Security audit Report
Rockside

Rockside.io Smart Contracts

Ref. : SEKOIA-Rockside_SmartContractReport-200814

REVISION HISTORY

Version	Date	Description	Author(s)
1.0	14/08/2020	Initial version	Bertrand MASIUS
1.1	11/09/2020	Addition of Rockside replies	Bertrand MASIUS
1.2	16/09/2020	Validation by SEKOIA	Bertrand MASIUS

DISTRIBUTION LIST

Recipient	Company
Vincent LE GALLIC	Rockside

SUMMARY

1. INTRODUCTION

1.1. AUDIT PURPOSE

The purpose of the code review is to detect potential vulnerabilities using the same means as attackers and to provide relevant recommendations to correct these vulnerabilities and improve the level of security of the audited platform.

1.2. AUDIT CONTEXT

The Smart Contracts reviewed are developped and deployed by Rockside in the Ethereum blockchain. Their purpose of the Smart Contracts is to relay transactions provided by Rockside.io customers through the API and refund the gas spent by Rockside relayers by sending back a compensation in Ether. Customers must supply Smart Contract wallet with Ethers.

The code review is not intrusive and no exploitation of vulnerability is attempted.

1.3. SCOPE

The scope of the code review is limited to these Smart Contracts:

- Forwarder.sol and dependencies
- Proxy.sol and dependencies

The source code is hosted on github.

We noted that the Forwarder implementation may be changed by the user, but this case in out of scope.

We noted that the user may change the relayers list and provide his own relayers, but this case il also out of scope.

1.4. CONTACT PRESENTATION

The SEKOIA team in charge of this mission is:

- Bertrand MASIUS, Audit,
- Arthur CHOVET, Audit Manager.

Mail: *first_name.last_name* at sekoia.fr

2. EXECUTIVE SUMMARY

2.1. RISK LEVEL ASSESSMENT

Impact				
Critical	Significant	Major	Critical	Critical
Major	Significant	**Major**	Major	Critical
Significant	Minor	Significant	Significant	Major
Low	Very low	Minor	Significant	Major
	Low	**Moderate**	High	Very high
		Likelihood		

The security code review performed on the Smart Contracts Proxy.sol and Forwarder.sol concluded with a **major risk level**.

The automated analysis did not reveal significant vulnerabilities in the code, its effectiveness was limited by the presence of multiple assembly part and interdependent calls.

The manual review covered more threats and reveal some major issues.

The vulnerabilities discovered expose the application and its users to the following risks:

- **Privilege escalation**: Some priviledged functions can be called by users calling them through the Rockside service. *Risk level*: **Major**.
- **Code bad practice**: Storage slots are used without being properly declared, leading to possible security impact in future development. *Risk level*: **Significant**.
- **Abuse of service**: a miner or a pool of miner can use the service to call estimateForward(), consuming a lot of gas paid by Rockside. *Risk level*: **Minor**.

Global correction effort is estimated as **Moderate**. The identified vulnerabilities require some fixes on the code, detailed in the document

3. TECHNICAL SUMMARY

3.1. VULNERABILITY LIST

The following table lists all vulnerabilities found during the audit. It is ordered by the vulnerability with highest risk level to the lowest.

ID	Vulnerability	Risk level
Vulnerability 2	**Privilege escalation**: Normally only the owner of the Proxy contract have control on it, but a vulnerability allows API key users to have owner right on the proxy.	**Major**
Vulnerability 1	**Storage layout is not explicitly preserved in proxy pattern:** In a proxy pattern, the storage slots of 'Proxy.sol' are used in the context of the code of 'Forwarder.sol', and both contracts must have the same storage layout. The storage layout is not explicitly declared and reserved in 'Proxy.sol', making 'Forwarder.sol' blindly using the storage slots that might be later used by 'Proxy.sol'.	**Significant**
Vulnerability 3	**estimateForward() protection might be partially defeated**: estimateForward() protection against execution in a transaction is partially effective since a reentrancy attack can defeat the first protection.	**Minor**
Vulnerability 4	**Possible front run attack during proxy deployment**: A proxied call to Fowarder.initialize() is necessary after proxy deployment. Deployment scripts might split proxy deployment in 2 separate transactions, creating the possibility of front run attack since execution of second call is not restricted to authorized accounts.	**Minor**
Vulnerability 5**Erreur ! Source du renvoi introuvable.**	**Potential execution of uncontrolled code**: Functions Proxy.upgradeTo() and Proxy.upgradeToAndCall() are accessible to users of Rockside.io, and allow them to change implementation of Forwarder.sol to something	**Minor**

	more interesting for them, such as a free forwarder with no gas refund.	
Vulnerability 6	**Code without effect**: Forwarder contract never uses state variables that are initialized at construction, because they are only placeholders for the state variables of the Proxy contract. Call to initialize() is without effect in the context of the Forwarder.	**Minor**

3.2. ACTION PLAN

The action plan is sort by priority order.

Recommendation	Associated vulnerability	Correction effort
Recommendation 1: Prefer 'external' scope for functions anywhere applicable	**Automated analysis**	Low
Recommendation 2: Declare explicitly in Proxy the slots used by Forwarder	**Vulnerability 1**	Low
Recommendation 3: Proxy contract must not be owner of itself.	**Vulnerability 2**	Moderate
Recommendation 4: Reconsider protection against execution of estimateForward()	**Vulnerability 3**	High
Recommendation 5: initialize() must be explicitly called by ProxyFactory.deployProxy(),or restrict the execution of initialize() to a specific account.	**Vulnerability 4**	Low
Recommendation 6: Implementation change operated by users must be controlled to check that Rockside interests are preserved.	**Vulnerability 5**	High
Recommendation 7: Forwarder constructor should not call initialize().	**Vulnerability 6**	Low

4. FULL REPORT

4.1. PERFORMED ANALYSIS

In this section, analysis performed during the audit are briefly explained.

4.1.1. SMART CONTRACT STATIC ANALYSIS

The SMT verifier of solc was used to list findings at compile time. There was no finding with security impact.

Slither analyzer was used to find possible vulnerabilities. Many findings regarding best practices were found, but none would have an impact on security.

Slither notices that many public functions are not called internally, and could be declared as external:

- In AuthorizedRelayers:
  - add
  - remove
  - transferOwnership
  - verify
- In Fowarder:
  - updateTrustedContracts
  - changeRelayersSource
  - withdraw
  - forward
- In Proxy:
  - upgradeToAndCall
  - version
  - implementation

Recommendation 1: Prefer ‘external’ scope for functions anywhere applicable

SEKOIA Validation: Fix is accepted in this commit.

Slither results can be found here.

4.1.2. SMART CONTRACT DYNAMIC ANALYSIS

Mythril did not discover exploitation path for the vulnerabilities the tool tries to detect.

There is only a false positive concerning call to an address controlled by the user: this result is normal since the user controls the Proxy implementation address.

Mythril results can be found here.

4.2. STORAGE LAYOUT IS NOT EXPLICITLY PRESERVED IN PROXY PATTERN

VULNERABILITY:	**ACTION PLAN:**
RISK: ●●○○	**CLASSIFICATION: SWC-124**
LIKELIHOOD: ●●○○	**CORRECTION EFFORT:** ●○○○
IMPACT: ●●●○	
TARGET: PROXY	
BUSINESS IMPACT: DENIAL OF SERVICE	

Vulnerability 1: In a proxy pattern, the storage slots of 'Proxy.sol' are used in the context of the code of 'Forwarder.sol', and both contracts must have the same storage layout. The storage layout is not explicitly declared and reserved in 'Proxy.sol', making 'Forwarder.sol' blindly using the storage slots that might be later used by 'Proxy.sol'.

Failing to declare and reserve the storage slots in 'Proxy.sol' creates a risk of malfunction and possibly security breach if storage variables are created in the future in slots used by Forwarder.sol'. Future developpers might miss this point and deliver code that still works but is vulnerable.

A good practice is to make Forwarder and Proxy both inherit of the same Contract declaring the state variables.

Recommendation 2: Declare explicitly in Proxy the slots used by Forwarder.

Rockside Response: We understand the risk as described but we decided to have a generic proxy that can work with any kind of implementations.

Moreover, a generic proxy requires less gas to be deployed.

We follow the proxy pattern as defined by openzeppelin as gnosis safe or argent contracts does. https://blog.openzeppelin.com/proxy-patterns/

The Proxy source file will include the missing declarations as comments with an explanation to avoid misunderstanding in future developments

SEKOIA Validation: The response is accepted as a fix is provided by this commit

4.3. PRIVILEGE ESCALATION ON THE PROXY

VULNERABILITY:	ACTION PLAN:
RISK: ●●●○	CLASSIFICATION: SWC-107
LIKELIHOOD: ●●○○	CORRECTION EFFORT: ●●○○
IMPACT: ●●●○	
TARGET: PROXY	
BUSINESS IMPACT: DENIAL OF SERVICE	

Vulnerability 2: Normally only the owner of the Proxy contract have control on it, but a vulnerability allows API key users to have owner right on the proxy.

Someone can craft a transaction and define the proxy itself as target.

When executed, the execution flow goes to the Forwarder code as normal, then back to the Proxy code since the Proxy contract is the target, having the **Proxy address as msg.sender** this time. The user can call any function of the proxy.

For example, changeRelayersSources() is restricted to owners, but the Proxy contract itself was declared as owner in the constructor, thus the call to changeRelayersSource() with the proxy address as msg.sender is allowed.

Figure 1: Proxy contract is owner of itself

```
constructor(address owner, bytes32 version, address implementation) payable public {
    owners[owner] = true;
    owners[address(this)] = true;
    _setVersion(version);
    _setImplementation(implementation);
}
```

Note that all functions restricted to owners are executable with this method, but the funds are safe since withdraw() is executable, but it sends funds to msg.sender, who is the Proxy contract itself.

Recommendation 3: Proxy contract must not be owner of itself.

Rockside Response: We moved the owners logic from the proxy, to implementation. So this is not on the responsibility of the proxy anymore.

We also changed the logic for the forwarder. The forwarder itself is not owner anymore. We originally added the forwarder as owner so administration tasks such as changing the implementation of the formarder can be done using MetaTx.
Even if we thought the attack surface was limited, because only people having the Rockside API Key would be able to execute transactions from the forwarder, we decided to remove it.

Here is the commit:
https://github.com/rocksideio/contracts/commit/a41c8670921b1751d9c05170abeb0f8a810190a8#diff-e9839443e2c0b012386b0555ffe30f98

SEKOIA Validation: The response is accepted and the fix is provided in this commit.

4.4. ESTIMATEFORWARD() PROTECTION MIGHT BE PARTIALLY DEFEATED

VULNERABILITY:	**ACTION PLAN:**
RISK: ●●○○	**CLASSIFICATION: SWC-107**
LIKELIHOOD: ●●○○	**CORRECTION EFFORT:** ●●●○
IMPACT: ●●○○	
TARGET: PROXY	
BUSINESS IMPACT: FINANCIAL LOSS	

Vulnerability 3: estimateForward() protection against execution in a transaction is partially effective since a reentrancy attack can defeat the first protection.

estimateForward() must not be executed in a transaction and has 2 protection for this. The first one is a filter to allow only ‘this’.

Figure 2: estimateForward execution is restricted to 'this'

```
function estimateForward(
        bytes calldata signature,
        address signer,
        address to,
        bytes calldata data,
        uint gasPriceLimit,
        uint256 nonce
)
        external
{
    require(msg.sender == address(this));
```

which means that only the proxy contract (through delegatecall), or the forwarder contract, can call the function.

A specially crafted transaction emitted through the API that target the proxy itself will check this condition, since the Forwarder uses the proxy address as msg.sender.

The second protection is effective against execution, but will consume all gas paid by the relayer.

Recommendation 4: Reconsider protection against execution of estimateForward()

Rockside Response: We make a dry run before sending the actual transaction. This protects us from possible on chain error. Intentional calls to this function through our API will fail before reaching the blockchain.

SEKOIA Validation: The response is accepted but we did not check that a call of this function through the API will fail since it is out of the scope of the audit.

4.5. FORWARDER.INITIALIZE() CALL IS NOT RESTRICTED

VULNERABILITY:	**ACTION PLAN:**
RISK: ●●○○	**CLASSIFICATION: SWC-114**
LIKELIHOOD: ●○○○	**CORRECTION EFFORT:** ●○○○
IMPACT: ●●●○	
TARGET: PROXY	
BUSINESS IMPACT: DENIAL OF SERVICE	

Vulnerability 4: A proxied call to Fowarder.initialize() is necessary after proxy deployment. Deployment scripts might split proxy deployment in 2 separate transactions, creating the possibility of front run attack since execution of second call is not restricted.

Proxy test source code shows that the contract creation and the initialization are performed in a single transaction, and it is assumed it is the same in production, but this behaviour is not enforced at blockchain level.

Figure 3: initialize() is supposed to be called here, but it is not mandatory

```
proxy = new Proxy{value:msg.value}(owner, version, implementation);
if (data.length > 0) {
    (bool success,) = address(proxy).call(data);
    require(success, "Failing call after deployment");
}
```

If a change in deployment script makes Forwarder.initialize() being called apart, a malicious person may watch proxy contract creations in the pending transactions pool and call Forwarder.initialize() right after, through the newly deployed proxy, taking control of trusted contracts list and authorized relayers list, i.e taking control of the proxy usage until the legitimate owner reset these values. Note that the GUI does not give access to the reset functions: they must be called manually.

Figure 4: initialize() is not restricted

```
function initialize(address relayersAddress, address[] memory _trustedContracts) public {
        require(!initialized, "Contract already initialized");
        initialized = true;

        relayers = AuthorizedRelayers(relayersAddress);
        hasTrustedContracts = _trustedContracts.length > 0;
```

The impact is a denial of service on the proxy until a manual fix is performed. Since this attack can be repeated on each newly created proxy, there is also a denial of service of support team that must fix each proxy contract in coordination with each user impacted.

Recommendation 5: initialize() must be explicitly called by ProxyFactory.deployProxy(), or restrict the execution of initialize() to a specific account..

Rockside Response: When deploying a forwarder contract the initialize method is directly called by the constructor. This way, no chance to have a non-initialized forwarder.

In the case of deploying a proxy that targets a forwarder as implementation, we wanted to keep our proxy factory as generic as possible, so it's the responsibility of the caller to specify the data to execute after the deployment of the contract.

In our case, we are covered by our acceptances test. They verify that a forwarder deployed through Rockside API is well initialized.

Moreover we documented the requirement that initialize() should be called in the same transaction as the constructor.

SEKOIA Validation: The response is accepted and a fix is provided by this commit and this commit.

4.6. USERS CAN CHANGE FORWARDER IMPLEMENTATION

VULNERABILITY:	**ACTION PLAN:**
RISK: ●●●●	**CLASSIFICATION: SWC-112**
LIKELIHOOD: ●●●●	**CORRECTION EFFORT: ●●●○**
IMPACT: ●●●○	
TARGET: PROXY	
BUSINESS IMPACT: FINANCIAL LOSS, REPUTATION, LEGAL RESPONSIBILITY	

Vulnerability 5: Functions Proxy.upgradeTo() and Proxy.upgradeToAndCall() are accessible to users of Rockside.io, and allow them to change implementation of Forwarder.sol to something more interesting for them, such as a free forwarder with no gas refund.

A specially crafted implementation may create financial loss by using relayers for free. It might also perform operations in the name of the relayer since it can preserve transaction context (with delegatecall) and have an impact on Rockside reputation or held Rockside liable for the actions performed on its name.

Rockside team is fully aware of the implication of this. This is a reminder that controls must be put in place to detect and react to implementation change to avoid financial loss, reputation attack or unwanted liability.

Recommendation 6: Implementation change operated by users must be controlled to check that Rockside interests are preserved.

Rockside Response: We will add a verification on the implementation registered by the proxy. Only transactions targeting a Forwarded with an authorized implementation will be authorized by the Rockside API.

SEKOIA Validation: The response is accepted.

4.7. FORWARDER CONTRACT CONSTRUCTOR SHOULD NOT CALL INITIALIZE()

VULNERABILITY:	**ACTION PLAN:**
RISK: ●○○○	**CLASSIFICATION: SWC-135**
LIKELIHOOD: ●○○○	**CORRECTION EFFORT:** ●○○○
IMPACT: ●○○○	
TARGET: FORWARDER	
BUSINESS IMPACT: NONE	

Vulnerability 6: Forwarder contract never uses state variables that are initialized at construction, because they are only placeholders for the state variables of the Proxy contract. Call to initialize() is without effect in the context of the Forwarder.

Recommendation 7: Forwarder constructor should not call initialize().

4.8. REMARKS FOR INFORMATION

These remarks have no impact on security.

Forwarder.verifySignature() documentation erroneously mentions a check that signer is an owner

The code does not contain such a condition check.

Remark 8: Fix documentation of verifySignature()

Since solidity v6.12, keccak256() of plain strings are computed at compilation time. Computed hashes used as constants may be explicitly computed in code. No need to precompute them

Remark 9: Constant variables assignment should call keccak() instead of hardcoded values, with ^0.6.12 pragma.

5. APPENDIX 1 : RISK ASSESSMENT MATRIX

The ranking system used by SEKOIA is based on risk assessment matrix proposed by ANSSI (French national cybersecurity agency).

Impact \ Likelihood	Low	Moderate	High	Very high
Critical	Significant	Major	Critical	Critical
High	Significant	Major	Major	Critical
Moderate	Minor	Significant	Significant	Major
Low	Minor	Minor	Significant	Major

The risk level of each vulnerability is assessed from the following scale:

- Minor: Minimal risk for the information system and might need a fix
- Significant: Moderate risk for the information system which needs to be fixed at medium term.
- Major: Major risk for the information system which needs to be fixed at short term.
- Critical: Critical risk for the information system which needs to be fixed immediately.

The likelihood is based on technical level and means required to exploit a vulnerability as well as the exposition level. It uses the following levels:

- Low: Vulnerability with very minimal exposure and/or need high skills in cybersecurity and/or important means
- Moderate: Vulnerability with minimal exposure and need some skills in cybersecurity or in tools development.
- High: Vulnerability with high exposure (Internet for example) and/or need low skills in cybersecurity or publicly tools.
- Very High: Vulnerability with very high exposure and/or without need any skills to exploit it.

Impact value corresponds to the consequences that the exploitation of the vulnerability can entail on the information system and/or the activity of audited company. This level is appreciated from the following scale:

- Minor: No direct consequence for the security of information system or company activity.
- High: Minor consequence on specific points on the audited information system.
- Major: Consequence which could have a high impact in the security of the information system or the company activity.
- Critical: Consequence which could have a very high impact in the security of the information system or the company activity

6. APPENDIX 2 : ANALYSIS TOOLS RESULTS

6.1. MYTHRIL

Table 1: Forwarder.sol analysis with Mythril

```
$ myth analyze Forwarder.sol
The analysis was completed successfully. No issues were
detected.
```

Proxy.sol analysis took too long time, a depth of 12 recurrent iterations was defined.

Table 2: Proxy.sol analysis with Mithril

```
$ myth analyze --max-depth 12 Proxy.sol
==== Delegatecall to user-supplied address ====
SWC ID: 112
Severity: High
Contract: Proxy
Function name: fallback
PC address: 276
Estimated Gas Usage: 1769 - 38261
The contract delegates execution to another contract with a
user-supplied address.
The smart contract delegates execution to a user-supplied
address.This could allow an attacker to execute arbitrary code
in the context of this contract account and manipulate the state
of the contract account or execute actions on its behalf.
--------------------
In file: Proxy.sol:60

delegatecall(gas(), _impl, ptr, calldatasize(), 0, 0)


--------------------
Initial State:
```

```
Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller:                    [CREATOR],                    calldata:
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000deadbeefdeadbeefdeadbeefdeadbeefde
adbeef..00, value: 0x0
Caller: [ATTACKER], function: unknown, txdata: 0x00, value: 0x0
```

6.2. SLITHER

Table 3: Proxy.sol analysis with slither

```
$ slither Proxy.sol
INFO:Detectors:
Proxy.constructor(address,bytes32,address).version
(Proxy.sol#24) shadows:
     - Proxy.version() (Proxy.sol#45-47) (function)
Proxy.constructor(address,bytes32,address).implementation
(Proxy.sol#24) shadows:
     - Proxy.implementation() (Proxy.sol#49-51) (function)
Reference:    https://github.com/crytic/slither/wiki/Detector-
Documentation#local-variable-shadowing
INFO:Detectors:
Proxy.fallback() (Proxy.sol#53-68) uses assembly
     - INLINE ASM None (Proxy.sol#57-67)
Proxy._implementation() (Proxy.sol#72-77) uses assembly
     - INLINE ASM None (Proxy.sol#74-76)
Proxy._setImplementation(address)    (Proxy.sol#79-85)    uses
assembly
```

```
	- INLINE ASM None (Proxy.sol#82-84)
Proxy._version() (Proxy.sol#87-92) uses assembly
	- INLINE ASM None (Proxy.sol#89-91)
Proxy._setVersion(bytes32) (Proxy.sol#94-100) uses assembly
	- INLINE ASM None (Proxy.sol#97-99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version>=0.6.0<0.7.0 (OwnersMap.sol#2) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Pragma version>=0.6.0<0.7.0 (Proxy.sol#2) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Proxy.upgradeToAndCall(bytes32,address,bytes) (Proxy.sol#39-43):
	- (success) = address(this).call{value: msg.value}(data) (Proxy.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
upgradeToAndCall(bytes32,address,bytes) should be declared external:
	- Proxy.upgradeToAndCall(bytes32,address,bytes) (Proxy.sol#39-43)
version() should be declared external:
	- Proxy.version() (Proxy.sol#45-47)
implementation() should be declared external:
	- Proxy.implementation() (Proxy.sol#49-51)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
INFO:Slither:Proxy.sol analyzed (2 contracts with 46 detectors), 13 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Table 4: Forwarder.sol analysis with slither

```
$ slither Forwarder.sol
INFO:Detectors:
Forwarder.forward(bytes,address,address,bytes,uint256,uint256) (Forwarder.sol#88-133) sends eth to arbitrary user
	Dangerous calls:
	- require(bool,string)(msg.sender.send(payment),Could not refund gas to relayer) (Forwarder.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
AuthorizedRelayers.constructor(address[]).i (AuthorizedRelayers.sol#15) is a local variable never initialized
AuthorizedRelayers.add(address[]).i (AuthorizedRelayers.sol#21) is a local variable never initialized
AuthorizedRelayers.remove(address[]).i (AuthorizedRelayers.sol#27) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
```

```
Forwarder.initialize(address,address[])  (Forwarder.sol#37-53)
uses assembly
     - INLINE ASM None (Forwarder.sol#49-51)
Forwarder.forward(bytes,address,address,bytes,uint256,uint256)
(Forwarder.sol#88-133) uses assembly
     - INLINE ASM None (Forwarder.sol#121-124)
Forwarder.signerIsValid(bytes32,bytes,bytes32,address)
(Forwarder.sol#236-273) uses assembly
     - INLINE ASM None (Forwarder.sol#252-256)
Reference:    https://github.com/crytic/slither/wiki/Detector-
Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used in :
     - Version used: ['>=0.6.0<0.7.0', '^0.6.0']
     - >=0.6.0<0.7.0 (AuthorizedRelayers.sol#2)
     - >=0.6.0<0.7.0 (Forwarder.sol#2)
     - >=0.6.0<0.7.0 (OwnersMap.sol#2)
     - ^0.6.0 (SafeMath.sol#2)
Reference:    https://github.com/crytic/slither/wiki/Detector-
Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma     version>=0.6.0<0.7.0     (AuthorizedRelayers.sol#2)
necessitates  versions  too  recent  to  be  trusted.  Consider
deploying with 0.5.11
Pragma  version>=0.6.0<0.7.0  (Forwarder.sol#2)  necessitates
versions  too  recent  to  be  trusted.  Consider  deploying  with
0.5.11
Pragma  version>=0.6.0<0.7.0  (OwnersMap.sol#2)  necessitates
versions  too  recent  to  be  trusted.  Consider  deploying  with
0.5.11
Pragma version^0.6.0 (SafeMath.sol#2) necessitates versions too
recent to be trusted. Consider deploying with 0.5.11
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Forwarder.forward(bytes,address,address,bytes,uint256,uint256) (Forwarder.sol#88-133):
	- (success) = to.call(abi.encodePacked(data,signer)) (Forwarder.sol#119)
Low level call in Forwarder.estimateForward(bytes,address,address,bytes,uint256,uint256) (Forwarder.sol#145-189):
	- (success) = to.call(abi.encodePacked(data,signer)) (Forwarder.sol#178)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Forwarder.initialize(address,address[])._trustedContracts (Forwarder.sol#37) is not in mixedCase
Parameter Forwarder.hashEIP712Domain(address,uint256)._verifyingContract (Forwarder.sol#278) is not in mixedCase
Parameter Forwarder.hashEIP712Domain(address,uint256)._chainId (Forwarder.sol#278) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
add(address[]) should be declared external:
	- AuthorizedRelayers.add(address[]) (AuthorizedRelayers.sol#20-24)
remove(address[]) should be declared external:
	- AuthorizedRelayers.remove(address[]) (AuthorizedRelayers.sol#26-30)
```

```
transferOwnership(address) should be declared external:
     - AuthorizedRelayers.transferOwnership(address)
(AuthorizedRelayers.sol#32-35)
verify(address) should be declared external:
     - AuthorizedRelayers.verify(address)
(AuthorizedRelayers.sol#37-39)
updateTrustedContracts(address[]) should be declared external:
     - Forwarder.updateTrustedContracts(address[])
(Forwarder.sol#59-65)
changeRelayersSource(address) should be declared external:
     - Forwarder.changeRelayersSource(address)
(Forwarder.sol#67-70)
withdraw(uint256) should be declared external:
     - Forwarder.withdraw(uint256) (Forwarder.sol#73-77)
forward(bytes,address,address,bytes,uint256,uint256) should be
declared external:
     -
Forwarder.forward(bytes,address,address,bytes,uint256,uint256)
(Forwarder.sol#88-133)
Reference:    https://github.com/crytic/slither/wiki/Detector-
Documentation#public-function-that-could-be-declared-as-
external
INFO:Slither:Forwarder.sol analyzed (4 contracts with 46
detectors), 25 result(s) found
INFO:Slither:[94mUse https://crytic.io/ to get access to
additional detectors and Github integration
```