    require(isValidSignature(_hash, addr, signature), "ModuleAuth#
 ↳ _signatureValidation: INVALID_SIGNATURE");
119
120    // Acumulate total weight of the signature
121    totalWeight += addrWeight;
122  } else {
123    revert("ModuleAuth#_signatureValidation INVALID_FLAG");
```

```
124   }
125
126   // Write weight and address to image
127   imageHash = keccak256(abi.encode(imageHash, addrWeight, addr));
128 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:L/D:N/Y:N/R:N/S:U (5.6)

Recommendation:

It is recommended to adapt the ImageHash verification logic to the specification, which is also used in 0xSequence smart contracts.

Remediation Plan:

SOLVED: the Immutable team solved this issues by modifying the specification instead of the code in the following commit:

- 6b91e4013b768254de3ad5fd8b4d85423f7204c5

4.2 (HAL-02) MODULECREATOR DOES NOT VERIFY WHETHER A CONTRACT HAS BEEN DEPLOYED - LOW (3.1)

Description:

There is a module named ModuleCreator that allows to deploy new smart contracts on behalf of the wallet through the create op-code in assembly.

However, the return value associated to executing the create op-code should be verified in order to confirm a smart contract has been created properly. In this case, the aforementioned return value should be different from zero address.

Code Location:

Listing 4: src/contracts/modules/commons/ModuleCreator.sol (Line 19)

```
18 function createContract(bytes memory _code) public override
↳ payable onlySelf returns (address addr) {
19     assembly { addr := create(callvalue(), add(_code, 32), mload(
↳ _code)) }
20     emit CreatedContract(addr);
21   }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation:

It is recommended to verify if the returned valued is zero address in order to handle possible errors.

Remediation Plan:

SOLVED: the Immutable team solved this issue by checking if the create instruction returns a zero address in the following commit:

- fdd8cd20d93f134f58396812214e02932d006f6f

4.3 (HAL-03) USING KECCAK256 RESULTS DIRECTLY FROM A STRING AS STORAGE KEYS COULD LEAD TO STORAGE COLLISIONS - LOW (3.1)

Description:

In the implementation, every contract that makes use of the EVM storage follows a key-value pattern implemented in the ModuleStorage library. This functionality allows storing a value in a specific slot of the storage defined by a key.

The way these contracts generate different keys for each value to be stored is by using the keccak256 hashing function, specifying a string that references the variable to be stored in the internal storage. However, since the pre-image of every keccak256 hash is known, its values could be overwritten in case a mistake is made. Then, it'd be convenient to use hashes as keys whose pre-images are not known, for instance, by subtracting 1 from the resulting hash since the pre-image for keccak256 (STRING) - 1 is not known.

Code Location:

Listing 5: src/contracts/modules/commons/ModuleIgnoreNonceCalls.sol

```
16 //                      NONCE_KEY = keccak256("org.arcadeum.
↳ module.calls.nonce");
17   bytes32 private constant NONCE_KEY = bytes32(0
↳ x8d0bf1fd623d628c741362c1289948e57b3e2905218c676d3e69abee36d6ae2e)
↳ ;
```

Listing 6: src/contracts/modules/commons/ModuleIgnoreAuthUpgradable.sol

```
16 //                          IMAGE_HASH_KEY = keccak256("org.arcadeum.
↳ module.auth.upgradable.image.hash");
17   bytes32 private constant IMAGE_HASH_KEY = bytes32(0
↳ xea7157fa25e3aa17d0ae2d5280fa4e24d421c61842aa85e45194e1145aa72bf8)
↳ ;
```

Listing 7: src/contracts/modules/commons/ModuleHooks.sol

```
16 //                          HOOKS_KEY = keccak256("org.arcadeum.
↳ module.hooks.hooks");
17   bytes32 private constant HOOKS_KEY = bytes32(0
↳ xbe27a319efc8734e89e26ba4bc95f5c788584163b959f03fa04e2d7ab4b9a120)
↳ ;
```

Listing 8: src/contracts/modules/commons/ModuleCalls.sol

```
13 //                          NONCE_KEY = keccak256("org.arcadeum.
↳ module.calls.nonce");
14   bytes32 private constant NONCE_KEY = bytes32(0
↳ x8d0bf1fd623d628c741362c1289948e57b3e2905218c676d3e69abee36d6ae2e)
↳ ;
```

Listing 9: src/contracts/modules/commons/ModuleAuthUpgradable.sol

```
12 //                          IMAGE_HASH_KEY = keccak256("org.arcadeum.
↳ module.auth.upgradable.image.hash");
13   bytes32 internal constant IMAGE_HASH_KEY = bytes32(0
↳ xea7157fa25e3aa17d0ae2d5280fa4e24d421c61842aa85e45194e1145aa72bf8)
↳ ;
14
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation:

As it was described, it is recommended to use hashes as keys in storage that do not have known pre-images.

Remediation Plan:

SOLVED: the Immutable team solved this issues by replacing the aforementioned hashes by randomized ones in the following commit:

- 99819fa06d8677c6ce6a5319b93fe9e4a72403a6

4.4 (HAL-04) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL (0.0)

Description:

Failed operations in this contract are reverted with an accompanying message selected from a set of hard-coded strings.

In EVM, emitting a hard-coded string in an error message costs ~50 more gas than emitting a custom error. Additionally, hard-coded strings increase the gas required to deploy the contract.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors. Usage of custom errors should look like this:

Listing 10

```
error CustomError();

// ...

if (condition)
    revert CustomError();
```

Remediation Plan:

ACKNOWLEDGED: The Immutable team acknowledged this finding.

4.5 (HAL-05) UNNECESSARY BOOLEAN EXPRESSION - INFORMATIONAL (0.0)

Description:

It has been identified expressions that always result in true value inside require statements. The evaluation of this kind of expression costs extra gas, bearing in mind that they are completely irrelevant in the code.

Code Location:

Listing 11: src/contracts/modules/commons/ModuleIgnoreNonceCalls.sol (Line 145)

```
144 require(
145       (providedNonce == currentNonce) || true,
146       "MainModule#_auth: INVALID_NONCE"
147     );
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Since the aforementioned expression will always result in a true value, it is recommended to remove that require.

Remediation Plan:

ACKNOWLEDGED: The Immutable team acknowledged this finding.

4.6 (HAL-06) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

The Wallet and Factory smart contracts use the floating pragma ^0.8. 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 new which has not been extensively tested.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

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: The Immutable team solved this issues by setting a fixed pragma version in the following commit:

- b82bf46b8e93d7c1ace6d668b1442d1458cdb3bb

UPGRADEABILITY

5.1 Description

Following one of the ‘Immutable team’s major concerns, this project should be upgradeable, thus checking whether a possible upgrade could incur issues is a priority in this case.

Since the logic of these contracts is executed through a proxy, there could exist risks of storage collisions. However, these contracts make use of a library to store bytes32 values in a specific slot in the storage following a key-value pattern. Therefore, as long as a key is not used more than once for several values, there should not exist any risk of storage collision between upgrades.

Below it’s described the different in-use slots in the storage in each module to be considered in order to avoid storage collisions.

5.2 Storage layouts

MainModuleDynamicAuth:

Slot	Variable
0x8d0bf1fd623d628c741362c1289948e5 7b3e2905218c676d3e69abee36d6ae2e	NONCE_KEY
0xea7157fa25e3aa17d0ae2d5280fa4e24 d421c61842aa85e45194e1145aa72bf8	IMAGE_HASH_KEY
address(this) - 0x0000000000000000 00000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	IMPLEMENTATION

MainModule:

Slot	Variable
0x8d0bf1fd623d628c741362c1289948e5 7b3e2905218c676d3e69abee36d6ae2e	NONCE_KEY
0xbe27a319efc8734e89e26ba4bc95f5c7 88584163b959f03fa04e2d7ab4b9a120	HOOKS_KEY
address(this) - 0x0000000000000000 00000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	IMPLEMENTATION

GuestModule:

Slot	Variable
0x8d0bf1fd623d628c741362c1289948e5 7b3e2905218c676d3e69abee36d6ae2e	NONCE_KEY
address(this) - 0x0000000000000000 00000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	IMPLEMENTATION

MainModuleGasEstimation:

Slot	Variable
0x8d0bf1fd623d628c741362c1289948e5 7b3e2905218c676d3e69abee36d6ae2e	NONCE_KEY
0xbe27a319efc8734e89e26ba4bc95f5c7 88584163b959f03fa04e2d7ab4b9a120	HOOKS_KEY
0xea7157fa25e3aa17d0ae2d5280fa4e24 d421c61842aa85e45194e1145aa72bf8	IMAGE_HASH_KEY
address(this) - 0x0000000000000000 00000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	IMPLEMENTATION

MainModuleUpgradable:

Slot	Variable
0x8d0bf1fd623d628c741362c1289948e5 7b3e2905218c676d3e69abee36d6ae2e	NONCE_KEY
0xbe27a319efc8734e89e26ba4bc95f5c7 88584163b959f03fa04e2d7ab4b9a120	HOOKS_KEY
0xea7157fa25e3aa17d0ae2d5280fa4e24 d421c61842aa85e45194e1145aa72bf8	IMAGE_HASH_KEY
address(this) - 0x0000000000000000 00000000XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	IMPLEMENTATION

5.3 Recommendation

As it was aforementioned, it is recommended to not use the same storage key to store more than one value to avoid possible storage collision issues.

In general, using a library in order to manage the storage following a key-value pattern is a robust practice. However, it could be even safer, as HAL-03 describes.

PROXY OPTIMIZATION

6.1 Description

One of the concerns in this project was optimizing the most the WalletProxy contract, since it will be deployed in many chains and will be executed in every transaction involving the wallet.

Initially, this contract was developed in YUL which is a intermediate level language for EVM supported natively by solc. Its code produces a minimalist bytecode thanks to the optimizer implemented in solc. However, it was suggested to try Huff which is a low-level assembly language capable of generating even more minimalist bytecode compared to the YUL one, that would be translated into less gas during deployment and execution time.

The proof of concept developed in Huff to be compared with YUL's implementation can be found in the Appendix of this document, and the difference in terms of gas usage is shown below.

6.2 Results

Gas usage with YUL contract:

src/contracts/Factory.sol:Factory contract					
Deployment Cost	Deployment Size				
625868	3342				
Function Name	min	avg	median	max	# calls
deploy	69687	69687	69687	69687	12
getAddress	1487	1487	1487	1487	19
grantRole	27213	27213	27213	27213	12

src/contracts/MultiCallDeploy.sol:MultiCallDeploy contract					
Deployment Cost	Deployment Size				
909558	4763				
Function Name	min	avg	median	max	# calls
deployAndExecute	6120	117584	152906	223078	22
grantExecutorRole	25039	25039	25039	25039	12

Gas usage with Huff contract:

src/contracts/Factory.sol:Factory contract						
Deployment Cost	Deployment Size					
625468	3340					
Function Name	min	avg	median	max	# calls	
deploy	69487	69487	69487	69487	12	
getAddress	1487	1487	1487	1487	19	
grantRole	27213	27213	27213	27213	12	

src/contracts/MultiCallDeploy.sol:MultiCallDeploy contract						
Deployment Cost	Deployment Size					
909558	4763					
Function Name	min	avg	median	max	# calls	
deployAndExecute	6122	117469	152708	222880	22	
grantExecutorRole	25039	25039	25039	25039	12	

6.3 Conclusion

Described Huff contract consumes **~200** gas less in deployment time and **~100-200** gas less in execution time (deployAndExecute function) compared to YUL's contract, which it's not a major difference. Moreover, Huff manages to generate a bytecode with a size 2 bytes smaller than YUL's, which is not a huge difference either.

MANUAL TESTING

7.1 SCENARIOS TESTED

In the manual testing phase, the following scenarios were simulated. It must be taken into account that it's been reviewed every test found in the repository, which almost covers each component of the project. The scenarios listed below were selected based on the features of the smart contracts and possible issues related to the scoped smart contracts:

- Test 1: Deployment without transactions.
- Test 2: Deployment with transactions.
- Test 3: Transaction execution after deployment.
- Test 4: Transaction execution with replayed signatures.
- Test 5: Transaction execution in not owned wallet.
- Test 6: Wallet can handle reverts.
- Test 7: Wallet can execute a batch of transactions without reverting if any of them fails.
- Test 8: Wallet works as intended after updating its implementation (revert flag).
- Test 9: Allowed `delegatecall` can modify the wallet's implementation.
- Test 10: Wallet can receive and send ether.
- Test 11: Wallet can execute transactions sent to itself.
- Test 12: Proxy can retrieve its implementation.

The code used for testing these aforementioned scenarios can be found in the Appendix of this document.

RESULTS:

```
[PASS] testDelegateCallManipulatesImpl() (gas: 1941750)
[PASS] testDeployExecuteTransactionAfter() (gas: 298444)
[PASS] testDeployExecuteWithRevert() (gas: 261749)
[PASS] testDeployKeepExecutingAfterRevert() (gas: 274233)
[PASS] testDeployUpdateImplementation() (gas: 1938218)
[PASS] testDeployWithTransaction() (gas: 250499)
[PASS] testDeployWithoutTransactions() (gas: 187516)
[PASS] testGetImplementation() (gas: 190475)
[PASS] testRevertWithNotOwnedWallet() (gas: 289626)
[PASS] testRevertWithReplayedSignature() (gas: 273099)
[PASS] testSelfExecute() (gas: 303814)
[PASS] testWalletReceiveAndSendEther() (gas: 283549)
```

AUTOMATED TESTING

8.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. 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:

```
INFO:Detectors:
GuestModule._executeGuest(bytes32,IModuleCalls.Transaction[]) (contracts/modules/GuestModule.sol#66-93) sends eth to arbitrary user
        Dangerous calls:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/GuestModule.sol#81-84)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/GuestModule.sol#81-84)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) sends eth to arbitrary user
        Dangerous calls:
        - (results[i].succeeded,results[i].result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#58-61)
        - (results[i].succeeded,results[i].result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#58-61)
ModuleCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleCalls.sol#96-126) sends eth to arbitrary user
        Dangerous calls:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#114-117)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#114-117)
ModuleIgnoreNonceCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#99-129) sends eth to arbitrary user
        Dangerous calls:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#117-120)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#117-120)
MultiCallUtils.multiCall(IModuleCalls.Transaction[]) (contracts/modules/utils/MultiCallUtils.sol#8-32) sends eth to arbitrary user
        Dangerous calls:
        - (_successes[i],_results[i]) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/utils/MultiCallUtils.sol#24-27)
        - (_successes[i],_results[i]) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/utils/MultiCallUtils.sol#24-27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
ModuleHooks.fallback() (contracts/modules/commons/ModuleHooks.sol#107-121) uses delegatecall to a input-controlled function id
        - (success,result) = target.delegatecall(msg.data) (contracts/modules/commons/ModuleHooks.sol#111)
ModuleCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleCalls.sol#96-126) uses delegatecall to a input-controlled function id
        - (success,result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#110-112)
        - (success,result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#110-112)
ModuleCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleCalls.sol#96-126) uses delegatecall to a input-controlled function id
        - (success,result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#110-112)
ModuleIgnoreNonceCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#99-129) uses delegatecall to a input-controlled function id
        - (success,result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#113-115)
ModuleIgnoreNonceCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#99-129) uses delegatecall to a input-controlled function id
        - (success,result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#113-115)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) uses delegatecall to a input-controlled function id
        - (results[i].succeeded,results[i].result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#50-52)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) uses delegatecall to a input-controlled function id
        - (results[i].succeeded,results[i].result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#50-52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
ModuleIgnoreNonceCalls._validateNonce(uint256) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#137-153) uses a Boolean constant improperly:
        -require(bool,string)((providedNonce == currentNonce) || true,MainModule#_auth: INVALID_NONCE) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#144-147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-boolean-constant
INFO:Detectors:
Contract locking ether found:
        Contract Factory (contracts/Factory.sol#11-59) has payable functions:
         - Factory.deploy(address,bytes32) (contracts/Factory.sol#49-58)
        But does not have a function to withdraw the ether
Contract locking ether found:
        Contract AlwaysRevertMock (contracts/mocks/AlwaysRevertMock.sol#5-9) has payable functions:
         - AlwaysRevertMock.fallback() (contracts/mocks/AlwaysRevertMock.sol#6-8)
        But does not have a function to withdraw the ether
Contract locking ether found:
        Contract CallReceiverMock (contracts/mocks/CallReceiverMock.sol#5-23) has payable functions:
         - CallReceiverMock.constructor() (contracts/mocks/CallReceiverMock.sol#11)
         - CallReceiverMock.testCall(uint256,bytes) (contracts/mocks/CallReceiverMock.sol#17-22)
```

```
        - CallReceiverMock.testCall(uint256,bytes) (contracts/mocks/CallReceiverMock.sol#17-22)
      But does not have a function to withdraw the ether
Contract locking ether found:
      Contract StartupWalletImpl (contracts/startup/StartupWalletImpl.sol#17-46) has payable functions:
       - StartupWalletImpl.fallback() (contracts/startup/StartupWalletImpl.sol#25-45)
      But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
ModuleAuth._signatureValidationWithUpdateCheck(bytes32,bytes).addr (contracts/modules/commons/ModuleAuth.sol#93) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
LibBytesImpl.readFirstUint16(bytes) (contracts/mocks/LibBytesImpl.sol#10-12) ignores return value by _data.readFirstUint16() (contracts/mocks/LibBytesImpl.sol#11)
LibBytesImpl.readUint8Uint8(bytes,uint256) (contracts/mocks/LibBytesImpl.sol#14-16) ignores return value by _data.readUint8Uint8(_index) (contracts/mocks/LibBytesImpl.sol#15)
LibBytesImpl.readAddress(bytes,uint256) (contracts/mocks/LibBytesImpl.sol#18-20) ignores return value by _data.readAddress(_index) (contracts/mocks/LibBytesImpl.sol#19)
LibBytesImpl.readBytes66(bytes,uint256) (contracts/mocks/LibBytesImpl.sol#22-24) ignores return value by _data.readBytes66(_index) (contracts/mocks/LibBytesImpl.sol#23)
LibBytesImpl.readUint16(bytes,uint256) (contracts/mocks/LibBytesImpl.sol#30-32) ignores return value by _data.readUint16(_index) (contracts/mocks/LibBytesImpl.sol#31)
LibBytesImpl.readBytes(bytes,uint256,uint256) (contracts/mocks/LibBytesImpl.sol#34-36) ignores return value by _data.readBytes(_index,_size) (contracts/mocks/LibBytesImpl.sol#35)
ModuleAuth._signatureValidationWithUpdateCheck(bytes32,bytes) (contracts/modules/commons/ModuleAuth.sol#73-132) ignores return value by (flag,addrWeight,rindex) = _signature.readUint8Uint8(rindex) (contracts/modules/commons/ModuleAuth.sol#94)
RequireUtils.publishInitialSigners(address,bytes32,uint256,bytes,bool) (contracts/modules/utils/RequireUtils.sol#122-228) ignores return value by (flag,addrWeight,rindex) = _signature.readUint8Uint8(rindex) (contracts/modules/utils/RequireUtils.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
GasEstimator.estimate(address,bytes)._to (contracts/modules/utils/GasEstimator.sol#7) lacks a zero-check on :
                - (success,result) = _to.call(_data) (contracts/modules/utils/GasEstimator.sol#12)
RequireUtils.publishConfig(address,uint256,RequireUtils.Member[],bool)._wallet (contracts/modules/utils/RequireUtils.sol#61) lacks a zero-check on :
                - (succeed,data) = _wallet.call(abi.encodePacked(IModuleAuthUpgradable(_wallet).imageHash.selector)) (contracts/modules/utils/RequireUtils.sol#73)
ImmutableSigner.constructor(address,address,address)._signer (contracts/signer/ImmutableSigner.sol#37) lacks a zero-check on :
                - primarySigner = _signer (contracts/signer/ImmutableSigner.sol#41)
ImmutableSigner.updateSigner(address)._newSigner (contracts/signer/ImmutableSigner.sol#62) lacks a zero-check on :
                - primarySigner = _newSigner (contracts/signer/ImmutableSigner.sol#64)
ImmutableSigner.updateSignerWithRolloverPeriod(address,uint256)._newSigner (contracts/signer/ImmutableSigner.sol#80) lacks a zero-check on :
                - primarySigner = _newSigner (contracts/signer/ImmutableSigner.sol#84)
                - primarySigner = _newSigner (contracts/signer/ImmutableSigner.sol#84)
LatestWalletImplLocator.changeWalletImplementation(address)._newImpl (contracts/startup/LatestWalletImplLocator.sol#34) lacks a zero-check on :
                - latestWalletImplementation = _newImpl (contracts/startup/LatestWalletImplLocator.sol#35)
StartupWalletImpl.constructor(address)._walletImplementationLocator (contracts/startup/StartupWalletImpl.sol#20) lacks a zero-check on :
                - walletImplementationLocator = _walletImplementationLocator (contracts/startup/StartupWalletImpl.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier Migrations.restricted() (contracts/migrations/Migrations.sol#13-16) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) has external calls inside a loop: (results[i].succeeded,results[i].result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#50-52)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) has external calls inside a loop: (results[i].succeeded,results[i].result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#50-52)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) has external calls inside a loop: (results[i].succeeded,results[i].result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#58-61)
MainModuleGasEstimation.simulateExecute(IModuleCalls.Transaction[]) (contracts/modules/MainModuleGasEstimation.sol#36-72) has external calls inside a loop: (results[i].succeeded,results[i].result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/MainModuleGasEstimation.sol#58-61)
SignatureValidator.isValidSignature(bytes32,address,bytes) (contracts/utils/SignatureValidator.sol#112-142) has external calls inside a loop: valid = ERC1271_MAGICVALUE_BYTES32 == IERC1271Wallet(_signer).isValidSignature(_hash,_signature) (contracts/utils/SignatureValidator.sol#131)
MultiCallUtils.multiCall(IModuleCalls.Transaction[]) (contracts/modules/utils/MultiCallUtils.sol#8-32) has external calls inside a loop: (_successes[i],_results[i]) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/utils/MultiCallUtils.sol#24-27)
MultiCallUtils.multiCall(IModuleCalls.Transaction[]) (contracts/modules/utils/MultiCallUtils.sol#8-32) has external calls inside a loop: (_successes[i],_results[i]) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/utils/MultiCallUtils.sol#24-27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in RequireUtils.publishConfig(address,uint256,RequireUtils.Member[],bool) (contracts/modules/utils/RequireUtils.sol#60-107):
      External calls:
      - (succeed,data) = _wallet.call(abi.encodePacked(IModuleAuthUpgradable(_wallet).imageHash.selector)) (contracts/modules/utils/RequireUtils.sol#73)
      State variables written after the call(s):
      - knownImageHashes[_wallet] = imageHash (contracts/modules/utils/RequireUtils.sol#94)
      - lastImageHashUpdate[imageHash] = block.number (contracts/modules/utils/RequireUtils.sol#105)
      - lastWalletUpdate[_wallet] = block.number (contracts/modules/utils/RequireUtils.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ModuleCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleCalls.sol#96-126):
```

```
Reentrancy in ModuleCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleCalls.sol#96-126):
        External calls:
        - (success,result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#110-112)
        - (success,result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/commons/ModuleCalls.sol#110-112
)
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleCalls.sol
#114-117)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/Module
Calls.sol#114-117)
        External calls sending eth:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleCalls.sol
#114-117)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/Module
Calls.sol#114-117)
        Event emitted after the call(s):
        - TxExecuted(_txHash) (contracts/modules/commons/ModuleCalls.sol#121)
        - TxFailed(_txHash,_reason) (contracts/modules/commons/ModuleCalls.sol#165)
                - _revertBytes(transaction,_txHash,result) (contracts/modules/commons/ModuleCalls.sol#123)
Reentrancy in ModuleIgnoreNonceCalls._execute(bytes32,IModuleCalls.Transaction[]) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#99-129):
        External calls:
        - (success,result) = transaction.target.delegatecall{gas: gasleft()()}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#113-1
15)
        - (success,result) = transaction.target.delegatecall{gas: transaction.gasLimit}(transaction.data) (contracts/modules/commons/ModuleIgnoreNonceCalls.
sol#113-115)
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleIgnoreNon
ceCalls.sol#117-120)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/Module
IgnoreNonceCalls.sol#117-120)
        External calls sending eth:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/commons/ModuleIgnoreNon
ceCalls.sol#117-120)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/commons/Module
IgnoreNonceCalls.sol#117-120)
        Event emitted after the call(s):
        - TxExecuted(_txHash) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#124)
        - TxFailed(_txHash,_reason) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#169)
                - _revertBytes(transaction,_txHash,result) (contracts/modules/commons/ModuleIgnoreNonceCalls.sol#126)
Reentrancy in GuestModule._executeGuest(bytes32,IModuleCalls.Transaction[]) (contracts/modules/GuestModule.sol#66-93):
        External calls:
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/GuestModule.sol#81-84)
        - (success,result) = transaction.target.call{gas: gasleft()(),value: transaction.value}(transaction.data) (contracts/modules/GuestModule.sol#81-84)
        - (success,result) = transaction.target.call{gas: transaction.gasLimit,value: transaction.value}(transaction.data) (contracts/modules/GuestModule.so
l#81-84)
        Event emitted after the call(s):
        - TxExecuted(_txHash) (contracts/modules/GuestModule.sol#88)
        - TxFailed(_txHash,_reason) (contracts/modules/commons/ModuleCalls.sol#165)
                - _revertBytes(transaction,_txHash,result) (contracts/modules/GuestModule.sol#90)
Reentrancy in RequireUtils.publishConfig(address,uint256,RequireUtils.Member[],bool) (contracts/modules/utils/RequireUtils.sol#60-107):
        External calls:
        - (succeed,data) = _wallet.call(abi.encodePacked(IModuleAuthUpgradable(_wallet).imageHash.selector)) (contracts/modules/utils/RequireUtils.sol#73)
        Event emitted after the call(s):
        - RequiredConfig(_wallet,imageHash,_threshold,abi.encode(_members)) (contracts/modules/utils/RequireUtils.sol#98)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
RequireUtils.requireNonExpired(uint256) (contracts/modules/utils/RequireUtils.sol#236-238) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= _expiration,RequireUtils#requireNonExpired: EXPIRED) (contracts/modules/utils/RequireUtils.sol#237)
ImmutableSigner.isValidSignature(bytes32,bytes) (contracts/signer/ImmutableSigner.sol#101-113) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= rolloverSigner.validUntil (contracts/signer/ImmutableSigner.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

- Flagged reentrancy issues are false positive and expected behaviors.
- Flagged arbitrary call issues are false positive and expected behaviors.
- Flagged controlled delegatecall issues are false positive, but it is true that they can pose a risk for the wallet if they are used wrongly.
- No major issues were found by Slither.

8.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

- No major issues found by MythX.

APPENDIX

Proof of Concept: WalletProxy.huff:

Listing 12: src/contracts/WalletProxy.huff

```
/// @title Optimized WalletProxy contract
/// @notice SPDX-License-Identifier: UNLICENSED
/// @author Alejandro Taibo (Halborn)

// Interface
#define function PROXY_getImplementation() view returns (address)

#define macro ADDRESS_SLOT() = takes(0) returns(1) {
    address                     // [address(this)] - returns contract address
}

#define macro CONSTRUCTOR() = takes(0) returns (0) {
    // Copy implementation address into memory
    0x53                        // [size] - byte size to copy
    __codesize(CONSTRUCTOR)     // [offset, size] - offset in the code to copy from
    returndatasize              // [mem, offset, size] - offset in memory to copy to
    codecopy                    // []

    // Copy implementation address from memory into storage
    __codesize(MAIN)            // [mainsize]
    dup1                        // [mainsize, mainsize]
    mload                       // [implementation_addr, mainsize] - load value into stack
    ADDRESS_SLOT() sstore       // [mainsize] - store implementation address into storage

    returndatasize return
}

#define macro MAIN() = takes(0) returns (0) {
    // Load implementation address
    ADDRESS_SLOT() sload        // [implementation_addr]

    // Store zero in stack to avoid using PUSH0 after delegatecall
    returndatasize              // [0, implementation_addr]

    // Load function signature from calldata
    returndatasize              // [0, 0, implementation_addr]
```

```
37      calldataload                // [calldata, 0,
↳ implementation_addr] - load calldata into stack
38      0xE0 shr                    // [signature, 0,
↳ implementation_addr] - shift 224 bits to right
39
40      // Function dispatcher
41      __FUNC_SIG(PROXY_getImplementation) eq getImplementation jumpi
42
43      // Execute delegatecall as a regular proxy
44      calldatasize                // [calldatasize, 0,
↳ implementation_addr]
45      returndatasize dup1         // [0, 0, calldatasize, 0,
↳ implementation_addr]
46      calldatacopy                // [0, implementation_addr] - copy
↳  calldata into memory
47
48      returndatasize dup1         // [0, 0, 0, implementation_addr]
49      calldatasize                // [calldatasize, 0, 0, 0,
↳ implementation_addr]
50      returndatasize              // [0, calldatasize, 0, 0, 0,
↳ implementation_addr]
51      dup6                        // [implementation_addr, 0,
↳ calldatasize, 0, 0, 0, implementation_addr] - load implementation
↳ address into stack
52      gas                         // [gas, implementation_addr, 0,
↳ calldatasize, 0, 0, 0, implementation_addr]
53      delegatecall                // [success, 0,
↳ implementation_addr]
54
55      returndatasize              // [returndatasize, success, 0,
↳ implementation_addr]
56      dup3 dup1                   // [0, 0, returndatasize, success,
↳  0, implementation_addr]
57      returndatacopy              // [success, 0,
↳ implementation_addr] - copy output data from delegatecall into
↳ memory
58
59      iszero error_delegatecall jumpi
60
61      returndatasize dup2 return
62
63      error_delegatecall:
64          returndatasize dup2 revert
65
```

```
66      // Return implementation address
67      getImplementation:
68          mstore                  // [implementation_addr] - copy
↳ implementation_addr into memory
69          0x20 returndatasize     // [0, 32, implementation_addr]
70          return                  // return implementation_addr
71  }
```

Code used in manual testing:

Listing 13: foundry_tests/TestWallet.t.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.17;
3
4  import "forge-std/Test.sol";
5
6  import "./helpers/Deployment.sol";
7  import "./helpers/WalletHelper.sol";
8
9  import { MainModuleDynamicAuth } from "src/contracts/modules/
↳ MainModuleDynamicAuth.sol";
10
11 import { CallReceiverMock } from "src/contracts/mocks/
↳ CallReceiverMock.sol";
12 import { DelegateCallStorageMock } from "src/contracts/mocks/
↳ DelegateCallStorageMock.sol";
13
14 import { IWalletProxy } from "src/contracts/IWalletProxy.sol";
15
16 contract TestWallet is Deployment, WalletHelper {
17
18     MainModuleDynamicAuth public mainmoduledynamicauth;
19
20     CallReceiverMock receiverMock;
21     DelegateCallStorageMock delegateMock;
22
23     function setUp() public {
24         // 5. Deploy & setup implementation in wallet
25         mainmoduledynamicauth = new MainModuleDynamicAuth(address(
↳ factory), address(startupwalletimpl));
26
```

```
27          vm.prank(IMPLCHANGER);
28          latestwalletimpllocator.changeWalletImplementation(address
↳ (mainmoduledynamicauth));
29
30          // 6. Deploy required mocks
31          receiverMock = new CallReceiverMock();
32          delegateMock = new DelegateCallStorageMock();
33      }
34
35      function testDeployWithoutTransactions() public {
36          IModuleCalls.Transaction[] memory txs = new IModuleCalls.
↳ Transaction[](0);
37          address wallet = _deployExecuteWallet("ALICE", txs);
38
39          assertTrue(wallet != address(0));
40      }
41
42      function testDeployWithTransaction() public {
43          IModuleCalls.Transaction[] memory txs = new IModuleCalls.
↳ Transaction[](1);
44          txs[0] = IModuleCalls.Transaction(
45              false,
46              true,
47              100_000,
48              address(receiverMock),
49              0,
50              abi.encodeWithSignature("testCall(uint256,bytes)",
↳ uint256(1337), bytes("ABCD"))
51          );
52
53          address wallet = _deployExecuteWallet("ALICE", txs);
54
55          assertTrue(wallet != address(0));
56          assertTrue(receiverMock.lastValA() == 1337);
57          assertEq(receiverMock.lastValB(), bytes("ABCD"));
58      }
59
60      function testDeployExecuteTransactionAfter() public {
61          IModuleCalls.Transaction[] memory txs = new IModuleCalls.
↳ Transaction[](1);
62          txs[0] = IModuleCalls.Transaction(
63              false,
64              true,
65              100_000,
```

```
66              address(receiverMock),
67              0,
68              abi.encodeWithSignature("testCall(uint256,bytes)",
↳ uint256(1337), bytes("ABCD"))
69          );
70
71          address wallet = _deployExecuteWallet("ALICE", txs);
72
73          assertTrue(wallet != address(0));
74          assertTrue(receiverMock.lastValA() == 1337);
75          assertEq(receiverMock.lastValB(), bytes("ABCD"));
76
77          txs[0] = IModuleCalls.Transaction(
78              false,
79              true,
80              100_000,
81              address(receiverMock),
82              0,
83              abi.encodeWithSignature("testCall(uint256,bytes)",
↳ uint256(4299), bytes("WXYZ"))
84          );
85
86          _deployExecuteWallet("ALICE", txs);
87
88          assertTrue(receiverMock.lastValA() == 4299);
89          assertEq(receiverMock.lastValB(), bytes("WXYZ"));
90      }
91
92      function testRevertWithReplayedSignature() public {
93          IModuleCalls.Transaction[] memory txs = new IModuleCalls.
↳ Transaction[](1);
94          txs[0] = IModuleCalls.Transaction(
95              false,
96              true,
97              100_000,
98              address(receiverMock),
99              0,
100             abi.encodeWithSignature("testCall(uint256,bytes)",
↳ uint256(1337), bytes("ABCD"))
101         );
102
103         address wallet = _deployExecuteWallet("ALICE", txs);
104
105         assertTrue(wallet != address(0));
```

```
106         assertTrue(receiverMock.lastValA() == 1337);
107         assertEq(receiverMock.lastValB(), bytes("ABCD"));
108 
109         (bool success, bytes memory data) = wallet.staticcall(
110             abi.encodeWithSignature("nonce()")
111         );
112 
113         if (!success)
114             revert("#testSimpleWithRepeatedNonce: error retrieving
 ↳  nonce");
115 
116         uint256 prevNonce = uint256(bytes32(data)) - 1;
117         address account = makeAddr("ALICE");
118         bytes32 imageHash = makeWalletImageHash(account, address(
 ↳ immutablesigner));
119 
120         (bytes memory signature, ) = makeWalletSignatureNonce(
121             wallet,
122             address(immutablesigner),
123             "ALICE",
124             "SIGNER",
125             txs,
126             prevNonce
127         );
128 
129         vm.startPrank(EXECUTOR);
130         {
131             vm.expectRevert("MainModule#_auth: INVALID_NONCE");
132             multicalldeploy.deployAndExecute(
133                 wallet,
134                 address(startupwalletimpl),
135                 imageHash,
136                 address(factory),
137                 txs,
138                 prevNonce,
139                 signature
140             );
141         }
142         vm.stopPrank();
143     }
144 
145     function testRevertWithNotOwnedWallet() public {
146         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
```

```
147         txs[0] = IModuleCalls.Transaction(
148             false,
149             true,
150             100_000,
151             address(receiverMock),
152             0,
153             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(1337), bytes("ABCD"))
154         );
155 
156         address wallet = _deployExecuteWallet("ALICE", txs);
157 
158         assertTrue(wallet != address(0));
159         assertTrue(receiverMock.lastValA() == 1337);
160         assertEq(receiverMock.lastValB(), bytes("ABCD"));
161 
162         address account = makeAddr("BOB");
163         bytes32 imageHash = makeWalletImageHash(account, address(
 ↳ immutablesigner));
164 
165         (bytes memory signature, uint256 nonce) =
 ↳ makeWalletSignature(
166             wallet,
167             address(immutablesigner),
168             "BOB",
169             "SIGNER",
170             txs
171         );
172 
173         vm.startPrank(EXECUTOR);
174         {
175             vm.expectRevert("ModuleCalls#execute:
 ↳ INVALID_SIGNATURE");
176             multicalldeploy.deployAndExecute(
177                 wallet,
178                 address(0),
179                 "",
180                 address(0),
181                 txs,
182                 nonce,
183                 signature
184             );
185         }
186         vm.stopPrank();
```

```
187     }
188
189     function testDeployExecuteWithRevert() public {
190         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
191         txs[0] = IModuleCalls.Transaction(
192             false,
193             true,
194             100_000,
195             address(receiverMock),
196             0,
197             abi.encodeWithSignature("setRevertFlag(bool)", bool(
 ↳ true))
198         );
199
200         address wallet = _deployExecuteWallet("ALICE", txs);
201
202         assertTrue(wallet != address(0));
203
204         txs[0] = IModuleCalls.Transaction(
205             false,
206             true,
207             100_000,
208             address(receiverMock),
209             0,
210             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(4299), bytes("WXYZ"))
211         );
212
213         address account = makeAddr("ALICE");
214         bytes32 imageHash = makeWalletImageHash(account, address(
 ↳ immutablesigner));
215
216         (bytes memory signature, uint256 nonce) =
 ↳ makeWalletSignature(
217             wallet,
218             address(immutablesigner),
219             "ALICE",
220             "SIGNER",
221             txs
222         );
223
224         vm.startPrank(EXECUTOR);
225         {
```

```
226             vm.expectRevert("CallReceiverMock#testCall:
 ↳ REVERT_FLAG");
227             multicalldeploy.deployAndExecute(
228                 wallet,
229                 address(0),
230                 "",
231                 address(0),
232                 txs,
233                 nonce,
234                 signature
235             );
236         }
237         vm.stopPrank();
238     }
239
240     function testDeployKeepExecutingAfterRevert() public {
241         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](4);
242         txs[0] = IModuleCalls.Transaction(
243             false,
244             true,
245             100_000,
246             address(receiverMock),
247             0,
248             abi.encodeWithSignature("setRevertFlag(bool)", bool(
 ↳ true))
249         );
250
251         txs[1] = IModuleCalls.Transaction(
252             false,
253             false,
254             100_000,
255             address(receiverMock),
256             0,
257             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(7331), bytes("DCBA"))
258         );
259
260         txs[2] = IModuleCalls.Transaction(
261             false,
262             true,
263             100_000,
264             address(receiverMock),
265             0,
```

```
266             abi.encodeWithSignature("setRevertFlag(bool)", bool(
 ↳ false))
267         );
268
269         txs[3] = IModuleCalls.Transaction(
270             false,
271             true,
272             100_000,
273             address(receiverMock),
274             0,
275             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(1337), bytes("ABCD"))
276         );
277
278         address wallet = _deployExecuteWallet("ALICE", txs);
279
280         assertTrue(wallet != address(0));
281         assertTrue(receiverMock.lastValA() == 1337);
282         assertEq(receiverMock.lastValB(), bytes("ABCD"));
283     }
284
285     function testDeployUpdateImplementation() public {
286         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
287         txs[0] = IModuleCalls.Transaction(
288             false,
289             true,
290             100_000,
291             address(receiverMock),
292             0,
293             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(1337), bytes("ABCD"))
294         );
295
296         address wallet = _deployExecuteWallet("ALICE", txs);
297         address prevImpl = IWalletProxy(wallet).
 ↳ PROXY_getImplementation();
298
299         assertTrue(wallet != address(0));
300         assertTrue(receiverMock.lastValA() == 1337);
301         assertEq(receiverMock.lastValB(), bytes("ABCD"));
302
303         address newMainModule = address(new MainModuleDynamicAuth(
 ↳ address(factory), address(startupwalletimpl)));
```

```
304         txs[0] = IModuleCalls.Transaction(
305             false,
306             true,
307             100_000,
308             address(wallet),
309             0,
310             abi.encodeWithSignature("updateImplementation(address)
 ↳ ", address(newMainModule))
311         );
312 
313         _deployExecuteWallet("ALICE", txs);
314         address currImpl = IWalletProxy(wallet).
 ↳ PROXY_getImplementation();
315 
316         assertTrue(currImpl != prevImpl);
317         assertEq(IWalletProxy(wallet).PROXY_getImplementation(),
 ↳ newMainModule);
318 
319         txs[0] = IModuleCalls.Transaction(
320             false,
321             true,
322             100_000,
323             address(receiverMock),
324             0,
325             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(4299), bytes("WXYZ"))
326         );
327 
328         _deployExecuteWallet("ALICE", txs);
329 
330         assertTrue(receiverMock.lastValA() == 4299);
331         assertEq(receiverMock.lastValB(), bytes("WXYZ"));
332     }
333 
334     function testDelegateCallManipulatesImpl() public {
335         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
336         txs[0] = IModuleCalls.Transaction(
337             false,
338             true,
339             100_000,
340             address(receiverMock),
341             0,
342             abi.encodeWithSignature("testCall(uint256,bytes)",
```

```
 ↳ uint256(1337), bytes("ABCD"))
343         );
344
345         address wallet = _deployExecuteWallet("ALICE", txs);
346         address prevImpl = IWalletProxy(wallet).
 ↳ PROXY_getImplementation();
347
348         assertTrue(wallet != address(0));
349         assertTrue(receiverMock.lastValA() == 1337);
350         assertEq(receiverMock.lastValB(), bytes("ABCD"));
351
352         address newMainModule = address(new MainModuleDynamicAuth(
 ↳ address(factory), address(startupwalletimpl)));
353         txs[0] = IModuleCalls.Transaction(
354             true,
355             true,
356             100_000,
357             address(delegateMock),
358             0,
359             abi.encodeWithSignature(
360                 "write(bytes32,bytes32)",
361                 bytes32(uint256(uint160(wallet))),
362                 bytes32(uint256(uint160(newMainModule)))
363             )
364         );
365
366         _deployExecuteWallet("ALICE", txs);
367         address currImpl = IWalletProxy(wallet).
 ↳ PROXY_getImplementation();
368
369         assertTrue(currImpl != prevImpl);
370         assertEq(IWalletProxy(wallet).PROXY_getImplementation(),
 ↳ newMainModule);
371
372         txs[0] = IModuleCalls.Transaction(
373             false,
374             true,
375             100_000,
376             address(receiverMock),
377             0,
378             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(4299), bytes("WXYZ"))
379         );
380
```

```
381         _deployExecuteWallet("ALICE", txs);
382
383         assertTrue(receiverMock.lastValA() == 4299);
384         assertEq(receiverMock.lastValB(), bytes("WXYZ"));
385     }
386
387     function testWalletReceiveAndSendEther() public {
388         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
389
390         address wallet = _deployExecuteWallet("ALICE", txs);
391
392         vm.deal(ALICE, 10 ether);
393         uint256 prevBalance = ALICE.balance;
394
395         vm.prank(ALICE);
396         (bool success, ) = wallet.call{value: prevBalance}("");
397
398         assertTrue(success);
399         assertEq(wallet.balance, prevBalance);
400
401         uint256 prevBalanceBob = BOB.balance;
402         uint256 prevBalanceWallet = wallet.balance;
403         txs[0] = IModuleCalls.Transaction(
404             false,
405             true,
406             100_000,
407             BOB,
408             prevBalanceWallet,
409             ""
410         );
411
412         _deployExecuteWallet("ALICE", txs);
413
414         assertEq(prevBalanceBob + prevBalanceWallet, BOB.balance);
415         assertEq(wallet.balance, 0);
416     }
417
418     function testSelfExecute() public {
419         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](1);
420
421         address wallet = _deployExecuteWallet("ALICE", txs);
422         assertTrue(wallet != address(0));
```

```
423
424         IModuleCalls.Transaction[] memory subTxs = new
 ↳ IModuleCalls.Transaction[](1);
425         subTxs[0] = IModuleCalls.Transaction(
426             false,
427             true,
428             100_000,
429             address(receiverMock),
430             0,
431             abi.encodeWithSignature("testCall(uint256,bytes)",
 ↳ uint256(1337), bytes("ABCD"))
432         );
433
434         txs[0] = IModuleCalls.Transaction(
435             false,
436             true,
437             500_000,
438             address(wallet),
439             0,
440             abi.encodeWithSelector(IModuleCalls.selfExecute.
 ↳ selector, subTxs)
441         );
442
443         _deployExecuteWallet("ALICE", txs);
444
445         assertTrue(receiverMock.lastValA() == 1337);
446         assertEq(receiverMock.lastValB(), bytes("ABCD"));
447     }
448
449     function testGetImplementation() public {
450         IModuleCalls.Transaction[] memory txs = new IModuleCalls.
 ↳ Transaction[](0);
451         address wallet = _deployExecuteWallet("ALICE", txs);
452
453         address implementation = IWalletProxy(wallet).
 ↳ PROXY_getImplementation();
454
455         assertTrue(implementation == address(mainmoduledynamicauth
 ↳ ));
456     }
457
458     function _deployExecuteWallet(
459         string memory _accountName,
460         IModuleCalls.Transaction[] memory _txs
```

```
461     ) internal returns (address) {
462         address account = makeAddr(_accountName);
463         bytes32 imageHash = makeWalletImageHash(account, address(
 ↳ immutablesigner));
464 
465         address expectedWallet = factory.getAddress(
466             address(startupwalletimpl),
467             imageHash
468         );
469 
470         (bytes memory signature, uint256 nonce) =
 ↳ makeWalletSignature(
471             expectedWallet,
472             address(immutablesigner),
473             _accountName,
474             "SIGNER",
475             _txs
476         );
477 
478         vm.prank(EXECUTOR);
479         multicalldeploy.deployAndExecute(
480             expectedWallet,
481             address(startupwalletimpl),
482             imageHash,
483             address(factory),
484             _txs,
485             nonce,
486             signature
487         );
488 
489         return expectedWallet;
490     }
491 
492 }
```

Listing 14: foundry_tests/helpers/Deployment.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.17;
3 
4 import "forge-std/Test.sol";
5 
6 import { Factory } from "src/contracts/Factory.sol";
7 import { MultiCallDeploy } from "src/contracts/MultiCallDeploy.sol
```

```
↳ ";
 8
 9 import { LatestWalletImplLocator } from "src/contracts/startup/
↳ LatestWalletImplLocator.sol";
10 import { StartupWalletImpl } from "src/contracts/startup/
↳ StartupWalletImpl.sol";
11
12 import { ImmutableSigner } from "src/contracts/signer/
↳ ImmutableSigner.sol";
13
14 abstract contract Deployment is Test {
15
16     address public ALICE = makeAddr("ALICE");
17     address public BOB = makeAddr("BOB");
18     address public CHARLIE = makeAddr("CHARLIE");
19
20     address public ADMIN = makeAddr("ADMIN");
21     address public DEPLOYER = makeAddr("DEPLOYER");
22     address public IMPLCHANGER = makeAddr("IMPLCHANGER");
23     address public SIGNER = makeAddr("SIGNER");
24     address public EXECUTOR = makeAddr("EXECUTOR");
25
26     bytes32 public constant DEPLOYER_ROLE = keccak256("
↳ DEPLOYER_ROLE");
27
28     Factory public factory;
29     MultiCallDeploy public multicalldeploy;
30
31     LatestWalletImplLocator public latestwalletimpllocator;
32     StartupWalletImpl public startupwalletimpl;
33
34     ImmutableSigner public immutablesigner;
35
36     constructor() {
37         // 1. The Factory initiates the deployment of the factory
↳ contract
38         factory = new Factory(ADMIN, DEPLOYER);
39         multicalldeploy = new MultiCallDeploy(ADMIN, DEPLOYER);
40
41         // 2. The Admin EOA grants DEPLOYER role to
↳ MultiCallDeploy
42         vm.startPrank(ADMIN);
43         {
44             factory.grantRole(DEPLOYER_ROLE, address(
```

```
↳ multicalldeploy));
45              multicalldeploy.grantExecutorRole(EXECUTOR);
46          }
47          vm.stopPrank();
48
49          // 3. Startup wallet implementation contracts deployment
50          latestwalletimpllocator = new LatestWalletImplLocator(
↳ ADMIN, IMPLCHANGER);
51          startupwalletimpl = new StartupWalletImpl(address(
↳ latestwalletimpllocator));
52
53          // 4. ImmutableSigner 2FA contract deployment
54          immutablesigner = new ImmutableSigner(ADMIN, SIGNER,
↳ SIGNER);
55      }
56
57 }
```

Listing 15: foundry_tests/helpers/WalletHelper.sol

```
 1 // SPDX-License-Identifier: MIT
 2 pragma solidity 0.8.17;
 3
 4 import "forge-std/Test.sol";
 5
 6 import "src/contracts/modules/commons/interfaces/IModuleCalls.sol"
↳ ;
 7
 8 abstract contract WalletHelper is Test {
 9
10     uint256 private constant FLAG_SIGNATURE = 0;
11     uint256 private constant FLAG_ADDRESS = 1;
12     uint256 private constant FLAG_DYNAMIC_SIGNATURE = 2;
13
14     uint256 private constant SIG_TYPE_EIP712 = 1;
15     uint256 private constant SIG_TYPE_ETH_SIGN = 2;
16     uint256 private constant SIG_TYPE_WALLET_BYTES32 = 3;
17
18     function makeWalletImageHash(
19         address _signerA,
20         address _signerB
21     ) internal pure returns (bytes32) {
22         bytes32 imageHash = keccak256(abi.encode(
23             keccak256(abi.encode(
```

```
                bytes32(uint256(2)),
                uint8(1), _signerA)),
            uint8(1), _signerB
        ));

        return imageHash;
    }

    function makeWalletSignature(
        address _wallet,
        address _immutableSigner,
        string memory _accountA,
        string memory _accountB,
        IModuleCalls.Transaction[] memory _txs
    ) internal returns (bytes memory, uint256) {
        uint256 nonce = _getLastNonce(_wallet);
        bytes32 digestData = _subDigest(_wallet, keccak256(abi.
↳ encode(nonce, _txs)));

        (, uint256 privA) = makeAddrAndKey(_accountA);
        (, uint256 privB) = makeAddrAndKey(_accountB);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(privA,
↳ digestData);
        bytes memory signatureA = abi.encodePacked(
            r, s, v, uint8(SIG_TYPE_EIP712)
        );

        (v, r, s) = vm.sign(privB, digestData);
        bytes memory signatureB = abi.encodePacked(
            r, s, v, uint8(SIG_TYPE_EIP712), uint8(
↳ SIG_TYPE_WALLET_BYTES32)
        );

        bytes memory txSignature = abi.encodePacked(
            uint16(2),                          // Threshold
            uint8(FLAG_SIGNATURE),              // Signer type
            uint8(1),                           // Signer weight
            signatureA,                         // Signature
            uint8(FLAG_DYNAMIC_SIGNATURE),      // Signer type
            uint8(1),                           // Signer weight
            _immutableSigner,                   // Contract
↳ address
            uint16(signatureB.length),          // Signature size
```

```
64              signatureB                          // Signature
65          );
66
67          return (txSignature, nonce);
68      }
69
70      function makeWalletSignatureNonce(
71          address _wallet,
72          address _immutableSigner,
73          string memory _accountA,
74          string memory _accountB,
75          IModuleCalls.Transaction[] memory _txs,
76          uint256 _nonce
77      ) internal returns (bytes memory, uint256) {
78          bytes32 digestData = _subDigest(_wallet, keccak256(abi.
↳ encode(_nonce, _txs)));
79
80          (, uint256 privA) = makeAddrAndKey(_accountA);
81          (, uint256 privB) = makeAddrAndKey(_accountB);
82
83          (uint8 v, bytes32 r, bytes32 s) = vm.sign(privA,
↳ digestData);
84          bytes memory signatureA = abi.encodePacked(
85              r, s, v, uint8(SIG_TYPE_EIP712)
86          );
87
88          (v, r, s) = vm.sign(privB, digestData);
89          bytes memory signatureB = abi.encodePacked(
90              r, s, v, uint8(SIG_TYPE_EIP712), uint8(
↳ SIG_TYPE_WALLET_BYTES32)
91          );
92
93          bytes memory txSignature = abi.encodePacked(
94              uint16(2),                          // Threshold
95              uint8(FLAG_SIGNATURE),              // Signer type
96              uint8(1),                           // Signer weight
97              signatureA,                         // Signature
98              uint8(FLAG_DYNAMIC_SIGNATURE),      // Signer type
99              uint8(1),                           // Signer weight
100             _immutableSigner,                   // Contract
↳ address
101             uint16(signatureB.length),          // Signature size
102             signatureB                          // Signature
103         );
```

```
104
105         return (txSignature, _nonce);
106     }
107
108     function _getLastNonce(address _wallet) private view returns (
 ↳ uint256) {
109         (bool success, bytes memory data) = _wallet.staticcall(
110             abi.encodeWithSignature("nonce()")
111         );
112
113         if (!success)
114             return 0;
115
116         return uint256(bytes32(data));
117     }
118
119     function _subDigest(address _wallet, bytes32 _digest) private
 ↳ view returns (bytes32) {
120         uint256 chainId; assembly { chainId := chainid() }
121         return keccak256(
122           abi.encodePacked(
123             "\x19\x01",
124             chainId,
125             _wallet,
126             _digest
127           )
128         );
129       }
130
131 }
```

Listing 16: src/contracts/mocks/DelegateCallStorageMock.sol

```
1 // SPDX-License-Identifier: Apache-2.0
2 pragma solidity 0.8.17;
3
4 import "../modules/commons/ModuleStorage.sol";
5
6 contract DelegateCallStorageMock {
7
8   function write(bytes32 _key, bytes32 _val) external {
9     ModuleStorage.writeBytes32(_key, _val);
10   }
11
```

```
12   function read(bytes32 _key) external view returns (bytes32){
13     return ModuleStorage.readBytes32(_key);
14   }
15
16 }
```

THANK YOU FOR CHOOSING

// HALBORN