certora

Security Assessment & Formal Verification Report Safe v1.5.0

Safe

Prepared for

Safe Ecosystem Foundation

Table of content

Project Summary **4**

Project Scope 4

Project Overview 4

Findings Summary 6

Severity Matrix 6

Detailed Findings **7**

Medium Severity Issues 9

M-01 ERC-777 compatibility isn't implemented correctly 9

Low Severity Issues 11

L-01 SafeProxy.fallback(): The dirty bits aren't correctly cleared 11

L-02 Wrong signature description 12

L-03 Possible mismatch between safeMethods and safeInterfaces 13

L-04 Possible dirty bits in getTransactionHash() 14

L-05 FallbackManager should staticcall instead of call the Fallback Handler 15

Informational Severity Issues **16**

I-01. Events lacking indexed fields 16

I-02. Some comments say keccak instead of keccak256 17

I-03. Inconsistency in formula for performCreate and performCreate2 18

Gas Optimization **19**

G-01. OwnerManager.removeOwner(): 1 SLOAD can be saved in the normal path 19

G-02. OwnerManager.changeThreshold(): 1 SLOAD can be saved by emitting an existing memory variable instead of reading from storage 20

G-03. ERC165Handler.setSupportedInterface(): Logic and storage access optimization 21

G-04. ExtensibleBase._setSafeMethod(): storage access optimization 22

G-05. Use iszero instead of eq(*, 0) 23

G-06. ExtensibleFallbackHandler._supportsInterface(): save gas via short-circuit evaluation 24

G-07. Use a mask instead of shifting left and right 25

G-08. Use shift right/left instead of division/multiplication if possible 26

G-09. Cache array length outside of loop 28

G-10. ++i costs less gas compared to i++ or i += 1 (same for --i vs i-- or i -= 1) 29

Formal Verification **32**

Verification Notations 32

Formal Verification Properties 33

Safe.sol **33**

P-01. Integrity of the Transaction Guard methods 33

P-02. Integrity of the Module Guard methods 34

P-03. Integrity of Execute Transaction and Execute Transaction from Module 35

P-04. Integrity of approveHash and approvedHashVal 36

P-05. Integrity of Setup 37

ExtensibleFallbackHandler.sol......38
P-01. Integrity of the Extensible Fallback Handler......38
Disclaimer......39
About Certora......39

Project Summary

Project Scope

Project Name	Repository (link)	Audited Commits	Platform
Safe Smart Account v1.5.0	https://github.com/safe-global/safe-smart-account	834e798 - initial 1c8b24a - latest including fixes	EVM/Solidity

Project Overview

This document describes the specification and verification of **Safe's Smart Account v1.5.0 Contracts** using the Certora Prover and manual code review findings. The work was undertaken from **Dec 10, 2024** to **Jan 14, 2025**.

The following contract list is included in our scope:

- `contracts/` `Safe.sol`
- `contracts/` `SafeL2.sol`
- `contracts/accessors` `SimulateTxAccessor.sol`
- `contracts/base` `Executor.sol`
- `contracts/base` `FallbackManager.sol`
- `contracts/base` `GuardManager.sol`
- `contracts/base` `ModuleManager.sol`
- `contracts/base` `OwnerManager.sol`
- `contracts/common` `NativeCurrencyPaymentFallback.sol`
- `contracts/common` `SecuredTokenTransfer.sol`
- `contracts/common` `SelfAuthorized.sol`
- `contracts/common` `SignatureDecoder.sol`
- `contracts/common` `Singleton.sol`
- `contracts/common` `StorageAccessible.sol`
- `contracts/external` `SafeMath`
- `contracts/handler/extensible` `ERC165Handler`
- `contracts/handler/extensible` `ExtensibleBase`
- `contracts/handler/extensible` `FallbackHandler`
- `contracts/handler/extensible` `MarshalLib`

- `contracts/handler/extensible` `SignatureVerifierMuxer`
- `contracts/handler/extensible` `TokenCallbacks`
- `contracts/handler` `CompatibilityFallbackHandler`
- `contracts/handler` `ExtensibleFallbackHandler`
- `contracts/handler` `HandlerContext`
- `contracts/handler` `TokenCallbackHandler`
- `contracts/libraries` `CreateCall`
- `contracts/libraries` `Enum`
- `contracts/libraries` `ErrorMessage`
- `contracts/libraries` `MultiSend`
- `contracts/libraries` `MultiSendCallOnly`
- `contracts/libraries` `SafeMigration`
- `contracts/libraries` `SafeStorage`
- `contracts/libraries` `SafeToL2Migration`
- `contracts/libraries` `SafeToL2Setup`
- `contracts/libraries` `SignMessageLib`
- `contracts/proxies` `SafeProxy`
- `contracts/proxies` `SafeProxyFactory`

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts**.** During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	1	1	1
Low	5	5	4
Informational	3	3	2
Gas Optimization	10	10	10
Total	**19**	**19**	**17**

Severity Matrix

		Likelihood		
		Low	Medium	High
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium

Detailed Findings

ID	Title	Severity	Status
M-01	ERC-777 compatibility isn't implemented correctly	Medium	Fixed
L-01	SafeProxy.fallback(): The dirty bits aren't correctly cleared	Low	Fixed
L-02	Wrong signature description	Low	Fixed
L-03	Possible mismatch between safeMethods and safeInterfaces	Low	Won’t Fix
L-04	Possible dirty bits in getTransactionHash()	Low	Fixed
L-05	FallbackManager should staticcall instead of call the Fallback Handler	Low	Fixed
I-01	Events lacking indexed fields	Info	Won’t Fix
I-02	Some comments say keccak instead of keccak256	Info	Fixed
I-03	Inconsistency in formula for performCreate and performCreate2	Info	Fixed
G-01	OwnerManager.removeOwner(): 1 SLOAD can be saved in the normal path	Gas	Fixed
G-02	OwnerManager.changeThreshold(): 1 SLOAD can be saved by emitting an existing memory variable instead of reading from storage	Gas	Fixed

G-03	ERC165Handler.setSupportedInterface(): Logic and storage access optimization	Gas	Fixed
G-04	ExtensibleBase._setSafeMethod(): storage access optimization	Gas	Fixed
G-05	Use iszero instead of eq(*, 0)	Gas	Fixed
G-06	ExtensibleFallbackHandler._supportsInterface(): save gas via short-circuit evaluation	Gas	Fixed
G-07	Use a mask instead of shifting left and right	Gas	Fixed
G-08	Use shift right/left instead of division/multiplication if possible	Gas	Fixed
G-09	Cache array length outside of loop	Gas	Fixed
G-10	++i costs less gas compared to i++ or i += 1 (same for --i vs i-- or i -= 1)	Gas	Fixed

Medium Severity Issues

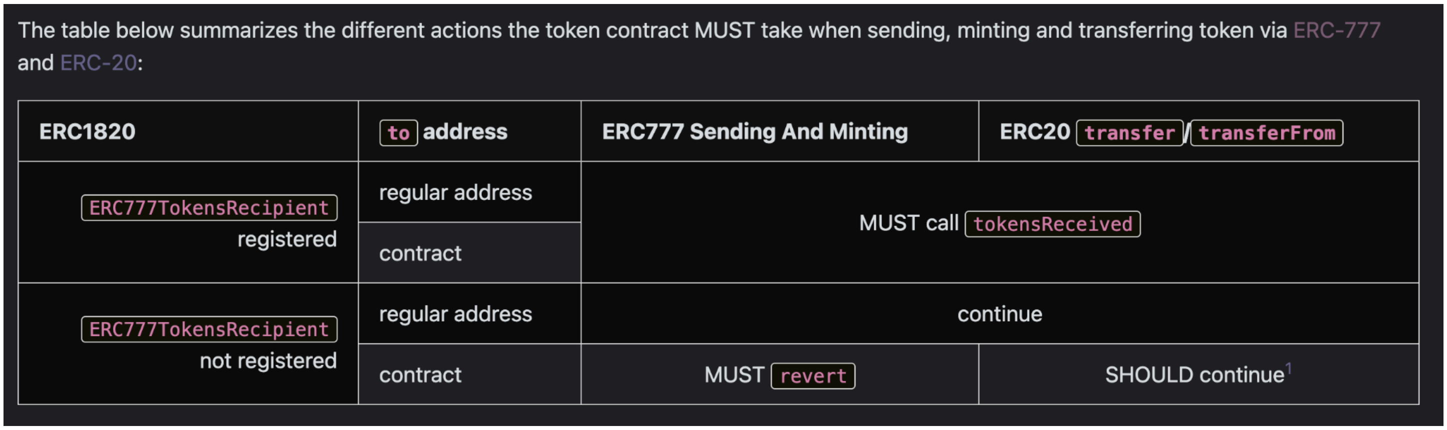
M-01 ERC-777 compatibility isn't implemented correctly

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: ● [CompatibilityFallbackHandler.sol](#) ● [TokenCallbackHandler.sol](#)	Status: Fixed	

Description: If we analyze how the compatibility with ERC-777 is implemented:

- ☑ `contract CompatibilityFallbackHandler is TokenCallbackHandler` and `contract TokenCallbackHandler is ... ERC777TokensRecipient`
- ☑ `function tokensReceived` is implemented
- ☑ `TokenCallbackHandler.supportsInterface()` doesn't declare `type(ERC777TokensRecipient).interfaceId` because the standard from ERC-777 doesn't need it: they use the ERC-1820 Registry
- □ Only the contract itself can register itself as a `ERC777TokensRecipient` on the ERC1820 Registry otherwise the calls to `ERC77.send()` or `ERC777.mint()` will revert.

Currently, ERC777 tokens will not recognize the contract as a recipient, although it's expected to be one. According to the ERC-777 standard, calls to the ERC777 token's `send()` or `mint()` functions with this contract as a recipient will revert.



The table below summarizes the different actions the token contract MUST take when sending, minting and transferring token via ERC-777 and ERC-20:

ERC1820	`to` address	ERC777 Sending And Minting	ERC20 `transfer`/`transferFrom`
`ERC777TokensRecipient` registered	regular address	MUST call `tokensReceived`	
	contract		
`ERC777TokensRecipient` not registered	regular address	continue	
	contract	MUST `revert`	SHOULD continue[1]

Existing workaround: Depending on what's inheriting from `CompatibilityFallbackHandler` (which isn't an abstract contract but a contract), there could exist a workaround requiring users to manually call `ERC1820Registry.setInterfaceImplementer()`, so a fix on the frontend could exist for older contracts.

Recommendations:
A call similar to `ERC1820Registry(registry).setInterfaceImplementer(address(this), keccak256("ERC777TokensRecipient"), address(this))` is missing for a real ERC-777 compatibility.

Safe's response: PR 885: This PR adds a detailed comment in the TokenCallbackHandler contract, clarifying the requirement for accounts to register the implementer via the ERC-1820 interface registry to receive ERC777 tokens. This update aims to improve clarity and understanding of the token reception process. No functional changes were made to the contract logic.

Low Severity Issues

L-01 SafeProxy.fallback(): The dirty bits aren't correctly cleared

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: SafeProxy.sol	Status: Fixed	

Description: At SafeProxy.sol#L43, the `shr(12, shl(12, _singleton))` operation is used because an `address` is 20 bytes long on the small-endian.

```
        assembly {
            let _singleton := sload(0)
            // 0xa619486e == keccak("masterCopy()"). The value is right padded
to 32-bytes with 0s
            if eq(calldataload(0),
0xa619486e00000000000000000000000000000000000000000000000000000000) {
                mstore(0, shr(12, shl(12, _singleton)))
                return(0, 0x20)
            }
```

As `let _singleton := sload(0)` operation loads a 32 bytes word, the aim is to clear the potential dirty 12 bytes. However, `shr` and `shl` operate in bits, not bytes. Therefore, the correct operation would instead be `shr(96, shl(96, _singleton))`

Safe's response: Fixed in PR 868

L-02 Wrong signature description

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: SignatureVerifierMuxer.sol	Status: Fixed	

Description: The comment `// 0x68 - 0x6C: encodeData length` at SignatureVerifierMuxer.sol#L124 is describing the length as being described 4 bytes.

However, a length is `0x20` (32 bytes).
Therefore the comment should be `// 0x68 - 0x88: encodeData length` (and the next few comments should reflect this change)
A user taking this as documentation would produce a wrong signature.

Safe's response: Fixed in PR 873

L-03 Possible mismatch between `safeMethods` and `safeInterfaces`		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: ERC165Handler.sol	Status: Won't fix	

Description:

Duplicates in the `handlerWithSelectors` array can lead to a mismatch between the `interfaceId` and the actual selectors added to `safeMethods` due to the XOR and `A ^ A = 0` (at ERC165Handler.sol#L53-L68).
Example:

- `handlerWithSelectors` is `[0x12345678, 0xabcdef01, 0x12345678]`
- Computed `interfaceId` is `0x12345678 ^ 0xabcdef01 ^ 0x12345678 = 0xabcdef01`
- Intended `interfaceId` is `0x12345678 ^ 0xabcdef01 = 0xbc99f579`

=> `safeMethods` will correctly store all the selectors, but the `interfaceId` won't represent the actual set of selectors due to the duplicate canceling out in the XOR computation.

This issue would be self-inflicted and easily cleaned with a call to `removeSupportedInterfaceBatch`.

A remediation could be to force an order (e.g. `handlerWithSelectors[i - 1] < handlerWithSelectors[i]`) but this would increase gas usage.

Safe's response:
Acknowledged but won't fix (self-inflicted and costs more gas)

L-04 Possible dirty bits in `getTransactionHash()`

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Safe.sol	Status: Fixed	

Description:

The solidity assembly types that are smaller than 256 bit can have dirty high bits according to the spec (see the Warning in the solidity docs). We have seen instances of these in practice. This means that technically the `getTransactionHash()`function is not correct as the high bits in the three `address` fields and the `enum` field (`enum` is an `uint8`) could be non-zero in some future compiler version.

Consider using a bitmask `and` operation to clean the higher bits out in the assembly block itself as the solidity docs suggest.

Safe's response:
Documented in PR 872. The current implementation is not affected as it reads these fields from calldata where the solidity compiler checks that the unused bits are cleared.

L-05 FallbackManager should `staticcall` instead of `call` the Fallback Handler

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: - [Safe.sol] - [FallbackManager.sol]	Status: Fixed	

Description: There's a mismatch between the description here:

```
File: Safe.sol
31:  *      - Fallback: Fallback handler is a contract that can provide
additional read-only functionality for Safe. Managed in `FallbackManager`.
```

And the actual implementation here:

```
File: FallbackManager.sol
80:             let success := call(gas(), handler, 0, ptr,
add(calldatasize(), 20), 0, 0)
```

Instead of a `call` which could induce a state change, this here should be a `staticcall`.
This gives a better protection against state changes (e.g. backdoor creation) and saves gas.

Safe's response:
PR 879: Updating the documentation.
The reasoning is pretty simple: we need to have a "CALL" opcode here to future-proof ourselves if a new standard requires a new method to be defined precisely at the deployed contract's address.
One recent example would be ERC-4337, which required an account to define the `validateUserOp` method, which also changes the blockchain's state (sending the pre-fund to the bundler) - with a static call that'd be impossible.

Informational Severity Issues

I-01. Events lacking `indexed` fields

Description:

The events at SafeL2.sol#L16-L32 are lacking indexed fields.
It would be relevant for `event SafeModuleTransaction(address module, address to, uint256 value, bytes data, Enum.Operation operation)` to have `address indexed module` and `address indexed to`.
It would also be relevant for `event SafeMultiSigTransaction` to have `address indexed to`

Safe's response:
We don't see any immediate benefit. Usually, the requirements for a field to be indexed come from our backend team that develops the indexer; they have never brought this up.

When we want to fetch the transactions, we need to fetch all of them first and then build whatever index is needed off-chain

I-02. Some comments say keccak instead of keccak256

Description:

Comments across the codebase always say keccak256 instead of keccak:

```
contracts/interfaces/ISignatureValidator.sol:
  6:     // bytes4(keccak256("isValidSignature(bytes32,bytes)")

contracts/libraries/SafeToL2Migration.sol:
  115:         // 0xef2624ae - bytes4(keccak256("migrateToL2(address)"))
  145:         // 0xd9a20812 -
bytes4(keccak256("migrateFromV111(address,address)"))
```

However, the following are an exception. For consistency and ease of copy-pasting for auditors and developers to test the signatures, we recommend this fix:

```
contracts/common/SecuredTokenTransfer.sol:
-  19:         // 0xa9059cbb - keccak("transfer(address,uint256)")
+  19:         // 0xa9059cbb - bytes4(keccak256("transfer(address,uint256)"))

contracts/proxies/SafeProxy.sol:
-  41:             // 0xa619486e == keccak("masterCopy()"). The value is right
padded to 32-bytes with 0s
+  41:             // 0xa619486e == bytes4(keccak256("masterCopy()")). The
value is right padded to 32-bytes with 0s
```

Safe's response: Fixed in PR 886

I-03. Inconsistency in formula for performCreate and performCreate2

Description:

performCreate2 has the following line:

```
File: CreateCall.sol
            newContract := create2(value, add(0x20, deploymentData), mload(deploymentData), salt)
```

performCreate has the following line:

```
File: CreateCall.sol
            newContract := create(value, add(deploymentData, 0x20), mload(deploymentData))
```

Commutativity makes the two additions being equivalent but we recommend the fix below for readability and to follow the standard given that:

- deploymentData gives a pointer to the start of the array (length position).
- Adding 0x20 skips the first 32 bytes (length field) to point directly to the start of the payload.

Recommendations:

```
File: CreateCall.sol
    function performCreate2(uint256 value, bytes memory deploymentData, bytes32 salt) public returns (address newContract) {
        /* solhint-disable no-inline-assembly */
        /// @solidity memory-safe-assembly
        assembly {
-           newContract := create2(value, add(0x20, deploymentData), mload(deploymentData), salt)
+           newContract := create2(value, add(deploymentData, 0x20), mload(deploymentData), salt)
        }
        /* solhint-enable no-inline-assembly */
        require(newContract != address(0), "Could not deploy contract");
        emit ContractCreation(newContract);
    }
```

Safe's response: Fixed in PR 887

Gas Optimization

G-01. OwnerManager.removeOwner(): 1 SLOAD can be saved in the normal path

Description:

The current code reads from storage twice with the `ownerCount` variable:

```
File: OwnerManager.sol
74:          if (ownerCount - 1 < _threshold) revertWithError("GS201");
//@audit ownerCount SLOAD 1
75:          // Validate owner address and check that it corresponds to owner
index.
76:          if (owner == address(0) || owner == SENTINEL_OWNERS)
revertWithError("GS203");
77:          if (owners[prevOwner] != owner) revertWithError("GS205");
78:          owners[prevOwner] = owners[owner];
79:          owners[owner] = address(0);
80:          ownerCount--;  //@audit ownerCount SLOAD 2 + SSTORE 1
```

However this is unfair to the normal and expected functioning scenario where the transaction is successful (doesn't revert with `"GS201"`).

The following would save gas under the usual and most probable scenario:

```
- 74:          if (ownerCount - 1 < _threshold) revertWithError("GS201");
//@audit ownerCount SLOAD 1
+ 74:          if (--ownerCount < _threshold) revertWithError("GS201");
//@audit ownerCount SLOAD 1 + SSTORE 1
75:          // Validate owner address and check that it corresponds to owner
index.
76:          if (owner == address(0) || owner == SENTINEL_OWNERS)
revertWithError("GS203");
77:          if (owners[prevOwner] != owner) revertWithError("GS205");
78:          owners[prevOwner] = owners[owner];
79:          owners[owner] = address(0);
- 80:          ownerCount--;  //@audit ownerCount SLOAD 2 + SSTORE 1
```

Safe's response: Fixed in PR 888

G-02. OwnerManager.changeThreshold(): 1 SLOAD can be saved by emitting an existing memory variable instead of reading from storage

Description:

```
File: OwnerManager.sol
    function changeThreshold(uint256 _threshold) public override authorized {
        // Validate that threshold is smaller than number of owners.
        if (_threshold > ownerCount) revertWithError("GS201");
        // There has to be at least one Safe owner.
        if (_threshold == 0) revertWithError("GS202");
        threshold = _threshold;
-       emit ChangedThreshold(threshold);
+       emit ChangedThreshold(_threshold);
    }
```

Safe's response: Fixed in PR 889

G-03. ERC165Handler.setSupportedInterface(): Logic and storage access optimization

Description:

When accessing a nested struct or mapping several times in storage, it's possible to save on gas by locally saving the reference using the `storage` keyword.
The following optimizes the logic and minimizes storage accesses:

```
File: ERC165Handler.sol
    function setSupportedInterface(bytes4 interfaceId, bool supported) public
override onlySelf {
        ISafe safe = ISafe(payable(_manager()));
        // invalid interface id per ERC165 spec
        require(interfaceId != 0xffffffff, "invalid interface id");
-         bool current = safeInterfaces[safe][interfaceId];
+         mapping(bytes4 => bool) storage safeInterface =
safeInterfaces[safe];
+         bool current = safeInterface[interfaceId];
-        if (supported && !current) {
-            safeInterfaces[safe][interfaceId] = true;
-            emit AddedInterface(safe, interfaceId);
-        } else if (!supported && current) {
-            delete safeInterfaces[safe][interfaceId];
-            emit RemovedInterface(safe, interfaceId);
-        }
+        if (supported != current) {
+            safeInterface[interfaceId] = supported;
+            if (supported) {
+                emit AddedInterface(safe, interfaceId);
+            } else {
+                emit RemovedInterface(safe, interfaceId);
+            }
+        }
    }
```

Safe's response: Fixed in PR 890

G-04. ExtensibleBase._setSafeMethod(): storage access optimization

Description:

Same as previously, locally saving the storage reference will save gas:

```
File: ExtensibleBase.sol
47:     function _setSafeMethod(ISafe safe, bytes4 selector, bytes32
newMethod) internal {
48:         (, address newHandler) = MarshalLib.decode(newMethod);
- 49:         bytes32 oldMethod = safeMethods[safe][selector];
+ 49:         mapping(bytes4 => bytes32) storage safeMethod =
safeMethods[safe];
+ 50:         bytes32 oldMethod = safeMethod[selector];
50:         (, address oldHandler) = MarshalLib.decode(oldMethod);
51:
52:         if (address(newHandler) == address(0) && address(oldHandler) !=
address(0)) {
- 53:             delete safeMethods[safe][selector];
+ 53:             delete safeMethod[selector];
54:             emit RemovedSafeMethod(safe, selector);
55:         } else {
- 56:             safeMethods[safe][selector] = newMethod;
+ 56:             safeMethod[selector] = newMethod;
57:             if (address(oldHandler) == address(0)) {
58:                 emit AddedSafeMethod(safe, selector, newMethod);
59:             } else {
60:                 emit ChangedSafeMethod(safe, selector, oldMethod,
newMethod);
61:             }
62:         }
63:     }
```

Safe's response: Fixed in PR 891

G-05. Use iszero instead of eq(*, 0)

Description:

This follows the steps of the following past fix:
https://github.com/safe-global/safe-smart-account/commit/7f79aaf05c33df71d9cb687f0bc8a73fa39d25d5

> Lastly, I noticed that there were some `eq(..., 0)` assembly calls which can be written as `iszero(...)` to save some gas and code...

```
File: SafeProxyFactory.sol
- 43:                 if eq(call(gas(), proxy, 0, add(initializer, 0x20),
mload(initializer), 0, 0), 0) {
+ 43:                 if iszero(call(gas(), proxy, 0, add(initializer, 0x20),
mload(initializer), 0, 0)) {
```

Safe's response: Fixed in PR 892

G-06. `ExtensibleFallbackHandler._supportsInterface()`: save gas via short-circuit evaluation

Description:

If it's expected, like the comment seems to explain, that `_supportsInterface` will most often be called for `ERC721 + ERC1155` tokens: consider reordering the `||` conditions to take advantage of the short-circuit evaluation:

```
File: ExtensibleFallbackHandler.sol
14: contract ExtensibleFallbackHandler is FallbackHandler,
SignatureVerifierMuxer, TokenCallbacks, ERC165Handler {
15:     /**
16:      * Specify specific interfaces (ERC721 + ERC1155) that this contract
supports.
17:      * @param interfaceId The interface ID to check for support
18:      */
19:     function _supportsInterface(bytes4 interfaceId) internal pure override
returns (bool) {
20:         return
+ 21:             interfaceId == type(ERC721TokenReceiver).interfaceId ||
+ 21:             interfaceId == type(ERC1155TokenReceiver).interfaceId ||
21:             interfaceId == type(ERC1271).interfaceId ||
22:             interfaceId == type(ISignatureVerifierMuxer).interfaceId ||
23:             interfaceId == type(ERC165Handler).interfaceId ||
- 24:             interfaceId == type(IFallbackHandler).interfaceId ||
+ 24:             interfaceId == type(IFallbackHandler).interfaceId
- 25:             interfaceId == type(ERC721TokenReceiver).interfaceId ||
- 26:             interfaceId == type(ERC1155TokenReceiver).interfaceId;
27:     }
28: }
```

Safe's response: Fixed in [PR 893]

G-07. Use a mask instead of shifting left and right

Description:

A direct bitmask (`and`) operation is cheaper than using `shl` followed by `shr` (saving 3 gas due to 1 less opcode):

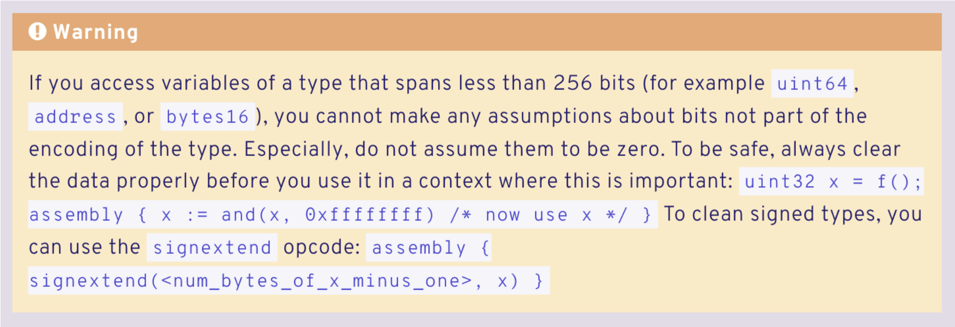
```
File: MarshalLib.sol
- 41:             handler := shr(96, shl(96, data))
+ 41:             handler := and(data,
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)

File: MarshalLib.sol
- 59:             handler := shr(96, shl(96, data))
+ 59:             handler := and(data,
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)

File: SignatureVerifierMuxer.sol
- 113:                 sigSelector := shl(224, shr(224,
calldataload(signature.offset)))
+ 113:                 sigSelector := and(calldataload(signature.offset),
0xFFFFFFFF00000000000000000000000000000000000000000000000000000000)
```

Safe's response: Can the conversion be omitted here at all?

Certora's response: The bits should be cleaned for the addresses as only a reuse in Solidity itself would clean them. If another assembly blocks uses it, the dirty bits will remain. Given that this is a lib, how it's used isn't guaranteed, so it's better to manually clean. See also the warning from the Solidity language documentation. However for bytes4 sigSelector, we're back to solidity after the assembly block (and the value isn't reused later), so it can indeed be removed.



Warning

If you access variables of a type that spans less than 256 bits (for example `uint64`, `address`, or `bytes16`), you cannot make any assumptions about bits not part of the encoding of the type. Especially, do not assume them to be zero. To be safe, always clear the data properly before you use it in a context where this is important: `uint32 x = f(); assembly { x := and(x, 0xffffffff) /* now use x */ }` To clean signed types, you can use the `signextend` opcode: `assembly { signextend(<num_bytes_of_x_minus_one>, x) }`

Safe's response: PR 894: Kept the handler cleaning, and removed the sigSelector one.

G-08. Use shift right/left instead of division/multiplication if possible

Description:

While the `DIV` / `MUL` opcode uses 5 gas, the `SHR` / `SHL` opcode only uses 3 gas. Furthermore, beware that Solidity's division operation also includes a division-by-0 prevention which is bypassed using shifting. Eventually, overflow checks are never performed for shift operations as they are done for arithmetic operations. Instead, the result is always truncated, so the calculation can be unchecked in Solidity version `0.8+`

- Use `>> 1` instead of `/ 2`
- Use `>> 2` instead of `/ 4`
- Use `<< 3` instead of `* 8`
- ...
- Use `>> 5` instead of `/ 2^5 == / 32`
- Use `<< 6` instead of `* 2^6 == * 64`

TL;DR:

- Shifting left by N is like multiplying by 2^N (Each bits to the left is an increased power of 2)
- Shifting right by N is like dividing by 2^N (Each bits to the right is a decreased power of 2)

Saves around 2 gas + 20 for unchecked per instance

Affected code:

- contracts/Safe.sol

```
# File: contracts/Safe.sol

- Safe.sol:168:         if (gasleft() < ((safeTxGas * 64) / 63).max(safeTxGas
+ 2500) + 500) revertWithError("GS010");
+ Safe.sol:168:         if (gasleft() < ((safeTxGas << 6) / 63).max(safeTxGas
+ 2500) + 500) revertWithError("GS010");
```

- contracts/common/StorageAccessible.sol

```
# File: contracts/common/StorageAccessible.sol

- StorageAccessible.sol:18:         bytes memory result = new bytes(length *
32);
+ StorageAccessible.sol:18:         bytes memory result = new bytes(length <<
5);
```

Safe's response: Fixed in PR 895

G-09. Cache array length outside of loop

Description:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Affected code:

- contracts/base/OwnerManager.sol

```
# File: contracts/base/OwnerManager.sol

OwnerManager.sol:38:         for (uint256 i = 0; i < _owners.length; i++) {
```

- contracts/handler/extensible/ERC165Handler.sol

```
# File: contracts/handler/extensible/ERC165Handler.sol

ERC165Handler.sol:56:         for (uint256 i = 0; i <
handlerWithSelectors.length; i++) {

ERC165Handler.sol:78:         for (uint256 i = 0; i < selectors.length; i++) {
```

Safe's response: Fixed in PR 896

G-10. `++i` costs less gas compared to `i++` or `i += 1` (same for `--i` vs `i--` or `i -= 1`)

Description:

Pre-increments and pre-decrements are cheaper.

For a `uint256 i` variable, the following is true with the Optimizer enabled at 10k:

Increment:

- `i += 1` is the most expensive form
- `i++` costs 6 gas less than `i += 1`
- `++i` costs 5 gas less than `i++` (11 gas less than `i += 1`)

Decrement:

- `i -= 1` is the most expensive form
- `i--` costs 11 gas less than `i -= 1`
- `--i` costs 5 gas less than `i--` (16 gas less than `i -= 1`)

Note that post-increments (or post-decrements) return the old value before incrementing or decrementing, hence the name *post-increment*:

```
uint i = 1;
uint j = 2;
require(j == i++, "This will be false as i is incremented after the
comparison");
```

However, pre-increments (or pre-decrements) return the new value:

```
uint i = 1;
uint j = 2;
require(j == ++i, "This will be true as i is incremented before the
comparison");
```

In the pre-increment case, the compiler has to create a temporary variable (when used) for returning `1` instead of `2`.

Consider using pre-increments and pre-decrements where they are relevant (meaning: not where post-increments/decrements logic are relevant).

Saves 5 gas per instance

Affected code:

- contracts/Safe.sol

```
# File: contracts/Safe.sol

Safe.sol:296:         for (i = 0; i < requiredSignatures; i++) {
```

- contracts/base/ModuleManager.sol

```
# File: contracts/base/ModuleManager.sol

ModuleManager.sol:215:             moduleCount++;
```

- contracts/base/OwnerManager.sol

```
# File: contracts/base/OwnerManager.sol

OwnerManager.sol:38:         for (uint256 i = 0; i < _owners.length; i++) {

OwnerManager.sol:63:         ownerCount++;

OwnerManager.sol:142:             index++;
```

- contracts/common/StorageAccessible.sol

```
# File: contracts/common/StorageAccessible.sol

StorageAccessible.sol:19:         for (uint256 index = 0; index < length;
index++) {
```

- contracts/handler/extensible/ERC165Handler.sol

```
# File: contracts/handler/extensible/ERC165Handler.sol

ERC165Handler.sol:56:         for (uint256 i = 0; i <
handlerWithSelectors.length; i++) {

ERC165Handler.sol:78:         for (uint256 i = 0; i < selectors.length; i++) {
```

- contracts/libraries/SafeToL2Migration.sol

```
# File: contracts/libraries/SafeToL2Migration.sol

SafeToL2Migration.sol:188:             index++;
```

Note: *Do not change the following line* as this would break the logic:

```
# File: contracts/Safe.sol

Safe.sol:140:                 nonce++
```

Safe's response: Fixed in PR 897

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

Formal Verification Properties

Safe.sol

Module General Assumptions

- Loop iterations: Any loop was unrolled at most 3 times (iterations)

Contract Properties

P-01. Integrity of the Transaction Guard methods.	
Status: Verified	

Rule Name	Status	Description	Link to rule report
guardAddressChange	Verified	*The only method that can change the transaction guard is setGuard.*	Report
setGetCorrespondenceGuard	Verified	*Making sure that set and get work as expected for the transaction guard.*	Report
setGuardReentrant	Verified	*setGuard can only be called by contract itself.*	Report
txnGuardCalled	Verified	*The transaction guard gets called both pre- and post- any execTransaction.*	Report

P-02. Integrity of the Module Guard methods.

Status: Verified

Rule Name	Status	Description	Link to rule report
moduleGuardAddressChange	Verified	*The only method that can change the module guard is setModuleGuard.*	[Report]
setGetCorrespondenceModuleGuard	Verified	*Making sure that set and get work as expected for the module guard.*	[Report]
setModuleGuardReentrant	Verified	*setModuleGuard can only be called by contract itself.*	[Report]
moduleGuardCalled	Verified	*The module guard gets called both pre- and post- any execTransactionFromModule.*	[Report]
moduleGuardCalledReturn	Verified	*The module guard gets called both pre- and post- any execTransactionFromModuleReturnData.*	[Report]

P-03. Integrity of Execute Transaction and Execute Transaction from Module

Status: Verified	

Rule Name	Status	Description	Link to rule report
execTxnModulePermissions	Verified	*A successful call to execTransactionFromModule must be from an enabled module.*	[Report]()
execTxnModuleReturnDataPermissions	Verified	*A call to execTransactionFromModuleReturnData that succeeds must be from an enabled module.*	[Report]()
executePermissions	Verified	*Execute can only be called by execTransaction or execTransactionFromModule.*	[Report]()
executeThresholdMet	Verified	*The number of signatures provided for any executing transaction meets the correct threshold.*	[Report]()

P-04. Integrity of approveHash and approvedHashVal.

Status: Verified	

Rule Name	Status	Description	Link to rule report
approvedHashesUpdate	Verified	*approvedHashes[user][hash] can only be changed by msg.sender==user.*	*Report*
approvedHashesSet	Verified	*approvedHashes is set when calling approveHash*	*Report*
transactionHashCantCollide	Verified	*The hash of two distinct transactions cannot be the same. Requires some munging of Safe.sol at line 456. Although the original assembly code is correct, it accesses the memory unaligned which breaks the Certora prover. The munging removes the unaligned access for verification. Note that this munged code must not be used in production, as it computes a slightly different hash.*	*Report*

P-05. Integrity of Setup	
Status: Verified	

Rule Name	Status	Description	Link to rule report
setupThresholdZeroAndSetsPositiveThreshold	Verified	*setup can only be called if threshold = 0 and setup sets threshold > 0*	[Report]

ExtensibleFallbackHandler.sol

Module General Assumptions

- Loop iterations: Any loop was unrolled at most 3 times (iterations).

Contract Properties

P-01. Integrity of the Extensible Fallback Handler	
Status: Verified	

Rule Name	Status	Description	Link to rule report
setFallbackIntegrity	Verified	*The fallback handler gets set by setFallbackHandler.*	Report
fallbackHandlerNeverSelf	Verified	*The address for the fallback handler slot is never set to the Safe contract.*	Report
simulateAndRevertReverts	Verified	*simulateAndRevert always reverts.*	Report
setSafeMethodSets	Verified	*setSafeMethod sets the handler.*	Report
setSafeMethodRemoves	Verified	*setSafeMethod removes the handler.*	Report
setSafeMethodChanges	Verified	*setSafeMethod changes the handler.*	Report
handlerCallableIfSet	Verified	*A handler, once set via setSafeMethod, is possible to call.*	Report
handlerCalledIfSet	Verified	*A handler, once set visa setSafeMethod, gets called under the expected conditions.*	Report

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.