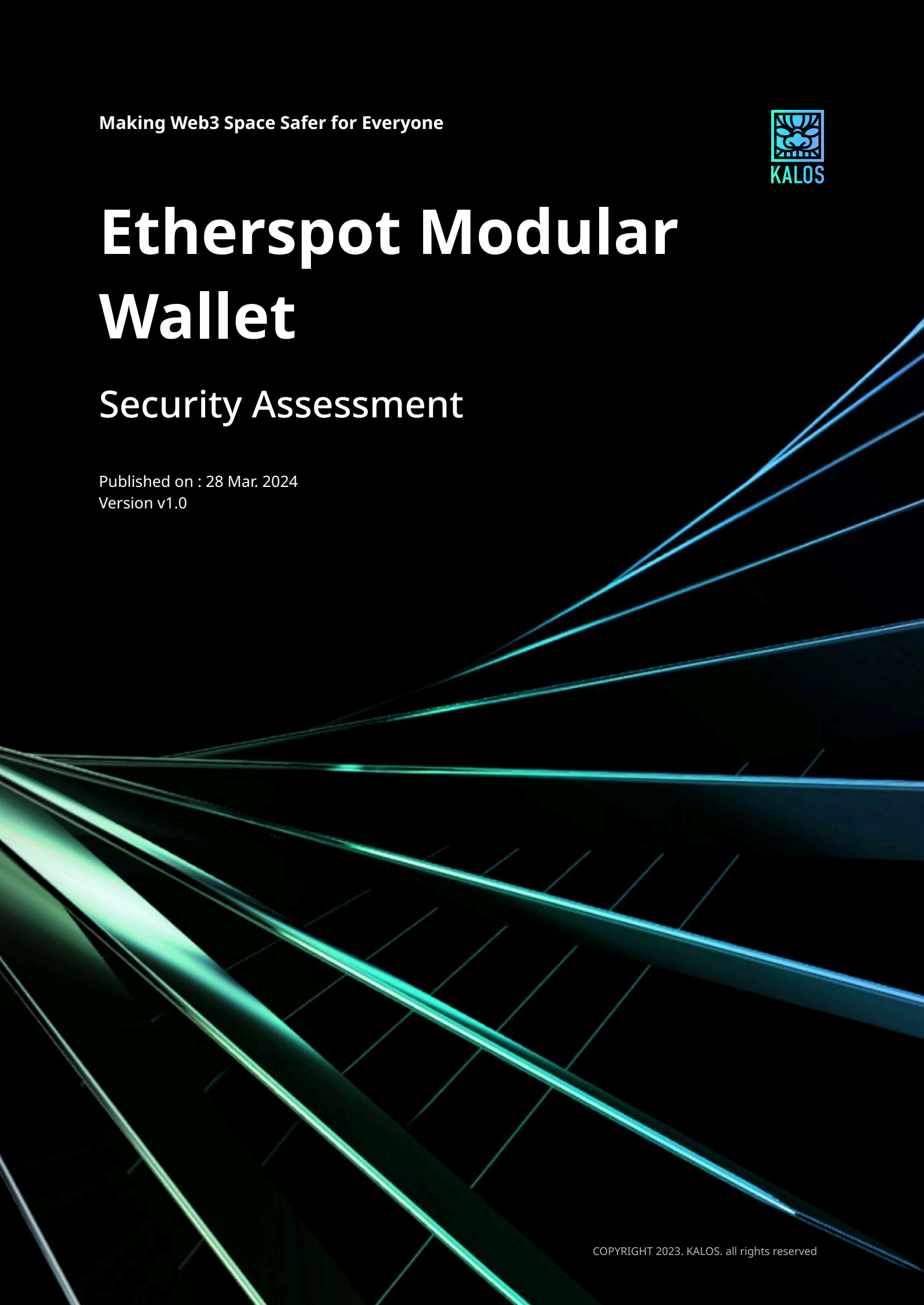
Making Web3 Space Safer for Everyone

KALOS

Etherspot Modular Wallet

Security Assessment

Published on : 28 Mar. 2024
Version v1.0



Security Report Published by KALOS

v1.0 28 Mar. 2024

Auditor : Jade

hojung han

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	2	2	-	-
Medium	-	-	-	-
Low	1	1	-	-
Tips	3	3	-	-

TABLE OF CONTENTS

TABLE OF CONTENTS
ABOUT US
Executive Summary
OVERVIEW
- Protocol overview
- Scope
- Access Controls

FINDINGS
- 1. Potential Security Risk in Implementation Contract Initialization
  - Issue
  - Recommendation
  - Patch Comment
- 2. Non-Compliance and Security Risk of DelegateCall in Fallback Handlers
  - Issue
  - Recommendation
  - Patch Comment
- 3. Logical Flaw in Handling Success Variable in _tryExecute and _tryExecuteDelegatecall
  - Issue
  - Recommendation
  - Patch Comment
- 4. Access Control Omission in Fallback Function Implementation
  - Issue
  - Recommendation
  - Patch Comment
- 5. Front-Running Risk in Proposals During Guardian Replacement
  - Issue
  - Recommendation
  - Patch Comment

6. Suppressed Specific Function Execution Due to receiverFallback Modifier
Issue
Recommendation
Patch Comment
Fix
Fix Comment
DISCLAIMER
Appendix. A
Severity Level
Difficulty Level
Vulnerability Category

ABOUT US

Making Web3 Space Safer for Everyone

Pioneering a safer Web3 space since 2018, KALOS proudly won 2nd place in the Paradigm CTF 2023. As a leader in the global blockchain industry, we unite the finest in Web3 security expertise.

Our team consists of top security researchers with expertise in blockchain/smart contracts and experience in bounty hunting. Specializing in the audit of mainnets, DeFi protocols, bridges, and the zkEVM protocol, KALOS has successfully safeguarded billions in crypto assets.

Supported by grants from the Ethereum Foundation and the Community Fund, we are dedicated to innovating and enhancing Web3 security, ensuring that our clients' digital assets are securely protected in the highly volatile and ever-evolving Web3 landscape.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Etherspot modular wallet contracts developed by the Etherspot team. KALOS conducted the audit focusing on whether the system created by the Etherspot team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the project. In detail, we have focused on the following

- Denial of Service
- Access Control of Various Storage Variables
- Access Control of Important Functions
- Freezing of User Assets
- Theft of User Assets
- Unhandled Exceptions

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/etherspot/etherspot-prime-contracts).

The last commit of the code used for this Audit is “8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4”.

The patch commit of the code is “1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4”.

Audit Timeline

Date	Event
2024/03/14	Audit Initiation (Etherspot Modular Wallet)
2022/03/28	Delivery of v1.0 report.

Findings

KALOS found 2 High and 1 Low severity issues. There are 3 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
Tips	Potential Security Risk in Implementation Contract Initialization	(Patched - v1.0)
High	Non-Compliance and Security Risk of DelegateCall in Fallback Handlers	(Patched - v1.0)
Tips	Logical Flaw in Handling Success Variable in _tryExecute and _tryExecuteDelegatecall	(Patched - v1.0)
Low	Access Control Omission in Fallback Function Implementation	(Patched - v1.0)
High	Front-Running Risk in Proposals During Guardian Replacement	(Patched - v1.0)
Tips	Suppressed Specific Function Execution Due to receiverFallback Modifier	(Patched - v1.0)

Remarks

ERC-7579 Modules created by third parties are not within the audit scope, and risk scenarios that may arise from third-party Modules have not been considered in this report.

OVERVIEW

Protocol overview

• **ModularEtherspotWallet**
This contract implements EIP-7579, designed to provide a framework for modular smart accounts with a focus on interoperability and simplicity. It integrates interfaces for execution, configuration, and fallback operations consistent with the specifications in EIP-7579, including compliance with ERC-165 and ERC-1271 for interaction with modules and signature verification. Features include support for batch and delegate call executions. It extends functionalities from AccessController, ExecutionHelper, ModuleManager, and HookManager to manage modules such as validators and executors effectively.

• **ModularEtherspotWalletFactory**
This contract facilitates the creation of ModularEtherspotWallet instances, aligning with the principles of EIP-7579 to support the efficient deployment of smart accounts within a minimalistic framework.

• **MultiOwnerECDSAValidator**
Aligned with the validator module type from EIP-7579, this contract provides multi-signature verification capabilities within the smart account system, particularly for the ERC-4337 user operation validation flow.

• **AccessController**
It incorporates access control mechanisms for managing permissions within modular smart accounts, which are crucial for their governance and operational integrity.

• **AccountBase**

The AccountBase contract sets a secure foundation for smart accounts by defining the address of an EntryPoint to facilitate wallet interactions and implementing a modifier that verifies whether msg.sender is the EntryPoint or the wallet itself. Additionally, It features the payPrefund modifier, designed to manage transaction fees by allowing the smart account to send necessary funds to the EntryPoint, ensuring smooth execution of transaction.

• **ExecutionHelper**
Offers transaction management utilities within smart accounts, including support for single, batch, and delegate calls, aligning with EIP-7579's execution mode encoding and error handling requirements.

• **HookManager**
Manages hooks that allow for the execution of custom logic before and after account operations, enhancing the account's operational flexibility.

• **ModuleManager**
Central to module management within the EIP-7579 framework, it oversees the integration and functioning of validators, executors, and fallback handlers, facilitating module interactions.

• **Receiver**
Addresses the receipt of ERC721 and ERC1155 tokens by smart accounts, ensuring compatibility and interoperability with these token standards.

Scope

```
├── access
│   └── AccessController.sol
├── erc7579-ref-impl
│   ├── core
│   │   ├── AccountBase.sol
│   │   ├── ExecutionHelper.sol
│   │   ├── HookManager.sol
│   │   ├── ModuleManager.sol
│   │   └── Receiver.sol
│   ├── interfaces
│   │   ├── IERC7579Account.sol
│   │   ├── IERC7579Module.sol
│   │   └── IMSA.sol
│   ├── libs
│   │   ├── ExecutionLib.sol
│   │   ├── ModeLib.sol
│   │   ├── ModuleTypeLib.sol
│   │   └── SentinelList.sol
│   └── utils
│       └── Bootstrap.sol
├── interfaces
│   ├── IAccessController.sol
│   └── IModularEtherspotWallet.sol
├── libraries
│   ├── ArrayLib.sol
│   └── ErrorsLib.sol
├── modules
│   └── MultipleOwnerECDSAValidator.sol
└── wallet
    ├── ModularEtherspotWallet.sol
    └── ModularEtherspotWalletFactory.sol
```

Access Controls

ModularEtherspotWallet contracts have the following access control mechanisms.

- withHook()
- onlyEntryPointOrSelf()
- onlyExecutorModule
- onlyOwnerOrSelf
- onlyGuardian
- onlyOwnerOrGuardianOrSelf

withHook() : The withHook modifier is tasked with the inspection of calldata whenever the execute or executeFromExecutor functions are invoked. This inspection is conducted through a pre-installed hook contract to determine the appropriateness of the calldata. This process ensures that the calldata meets the specified criteria for execution, thereby enhancing the security and integrity of the transaction.

onlyEntryPointOrSelf() : The onlyEntryPointOrSelf modifier serves as a security measure, designed to ensure that the msg.sender is either the wallet itself or the designated entryPoint. This modifier is integrally executed in conjunction with the execute, executeUserOp, installModule, and uninstallModule functions. Implemented within the wallet contract, its primary purpose is to fortify security by restricting access and preventing arbitrary invocation of these critical functions.

onlyExecutorModule() : The onlyExecutorModule modifier is a security mechanism designed to validate that the msg.sender is a pre-installed executor. Its implementation is specifically targeted for functions that must be exclusively called by the installed executor.

onlyOwnerOrSelf() : The onlyOwnerOrSelf modifier is a security feature designed to ascertain whether the msg.sender is the owner or the wallet itself. This modifier is employed alongside the execution of various functions including addOwner, removeOwner, addGuardian, removeGuardian, and changeProposalTimelock. The primary intent behind this implementation is to ensure that critical functions related to ownership and guardianship management are securely controlled. By restricting the execution of these functions to the owner or the wallet, it effectively safeguards the system against unauthorized access and potential security breaches.

onlyGuardian() : The onlyGuardian modifier implements a critical security check by verifying that the msg.sender is indeed a guardian. This verification process is automatically triggered alongside the execution of the guardianPropose and guardianCosign functions. The primary purpose of this modifier is to stringently restrict

access to functions that facilitate the addition of new owners, ensuring that only guardians are granted access rights.

onlyOwnerOrGuardianOrSelf() : The onlyOwnerOrGuardianOrSelf modifier acts as a crucial security check within the system. It is designed to verify that the msg.sender corresponds to one of the following roles: guardian, owner, or the wallet itself.

FINDINGS

1. Potential Security Risk in Implementation Contract Initialization

ID: Etherspot-1-1
Severity: Tips
Type: Logic Error
Difficulty: -
File: wallet/ModularEtherspotWallet.sol

Issue

In the ModularEtherSpotWallet's implementation contract (excluding proxy contracts), it is observed that the initializeAccount function may be exposed to potential security risks associated with the selfdestruct operation. This function is critical for setting up new wallet instances, as it involves initializing module managers and setting owners through delegate calls. Here's the target function:

```
function initializeAccount(bytes calldata data) public payable virtual {
    _initModuleManager();
    (address owner, address bootstrap, bytes memory bootstrapCall) = abi
        .decode(data, (address, address, bytes));
    _addOwner(owner);
    (bool success, ) = bootstrap.delegatecall(bootstrapCall);
    if (!success) revert AccountInitializationFailed();
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol#L322-L329]

Although the Cancun upgrade has been implemented and EIP-6780 activated on several chains like Arbitrum and Optimism, effectively mitigating the threat posed by selfdestruct, this protection is not universally available. Chains that have not activated EIP-6780 remain vulnerable to these risks, which could compromise the wallet.

Recommendation

It is recommended to prevent the call to the initializeAccount function in the implementation contract by referencing the onlyProxy modifier from OpenZeppelin.

Patch Comment

We have confirmed that the code has been modified according to our recommendation. (https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol#L339)

2. Non-Compliance and Security Risk of DelegateCall in Fallback Handlers

ID: Etherspot-1-2 Severity: High
Type: Storage Collision Difficulty: Low
File: erc-7579-ref-impl/core/ModuleManager.sol

Issue

The ERC-7579 specification mandates that fallback handlers must only be executed through a call. Despite this, the ModularEtherspotWallet supports delegatecall, which contravenes the specified standard. This provision for delegatecall inadvertently introduces a risk of contaminating the predefined storage slots in the AccessControl contract, a situation that can occur even without malicious intent from third-party developers. The significance of this issue is highlighted upon reviewing the ModularEtherspotWallet's storage layout, which reveals the critical variables and their allocated storage slots, underscoring the potential for inadvertent contamination by third-party developed fallback handlers:

Name	Type	Slot	Offset	Bytes	Contract
ownerCount	uint256	0	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
guardianCount	uint256	1	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
proposalId	uint256	2	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
proposalTimelock	uint256	3	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
_owners	mapping(address => bool)	4	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
_guardians	mapping(address => bool)	5	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet
_proposals	mapping(uint256 => struct IAccessController.NewOwnerProposal)	6	0	32	src/modular-etherspot-wallet/wallet/ModularEtherspotWallet.sol:ModularEtherspotWallet

[ModularEtherspotWallet Storage Layout]

The above layout distinctly outlines how critical storage variables, such as owner and guardian counts and proposals, are organized within the contract's storage layout. The potential for these structures to be altered or corrupted due to the use of delegatecall in fallback handler execution presents a clear danger to the integrity of the contract's operational logic and security.

The relevant code is as follows:

```
fallback() external payable override(Receiver) receiverFallback {
    ...
    if (calltype == CALLTYPE_DELEGATECALL) {
        assembly {
            calldatacopy(0, 0, calldatasize())
            let result := delegatecall(gas(), handler, 0, calldatasize(), 0, 0)
            returndatacopy(0, 0, returndatasize())
            switch result
```

```
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ModuleManager.sol#L300-L309]

Recommendation

It is recommended to remove the use of delegateCall for executing fallback handlers.

Patch Comment

We have confirmed that the code has been modified according to our recommendation. (https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ModuleManager.sol#L300-L376)

3. Logical Flaw in Handling Success Variable in _tryExecute and _tryExecuteDelegatecall

ID: Etherspot-1-3 Severity: Tips
Type: Logic Error Difficulty: -
File: erc7579-ref-impl/core/ExecutionHelper.sol

Issue

The provided functions _tryExecute and _tryExecuteDelegatecall within the smart contract misinterpret the execution result due to the use of the iszero statement in assembly.

For _tryExecute, the success of the call operation is determined inversely by using iszero on the call result.

Similarly, in _tryExecuteDelegatecall, the delegatecall success outcome is incorrectly assessed using iszero, leading to a reversed interpretation.

This inversion can lead to misleading interpretations of the function executions, where a successful operation is incorrectly marked as failed and vice versa.

```
function _tryExecute(
    address target,
    uint256 value,
    bytes calldata callData
)
    internal
    virtual
    returns (bool success, bytes memory result)
{
    /// @solidity memory-safe-assembly
    assembly {
        result := mload(0x40)
        calldatacopy(result, callData.offset, callData.length)
        success := iszero(call(gas(), target, value, result, callData.length, codesize(), 0x00))
        mstore(result, returndatasize()) // Store the length.
        let o := add(result, 0x20)
        returndatacopy(o, 0x00, returndatasize()) // Copy the returndata.
        mstore(0x40, add(o, returndatasize())) // Allocate the memory.
    }
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ExecutionHelper.sol#L68-L87]

```
function _tryExecuteDelegatecall(
    address delegate,
    bytes calldata callData
)
    internal
    returns (bool success, bytes memory result)
{
    /// @solidity memory-safe-assembly
    assembly {
        result := mload(0x40)
        calldatacopy(result, callData.offset, callData.length)
        // Forwards the `data` to `delegate` via delegatecall.
        success :=
            iszero(delegatecall(gas(), delegate, result, callData.length, codesize(), 0x00))
        mstore(result, returndatasize()) // Store the length.
        let o := add(result, 0x20)
        returndatacopy(o, 0x00, returndatasize()) // Copy the returndata.
        mstore(0x40, add(o, returndatasize())) // Allocate the memory.
    }
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ExecutionHelper.sol#L115-L134]

Recommendation

It is recommended to eliminate the iszero statement to ensure that success is accurately set to true upon the successful execution of call or delegatecall.

Patch Comment

We have confirmed that the code has been modified according to our recommendation. (https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ExecutionHelper.sol#L81,
https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ExecutionHelper.sol#L127)

4. Access Control Omission in Fallback Function Implementation

ID: Etherspot-1-4 Severity: Low
Type: Access Control Omission Difficulty: Low
File: erc7579-ref-impl/core/ModuleManager.sol

Issue

When considering the fallback handlers called through the ModuleManager contract's fallback function, it is not safe to assume that third-party developers creating these handlers will always include the necessary access control code within the fallback handlers themselves. This potential oversight stems from the possibility that developers unfamiliar with the ERC-7579 internal process might omit essential access control-related code.

```
fallback() external payable override(Receiver) receiverFallback {
    FallbackHandler storage $fallbackHandler = $moduleManager().$fallbacks[msg.sig];
    address handler = $fallbackHandler.handler;
    CallType calltype = $fallbackHandler.calltype;
    if (handler == address(0)) revert NoFallbackHandler(msg.sig);

    // Static, Single, and DelegateCall execution logic
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ModuleManager.sol#L244-L248]

For instance, if the fallback handler contains logic for transferring tokens, it can pose a serious security risk. Additionally, there are other scenarios that could potentially lead to security vulnerabilities, hence it is crucial to add appropriate access control-related code.

Recommendation

It is recommended to include strict access control measures in fallback handlers, considering they may contain token transfer logic or other sensitive operations, to avert potential security risks.

Patch Comment

We have confirmed that the code has been modified according to our recommendation. (https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e

825f8f0c9c8d163a4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ModuleManager.sol#L301)

5. Front-Running Risk in Proposals During Guardian Replacement

ID: Etherspot-1-5
Type: Front Running
File: access/AccessController.sol

Severity: High
Difficulty: Low

Issue

The issue at hand pertains to the potential for unintended consequences during changing Guardians within a ModularEtherspotWallet, explicitly leading to the accidental addition of an Owner due to front running. This problem arises from the dynamic nature of the guardianCount variable during guardian replacement events. To illustrate, consider a scenario where there are initially five Guardians. If, for reasons such as private key leakage, the Owner decides to replace all five Guardians after another, a proposal to add a new Owner is in motion. Under normal circumstances, with five active Guardians, achieving a quorum for the proposal necessitates the approval of more than three Guardians. Yet, as each Guardian is individually replaced, thereby decrementing the guardianCount, the threshold to reach a quorum is inadvertently lowered, as demonstrated by the function _checkQuorumReached:

```
function _checkQuorumReached(
    uint256 _proposalId
) internal view returns (bool) {
    return ((proposals[_proposalId].approvalCount * MULTIPLY_FACTOR) /
        guardianCount >=
        SIXTY_PERCENT);
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/access/AccessController.sol#L283-L289]

This mechanism inadvertently simplifies reaching the required quorum for proposals, potentially compromising the intended security and governance protocols of the ModularEtherWallet.

Recommendation

It is recommended to automatically discard the current proposal whenever a Guardian is added or removed, specifically during the execution of the addGuardian or removeGuardian functions.

Patch Comment

We have confirmed that the code has been modified according to our recommendation. (https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/access/AccessController.sol#L269-L272, https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/access/AccessController.sol#L280-L284)

6. Suppressed Specific Function Execution Due to receiverFallback Modifier

ID: Etherspot-1-6
Severity: Tips
Type: Logic Error
Difficulty: -
File: erc7579-ref-impl/core/Receiver.sol

Issue

This issue concerns the receiverFallback modifier's handling of certain function signatures (onERC721Received, onERC1155Received, onERC1155BatchReceived). When msg.sig matches these specific signatures, the modifier does not proceed with executing the logic defined within these functions. Instead, it merely returns the function signature. This behavior significantly restricts users' ability to define and execute additional logic in the mentioned functions, thereby limiting the functionality and flexibility of the contract.

```
modifier receiverFallback() virtual {
    /// @solidity memory-safe-assembly
    assembly {
        let s := shr(224, calldataload(0))
        // Matching signatures for onERC721Received, onERC1155Received, onERC1155BatchReceived.
        if or(eq(s, 0x150b7a02), or(eq(s, 0xf23a6e61), eq(s, 0xbc197c81))) {
            mstore(0x20, s) // Store `msg.sig`.
            return(0x3c, 0x20) // Return `msg.sig`.
        }
    }
    _; // Proceed with function execution
}
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/Receiver.sol#L18-L31]

```
fallback() external payable override(Receiver) receiverFallback {
    FallbackHandler storage $fallbackHandler = $moduleManager().$fallbacks[msg.sig];
    address handler = $fallbackHandler.handler;
    CallType calltype = $fallbackHandler.calltype;
    if (handler == address(0)) revert NoFallbackHandler(msg.sig);
    ...
```

[https://github.com/etherspot/etherspot-prime-contracts/blob/8206155066d9a0dcd96bd5d5c5f5b32a4b5ab3e4/src/modular-etherspot-wallet/erc7579-ref-impl/core/ModuleManager.sol#L244-L248]

Recommendation

It is recommended to remove the code defined within the receiverFallback modifier.

Patch Comment

We have confirmed that the receiverFallback modifier code has been removed as per our recommendation.
(https://github.com/etherspot/etherspot-prime-contracts/blob/1e88fd1e3f6b189e77b521e825f8f0c9c8d163a4/src/modular-etherspot-wallet/erc7579-ref-impl/core/Receiver.sol)

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document