// HALBORN

Rarimo - Solidity Bridge

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 12th, 2022 - December 16th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/13/2022	Alejandro Taibo
0.2	Document Update	12/16/2022	Alejandro Taibo
0.3	Draft Review	12/16/2022	Gabi Urrutia
1.0	Remediation Plan	12/27/2022	Alejandro Taibo
1.1	Remediation Plan Review	12/27/2022	Gokberk Gulgun
1.2	Remediation Plan Review	12/28/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Alejandro Taibo	Halborn	Alejandro.Taibo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Rarimo engaged Halborn to conduct a security audit on their smart contracts beginning on December 12th, 2022 and ending on December 16th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were accepted and acknowledged by the Rarimo team.

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE, Ganache, Foundry)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

CODE REPOSITORIES:

- Repository: evm-bridge
- Commit ID: 8d5efd4072e7ced93f8c3400033684115d3b91f2
- Smart contracts in scope:
  - contracts/bridge/Bridge.sol
  - contracts/bridge/proxy/UUPSSignableUpgradeable.sol
  - contracts/bundle/Bundler.sol
  - contracts/bundle/proxy/BundleExecutorImplementation.sol
  - contracts/bundle/proxy/BundleExecutorProxy.sol
  - contracts/handlers/ERC1155Handler.sol
  - contracts/handlers/ERC20Handler.sol
  - contracts/handlers/ERC721Handler.sol
  - contracts/handlers/NativeHandler.sol
  - contracts/interfaces/bridge/IBridge.sol
  - contracts/interfaces/bundle/IBundler.sol
  - contracts/interfaces/handlers/IERC1155Handler.sol
  - contracts/interfaces/handlers/IERC20Handler.sol
  - contracts/interfaces/handlers/IERC721Handler.sol
  - contracts/interfaces/handlers/INativeHandler.sol
  - contracts/interfaces/tokens/IERC1155MintableBurnable.sol
  - contracts/interfaces/tokens/IERC20MintableBurnable.sol
  - contracts/interfaces/tokens/IERC721MintableBurnable.sol
  - contracts/libs/Encoder.sol
  - contracts/tokens/ERC1155MintableBurnable.sol
  - contracts/tokens/ERC20MintableBurnable.sol
  - contracts/tokens/ERC721MintableBurnable.sol
  - contracts/utils/Hashes.sol
  - contracts/utils/Signers.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	2	4

LIKELIHOOD

IMPACT

(HAL-04)	(HAL-02)			
(HAL-05)		(HAL-03)	(HAL-01)	
(HAL-06) (HAL-07) (HAL-08) (HAL-09)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - SELFDESTRUCT DEPRECATION	Medium	RISK ACCEPTED
HAL-02 - BUNDLE IMPLEMENTATION IS PRONE TO DOS	Medium	RISK ACCEPTED
HAL-03 - REMAINING TOKENS ARE NOT RECOVERED AFTER BUNDLE DESTRUCTION	Medium	RISK ACCEPTED
HAL-04 - LACK OF RE-ENTRANCY PROTECTION	Low	RISK ACCEPTED
HAL-05 - FLOATING PRAGMA	Low	RISK ACCEPTED
HAL-06 - ZERO ADDRESS NOT CHECKED AFTER CREATE2 EXECUTION	Informational	NOT APPLICABLE
HAL-07 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational	ACKNOWLEDGED
HAL-08 - USE OF INLINE ASSEMBLY	Informational	NOT APPLICABLE
HAL-09 - MISSING ZERO ADDRESS CHECK	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SELFDESTRUCT DEPRECATION - MEDIUM

Description:

The BundleExecutorProxy smart contract executes self-destruct opcode in order to destruct the smart contract and sending the remaining ether back to the bridge.

Following the recent EIP-6049, the self-destruct opcode will be deprecated, and hence, modifying the functionality of this opcode. Moreover, this EIP warns against its usage.

Code Location:

Listing 1: contracts/bundle/proxy/BundleExecutorProxy.sol

```
17 function destroy() external {
18     address bridge_ = _BRIDGE;
19
20     assembly {
21         if iszero(eq(caller(), bridge_)) {
22             revert(0, 0)
23         }
24
25         selfdestruct(caller())
26     }
27 }
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to stop using this opcode in order to avoid broken functionalities in the future.

Reference:

EIP-6049: Deprecate SELFDESTRUCT

Remediation Plan:

RISK ACCEPTED: The Rarimo team accepted the risk of this finding. The Rarimo team stated:

“We are aware of this and it only warns against the usage. In the backward compatibility” section, the EIP clearly (rather poorly) states, this EIP updates non-normative text in the Yellow Paper. No changes to clients is applicable.”

3.2 (HAL-02) BUNDLE IMPLEMENTATION IS PRONE TO DOS - MEDIUM

Description:

The BundleExecutorImplementation smart contract performs several external calls in a loop, this pattern adds an extra dependency for a successful execution of the transaction, since each external call should not revert or consume the remaining gas.

This condition might lead to a denial-of-service (DOS) attacks, since a malicious smart contract could revert the transaction after executing a revert or draining the remaining gas.

In this case, this issue does not pose a high risk since the Bridge smart contract is using a try-catch statement to avoid blocking the assets transferred in the transaction, but a DOS attack would block executing the rest of the actions specified in a bundle.

Code Location:

Listing 2: contracts/bundle/proxy/BundleExecutorImplementation.sol (Line 14)

```
14 for (uint256 i = 0; i < contracts_.length; i++) {
15     (bool success_, ) = payable(contracts_[i]).call{value: values_
↳ [i]}(data_[i]);
16
17     require(success_, "BundleExecutorImplementation: call reverted
↳ ");
18 }
```

Proof Of Concept:

Listing 3

```
function testDOS() {
    bundleExecutorImplementation.execute(anyBundleData);
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

If possible, it is highly recommended to use pull over push strategies for these situations.

Remediation Plan:

RISK ACCEPTED: The Rarimo team accepted the risk of this finding. The Rarimo team stated:

"The bundle workflow is intended. The DOS might only affect the funds of the exact user who executed this bundle, yet this user would be the one who created the bundle as well. Basically, it is on the user's shoulders (and front end) to correctly assemble the bundle.

Moreover, the try-catch logic will remedy the situation in the case of user errors or dedicated DOS."

3.3 (HAL-03) REMAINING TOKENS ARE NOT RECOVERED AFTER BUNDLE DESTRUCTION - MEDIUM

Description:

The BundleExecutorProxy smart contract oversees making delegate calls to the BundleExecutorImplementation smart contract. After a successful execution, BundleExecutorProxy smart contract is destroyed by the execution of its destroy function, transferring back its balance of ether to the bridge by executing selfdestruct.

Therefore, if a bundle aiming to transfer tokens such as ERC20, ERC721 or ERC1155 does not transfer all the specified tokens in the bundle, these remaining tokens would keep associated to the destroyed account instead of getting transferred to the bridge since selfdestruct function only transfer ether.

Code Location:

Listing 4: contracts/bundle/Bundler.sol (Line 32)

```
23 function _bundleUp(Bundle calldata bundle_) internal {
24     address payable executor = payable(
25         new BundleExecutorProxy{salt: bundle_.salt}(
26             bundleExecutorImplementation,
27             address(this)
28         )
29     );
30
31     BundleExecutorImplementation(executor).execute(bundle_.bundle)
↳ ;
32     BundleExecutorProxy(executor).destroy();
33 }
```

Listing 5: contracts/bundle/proxy/BundleExecutorProxy.sol

```
17 function destroy() external {
18     address bridge_ = _BRIDGE;
19
20     assembly {
21         if iszero(eq(caller(), bridge_)) {
22             revert(0, 0)
23         }
24
25         selfdestruct(caller())
26     }
27 }
```

Proof Of Concept:

Listing 6

```
1 function testRemainingFunds() {
2    bundleExecutor.destroy();
3 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to verify whether the deployed smart contract has tokens associated to transfer them back to the bridge before destroying the contract.

Remediation Plan:

RISK ACCEPTED: The Rarimo team accepted the risk of this finding. The Rarimo team stated:

"The described logic is also expected. We would have left the native tokens on the same proxy address, however, `selfdestruct(payable(address(this)))` basically burns ether, so we are sending it back to the bridge. This approach is safe because the proxy address gets determined with `tx.origin` salt, so no one would be able to steal not their own funds.

The checks for 0 tokens withdrawals will be discarded to simplify the process of proxy recreation."

3.4 (HAL-04) LACK OF RE-ENTRANCY PROTECTION - LOW

Description:

The NativeHandler, ERC721Handler, ERC1155Handler and BundleExecutorImplementation smart contracts perform several arbitrary external calls without caring about recursive calls to its functions.

It is known that calling external contracts is dangerous if some functions and variables are called after the external call. An attacker could use a malicious contract to perform recursive calls, taking over the control flow.

In the case of BundleExecutorImplementation smart contract, executing recursive calls to execute function would drain the smart contract's balance but, at the same time, it would also mean reverting the transaction since the rest of the external calls in the loop should use the drained ether by specifying it as value argument.

By the other hand, in NativeHandler, ERC721Handler and ERC1155Handler smart contracts, an attacker would require of a valid signature to perform recursive calls to withdraw functions since after every execution the originHash is blacklisted avoiding more than one usage.

Therefore, despite not posing a risk to scoped smart contracts, it is worth to avoid this code patterns since they could be dangerous and implement countermeasures such as locks/mutex in order to avoid unintended recursive calls to smart contract's functions.

Code Location:

Listing 7: contracts/bundle/proxy/BundleExecutorImplementation.sol (Line 15)

```
14 for (uint256 i = 0; i < contracts_.length; i++) {
15     (bool success_, ) = payable(contracts_[i]).call{value: values_
↳ [i]}(data_[i]);
16
17     require(success_, "BundleExecutorImplementation: call reverted
↳ ");
18 }
```

Listing 8: contracts/handlers/NativeHandler.sol (Line 48)

```
42 function _withdrawNative(bytes calldata tokenData_, address
↳ receiver_, bool) internal {
43     uint256 amount_ = _decodeNativeTokenData(tokenData_);
44
45     require(amount_ > 0, "NativeHandler: amount is zero");
46     require(receiver_ != address(0), "NativeHandler: receiver is
↳ zero");
47
48     (bool success_, ) = payable(receiver_).call{value: amount_}("
↳ );
49
50     require(success_, "NativeHandler: failed to send eth");
51 }
```

Listing 9: contracts/handlers/ERC721Handler.sol (Line 71)

```
66 IERC721MintableBurnable erc721_ = IERC721MintableBurnable(token_);
67
68 if (isWrapped_) {
69     erc721_.mintTo(receiver_, tokenId_, tokenURI_);
70 } else {
71     erc721_.safeTransferFrom(address(this), receiver_, tokenId_);
72 }
```

Listing 10: contracts/handlers/ERC1155Handler.sol (Line 78)

```
73 IERC1155MintableBurnable erc1155_ = IERC1155MintableBurnable(
↳ token_);
74
75 if (isWrapped_) {
76     erc1155_.mintTo(receiver_, tokenId_, amount_, tokenURI_);
77 } else {
78     erc1155_.safeTransferFrom(address(this), receiver_, tokenId_,
↳ amount_, "");
79 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to protect against reentrancy attacks by using a mutex mechanism as mentioned above. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against the recursive calls.

Remediation Plan:

RISK ACCEPTED: The Rarimo team accepted the risk of this finding. The Rarimo team stated:

“The code is indeed re-entrant, however, there are no benefits for the attacker to actually execute the . would just increase the execution cost.”

3.5 (HAL-05) FLOATING PRAGMA - LOW

Description:

Smart contracts use the floating pragma ^0.8.9. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 11: contracts/*

```
2 pragma solidity ^0.8.9;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider locking the pragma version with known bugs for the compiler version by removing the caret (^) symbol. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

RISK ACCEPTED : The Rarimo team accepted the risk of this finding

3.6 (HAL-06) ZERO ADDRESS NOT CHECKED AFTER CREATE2 EXECUTION - INFORMATIONAL

Description:

The Bundler smart contracts make use of CREATE2 opcode to deploy a proxy smart contract in a pre-computable address by specifying the salt argument during the deployment.

This opcode can return zero address whether an error occurs during the construction of the smart contract or a smart contract has been already deployed in the pre-computed address. In the latter case, the same salt and deployment bytecode should be used.

Code Location:

Listing 12: contracts/bundle/Bundler.sol (Lines 22-24)

```
20     function _bundleUp(Bundle calldata bundle_) internal {
21         address payable executor = payable(
22             new BundleExecutorProxy{salt: bundle_.salt}(
23                 bundleExecutorImplementation,
24                 address(this)
25             )
26         );
27
28         BundleExecutorImplementation(executor).execute(bundle_.
↳ bundle);
29         BundleExecutorProxy(executor).destroy();
30     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to verify whether CREATE2 has been executed successfully by checking if the returned value is different from the zero address.

Remediation Plan:

NOT APPLICABLE: The Rarimo team stated that this issue is not applicable, since the high-level solidity salted creation reverts in case of failure.

3.7 (HAL-07) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL

Description:

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

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

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors.

Remediation Plan:

ACKNOWLEDGED : The Rarimo team acknowledged this issue. The Rarimo team will consider using custom errors in the future.

3.8 (HAL-08) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features of Solidity, and the static compiler. Since the EVM is a stack machine, it is often hard to address the correct stack slot and provide arguments to opcodes at the correct point on the stack. Solidity's inline assembly tries to facilitate that and other issues arising when writing manual assembly. Assembly is much more difficult to write because the compiler does not perform checks, so the developer of the contract should be aware of this warning.

Code Location:

Listing 13: contracts/bundle/proxy/BundleExecutorProxy.sol

```
20 assembly {
21     if iszero(eq(caller(), bridge_)) {
22         revert(0, 0)
23     }
24
25     selfdestruct(caller())
26 }
```

Listing 14: contracts/bundle/proxy/BundleExecutorProxy.sol

```
30 assembly {
31     calldatacopy(0, 0, calldatasize())
32
33     let result_ := delegatecall(gas(), implementation_, 0,
↳ calldatasize(), 0, 0)
34
35     returndatacopy(0, 0, returndatasize())
36
37     switch result_
```

```
38      case 0 {
39          revert(0, returndatasize())
40      }
41      default {
42          return(0, returndatasize())
43      }
44  }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The contracts should avoid using inline assembly because it interacts with the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many essential safety features of Solidity.

Remediation Plan:

NOT APPLICABLE : The Rarimo team acknowledged this issue. The Rarimo team considers this issue as an intended.

3.9 (HAL-09) MISSING ZERO ADDRESS CHECK - INFORMATIONAL

Description:

The Bridge smart contract is missing the zero address validation in critical setters such as changeSigner and changeBundleExecutorImplementation functions. It is possible to configure signer and bundleExecutorImplementation fields to point to the zero address, which may cause issues with contract execution.

Code Location:

Listing 15: contracts/bridge/Bridge.sol

```
195 function changeSigner(address newSigner_, bytes memory signature_)
 ↳  external {
196     _checkSignatureAndIncrementNonce(_getAddressChangeHash(
 ↳ newSigner_), signature_);
197
198     signer = newSigner_;
199 }
```

Listing 16: contracts/bridge/Bridge.sol

```
201 function changeBundleExecutorImplementation(
202     address newImplementation_,
203     bytes memory signature_
204 ) external {
205     _checkSignatureAndIncrementNonce(_getAddressChangeHash(
 ↳ newImplementation_), signature_);
206
207     bundleExecutorImplementation = newImplementation_;
208 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding a check to ensure signer and bundleExecutorImplementation addresses are different from the zero address.

Remediation Plan:

ACKNOWLEDGED : The Rarimo team acknowledged this issue. The Rarimo team will fix the issue in the future.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
NativeHandler._withdrawNative(bytes,address,bool) (contracts/handlers/NativeHandler.sol#42-51) sends eth to arbitrary user
        Dangerous calls:
        - (success_) = address(receiver_).call{value: amount_}() (contracts/handlers/NativeHandler.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Contract locking ether found:
        Contract BundleExecutorProxy (contracts/bundle/proxy/BundleExecutorProxy.sol#4-46) has payable functions:
        - BundleExecutorProxy.fallback() (contracts/bundle/proxy/BundleExecutorProxy.sol#13-15)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#92) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#65-74) ignores return value by Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#72)
ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#81-99) ignores return value by IERC1822Proxiable(newImplementation).proxiableUUID() (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#92-96)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#174-184) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Bundler.__Bundler_init(address).bundleExecutorImplementation_ (contracts/bundle/Bundler.sol#19) lacks a zero-check on :
                - bundleExecutorImplementation = bundleExecutorImplementation_ (contracts/bundle/Bundler.sol#20)
Signers.__Signers_init(address,string).signer_ (contracts/utils/Signers.sol#17) lacks a zero-check on :
                - signer = signer_ (contracts/utils/Signers.sol#18)
Bridge.changeSigner(address,bytes).newSigner_ (contracts/bridge/Bridge.sol#195) lacks a zero-check on :
                - signer = newSigner_ (contracts/bridge/Bridge.sol#198)
Bridge.changeBundleExecutorImplementation(address,bytes).newImplementation_ (contracts/bridge/Bridge.sol#202) lacks a zero-check on :
                - bundleExecutorImplementation = newImplementation_ (contracts/bridge/Bridge.sol#207)
BundleExecutorProxy.constructor(address,address)._implementation_ (contracts/bundle/proxy/BundleExecutorProxy.sol#8) lacks a zero-check on :
                - _IMPLEMENTATION = implementation_ (contracts/bundle/proxy/BundleExecutorProxy.sol#9)
BundleExecutorProxy.constructor(address,address)._bridge_ (contracts/bundle/proxy/BundleExecutorProxy.sol#8) lacks a zero-check on :
                - _BRIDGE = bridge_ (contracts/bundle/proxy/BundleExecutorProxy.sol#10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

BundleExecutorImplementation.execute(bytes) (contracts/bundle/proxy/BundleExecutorImplementation.sol#8-19) has external calls inside a loop: (success_) = address(contracts_[i]).call{value: values_[i]}(data_[i]) (contracts/bundle/proxy/BundleExecutorImplementation.sol#15)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-inside-a-loop

Variable 'ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#92)' in ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#81-99) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#93)
Variable 'ECDSA.tryRecover(bytes32,bytes).r (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#62)' in ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-86) potentially used before declaration: r = mload(uint256)(signature + 0x20) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in ERC115SHandler.depositERC1155(address,uint256,uint256,IBundler.Bundle,string,string,bool) (contracts/handlers/ERC115SHandler.sol#16-46):
        External calls:
        - erc115S_.burnFrom(msg.sender,tokenId_,amount_) (contracts/handlers/ERC115SHandler.sol#31)
        - erc115S_.safeTransferFrom(msg.sender,address(this),tokenId_,amount_,) (contracts/handlers/ERC115SHandler.sol#33)
        Event emitted after the call(s):
        - DepositedERC1155(token_,tokenId_,amount_,bundle_.salt.encode(),bundle_.bundle,network_,receiver_,isWrapped_) (contracts/handlers/ERC115SHandler.sol#36-45)
Reentrancy in ERC20Handler.depositERC20(address,uint256,IBundler.Bundle,string,string,bool) (contracts/handlers/ERC20Handler.sol#17-46):
        External calls:
        - erc20_.burnFrom(msg.sender,amount_) (contracts/handlers/ERC20Handler.sol#31)
        - erc20_.safeTransferFrom(msg.sender,address(this),amount_) (contracts/handlers/ERC20Handler.sol#33)
        Event emitted after the call(s):
        - DepositedERC20(token_,amount_,bundle_.salt.encode(),bundle_.bundle,network_,receiver_,isWrapped_) (contracts/handlers/ERC20Handler.sol#36-44)
Reentrancy in ERC721Handler.depositERC721(address,uint256,IBundler.Bundle,string,string,bool) (contracts/handlers/ERC721Handler.sol#16-43):
        External calls:
        - erc721_.burnFrom(msg.sender,tokenId_) (contracts/handlers/ERC721Handler.sol#29)
        - erc721_.safeTransferFrom(msg.sender,address(this),tokenId_) (contracts/handlers/ERC721Handler.sol#31)
        Event emitted after the call(s):
        - DepositedERC721(token_,tokenId_,bundle_.salt.encode(),bundle_.bundle,network_,receiver_,isWrapped_) (contracts/handlers/ERC721Handler.sol#34-42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
StorageSlot.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#52-56) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#53-55)
StorageSlot.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#61-65) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#62-64)
StorageSlot.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#70-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#71-73)
StorageSlot.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#79-83) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#80-82)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-86) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#67-71)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#78-81)
MerkleProof._efficientHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#58-64) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#59-63)
BundleExecutorProxy.destroy() (contracts/bundle/proxy/BundleExecutorProxy.sol#17-27) uses assembly
        - INLINE ASM (contracts/bundle/proxy/BundleExecutorProxy.sol#20-26)
BundleExecutorProxy._delegate(address) (contracts/bundle/proxy/BundleExecutorProxy.sol#29-45) uses assembly
        - INLINE ASM (contracts/bundle/proxy/BundleExecutorProxy.sol#30-44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Different versions of Solidity are used:
	- Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.9']
	- ^0.8.0 (node_modules/@openzeppelin/contracts/interfaces/draft-IERC1822.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#4)
	- ^0.8.2 (node_modules/@openzeppelin/contracts/proxy/beacon/IBeacon.sol#4)
	- ^0.8.2 (node_modules/@openzeppelin/contracts/proxy/utils/Initializable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/utils/ERC1155Receiver.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol#4)
	- ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
	- ^0.8.9 (contracts/bridge/Bridge.sol#2)
	- ^0.8.9 (contracts/bridge/proxy/UUPSSignableUpgradeable.sol#2)
	- ^0.8.9 (contracts/bundle/Bundler.sol#2)
	- ^0.8.9 (contracts/bundle/proxy/BundleExecutorImplementation.sol#2)
	- ^0.8.9 (contracts/bundle/proxy/BundleExecutorProxy.sol#2)
	- ^0.8.9 (contracts/handlers/ERC1155Handler.sol#2)
	- ^0.8.9 (contracts/handlers/ERC20Handler.sol#2)
	- ^0.8.9 (contracts/handlers/ERC721Handler.sol#2)
	- ^0.8.9 (contracts/handlers/NativeHandler.sol#2)
	- ^0.8.9 (contracts/interfaces/bridge/IBridge.sol#2)
	- ^0.8.9 (contracts/interfaces/bundle/IBundler.sol#2)
	- ^0.8.9 (contracts/interfaces/handlers/IERC1155Handler.sol#2)
	- ^0.8.9 (contracts/interfaces/handlers/IERC20Handler.sol#2)
	- ^0.8.9 (contracts/interfaces/handlers/IERC721Handler.sol#2)
	- ^0.8.9 (contracts/interfaces/handlers/INativeHandler.sol#2)
	- ^0.8.9 (contracts/interfaces/tokens/IERC1155MintableBurnable.sol#2)
	- ^0.8.9 (contracts/interfaces/tokens/IERC20MintableBurnable.sol#2)
	- ^0.8.9 (contracts/interfaces/tokens/IERC721MintableBurnable.sol#2)
	- ^0.8.9 (contracts/libs/Encoder.sol#2)
	- ^0.8.9 (contracts/utils/Hashes.sol#2)
	- ^0.8.9 (contracts/utils/Signers.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#130-136) is never used and should be removed
ECDSA.recover(bytes32,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#181-190) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#214-216) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#200-204) is never used and should be removed
ECDSA.toTypedDataHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#227-229) is never used and should be removed
ERC1155Handler._getERC1155TokenDataLeaf(bytes) (contracts/handlers/ERC1155Handler.sol#82-95) is never used and should be removed
ERC1967Upgrade._changeAdmin(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#133-136) is never used and should be removed
ERC1967Upgrade._getAdmin() (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#116-118) is never used and should be removed
ERC1967Upgrade._getBeacon() (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#152-154) is never used and should be removed
ERC1967Upgrade._setAdmin(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#123-126) is never used and should be removed
ERC1967Upgrade._setBeacon(address) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#159-166) is never used and should be removed
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#174-184) is never used and should be removed
ERC20Handler._getERC20TokenDataLeaf(bytes) (contracts/handlers/ERC20Handler.sol#76-84) is never used and should be removed
ERC721Handler._getERC721TokenDataLeaf(bytes) (contracts/handlers/ERC721Handler.sol#75-83) is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts/proxy/utils/Initializable.sol#129-131) is never used and should be removed
MerkleProof.verify(bytes32[],bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#27-33) is never used and should be removed
NativeHandler._getNativeTokenDataLeaf(bytes) (contracts/handlers/NativeHandler.sol#53-59) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
StorageSlot.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#70-76) is never used and should be removed
StorageSlot.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#79-83) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#48-51) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#56-66) is never used and should be removed
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#15-35) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/interfaces/draft-IERC1822.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/proxy/beacon/IBeacon.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC1155/utils/ERC1155Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version^0.8.9 (contracts/bridge/Bridge.sol#2) allows old versions
Pragma version^0.8.9 (contracts/bridge/proxy/UUPSSignableUpgradeable.sol#2) allows old versions
Pragma version^0.8.9 (contracts/bundle/Bundler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/bundle/proxy/BundleExecutorImplementation.sol#2) allows old versions
Pragma version^0.8.9 (contracts/bundle/proxy/BundleExecutorProxy.sol#2) allows old versions
Pragma version^0.8.9 (contracts/handlers/ERC1155Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/handlers/ERC20Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/handlers/ERC721Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/handlers/NativeHandler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/bridge/IBridge.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/bundle/IBundler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/handlers/IERC1155Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/handlers/IERC20Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/handlers/IERC721Handler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/handlers/INativeHandler.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/tokens/IERC1155MintableBurnable.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/tokens/IERC20MintableBurnable.sol#2) allows old versions
Pragma version^0.8.9 (contracts/interfaces/tokens/IERC721MintableBurnable.sol#2) allows old versions
Pragma version^0.8.9 (contracts/libs/Encoder.sol#2) allows old versions
Pragma version^0.8.9 (contracts/utils/Hashes.sol#2) allows old versions
Pragma version^0.8.9 (contracts/utils/Signers.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
	- (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
	- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
	- (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
	- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Low level call in BundleExecutorImplementation.execute(bytes) (contracts/bundle/proxy/BundleExecutorImplementation.sol#8-19):
	- (success_) = address(contracts_[i]).call{value: values_[i]}(data_[i]) (contracts/bundle/proxy/BundleExecutorImplementation.sol#15)
Low level call in NativeHandler._withdrawNative(bytes,address,bool) (contracts/handlers/NativeHandler.sol#42-51):
	- (success_) = address(receiver_).call{value: amount_}() (contracts/handlers/NativeHandler.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol#23) is not in mixedCase
Function Bridge.__Bridge_init(address,address,string) (contracts/bridge/Bridge.sol#31-38) is not in mixedCase
Function Bundler.__Bundler_init(address) (contracts/bundle/Bundler.sol#19-21) is not in mixedCase
Variable BundleExecutorProxy._IMPLEMENTATION (contracts/bundle/proxy/BundleExecutorProxy.sol#6) is not in mixedCase
Variable BundleExecutorProxy._BRIDGE (contracts/bundle/proxy/BundleExecutorProxy.sol#8) is not in mixedCase
Parameter Encoder.encode(function(bytes) returns(bytes),bytes,Bundler.bundle_bytes3,string,address)._getTokenDataLeaf (contracts/libs/Encoder.sol#12) is not in mixedCase
Function Signers.__Signers_init(address,string) (contracts/utils/Signers.sol#17-20) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable BundleExecutorProxy._IMPLEMENTATION (contracts/bundle/proxy/BundleExecutorProxy.sol#5) is too similar to BundleExecutorProxy.constructor(address,address).implementation_ (contracts/bundle/proxy/BundleExecutorProxy.sol#8)
Variable BundleExecutorProxy._IMPLEMENTATION (contracts/bundle/proxy/BundleExecutorProxy.sol#5) is too similar to BundleExecutorProxy._delegate(address).implementation_ (contracts/bundle/proxy/BundleExecutorProxy.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

Bundler.determineProxyAddress(bytes32) (contracts/bundle/Bundler.sol#35-56) uses literals with too many digits:
	- address(uint160(uint256(keccak256(bytes)(abi.encodePacked(bytes1(0xff),address(this),salt_,keccak256(bytes)(abi.encodePacked(type()(BundleExecutorProxy).creationCode,abi.encode(bundleExecutorImplementation,address(this))))))))) (contracts/bundle/Bundler.sol#36-56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

- All the re-entrancy issues were checked individually, and those not described above in the report do not pose any risk.
- Contract locking ether issue is a false positive since this smart contract is destroyed after its execution.
- No major issues found by Slither.

THANK YOU FOR CHOOSING

// HALBORN