Brahma Security Review

Auditors

Saw-mon and Natalie, Lead Security Researcher

Gerard Persoon, Lead Security Researcher

Xiaoming90, Security Researcher

Philogy, Security Researcher

Report prepared by: Lucas Goiriz

November 22, 2023

Contents

1 About Spearbit **3**

2 Introduction **3**

3 Risk classification **3**
3.1 Impact . . . 3
3.2 Likelihood . . . 3
3.3 Action required for severity levels . . . 3

4 Executive Summary **4**

5 Findings **5**
5.1 Medium Risk . . . 5
5.1.1 Governance can remove policy check due to upgradability . . . 5
5.1.2 Governance can backdoor new Safes via malicious upgrade . . . 5
5.1.3 Changes in modules not detected . . . 6
5.1.4 Governance can brick Safes by blocking moderator override . . . 6
5.1.5 Subaccount can be console account . . . 7
5.1.6 `registerWallet()` doesn't verify the wallet is a real `Safe` . . . 8
5.2 Low Risk . . . 9
5.2.1 Upgrade of `trustedValidator` could circumvent Policy checks . . . 9
5.2.2 `Console` can brick its sub accounts in some scenarios if it removes itself as a module . . . 9
5.2.3 Anyone can deploy a `console` with the same set of parameters but with a different `_policyCommit` . . . 10
5.2.4 Changes between signing and execution could yield results that are outside the bounds of the policy . . . 10
5.3 Gas Optimization . . . 11
5.3.1 Function `updatePolicy()` can be optimized . . . 11
5.3.2 Unnecessary custom getters . . . 11
5.3.3 Use of storage for immutable registry addresses in `AddressProvider` . . . 12
5.3.4 Unneeded helper contract `SafeEnabler` . . . 13
5.3.5 Access `msg.sender` within the function . . . 14
5.3.6 Location of `_genNonce()` can be optimized . . . 15
5.3.7 Function `isPolicySignatureValid()` can be optimized . . . 16
5.3.8 Check for lacking policies can be done earlier . . . 18
5.3.9 Addresses retrieved twice in `executeTransaction()` and `validatePostExecutorTransaction()` . . . 18
5.3.10 `keccak256` result can be cached . . . 19
5.3.11 Function `updatePolicy` can be optimized . . . 20
5.4 Informational . . . 21
5.4.1 Suggestions for the deployment script `ConsoleFactory` . . . 21
5.4.2 Initialization of hashes in `Constants` . . . 22
5.4.3 Enum conversion in `_packMultisendTxns()` not obvious . . . 23
5.4.4 How to recover from a bug in `ExecutorPlugin` . . . 24
5.4.5 Re-implementation of Ownership with 2-step transfer pattern . . . 24
5.4.6 Type hash is not aligned with `ValidationParams` struct . . . 25
5.4.7 Trusted validator weakens Safe access control via `ExecutorPlugin` . . . 26
5.4.8 Use builtin functions to compute constant values . . . 26
5.4.9 Make sure the chain used to deploy the protocol implements the desired precompiles at the correct addresses . . . 27
5.4.10 Best to use `uint64` as a type for `timestamp` . . . 27
5.4.11 Incomplete or incorrect comments . . . 27
5.4.12 `abi.encodeCall` should be used instead of `abi.encodePacked` . . . 28
5.4.13 `AccountSecurityConfig` checks could be expanded . . . 28
5.4.14 Misleading function names . . . 29

5.4.15 Function `_decompileSignatures()` could revert without descriptive error 29
5.4.16 Unused code 30
5.4.17 `isValidSignature()` does fewer checks than `SafeModerator` 30
5.4.18 Typographical errors 31
5.4.19 `isValidSignature()` doesn't check policy in combination with `signedMessages()` 31
5.4.20 Function `_executeOnSafe()` isn't used 32

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Brahma Console is a specialised custody and execution environment for on-chain execution. With Console, users can segregate risk and delegate operations, as well as execute and automate transactions on any dApp.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brahma console-core-v2 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 8 days in total, Brahma engaged with Spearbit to review the console-core-v2 protocol. In this period of time a total of **41** issues were found.

Summary

Project Name	Brahma
Repository	console-core-v2
Commit	85c675...ec4c87
Type of Project	DeFi
Audit Timeline	Oct 30 to Nov 8
Two week fix period	Nov 8 - Nov 18

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	6	2	4
Low Risk	4	0	4
Gas Optimizations	11	9	2
Informational	20	9	11
Total	**41**	**20**	**21**

5 Findings

5.1 Medium Risk

5.1.1 Governance can remove policy check due to upgradability

Severity: Medium Risk

Context: AddressProvider.sol#L77

Description: Governance has the possiblity to upgrade the values for several configurations. This allows circumventing the Policy checks, due to a bug, mistake, or malicious behaviour. Combined with a lack of security or checks on other places (e.g. multisig signers, too low a threshold) funds could be lost or stolen.

```
_getAuthorizedAddress(_TRANSACTION_VALIDATOR_HASH)
_getAuthorizedAddress(_TRUSTED_VALIDATOR_HASH)
_getAuthorizedAddress(_POLICY_VALIDATOR_HASH)
```

Recommendation: Determine a path to make them immutable, or have a way where an explicit Safe operation (signed by owner) is required to upgrade to a new version.

Brahma: Acknowledged, within the trust scope of governance. Given that the governance and team is known and not anon, that increases the deterrence of such attacks

Spearbit: Acknowledged

5.1.2 Governance can backdoor new Safes via malicious upgrade

Severity: Medium Risk

Context: AddressProvider#L77-L90, SafeDeployer#L223-L224

Description: The `SafeDeployer` contract is responsible for deploying and configuring new "top-level" console accounts and sub-accounts. These are initialized via the authorized safe factory address with the authorized reference singleton (the implementation contract for Safe multi-signature wallets). These addresses upon which the `SafeDeployer` relies can be set by the `governance` address of the `AddressProvider`.

Malicious governance could manipulate these addresses to have new accounts be deployed with an invalid, faulty or even backdoored safe, meaning users could start unsuspectingly using their accounts and one day unsuspectingly be exploited by a backdoor that governance previously installed. This could be obfuscated by making the malicious configuration part of a sandwich attack whereby deploying transactions are wrapped with 2 transactions that set and unset the malicious variants, this would prevent users from detecting this by simply querying and verifying the address configured in the `AddressProvider`.

The likelihood for this happening is low, but if it would happen the impact for the users is high. Therefore we've set this to medium risk.

Recommendation: This should be fixed by either hardcoding the safe factory and singleton addresses into the `SafeDeployer` contract or allowing users to pass them in as parameters, avoiding the provider as the core source of truth. This not only solves this issue but will also improve the gas usage of the safe creation method as the addresses will be available locally (either via calldata parameter or hardcoded as a constant) meaning the expensive external call + storage access will be avoided.

Brahma: Acknowledged, within the trust scope of governance. Need singleton to be upgradable in case new bugs are discovered. Also severity for governance abuse and impact should be on low likelihood.

Spearbit: Acknowledged.

5.1.3 Changes in modules not detected

Severity: Medium Risk

Context: TransactionValidator.sol#L134-L150

Description: The function `validatePostExecutorTransaction()` checks the executor plugin is still enabled. However other modules are not checked. As modules are very powerful, it they would be "sneaked" in, they would pose risks to the safes.

```
function validatePostExecutorTransaction(address, /*msgSender */ address account) external view {
    // ...
    // Check if account has executor plugin still enabled as a module on it
    if (!IGnosisSafe(account).isModuleEnabled(AddressProviderService._getAuthorizedAddress(_EXECUTOR_PL
↪  UGIN_HASH)))
↪  {
        revert InvalidExecutorPlugin();
    }
    // ...
}
```

Recommendation: Consider checking the list of modules. This could be done by letting the trusted verifier sign the list of modules that should be present after the transaction and verifying that in the Post functions. For `validatePostExecutorTransaction()` this is relatively straightforward. For the `validatePostTransaction` functions, see the issue "Changes between signing and execution could yield results that are outside the bounds of the policy" for a potential approach to perform checks.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.1.4 Governance can brick Safes by blocking moderator override

Severity: Medium Risk

Context: AddressProvider#L77-L90, SafeModeratorOverridable#L52-L53

Description: The `SafeModeratorOverridable` contract is a guard that enables additional per-transaction validation for safes via the policy validator accessed via the transaction validator. Unlike its `SafeModerator` counterpart, `SafeModeratorOverridable` is meant to be overridable allowing the safe owners to override and disable it by setting the `guard` back to zero. This is done by:

1. Checking if the attempted safe call is a call to `setGuard(address(0))`.
2. Skipping further transaction validation.

The issue with this system is that the override logic is implemented in the separate `TransactionValidator` contract which is simply called by the safe moderator. The source of truth for the address of the transaction validator is the `AddressProvider` which the `governance` address can change. This allows a malicious `governance` address to effectively take all safes hostage by e.g. changing the "authorized transaction validator" to a contract that blocks all transactions unless a specific "release ransom" is paid.

Recommendation: This should be fixed by moving out the override logic from the `TransactionValidator` into the immutable `SafeModeratorOverridable` contract itself. The override should cause the `SafeModeratorOverridable` contract to skip the retrieval of the authorized transaction validator and simply terminate its call successfully such that the transaction can be executed.

Brahma: Solved in PR 62.

Spearbit: Verified.

5.1.5 Subaccount can be console account

Severity: Medium Risk

Context: WalletRegistry.sol#L49-L55, ExecutorRegistry.sol#L77-L87, PolicyRegistry.sol#L35-L59

Description: Suppose someone sets up a `pseudo subaccount safe` by directly calling `createProxyWithNonce()`, with the same parameters that would be used to create a subAccount. Then he uses this `pseudo subaccount safe` to call `registerWallet()`. This will be possible because `subAccountToWallet[]` hasn't been filled for this `pseudo subaccount safe`. Next he removes the `pseudo subaccount safe` via `selfdestruct`.

After this he creates a subaccount via `deploySubAccount()`, which result in the same address. The result is that the subaccount is also a console account.

```
function registerSubAccount(address _wallet, address _subAccount) external {
    if (msg.sender != AddressProviderService._getAuthorizedAddress(_SAFE_DEPLOYER_HASH)) revert
↪   InvalidSender();
    if (subAccountToWallet[_subAccount] != address(0)) revert AlreadyRegistered();
    subAccountToWallet[_subAccount] = _wallet;
    walletToSubAccountList[_wallet].push(_subAccount);
    emit RegisterSubAccount(_wallet, _subAccount);
}
```

This will make other functions work in an unexected way. For example `_validateMsgSenderConsoleAccount()` will return `true` and allow a subaccount to do `registerExecutor()`.

```
function _validateMsgSenderConsoleAccount(address _account) internal view {
    // ...
    // msg.sender is console account
    if (msg.sender == _account && _walletRegistry.isWallet(msg.sender)) return;
    // ...
}
```

Also `updatePolicy()` will allow the subaccount to update a policy.

```
function updatePolicy(address account, bytes32 policyCommit) external {
    // ...
    } else if (msg.sender == account && walletRegistry.isWallet(account)) {
        // In case invoker is a registered wallet
    } else {
        revert UnauthorizedPolicyUpdate();
    }
    // solhint-enable no-empty-blocks
    _updatePolicy(account, policyCommit, currentCommit);
}
```

Recommendation: Consider checking the `_subAccount` isn't registered as a wallet in `registerSubAccount()`.

```
function registerSubAccount(address _wallet, address _subAccount) external {
    // ...
+   if (isWallet[_subAccount]) revert AlreadyRegistered();
    // ...
}
```

Brahma: Solved in PR 56.

Spearbit: Verified.

5.1.6 `registerWallet()` doesn't verify the wallet is a real `Safe`

Severity: Medium Risk

Context: WalletRegistry.sol#L35-L40

Description: A wallet can register itself via `registerWallet()`. However no check is done this wallet is actually a `Safe`. Also no check is done this wallet has a supperted version. As the Brahma logic uses low level details of the `Safe` not all versions are supported, for example the future 1.5.x versions will not automatically be supported. This could potentially leave subaccounts, that are derived from the wallet, unprotected.

```
function registerWallet() external {
    if (isWallet[msg.sender]) revert AlreadyRegistered();
    if (subAccountToWallet[msg.sender] != address(0)) revert IsSubAccount();
    isWallet[msg.sender] = true;
    emit RegisterWallet(msg.sender);
}
```

> **Note** The risk would be for the safe that registers itself. However if a user (accidentally) registers a not supported safe (very old or very new), unexpected issues could occur. The probability of the combination that a not supported safe is registired and also unrecoverable issues occur is low. However if it would happen then the impact for the safe would be high as the funds might be lost or inaccessible. Therefore we set the severity to medium.

Recommendation: Verify that the wallet is indeed a `Safe` wallet and has a supported version. Here is a piece of to show how this could work. Also see verify-safe-creation and Functionality to get masterCopy from a Proxy instance externally.

```
contract ISafeProxy {
    function masterCopy() public returns (address) {}
}
function isLegitimateSafe(address p) public returns(bool){
    bytes32 hash = address(p).codehash;
    // check with the codehash of proxies generated by
↪   https://github.com/safe-global/safe-deployments/tree/main/src/assets
    address singleton = ISafeProxy(p).masterCopy();
    // check with the singleton addresses of `Safe` / `SafeL2` of the appropriate chain / versions
    // which can be found here: https://github.com/safe-global/safe-deployments/tree/main/src/assets
}
```

> **Note** The function `proxyRuntimeCode()`, which is present in older versions of the `SafeProxyFactory`, would be useful to deterimine the hash for the proxy. However it has been removed, see CHANGELOG.

Brahma: This would increase gas usage for our users while decreasing the flexibility in adding safes. In addition to it, the issue would be experienced by users who choose to interact with contracts directly on chain and bypass our UI. So we wont be fixing this.

Spearbit: Acknowledged.

5.2 Low Risk

5.2.1 Upgrade of `trustedValidator` could circumvent Policy checks

Severity: Low Risk

Context: PolicyValidator.sol#L97-L143, SignatureCheckerLib.sol#L32-L115

Description: If the `trustedValidator` is a smart contract, then its called via `isValidSignature`. If it is a contract it could be upgraded via:

- `_getAuthorizedAddress(_TRUSTED_VALIDATOR_HASH)`.
- Or, if it is deployed via a proxy, the proxy could be upgraded.

If there would be a bug in the `trustedValidator` contract or it is maliciously upgraded then Policy checks could be circumvented.

```
function isPolicySignatureValid(address account, bytes32 executionStructHash, bytes calldata
↪  signatures) ... {
    return SignatureCheckerLib.isValidSignatureNow(trustedValidator, txnValidityDigest,
↪  validatorSignature);
}
function isValidSignatureNow(address signer, bytes32 hash, bytes memory signature) ... {
    // ...
    mstore(m, f) // `bytes4(keccak256("isValidSignature(bytes32,bytes)"))`.
    // ...
    staticcall(
        gas(), // Remaining gas.
        signer, // The `signer` address.
        m, // Offset of calldata in memory.
        add(returndatasize(), 0x44), // Length of calldata in memory.
        d, // Offset of returndata.
        0x20 // Length of returndata to write.
    )
   // ...
}
```

Recommendation: Have a way where an explicit Safe operation (signed by owner) is required to upgrade to a new `trustedValidator`. If its a contract then don't use a proxy.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.2.2 `Console` can brick its sub accounts in some scenarios if it removes itself as a module

Severity: Low Risk

Context: SafeDeployer.sol#L176-L182, TransactionValidator.sol#L212-L213

Description: If the `console` detaches itself as a module from its sub account that sub account will be bricked since the `console` is still attached to the subaccount in the wallet registry and the sub account guard has a post tx check for this requirement (the same check exists for execution plugin module):

```
function _checkSubAccountSecurityConfig(address _subAccount) internal view {
    // ...
    address ownerConsole =
        WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH)).subAccountToWallet(_
↪  subAccount);

    // Ensure owner console as a module has not been disabled
    if (!IGnosisSafe(_subAccount).isModuleEnabled(ownerConsole)) revert InvalidModule();
}
```

And so all submitted transaction to the sub account will be reverted with `InvalidModule()`.

Recommendation: Therefore, the `console` first needs to remove or modify the guard for the sub account and then remove itself as a module. Perhaps a safe tx builder for this scenario can be added to `ConsoleOpBuilder` and the above scenario can be documented.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.2.3 Anyone can deploy a `console` with the same set of parameters but with a different `_policyCommit`

Severity: Low Risk

Context: SafeDeployer.sol#L65-L69, PolicyRegistry.sol#L45-L56

Description: Anyone can set the first `_policyCommit` for a `console` by deploying it first with the desired parameters as what the original `owners` of the `safe` want.

This is due to the fact that the `PolicyRegistry` allows the `SafeDeployer` to provide non-zero `_policyCommit` for the `console` when the already stored value for this parameter is `bytes32(0)`:

```
function updatePolicy(address account, bytes32 policyCommit) external {
    // ...
    if (
        currentCommit == bytes32(0)
            && msg.sender == AddressProviderService._getAuthorizedAddress(_SAFE_DEPLOYER_HASH)
    ) {
        // In case invoker is safe  deployer
    } // ...
}
```

Recommendation: Either:

- Warning comment needs to be added for the devs/users that: *the owners need to check and change this policy before using the deployed `console`.* Or...
- The initial `_policyCommit` needs to be signed by enough number of owners (or all) and then this requirement checked during the deployment of the `console`.

Also note that the entity calling `SafeDeployer` can extend the set of `owners` (in some cases minimise and also change `threshold`), which in nefarious scenarios lots of malicious owners can be added so that together they could submit transactions to the `console` and pass the threshold requirement.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.2.4 Changes between signing and execution could yield results that are outside the bounds of the policy

Severity: Low Risk

Context: TransactionValidator.sol

Description: In theory things could have changed between the signing by the `trusted validator` and the execution of the transaction. For example code that depends on a timestamp. Or a transaction that has justed executed before this one (frontrunning or sandwiching) and has influenced for the value of a token. Or a transaction which is executed via a module without updating the Safe nonce. This way the transaction could yield results that are outside the bounds of the policy. **Note:** as the project informed us the current polices are based on outflows of tokens, which well not be influenced by this risk:

- The policy limits the amount of tokens send.
- If too few tokens are available the transaction reverts.

Recommendation: A potential way to reduce the risk would be to check token balances and token allowances in the Post checks: `validatePostTransaction()`, `validatePostTransactionOverridable()`, `validatePostExecutorTransaction()`.

Because the functions `validatePostTransaction()` and `validatePostTransactionOverridable()` only receive the `TransactionHash`, the transaction parameters have to be stored somewhere to allow the function to retrieve them with the help of the `TransactionHash`. This should be done before the call to `execTransaction()`.

In the case of `validatePostExecutorTransaction()`, the function `executeTransaction()` could be enhanced to supply more information.

Brahma: Acknowledged, within the trust scope of policy definitions.

Spearbit: Acknowledged.

5.3 Gas Optimization

5.3.1 Function `updatePolicy()` can be optimized

Severity: Gas Optimization

Context: PolicyRegistry.sol#L35-L59

Description: Accessing `msg.sender` is cheaper than accessing a memory variable. So if a variable is certain the to equal to `msg.sender` it can be replaced by `msg.sender`.

```
function updatePolicy(address account, bytes32 policyCommit) external {
    // ...
    if (msg.sender == account && walletRegistry.isWallet(account)) {
        // ...
    }
}
```

Recommendation: Consider changing the code to:

```
- if (msg.sender == account && walletRegistry.isWallet(account)) {
+ if (msg.sender == account && walletRegistry.isWallet(msg.sender)) {
```

Brahma: Solved in PR 59.

Spearbit: Verified.

5.3.2 Unnecessary custom getters

Severity: Gas Optimization

Context: AddressProvider.sol#L112-L123, WalletRegistry.sol#L63-L65

Description: `authorizedAddresses` and `registries` are already `public` storage variable so custom getter functions are not necessary.

```
contract AddressProvider {
    mapping(bytes32 => address) public authorizedAddresses;
    mapping(bytes32 => address) public registries;

    function getAuthorizedAddress(bytes32 _key) external view returns (address) {
        return authorizedAddresses[_key];
    }
    function getRegistry(bytes32 _key) external view returns (address) {
        return registries[_key];
    }
// ...
```

`walletToSubAccountList` is already a `public` storage variable so custom getter functions are not necessary. The getter function created would be of the form:

```
/// YUL
function getter_fun_walletToSubAccountList_ID(wallet, subAccountIndex) -> subAccount
```

The function `getSubAccountsForWallet()` does have the added advantage that it retrieves an entire list, but this also contains the risk of an out of gas error if the list grows too long.

```
contract WalletRegistry is AddressProviderService {
    mapping(address wallet => address[] subAccountList) public walletToSubAccountList;
    function getSubAccountsForWallet(address _wallet) external view returns (address[] memory) {
        return walletToSubAccountList[_wallet];
    }
}
```

Recommendation: Consider using the automatically generated getter functions. For mappings with values of array type one can add a getter function for the array lengths, retrieve the inner values by length and the automatically generated getter functions or one can also introduce a paginated getter function so that out of gas errors would be prevented when the size of arrays grow.

If the custom getter function are needed, one can avoid the compiler creating the automatically created getter functions by changing the visibility of these variable declarations to `internal`.

Brahma: Solved in PR 63 by making the variables internal.

Spearbit: Verified.

5.3.3 Use of storage for immutable registry addresses in `AddressProvider`

Severity: Gas Optimization

Context: AddressProvider.sol#L97-L105

Description: The `AddressProvider` manages two separate key-value stores. One is the mutable authorized address store and the address the registry store (`mapping(bytes32 => address) public registries`). Unlike authorized addresses however the registry values in the registry store are immutable, once set they cannot be changed:

```
_ensureAddressProvider(_registry);

if (registries[_key] != address(0)) revert RegistryAlreadyExists();
registries[_key] = _registry;
```

Note: that `_ensureAddressProvider` calls the `_registry` address and checks a return value implicitly verifying that the `_registry` address is not zero.

This means that using storage for these addresses is quite inefficient because the consumers of the `AddressProvider` contract have to trigger a storage read and have to do an added external call, this is an unnecessary expense considering the addresses are essentially immutable.

Recommendation: To avoid the external call and storage load these immutable addresses should be configured as constants. This can be done in two ways:

1. Configure the deployment such that the immutable registries are deployed using a deterministic deployment procedure allowing you to predict and store the addresses of the contracts in advance. The recommended way to do this is via `CREATE2` factories as they remove the dependency on deploy wallet nonces and allow you to know the addresses in advance. You can then hardcode them into a library/contract similar to the existing `src/core/Constants.sol` contract and share them with the dependent contracts.
2. Alternatively, you could define `immutable` constants in the dependent contracts and individually pass to the registry addresses as constructor arguments.

Brahma: Solved in PR 64.

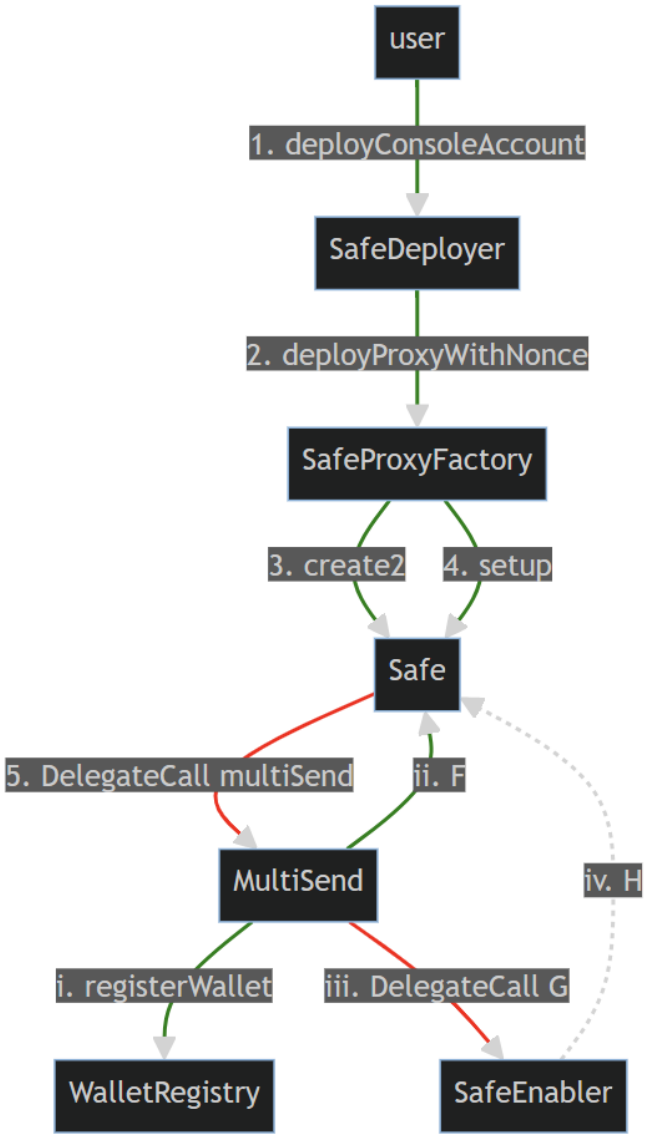
Spearbit: Verified.

5.3.4 Unneeded helper contract `SafeEnabler`

Severity: Gas Optimization

Context: SafeEnabler.sol, SafeDeployer.sol#L123-L160, SafeDeployer.sol#L172-L205

Description: Currently in the `SafeDeployer` the initializer call for newly deployed safes is aimed at the standard multi-send, the sub-calls for which are the actual initializing calls. Some of the calls are delegate-calls to the `SafeEnabler` helper contract which copies the storage layout and is used to directly add modules and guards.



Note In the diagram `iv. H` doesn't represent a function call but merely a storage update.

However, a separate contract is not only not required for this but adds gas overhead because of the separate cold address access and authorized address retrieval that needs to be done for this contract.

The methods contained within `SafeEnabler` are almost full 1:1 copies of the original Safe logic and can just be called directly. Beyond added complexity and gas inefficiency, this approach has the added downside of increasing

attack surface. Mimicking the storage layout of another contract is prone to subtle, difficult-to-detect errors that can have critical impacts.

Recommendation: Fork Safe's `MultiSend` contract making a small modification to the main `multiSend` function to allow implicitly specifying your safe's address by setting the recipient as `address(0)`:

```
   // We offset the load address by 1 byte (operation byte)
   // We shift it right by 96 bits (256 - 160 [20 address bytes]) to right-align the data and zero out
↪  unused data.
   let to := shr(0x60, mload(add(transactions, add(i, 0x01))))
+  // Defaults `to` to `address(this)` if `address(0)` is provided.
+  to := or(to, mul(iszero(to), address()))
   // We offset the load address by 21 byte (operation byte + 20 address bytes)
   let value := mload(add(transactions, add(i, 0x15)))
   // We offset the load address by 53 byte (operation byte + 20 address bytes + 32 value bytes)
```

The expression uses a binary-OR combined with a multiplication to implement a branch-less version of the following statement:

```
to = to == address(0) ? address(this) : to;
```

Here's the reasoning for why `or(to, mul(iszero(to), address())` is equivalent to the above ternary expression:

```
if (to == 0) {
    to = 0
    iszero(to) = 1
    mul(address(), iszero(to)) = mul(address(), 1) = address()
    or(to, mul(address(), iszero(to)) = or(0, address()) = address()
} else /* to != 0 */ {
    to = [1, 2^160-1]
    iszero(to) = 0
    mul(address(), iszero(to)) = mul(address(), 0) = 0
    or(to, mul(address(), iszero(to)) = or(to, 0) = to
}
```

You can then remove the `SafeEnabler` contract and change the `to` target in the `SafeDeployer` contract to `address(0)`.

We have created a Safe Contracts PR 695 for the Safe protocol to implement.

Brahma: This has been a very helpful discussion and we hope Safe Wallet thinks about enabling this usecase. For the time being, we chose to use `SafeEnabler` as is.

Spearbit: Acknowledged.

5.3.5 Access `msg.sender` within the function

Severity: Gas Optimization

Context: SafeDeployer.sol#L95

Description: Using `msg.sender` internally within the `_setupSubAccount` function rather than passing it as an argument can save gas.

```
_subAcc = _createSafe(_owners, _setupSubAccount(_owners, _threshold, msg.sender), _salt);
```

Recommendation: Consider accessing `msg.sender` directly within a function.

Brahma: Solved in PR 59.

Spearbit: Verified.

5.3.6 Location of `_genNonce()` can be optimized

Severity: Gas Optimization

Context: SafeDeployer.sol#L219-L246

Description: The function `_createSafe()` has two instances of the call to `_genNonce()`. They can be combined to save some deployment gas.

```
function _createSafe(address[] calldata _owners, bytes memory _initializer, bytes32 _salt) ... {
    // ...
    uint256 nonce = _genNonce(ownersHash, _salt);
    do {
        try ...createProxyWithNonce(...) returns (...) {
            _safe = ...
        } catch Error(string memory reason) {
            // ...
            nonce = _genNonce(ownersHash, _salt);
            // ...
        }
    } while (_safe == address(0));
}
```

Recommendation: Consider changing the code to:

```
function _createSafe(address[] calldata _owners, bytes memory _initializer, bytes32 _salt) ... {
    // ...
-   uint256 nonce = _genNonce(ownersHash, _salt);
    do {
+       uint256 nonce = _genNonce(ownersHash, _salt);
        try ...createProxyWithNonce(...) returns (...) {
            _safe = // ...
        } catch Error(string memory reason) {
            // ...
-           nonce = _genNonce(ownersHash, _salt);
            // ...
        }
    } while (_safe == address(0));
}
```

```
testSubAccountDeploymentFrontrun(uint256) (gas: 8 (0.000%))
testConsoleAccountDeploymentFrontrun(uint256) (gas: -16 (-0.000%))
testExecuteValidTxn() (gas: 3 (0.001%))
testDeploySubAccount_Successful() (gas: 3 (0.001%))
testKillAllSubAccounts() (gas: 6 (0.001%))
testKillSubAccounts() (gas: 6 (0.001%))
testDeployConsoleAccount_WithPolicyCommit() (gas: 3 (0.001%))
testDeployConsoleAccount_WithoutPolicyCommit() (gas: 3 (0.001%))
testFailExecuteInvalidTxn() (gas: 3 (0.001%))
testRevertDeployConsoleAccount_Create2CallFailed() (gas: 3 (0.001%))
testRevertDeployConsoleAccount_OwnerZeroAddress() (gas: 3 (0.002%))
testMultichainDeployment() (gas: -8804 (-0.017%))
testDeploy() (gas: -2207 (-0.150%))
testEnableExecutorMulticall(uint160) (gas: -161076 (-2.757%))
testDisableExecutorMulticall(uint160) (gas: -2230659 (-38.208%))
Overall gas change: -2402721 (-0.000%)
```

Brahma: Solved in PR 59.

Spearbit: Verified.

5.3.7 Function `isPolicySignatureValid()` can be optimized

Severity: Gas Optimization

Context: PolicyValidator.sol#L97-L143

Description: The function `isPolicySignatureValid()` uses assembly which can be replaced with Solidity. Additionally the check for the code length can only be done only when necessary to save some gas.

```
function isPolicySignatureValid(...) ... {
    // ...
    assembly {
        _codesize := extcodesize(trustedValidator)
    }
    if (_codesize == 0 && validatorSignature.length == 0) {
        revert ...
    }
}
```

Recommendation: Consider changing the code to:

```
function isPolicySignatureValid(...) ... {
    // ...
-    uint256 _codesize;
-    assembly {
-        _codesize := extcodesize(trustedValidator)
-    }
-   if (_codesize == 0 && validatorSignature.length == 0) {
+   if (validatorSignature.length == 0 && trustedValidator.code.length == 0) {
        revert ...
    }
}
```

```
testSubAccountDeploymentFrontrun(uint256) (gas: -30 (-0.000%))
testValidateSafe_RevertInvalidSignature() (gas: 6 (0.005%))
testValidateSafe_ReturnTrue_ValidTrustedValidatorContractSignature() (gas: 15 (0.009%))
testValidateSafe_ReturnFalse_InvalidTrustedValidatorContractSignature() (gas: 15 (0.013%))
testMultichainDeployment() (gas: 18392 (0.036%))
testValidateModule_ReturnFalseInvalidContractSignature() (gas: -99 (-0.066%))
testValidateModule_ReturnTrueValidContractSignature() (gas: -99 (-0.066%))
testValidateSafe_ReturnFalseInvalidContractSignature() (gas: -99 (-0.068%))
testValidateSafe_ReturnTrueValidContractSignature() (gas: -99 (-0.068%))
testExecuteTransactionSubSafe_ShouldRevertTxnUnAuthorized() (gas: -220 (-0.068%))
testExecuteTransactionMainSafe_ShouldRevertTxnUnAuthorized() (gas: -220 (-0.073%))
testValidateModule_ReturnFalseInvalidEOASignature() (gas: -110 (-0.080%))
testValidateSafe_ReturnFalseInvalidEOASignature() (gas: -110 (-0.082%))
testCheckTransactionOverridable_ShouldRevertInvalidValidator() (gas: -110 (-0.103%))
testCheckTransactionOverridable_ShouldRevertInvalidCommit() (gas: -110 (-0.103%))
testCheckTransaction_ShouldRevertInvalidValidator() (gas: -110 (-0.103%))
testCheckTransaction_ShouldRevertInvalidCommit() (gas: -110 (-0.103%))
testIsValidSignature_ShouldRevert_InvalidPolicy_PreSign_NotTrustedValidator() (gas: -110 (-0.112%))
testIsValidSignature_ShouldReturnMagicValue_InvalidPolicy_ECDSAEIP191() (gas: -110 (-0.145%))
testIsValidSignature_ShouldRevert_InvalidPolicy_ECDSA() (gas: -110 (-0.146%))
testIsValidSignature_ShouldRevert_InvalidPolicy_ContractSignature() (gas: -110 (-0.149%))
testValidateSafe_ReturnTrue_ValidTrustedValidatorContractSignature_ValidExecutorContractSignature()
↪  (gas: -2595 (-0.688%))
testValidateSafe_ReturnTrue_ValidTrustedValidatorContractSignature_ValidExecutorEOA() (gas: -2709
↪  (-0.691%))
testExecuteTransaction_ShouldRevertSigReplay() (gas: -2720 (-0.697%))
testCheckTransactionExecutor_Valid() (gas: -2720 (-0.725%))
testExecuteTransactionSubSafe_SuccessfulContractSigner() (gas: -2720 (-0.730%))
testExecuteTransactionSubSafe_SuccessfulEOASigner() (gas: -2720 (-0.732%))
```

```
testExecuteTransactionMainSafe_SuccessfulContractSigner() (gas: -2720 (-0.772%))
testValidateSafe_ReturnFalse_InvalidTrustedValidatorContractSignature_ValidExecutorEOA() (gas: -2595
↪  (-0.774%))
testExecuteTransactionMainSafe_SuccessfulEOASigner() (gas: -2720 (-0.774%))
testCheckAfterExecutor_ShouldRevertInvalidFallbackHandler() (gas: -2720 (-0.778%))
testCheckAfterExecutor_ShouldRevertInvalidModuleWhenMainSafeRemoved() (gas: -2720 (-0.781%))
testExecuteTransactionSubSafe_WhenMainSafeRemoved_ShouldRevertInvalidModule() (gas: -2720 (-0.781%))
testExecuteTransactionSubSafe_WhenFallbackHandlerRemoved_ShouldRevertInvalidGuard() (gas: -2720
↪  (-0.793%))
testCheckAfterExecutor_ShouldRevertInvalidGuardWhenTransactionGuardRemoved() (gas: -2720 (-0.795%))
testExecuteTransactionSubSafe_WhenTransactionGuardRemoved_ShouldRevertInvalidGuard() (gas: -2720
↪  (-0.795%))
testValidateSafe_ReturnFalse_ValidTrustedValidatorContract_InvalidExecutorContract() (gas: -2610
↪  (-0.818%))
testExecuteTransactionMainSafe_WhenFallbackHandlerRemoved_ShouldRevertInvalidGuard() (gas: -2720
↪  (-0.833%))
testExecuteTransactionMainSafe_WhenTransactionGuardRemoved_ShouldRevertInvalidGuard() (gas: -2720
↪  (-0.842%))
testExecuteTransactionSubSafe_ShouldRevertInvalidExecutorSignature() (gas: -2610 (-0.848%))
testExecuteTransactionSubSafe_WhenExecPluginRemoved_ShouldRevertInvalidGuard() (gas: -2720 (-0.868%))
testExecuteTransactionMainSafe_ShouldRevertInvalidContractSigner() (gas: -2610 (-0.870%))
testExecuteTransactionSubSafe_ShouldRevertInvalidContractSigner() (gas: -2610 (-0.906%))
testExecuteTransactionMainSafe_ShouldRevertInvalidExecutorSignature() (gas: -2610 (-0.907%))
testExecuteTransactionMainSafe_WhenExecPluginRemoved_ShouldRevertInvalidGuard() (gas: -2720 (-0.927%))
testExecuteTransactionSafe_ShouldRevertInvalidSignature() (gas: -2610 (-0.933%))
testExtractExecutorSignature_WithoutExecutorSig1271() (gas: -2610 (-0.948%))
testExecuteTransaction_ShouldRevertWhenContractSignerInvalid() (gas: -2610 (-0.994%))
testExecuteTransactionMainSafe_ShouldRevertInvalidSignature() (gas: -2610 (-1.004%))
testDisablePolicyMulticall() (gas: -2610 (-1.178%))
testExecuteTransactionSubSafe_ShouldRevertModuleExecutionFailed() (gas: -2610 (-1.267%))
testExecuteTransactionMainSafe_ShouldRevertModuleExecutionFailed() (gas: -2610 (-1.280%))
testCheckTransaction_Valid() (gas: -2610 (-1.519%))
testCheckTransactionOverridable_Valid() (gas: -2610 (-1.770%))
testValidateTransactionForConsoleGuard() (gas: -2610 (-1.770%))
testCheckAfterExecution_ShouldRevertModuleRemoval() (gas: -2610 (-1.797%))
testEnableExecutorMulticall(uint160) (gas: 106748 (1.827%))
testValidateModule_ReturnTrueValidEOASignature() (gas: -2610 (-1.902%))
testValidateSafe_ReturnTrueValidEOASignature() (gas: -2610 (-1.952%))
testCheckAfterExecution_ShouldRevertFallbackHandlerRemoval() (gas: -2610 (-2.053%))
testCheckAfterExecution_ShouldRevertGuardRemoval() (gas: -2610 (-2.105%))
testIsValidSignature_ShouldReturnMagicValue_ValidPolicy_PreSign() (gas: -2610 (-2.525%))
testIsValidSignature_ShouldReturnMagicValue_ValidPolicy_InvalidECDSAEIP191() (gas: -2610 (-3.007%))
testIsValidSignature_ShouldReturnMagicValue_ValidPolicy_ECDSAEIP191() (gas: -2610 (-3.016%))
testIsValidSignature_ShouldRevert_ValidPolicy_InvalidECDSA() (gas: -2610 (-3.023%))
testIsValidSignature_ShouldReturnMagicValue_ValidPolicy_ECDSA() (gas: -2610 (-3.033%))
testIsValidSignature_ShouldRevert_ValidContractSignature_InvalidOwner() (gas: -2610 (-3.109%))
testIsValidSignature_ShouldRevert_InvalidContractSignature() (gas: -2610 (-3.289%))
testIsValidSignature_ShouldRevert_ValidPolicy_PreSign_HashNotApproved() (gas: -2610 (-3.431%))
testDisableExecutorMulticall(uint160) (gas: -553317 (-9.477%))
Overall gas change: -554606 (-0.000%)
```

Brahma: Solved in PR 63 by switching the order of the `and` expressions.

Spearbit: Verified.

5.3.8 Check for lacking policies can be done earlier

Severity: Gas Optimization

Context: ExecutorPlugin.sol#L74

Description: Suppose a console account does not have any policies. The console account allows an executor to make module transactions on behalf of his console account. However, without any policies, the hook (`validatePostExecutorTransaction` → `_checkConsoleAccountSecurityConfig`) will always revert whenever the executor tries to execute a transaction as the `guard` and/or `fallbackHandler` are not initialized.

```
function _checkConsoleAccountSecurityConfig(address _consoleAccount) internal view {
    address guard = SafeHelper._getGuard(_consoleAccount);
    address fallbackHandler = SafeHelper._getFallbackHandler(_consoleAccount);

    // Ensure guard has not been disabled
    if (guard != AddressProviderService._getAuthorizedAddress(_SAFE_MODERATOR_OVERRIDABLE_HASH)) {
        revert InvalidGuard();
    }

    // Ensure fallback handler has not been altered
    if (fallbackHandler !=
↪   AddressProviderService._getAuthorizedAddress(_CONSOLE_FALLBACK_HANDLER_HASH)) {
        revert InvalidFallbackHandler();
    }
}
```

Recommendation: If the intention is to prevent console accounts from enabling the executor plugin without having a guard set and policy enabled, it is recommended to explicitly block it early (e.g. either on `validatePreExecutorTransaction()` or even on `registerExecutor()`) instead of blocking/catching it via a revert due to an error.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.3.9 Addresses retrieved twice in `executeTransaction()` and `validatePostExecutorTransaction()`

Severity: Gas Optimization

Context: ExecutorPlugin.sol#L68-L77, ExecutorPlugin.sol#L106-L152, TransactionValidator.sol#L134-L150, TransactionValidator.sol#L197-L214

Description: Both the functions `executeTransaction()` and `_validateExecutionRequest()` retrieve the `TransactionValidator` address. This could be combined and the address could be supplied to `_validateExecutionRequest()`, to save some gas.

```
function executeTransaction(ExecutionRequest calldata execRequest) external nonReentrant returns (bytes
↪   memory) {
    _validateExecutionRequest(execRequest);
    // ...
    TransactionValidator(AddressProviderService._getAuthorizedAddress(_TRANSACTION_VALIDATOR_HASH))
        .validatePostExecutorTransaction(...);
    return txnResult;
}
function _validateExecutionRequest(ExecutionRequest calldata execRequest) internal {
    // ...
    TransactionValidator(AddressProviderService._getAuthorizedAddress(_TRANSACTION_VALIDATOR_HASH))
        .validatePreExecutorTransaction(...);

}
```

Both functions `validatePostExecutorTransaction()` and `_checkSubAccountSecurityConfig()` retrieve the `WalletRegistry` address. This could be combined and the address could be supplied to `_checkSubAccountSecurityConfig()`, to save some gas.

```
function validatePostExecutorTransaction(address, /*msgSender */ address account) external view {
    WalletRegistry _walletRegistry =
↪   WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH));
     // ...
     _checkSubAccountSecurityConfig(account);
     // ...
}
function _checkSubAccountSecurityConfig(address _subAccount) internal view {
    // ...
    address ownerConsole = WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH))
        .subAccountToWallet(_subAccount);
    // ...
}
function validatePostTransaction(bytes32, /*txHash */ bool, /*success */ address subAccount) external
↪   view {
        _checkSubAccountSecurityConfig(subAccount); // also has to be adapted
}
```

> **Warning** In case the value of `_getAuthorizedAddress(...)` would be updated as a side effect of one of the function calls the results wouldn't be identical. However shouldn't happen. Also see issue Governance can remove policy check due to upgradability.

Recommendation: Consided retrieving the addresses only once and supplying them as a parameter to the called functions. **Note:** function `validatePostTransaction()` also calls `_checkSubAccountSecurityConfig()`, so should also be adapted.

Brahma: Solved in PR 49 and PR 66.

Spearbit: Verified.

5.3.10 `keccak256` result can be cached

Severity: Gas Optimization

Context: TransactionValidator.sol#L182

Description: The current implementation redundantly recalculates `keccak256(_data)` on each execution.

```
if (SafeHelper._GUARD_REMOVAL_CALLDATA_HASH == keccak256(_data)) {
    return true;
} else if (SafeHelper._FALLBACK_REMOVAL_CALLDATA_HASH == keccak256(_data)) {
```

Recommendation: Consider caching the result of `keccak256(_data)`.

Brahma: Solved in PR 62.

Spearbit: Verified.

5.3.11 Function `updatePolicy` can be optimized

Severity: Gas Optimization

Context: PolicyRegistry.sol#L35-L59

Description: Within function `updatePolicy()`, the retrieval of the `walletRegistry` could be moved inside the `else` statement, to only retrieve it when necessary. This saves some gas.

```
function updatePolicy(address account, bytes32 policyCommit) external {
    // ...
    WalletRegistry walletRegistry =
↪   WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH));
    // ...
    if (currentCommit == bytes32(0) && msg.sender ==  ... ) {
    } else if (walletRegistry.subAccountToWallet(account) == msg.sender) {
    } else if (msg.sender == account && walletRegistry.isWallet(account)) {
    } else {
        revert UnauthorizedPolicyUpdate();
    }
    // ...
}
```

Recommendation: Consider changing the code to:

```
function updatePolicy(address account, bytes32 policyCommit) external {
    // ...
-   WalletRegistry walletRegistry =
↪   WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH));
    // ...
    if (currentCommit == bytes32(0) && msg.sender ==  ... ) {
    } else
+   {
+       WalletRegistry walletRegistry =
↪   WalletRegistry(AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH));
        if (walletRegistry.subAccountToWallet(account) == msg.sender) {
        } else if (msg.sender == account && walletRegistry.isWallet(account)) {
        } else {
            revert UnauthorizedPolicyUpdate();
        }
+   }
    // ...
}
```

```
testSubAccountDeploymentFrontrun(uint256) (gas: -996 (-0.000%))
testDisablePolicyMulticall() (gas: -2 (-0.001%))
testEnablePolicyMulticall() (gas: -2 (-0.001%))
testUpdatePolicy_MainSafeToMainSafe() (gas: -2 (-0.001%))
testUpdatePolicy_MainSafeToSubSafe() (gas: -2 (-0.001%))
testValidateModule_ReturnFalseInvalidContractSignature() (gas: -2 (-0.001%))
testValidateModule_ReturnTrueValidContractSignature() (gas: -2 (-0.001%))
testValidateSafe_ReturnFalseInvalidContractSignature() (gas: -2 (-0.001%))
testValidateSafe_ReturnTrueValidContractSignature() (gas: -2 (-0.001%))
testValidateModule_ReturnFalseInvalidEOASignature() (gas: -2 (-0.001%))
testValidateModule_ReturnTrueValidEOASignature() (gas: -2 (-0.001%))
testValidateSafe_ReturnFalseInvalidEOASignature() (gas: -2 (-0.001%))
testValidateSafe_ReturnTrueValidEOASignature() (gas: -2 (-0.001%))
testValidateSafe_RevertInvalidLength() (gas: -2 (-0.002%))
testValidateModule_ShouldRevertWhenExpired() (gas: -2 (-0.002%))
testValidateSafe_ShouldRevertWhenExpired() (gas: -2 (-0.002%))
testValidateSafe_RevertInvalidSignature() (gas: -2 (-0.002%))
testUpdatePolicy_ShouldRevertArbitrarySenderToSubAccount(uint256) (gas: -3 (-0.002%))
testUpdatePolicy_ShouldRevertArbitrarySenderToMainSafe(uint256) (gas: -3 (-0.002%))
testUpdatePolicy_ShouldRevertSubAccountToSubAccount() (gas: -3 (-0.002%))
testUpdatePolicy_ShouldRevertSubAccountToMainSafe() (gas: -3 (-0.002%))
testUpdatePolicy_MainSafeCanOverwrite() (gas: 7 (0.004%))
testUpdatePolicy_ShouldRevertSafeDeployerNonInit() (gas: 12 (0.006%))
testMultichainDeployment() (gas: -3984 (-0.008%))
testRevert_OneRevertRevertsEntireMultiCall() (gas: 8 (0.008%))
testDeploySubAccount_Successful() (gas: -996 (-0.174%))
testKillAllSubAccounts() (gas: -1992 (-0.199%))
testKillSubAccounts() (gas: -1992 (-0.203%))
testDeployConsoleAccount_WithPolicyCommit() (gas: -996 (-0.207%))
testUpdatePolicy_SafeDeployerInitMainSafe() (gas: -3992 (-2.170%))
testDisableExecutorMulticall(uint160) (gas: 506439 (8.674%))
testEnableExecutorMulticall(uint160) (gas: -1836041 (-31.423%))
Overall gas change: -1344567 (-0.000%)
```

Brahma: Solved in PR 59.

Spearbit: Verified.

5.4 Informational

5.4.1 Suggestions for the deployment script `ConsoleFactory`

Severity: Informational

Context: ConsoleFactory.s.sol#L75, ConsoleFactory.s.sol#L166

Description:

- ConsoleFactory.s.sol#L75, in 2 extra addresses are provided to `create3Deploy(...)` since `AddressProvider`'s `constructor` only takes one argument.
- ConsoleFactory.s.sol#L166, logic check for `consoleFallbackHandler` has been skipped since `_overrideCheck` is true:

```
/// @dev skips checks for supported `addressProvider()` if `_overrideCheck` is true
if (!_overrideCheck) {
    /// @dev skips checks for supported `addressProvider()` if `_authorizedAddress` is an EOA
    if (_authorizedAddress.code.length != 0) _ensureAddressProvider(_authorizedAddress);
}
```

Recommendation: The following diff can be applied:

```
testConsoleAccountDeploymentFrontrun(uint256) (gas: -54 (-0.000%))
testSubAccountDeploymentFrontrun(uint256) (gas: -101 (-0.000%))
testMultichainDeployment() (gas: 1738 (0.003%))
testDisableExecutorMulticall(uint160) (gas: -494854 (-8.476%))
testEnableExecutorMulticall(uint160) (gas: -1246445 (-21.332%))
Overall gas change: -1739716 (-0.000%)
```

```
diff --git a/script/utils/ConsoleFactory.s.sol b/script/utils/ConsoleFactory.s.sol
index 093d285..3d98033 100644
--- a/script/utils/ConsoleFactory.s.sol
+++ b/script/utils/ConsoleFactory.s.sol
@@ -72,7 +72,7 @@ contract ConsoleFactory is Constants, ConstantSetup, AddressRegistry {
         }
         deployer = new Deployer{salt:"deployer"}();
         addressProvider = AddressProvider(
-            deployer.create3Deploy(type(AddressProvider).creationCode, addressProviderSalt,
↪  abi.encode(gov, gov, gov))
+            deployer.create3Deploy(type(AddressProvider).creationCode, addressProviderSalt,
↪  abi.encode(gov))
         );

         consoleFallbackHandler = ConsoleFallbackHandler(
@@ -163,7 +163,7 @@ contract ConsoleFactory is Constants, ConstantSetup, AddressRegistry {
         addressProvider.setAuthorizedAddress(_GNOSIS_SINGLETON_HASH, singleton, true);
         addressProvider.setAuthorizedAddress(_GNOSIS_MULTI_SEND_HASH, gnosisMultisend, true);
         addressProvider.setAuthorizedAddress(_GNOSIS_FALLBACK_HANDLER_HASH, fallbackHandler, true);
-        addressProvider.setAuthorizedAddress(_CONSOLE_FALLBACK_HANDLER_HASH,
↪  address(consoleFallbackHandler), true);
+        addressProvider.setAuthorizedAddress(_CONSOLE_FALLBACK_HANDLER_HASH,
↪  address(consoleFallbackHandler), false);

         if (broadcast) {
             vm.stopBroadcast();
```

Brahma: Solved in commit ad4274dc.

Spearbit: Verified.

5.4.2 Initialization of hashes in `Constants`

Severity: Informational

Context: Constants.sol#L13-L20

Description: The contract `Constants` contains several constants that are initialized via a hex string. They could also be initialized via a `keccak256` expression, which would make the code easier to maintain and verify. Perhaps this is the result of previous optimizations. With optimization of the SOLC switch on this, it shouldn't make a difference.

Additionally the hashes, which are determined by the Brahma protocol (and thus don't have to match a value in the Safe contracts), can be prefixed by a protocol specific path. This is to prevent that values are accidentally reused.

```
abstract contract Constants {
    // ...
    /// @notice keccak256("ExecutorRegistry")
    bytes32 internal constant _EXECUTOR_REGISTRY_HASH =
        0x165eedff3947ccfbe9739de5f67209b9935e684faef9ce859fb3dc46d33317f1;
    // ...
}
```

Recommendation:

1. Doublecheck the gas usage initializing via a hash and the consider using that.
2. Consider offsetting the Brahma defined hashed by one (e.g. subtract 1) to avoid potential compiler storage collisions
3. Prefix the hashed string with protocol specific path to prevent using the same strings from other protocols:

```
bytes32 internal constant NAME =
    bytes32(uint256(keccak256("PROTOCOL_SPECIFIC_PATH_PREFIX.NAME")) - 1);
```

Brahma: Solved in PR 55.

Spearbit: Verified.

5.4.3 Enum conversion in `_packMultisendTxns()` not obvious

Severity: Informational

Context: SafeHelper.sol#L103-L135, Types.sol#L5-L9, IGnosisSafe.sol#L8-L13

Description: The logic to convert from `Enum Operation` to `Enum CallType` is fragile to future changes in the enum. If any enum variants are added they'd fall through the check and cause the `call` variable to default `0`.

```
contract Enum {
    enum Operation {
        Call,
        DelegateCall
    }
}
interface Types {
    enum CallType {
        CALL,
        DELEGATECALL,
        STATICCALL
    }
}
function _packMultisendTxns(Types.Executable[] memory _txns) internal pure returns (bytes memory
↪  packedTxns) {
    // ...
    uint8 call;  // initially 0 and thus CALL
    if (_txns[i].callType == Types.CallType.DELEGATECALL) {
        call = uint8(Enum.Operation.DelegateCall);
    } else if (_txns[i].callType == Types.CallType.STATICCALL) {
        revert InvalidMultiSendCall(i);
    }
    // ...
}
```

Recommendation: Consider changing the code to:

```
uint8 call = uint8(_txns[i].callType);
if (call == uint8(Types.CallType.STATICCALL)
   revert InvalidMultiSendCall(i);
```

This ensures that future extensions of the enum variant will still be converted to their unique `uint8` representation. Also the suggestion still assumes that the both `Types.CallType` and `Enum.Operation` have the `CALL` and `DELEGATECALL` at the same indexes and no other operations.

Alternatively, each variant could be handled explicitly and individually this would ensure that extensions of the enum would lead to a runtime error.

Brahma: Solved in PR 54.

Spearbit: Verified.

5.4.4 How to recover from a bug in `ExecutorPlugin`

Severity: Informational

Context: ConsoleOpBuilder.sol#L96-L189, TransactionValidator.sol#L134-L150

Description: In case a bug is found in `ExecutorPlugin` it will be difficult to replace this:

- All Safes would have to remove the old `ExecutorPlugin` from the list of modules and install a new one;
- The old `ExecutorPlugin`doesn't work anymore once `AddressProviderService._getAuthorizedAddress(_EXECUTOR_PLUGIN_HASH)` is updated, due to the `isModuleEnabled()` check in `validatePostExecutorTransaction()`, which refers to the new `ExecutorPlugin`.
  - This is good when a bug is present but
  - `disableExecutorPluginOnSubAccount(...)` can't be used anymore

```
function enableExecutorPluginOnSubAccount(address subAccount, address[] memory executors) ... {
    // ...
    ... IGnosisSafe.enableModule, (AddressProviderService._getAuthorizedAddress(_EXECUTOR_PLUGIN_HASH))
↪   ...
    // ...
}
function disableExecutorPluginOnSubAccount(address subAccount, address previousModule) ... {
    // ...
    .. IGnosisSafe.disableModule,previousModule,
↪   AddressProviderService._getAuthorizedAddress(_EXECUTOR_PLUGIN_HASH)) ...
    // ...
}
function validatePostExecutorTransaction(address, /*msgSender */ address account) external view {
    // ...
    // Check if account has executor plugin still enabled as a module on it
    if (!IGnosisSafe(account).isModuleEnabled(AddressProviderService._getAuthorizedAddress(_EXECUTOR_PL
↪   UGIN_HASH)))
↪   {
        revert InvalidExecutorPlugin();
    }
    // ...
}
```

Recommendation: Think about an approach to effective recover from a bug in `ExecutorPlugin`

Brahma: Anything attached to safe is difficult to upgrade. Guards (`SafeModerator`, `SafeModeratorOverridable`) and Modules (`ExecutorPlugin`) need cooperation from user before upgrade if necessary.

Spearbit: Acknowledged.

5.4.5 Re-implementation of Ownership with 2-step transfer pattern

Severity: Informational

Context: AddressProvider.sol#L48-L69

Description: The `AddressProvider` implements the ownership with 2-step ownership transfers pattern, allowing the contract to keep track of an "owner" (labelled `governance` in `AddressProvider`) and only allowing transfers of said ownership via a 2-step process whereby the current owner can only "propose" a new owner which then has to manually accept ownership to confirm the transfer.

The issue is the replication of this code. The repo already has `openzeppelin-contracts` as a dependency which has this exact pattern as an existing, heavily verified and audited implementation under `contracts/access/Ownable2Step.sol`. Besides the naming, they are mostly identical. The only functional difference is that the custom implementation disallows setting the pending owner to `address(0)` meaning that for cancelling pending transfers the owner of `AddressProvider` would need to use `address(1)`. Furthermore the use of string

errors vs. custom errors is also a difference. However, if that is the main deciding factor a newer version of the OpenZeppelin library can be chosen that uses custom errors.

Recommendation: While the implementation is correct it's generally recommended to not rewrite existing patterns from scratch and rely on existing verified and reviewed code whenever possible unless there's an important reason not to. It's recommended to replace the custom ownership with 2-step transfer logic with use of OpenZeppelin's `Ownable2Step` mixin.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.6 Type hash is not aligned with `ValidationParams` struct

Severity: Informational

Context: TypeHashHelper.sol#L56

Description: The `ValidationParams` struct implemented in the codebase is as follows:

```
struct ValidationParams {
     uint32 expiryEpoch;
     bytes32 executionStructHash;
     bytes32 policyHash;
 }
```

However, the type hash is derived from the following `ValidationParams` struct, which is different from the struct used within the codebase:

```
struct ValidationParams {
     uint32 expiryEpoch;
     ExecutionParams executionParams
     bytes32 policyHash;
 }
```

```
/*
 * ...
 * @dev keccak256("ValidationParams(uint32 expiryEpoch,ExecutionParams executionParams,bytes32
↪  policyHash)ExecutionParams(uint8 operation,address to,address account,address executor,uint256
↪  value,uint256 nonce,bytes data)")
 */
bytes32 public constant VALIDATION_PARAMS_TYPEHASH =
    0x68883b91861c8baad1e8d9f6fd31a22216bb9cd1aec71362de9879112a14cde4;
```

Recommendation: Update the type hash to be aligned with the `ValidationParams` struct used within the codebase to ensure EIP712 compatibility.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.7 Trusted validator weakens Safe access control via `ExecutorPlugin`

Severity: Informational

Context: PolicyValidator.sol#L97-L143

Description: The `PolicyValidator` contract validates signatures from the "transaction validator" ("validator" herein) on messages. This is used in the safe moderator contracts (`SafeModerator`, `SafeModeratorOverridable`) and `ExecutorPlugin` contracts to validate attempted transactions externally. For the safe moderators, this simply serves as an extra check to the minimum quorum of safe signers, for the `ExecutorPlugin` this authenticates arbitrary, direct module transactions.

Executed from the `ExecutorPlugin`, if configured as a module can execute anything from asset transfers to other configuration transactions (e.g. removing modules, setting guards, adjusting members etc...). This means that by colluding with the validator, a single executor can take over complete control of the safe, removing event the parent console account's ability to override. This is because the `ExecutorPlugin` has no on-chain limits or checks on what signatures can be executed through it, the validator is fully trusted. This means that when the `ExecutorPlugin` is configured as a module and executors added to a sub-account the authorization required is weakened from

```
[sub-account signer quorum (+ optional guard)] OR [parent console account approval  (+ optional guard)]
```

to

```
[sub-account signer quorum (+ optional guard)] OR [parent console account approval  (+ optional guard)]
↪  OR [[ANY 1 executor] AND [validator]]
```

Recommendation: Acknowledge and clearly document the effect adding the `ExecutorPlugin` to safe has. Additionally, add the ability to explicitly validate and limit the type of transactions the `ExecutorPlugin` can execute on-chain, this will allow the impact of a compromised validator to be limited.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.8 Use builtin functions to compute constant values

Severity: Informational

Context: ConsoleFallbackHandler.sol#L27

Description: ConsoleFallbackHandler.sol#L27, `0x1626ba7e` is:

```
UPDATED_MAGIC_VALUE = bytes4(keccak256("isValidSignature(bytes32,bytes)"))
```

Note that the original `Safe` implementation also does not use builtin functions. A related issue is "Initialization of hashes in `Constants`".

Recommendation: Use builtin functions mentioned above to compute constant values. If this change creates more deployment/runtime gas, at least adding a comment would be recommended.

Brahma: `ConsoleFallbackHandler` is derived strictly from original `Safe` implementation. Acknowledged.

Spearbit: Acknowledged.

5.4.9 Make sure the chain used to deploy the protocol implements the desired precompiles at the correct addresses

Severity: Informational

Context: ExecutorPlugin.sol#L140, PolicyValidator.sol#L142

Description: When validating a signature the project uses `Solday` library to perform the check. The function used is `SignatureCheckerLib.isValidSignatureNow` that has the following assumption for the chain used for deployment:

```
/// @dev Note:
/// - The signature checking functions use the ecrecover precompile (0x1).
/// - The `bytes memory signature` variants use the identity precompile (0x4)
///   to copy memory internally.
```

Recommendation: Before deploying to the desired chains make sure the above requirements for the precompiles are satisfied.

Brahma: Verified for all the target chains, ecrecover precompile exists.

Spearbit: Acknowledged

5.4.10 Best to use `uint64` as a type for `timestamp`

Severity: Informational

Context: PolicyValidator.sol#L160, TypeHashHelper.sol#L40

Description: Currently the timestamps in the codebase use `uint32` as a type. But the max value for this type will be reached within approximately 80 years.

Recommendation: It is best to use `uint64` for timestamps to avoid the above issue.

Brahma: `uint32` has been chosen to save gas. We think 82 years is quite enough for this implementation.

Spearbit: Acknowledged

5.4.11 Incomplete or incorrect comments

Severity: Informational

Context: PolicyValidator.sol#L81-L84, Constants.sol#L49-L51, Constants.sol#L27-L90

Description/Recommendation:

- PolicyValidator.sol#L81-L84, Inconsistent with other comments regarding `signatures`:

```
    /**
     ...
     * @dev signatures = abi.encodePacked(safeSignature, validatorSignature, validatorSignatureLength,
↪  expiryEpoch)
     *  safeSignature = safe owners signatures (arbitrary bytes length)
     *  validatorSignature = EIP 712 digest signature (arbitrary bytes length)
     *  validatorSignatureLength = length of `validatorSignature` (4 bytes)
     *  expiryEpoch = expiry timestamp (4 bytes)
     ...
```

In all call sites the `safeSignature` is also used. Also since these signatures are decoded backwards in general one could have:

```
signatures = abi.encodePacked(x0, x1, ..., xN, validatorSignature, validatorSignatureLength,
↪  expiryEpoch)
```

- Constants.sol#L49-L51, the hashed string does not match with the other patterns instead of just `Gnosis`, `GnosisSafe` is used.

```
/// @notice keccak256("GnosisSingleton")
bytes32 internal constant _GNOSIS_SINGLETON_HASH =
    0xf3ba33e8d2539358a3ce532b82a27f60ffeff80d21521e08296b682cfa51b06b;
```

Also note, the name `Gnosis` itself isn't used anymore either, its all rebranded to just `Safe`.

- Constants.sol#L27-L90, Mention that the `core` and `roles` hashes are used as keys in the mapping of authorised addresses. And that these addresses are not immutable and can be updated by the governance of the `AddressProvider`:

```
/**
 * @notice keccak256 hash of authorizedAddress keys mapped to their addresses
 * @dev authorizedAddresses are updatable by governance
 */
mapping(bytes32 => address) public authorizedAddresses;
```

Brahma: Solved in PR 55.

Spearbit: Verified.

5.4.12 `abi.encodeCall` should be used instead of `abi.encodePacked`

Severity: Informational

Context: SafeDeployer.sol#L144

Description: It is recommended to always use `abi.encodeCall` even if the method has no arguments as `abi.encodeCall` provides type checking:

```
txns[0] = Types.Executable({
    callType: Types.CallType.CALL,
    target: AddressProviderService._getRegistry(_WALLET_REGISTRY_HASH),
    value: 0,
    data: abi.encodePacked(WalletRegistry.registerWallet.selector)
    // ...
```

Recommendation: Consider applying the recommendation shown above.

Brahma: Solved in PR 57.

Spearbit: Verified.

5.4.13 `AccountSecurityConfig` checks could be expanded

Severity: Informational

Context: TransactionValidator.sol#L197-L234

Description: The functions `_checkSubAccountSecurityConfig()` and `_checkConsoleAccountSecurityConfig()` do perform some checks but these checks could be expanded for more security.

Recommendation: Consider checking the `Safe` contract hasn't been updated, e.g. the `singleton` hasn't been changed, or if it is changed, its changed to a valid `singleton`. This can be done by checking the `Safe` is (still) a valid `Safe`.

> **Note** This does increase reliance on Brahma governance. Upgrading to the latest singleton can only be done once its whitelisted. A solution for that might be to have a specific action like "_isConsoleBeingOverridden" to be able to do replace the singleton.
>
> **Note** This might be most effective for SubAccounts, because the singleton is set by `SafeDeployer`. See issue `registerWallet()` doesn't verify the wallet is a real `Safe`.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.14 Misleading function names

Severity: Informational

Context: ConsoleOpBuilder.sol#L96, ConsoleOpBuilder.sol#L146, ExecutorRegistry.sol#L73

Description: 1) `enableExecutorPluginOnSubAccount`: The `SubAccount` on the function name and the input parameter are misleading as this function can also be called on the console account.

```
function enableExecutorPluginOnSubAccount(address subAccount, address[] memory executors)
```

2) `disableExecutorPluginOnSubAccount`: The `SubAccount` on the function name and the input parameter are misleading as this function can also be called on the console account.

```
function disableExecutorPluginOnSubAccount(address subAccount, address previousModule)
```

3) `getExecutorsForSubAccount`: The `SubAccount` on the function name and the input parameter are misleading as this function can also be called to query the list of executors for main console accounts.

```
function getExecutorsForSubAccount(address _account) external view returns (address[] memory)
```

Recommendation: Update the name of the affected functions and their related parameters to be in line with their intention.

Brahma: Solved in PR 57.

Spearbit: Verified.

5.4.15 Function `_decompileSignatures()` could revert without descriptive error

Severity: Informational

Context: PolicyValidator.sol#L157-L167

Description: The function `_decompileSignatures()` checks `(_signatures.length < 8)`. However if `_signatures.length - 8 - sigLength` results in a negative value, the transaction will revert without a descriptive error. This might make troubleshooting a failed transaction more difficult.

```
function _decompileSignatures(bytes calldata _signatures) ... {
    if (_signatures.length < 8) revert InvalidSignatures();
    uint32 sigLength = uint32(bytes4(_signatures[_signatures.length - 8:_signatures.length - 4]));
    expiryEpoch = uint32(bytes4(_signatures[_signatures.length - 4:_signatures.length]));
    validatorSignature = _signatures[_signatures.length - 8 - sigLength:_signatures.length - 8];
}
```

Recommendation: Consider adding the following check:

```
function _decompileSignatures(bytes calldata _signatures) ... {
    if (_signatures.length < 8) revert InvalidSignatures();
    uint32 sigLength = uint32(bytes4(_signatures[_signatures.length - 8:_signatures.length - 4]));
    // ...
+   if (_signatures.length - 8 < sigLength) revert InvalidSignatures();
    validatorSignature = _signatures[_signatures.length - 8 - sigLength:_signatures.length - 8];
}
```

Alternatively consider removing `if (_signatures.length < 8) revert InvalidSignatures();` to be consistent.

Brahma: Solved in PR 58.

Spearbit: Verified.

5.4.16 Unused code

Severity: Informational

Context: AddressProviderService.sol#L62, Constants.sol#L86

Description: 1) The `_onlyGov` function is not used within the codebase:

```
function _onlyGov() internal view {
    if (msg.sender != addressProvider.governance()) {
        revert NotGovernance(msg.sender);
    }
}
```

2) The following `_GUARDIAN_HASH` constant is not used within the codebase:

```
bytes32 internal constant _GUARDIAN_HASH =
↪  0x424560fc12b0242dae8bb63e27dad69d2589059728e8daf9b2ff8557998f3402;
```

Recommendation: Consider the following actions:

1) Remove the unused `_onlyGov()` function from the codebase. The `error NotGovernance(address)` can also be removed.
2) Remove the unused `_GUARDIAN_HASH` constant.

Brahma: Solved in PR 55.

Spearbit: Verified.

5.4.17 `isValidSignature()` does fewer checks than `SafeModerator`

Severity: Informational

Context: ConsoleFallbackHandler.sol#L40-L57

Description: The function `isValidSignature()` does `isPolicySignatureValid()`, which is the equivalent of the `validatePre...Transaction...()` call of the `SafeModerator`s. However there is no equivalent of the checks done in the `validatePost...Transaction...()` call. The function `isValidSignature()` doesn't allow checks after the fact, but it could to the checks from `validatePost...Transaction...()`.

```
function isValidSignature(bytes memory _data, bytes memory _signature) public view override returns
↪  (bytes4) {
    // ...
    if (_signature.length == 0) {
        require(safe.signedMessages(messageHash) != 0, "Hash not approved");
    } else {
        // ...
        require(policyValidator.isPolicySignatureValid(msg.sender, messageHash, _signature), "Policy
↪  not approved");
        // ...
    }
    // ...
}
```

Recommendation: Consider adding checks from `validatePost...Transaction...()` to the function `isValidSignature()`. This would mean calling `_checkSubAccountSecurityConfig()`, `_checkConsoleAccountSecurityConfig()` to check the integrity of the Safe before potentially executing a transaction, so it would check the result of the previous transaction.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.18 Typographical errors

Severity: Informational

Context: SafeModeratorOverridable.sol#L14, TransactionValidator.sol#L80, SafeDeployer.sol#L176

Description: Several typographical errors were identified within the code comments:

```
// SafeModeratorOverridable.sol#L14
* @notice A guard that validates transactions and allows only policy abiding txns, on Brhma console
↪  account and can be overriden by removal of guard

// TransactionValidator.sol#L80
* @notice Provides on-chain guarantees on security critical expected states of a Brhma console account

// SafeDeployer.sol#L172
* Enable Brhma Console account as module on sub Account
```

Recommendation: Update the above typographical errors.

```
- ... Brhma ...
+ ... Brahma ...
```

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.19 `isValidSignature()` doesn't check policy in combination with `signedMessages()`

Severity: Informational

Context: ConsoleFallbackHandler.sol#L40-L57

Description: The function `isValidSignature()` doesn't check a policy if `signedMessages()` are used. The `signedMessages()` could be used in the following ways:

1) Delegate call to signMessage of SignMessageLib and checked via the Policy.
2) Delegate call to signMessage of SignMessageLib and via an extra installed module, outside of the knowledge of the protocol.
3) Left over from a pre-existing safe.

Situation 1 is checked and should be allowed. Situation 2 and 3 are not checked by a Policy.

```
function isValidSignature(bytes memory _data, bytes memory _signature) public view override returns
↪  (bytes4) {
    // ...
    if (_signature.length == 0) {
        require(safe.signedMessages(messageHash) != 0, "Hash not approved");
    } else {
        // ...
        require(policyValidator.isPolicySignatureValid(msg.sender, messageHash, _signature), "Policy
↪  not approved");
        // ...
    }
    // ...
}
```

Recommendation: Consider checking if its possible to differentiate between the different situations, by remembering `signedMessages()` that have been added in combination with a policy check. After that checked `signedMessages()` could be allowed, while others could be blocked.

Brahma: Acknowledged.

Spearbit: Acknowledged.

5.4.20 Function `_executeOnSafe()` isn't used

Severity: Informational

Context: SafeHelper.sol#L63-L78

Description: The function `_executeOnSafe()` isn't used in the current code. **Note:** it is used in some of the tests.

```
function _executeOnSafe(address safe, address target, Enum.Operation op, bytes memory data) internal {
    bool success = IGnosisSafe(safe).execTransaction(...);
    if (!success) revert SafeExecTransactionFailed();
}
```

Recommendation: Doublecheck the relevance of `_executeOnSafe()` and remove if it is no longer used.

Brahma: Solved in PR 61.

Spearbit: Verified.