deBridge - Solidity Contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 25th, 2022 - April 20th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/22/2022	Mustafa Hasan
0.2	Document Edits	04/22/2022	Mustafa Hasan
0.3	Document Edits	04/25/2022	Mustafa Hasan
0.5	Draft Review	04/25/2022	Timur Guvenkaya
0.5	Draft Final Review	04/22/2022	Gabi Urrutia
1.0	Remediation Plan	05/19/2022	Mustafa Hasan
1.1	Remediation Plan Review	05/25/2022	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
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com
Mustafa Hasan	Halborn	Mustafa.Hasan@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

deBridge engaged Halborn to conduct a security assessment on their smart contracts beginning on March 25th, 2022 and ending April 20th, 2022. deBridge is a cross-chain interoperability and liquidity transfer protocol that allows truly decentralized transfer of assets between various blockchains.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned one full-time security engineer to audit the security of the assets in scope. The 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 achieve the following:

- Identify potential security issues with the smart contracts.

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

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual testing of core functions through Hardhat and Ganache
- Manual testing with custom scripts.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Testnet deployment (Remix IDE)

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

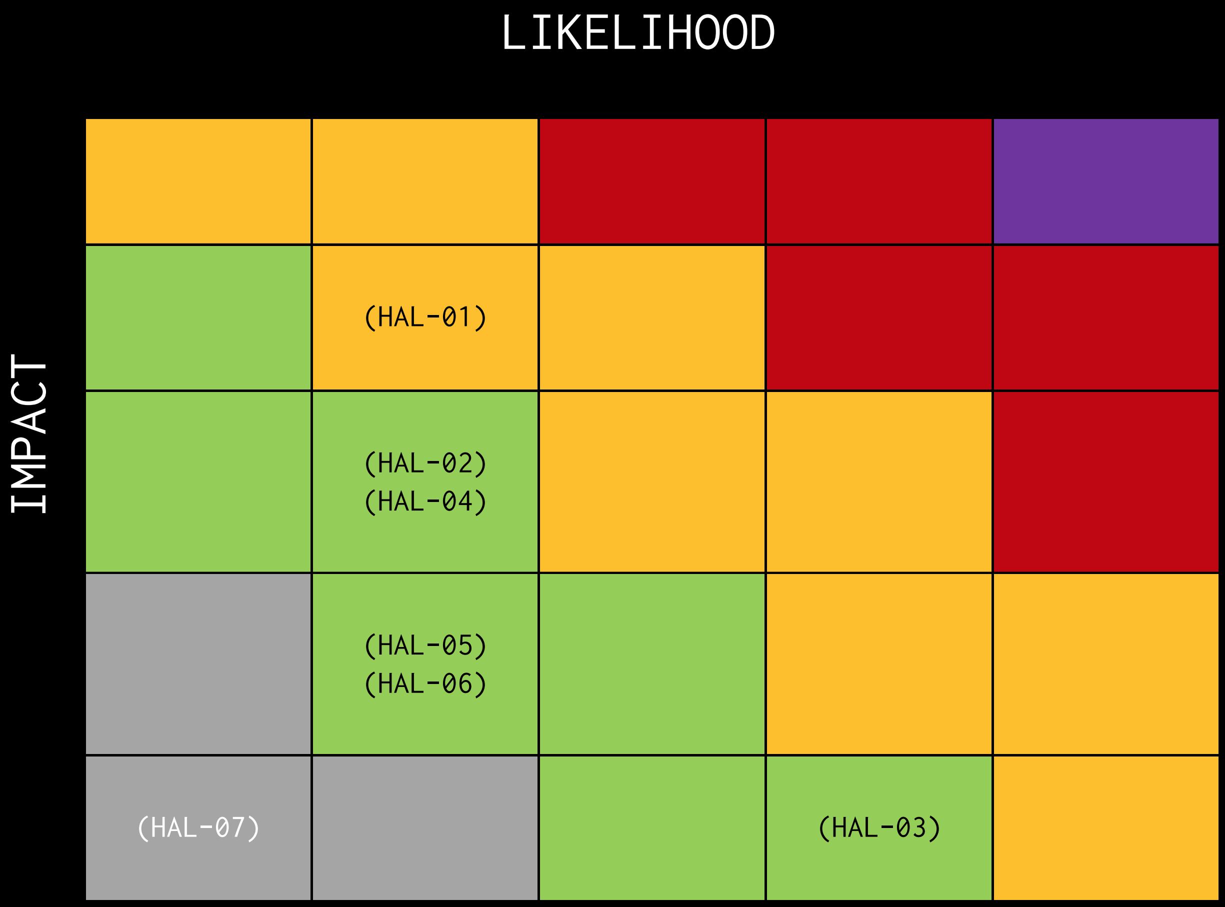
The review was scoped to contracts and scripts in the following:
master branch on https://github.com/debridge-finance/debridge-contracts-develop

Smart contracts:

- DeBridgeGate.sol
- DeBridgeTokenDeployer.sol
- OraclesManager.sol
- SignatureVerifier.sol
- WethGate.sol
- BatchBalance.sol
- CallProxy.sol
- DeBridgeToken.sol
- DefiController.sol
- FreeProxy.sol
- FeesCalculator.sol
- PriceConsumer.sol
- SimpleFeeProxy.sol
- SwapProxy.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	5	1



🟨	🟨	🟥	🟥	🟪
🟩	🟨 (HAL-01)	🟨	🟥	🟥
🟩	🟩 (HAL-02) (HAL-04)	🟨	🟨	🟥
⬜	🟩 (HAL-05) (HAL-06)	🟩	🟨	🟨
⬜ (HAL-07)	⬜	🟩	🟩 (HAL-03)	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 ADMIN CAN BE SET AS A MINTER	Medium	SOLVED - 05/17/2022
HAL-02 THE DOMAIN SEPARATOR IS NOT RECALCULATED AFTER A HARD FORK HAPPENS	Low	FUTURE RELEASE
HAL-03 LACK OF ZERO ADDRESS CHECK	Low	NOT APPLICABLE
HAL-04 LACK OF MINIMUM REQUIRED ORACLES CHECK	Low	NOT APPLICABLE
HAL-05 LACK OF SEPARATION OF PRIVILEGES FOR ADMIN AND PAUSER	Low	RISK ACCEPTED
HAL-06 FLOATING PRAGMA	Low	SOLVED - 05/17/2022
HAL-07 USE OF BLOCK.TIMESTAMP	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) HAL-01 ADMIN CAN BE SET AS A MINTER - MEDIUM

Description:

The initialize() function takes a list of minters and assigns them the token minting privileges; however, there are no checks to make sure that the address of the contract admin is not included in the list of minters. This defeats the separation of privileges and decentralization principles.

Additionally, the admin can pass an empty list of minters, effectively rendering the burn() function useless.

Code Location:

Listing 1: contracts/periphery/DeBridgeToken.sol (Lines 71,72)

```
54 function initialize(
55     string memory name_,
56     string memory symbol_,
57     uint8 decimals_,
58     address admin,
59     address[] memory minters
60 ) public initializer {
61     _decimals = decimals_;
62     name_ = string(abi.encodePacked("deBridge ",
63         bytes(name_).length == 0 ? symbol_ : name_));
64     symbol_ = string(abi.encodePacked("de", symbol_));
65
66     __ERC20_init_unchained(name_, symbol_);
67
68     _setupRole(DEFAULT_ADMIN_ROLE, admin);
69     _setupRole(PAUSER_ROLE, admin);
70     uint256 mintersCount = minters.length;
71     for (uint256 i = 0; i < mintersCount; i++) {
72         _setupRole(MINTER_ROLE, minters[i]);
73     }
74 ...
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Separation of privileges should be implemented through logic that makes sure the roles are assigned to different addresses.

Remediation Plan:

SOLVED: The deBridge team solved the issue in commit 1849deeb0896761fb6baf5931afb670129

3.2 (HAL-02) HAL-02 THE DOMAIN SEPARATOR IS NOT RECALCULATED AFTER A HARD FORK HAPPENS - LOW

Description:

The variable DOMAIN_SEPARATOR in DeBridgeToken.sol is cached in the contract storage and will not change after the contract is initialized. However, if a hard fork happens after the contract deployment, the DOMAIN_SEPARATOR would become invalid on one of the forked chains due to the chainId has changed.

Code Location:

Listing 2: contracts/periphery/DeBridgeToken.sol (Lines 75,79)

```
54 function initialize(
55         string memory name_,
56         string memory symbol_,
57         uint8 decimals_,
58         address admin,
59         address[] memory minters
60     ) public initializer {
61         _decimals = decimals_;
62         name_ = string(abi.encodePacked("deBridge ",
63             bytes(name_).length == 0 ? symbol_ : name_));
64         symbol_ = string(abi.encodePacked("de", symbol_));
65
66         __ERC20_init_unchained(name_, symbol_);
67
68         _setupRole(DEFAULT_ADMIN_ROLE, admin);
69         _setupRole(PAUSER_ROLE, admin);
70         uint256 mintersCount = minters.length;
71         for (uint256 i = 0; i < mintersCount; i++) {
72             _setupRole(MINTER_ROLE, minters[i]);
73         }
74
75         uint256 chainId;
76         assembly {
```

```
            chainId := chainid()
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(
                    "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
                ),
                keccak256(bytes(name_)),
                keccak256(bytes("1")),
                chainId,
                address(this)
            )
        );
    }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider using the implementation from OpenZeppelin, which recalculates the domain separator if the current chainid is not the cached chain ID.

Remediation Plan:

PENDING: The deBridge team will deploy a fix in the next protocol upgrade.

3.3 (HAL-03) HAL-03 LACK OF ZERO ADDRESS CHECK - LOW

Description:

Multiple functions are used to set addresses by an admin; however, no checks exist to confirm an address is not the zero address before being set. Affected functions include setTreasury() and setFeeProxyAddress().

Code Location:

Listing 3: contracts/periphery/SimpleFeeProxy.sol

```
function setTreasury(address _treasury) external onlyAdmin {
    treasury = _treasury;
}
```

Listing 4: contracts/periphery/FeeProxy.sol

```
function setFeeProxyAddress(uint256 _chainId, bytes memory _address) external onlyAdmin {
    feeProxyAddresses[_chainId] = _address;
}
```

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

Checks should be employed to make sure the zero address cannot be set as the values of said items.

Remediation Plan:

NOT APPLICABLE: The deBridge team confirmed this is intentional and that their logic accounts for such cases without breaking.

3.4 (HAL-04) HAL-04 LACK OF MINIMUM REQUIRED ORACLES CHECK - LOW

Description:

The requiredOraclesCount value is never checked to make sure it doesn't fall below a certain acceptable value. This allows an admin to call addOracles() with a list of oracles that doesn't include any required oracle instances, and signature verification will then pass without needing any required oracles.

Code Location:

Listing 5: contracts/transfers/OraclesManager.sol

```
function addOracles(
    address[] memory _oracles,
    bool[] memory _required
) external onlyAdmin {
    if (_oracles.length != _required.length) revert WrongArgument();
    if (minConfirmations < (oracleAddresses.length + _oracles.length) / 2 + 1) revert LowMinConfirmations();

    for (uint256 i = 0; i < _oracles.length; i++) {
        OracleInfo storage oracleInfo = getOracleInfo[_oracles[i]];
        if (oracleInfo.exist) revert OracleAlreadyExist();

        oracleAddresses.push(_oracles[i]);

        if (_required[i]) {
            requiredOraclesCount += 1;
        }

        oracleInfo.exist = true;
        oracleInfo.isValid = true;
        oracleInfo.required = _required[i];

        emit AddOracle(_oracles[i], _required[i]);
    }
}
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

A minimum value must be implemented and checked against to make sure the number of required oracles at least meets that value.

Remediation Plan:

NOT APPLICABLE: The deBridge team confirmed this is intentional, as the requiredOraclesCount calculation depends on OracleInfo.

3.5 (HAL-05) HAL-05 LACK OF SEPARATION OF PRIVILEGES FOR ADMIN AND PAUSER - LOW

Description:

The initialize() function takes an admin address and sets it as the admin as well as the pauser of the contract. This is bad practice since it defeats the concept of separation of privileges and decentralization.

The same is valid for “contracts/periphery/SimpleFeeProxy.sol”

Code Location:

Listing 6: contracts/periphery/DeBridgeToken.sol, (Lines 68,69)

```
54 function initialize(
55     string memory name_,
56     string memory symbol_,
57     uint8 decimals_,
58     address admin,
59     address[] memory minters
60 ) public initializer {
61     _decimals = decimals_;
62     name_ = string(abi.encodePacked("deBridge ",
63         bytes(name_).length == 0 ? symbol_ : name_));
64     symbol_ = string(abi.encodePacked("de", symbol_));
65
66     __ERC20_init_unchained(name_, symbol_);
67
68     _setupRole(DEFAULT_ADMIN_ROLE, admin);
69     _setupRole(PAUSER_ROLE, admin);
70     uint256 mintersCount = minters.length;
71     for (uint256 i = 0; i < mintersCount; i++) {
72         _setupRole(MINTER_ROLE, minters[i]);
73     }
74
75     uint256 chainId;
76     assembly {
```

```
            chainId := chainid()
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(
                    "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
                ),
                keccak256(bytes(name_)),
                keccak256(bytes("1")),
                chainId,
                address(this)
            )
        );
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Separation of the admin and pauser roles should be implemented.

Remediation Plan:

RISK ACCEPTED: The deBridge team confirmed this is intentional and admins are supposed to be pausers as well.

3.6 (HAL-06) HAL-06 FLOATING PRAGMA - LOW

Description:

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, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Code Location:

- BytesLib.sol

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Lock the pragma version and also consider known bugs (https://github.com/ethereum/soli
for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma to compile locally.

Remediation Plan:

SOLVED: The deBridge team solved the issue in commit 2158d8b866a60890a3770bab685388751

3.7 (HAL-07) HAL-07 USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

block.timestamp can be influenced by miners to a certain degree, so developers should be aware that this may have some risk if miners collude on time manipulation to influence the price oracles.

Code Location:

Listing 7: contracts/periphery/DeBridgeToken.sol, (Line 119)

```
110 function permit(
111     address _owner,
112     address _spender,
113     uint256 _value,
114     uint256 _deadline,
115     uint8 _v,
116     bytes32 _r,
117     bytes32 _s
118 ) external override {
119     require(_deadline >= block.timestamp, "permit: EXPIRED");
120     bytes32 digest = keccak256(
121         abi.encodePacked(
122             "\x19\x01",
123             DOMAIN_SEPARATOR,
124             keccak256(
125                 abi.encode(
126                     PERMIT_TYPEHASH,
127                     _owner,
128                     _spender,
129                     _value,
130                     nonces[_owner]++,
131                     _deadline
132                 )
133             )
134         )
135     );
136     address recoveredAddress = ecrecover(digest, _v, _r, _s);
137     require(
        recoveredAddress != address(0) && recoveredAddress == _owner,
        "permit: invalid signature"
    );
    _approve(_owner, _spender, _value);
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of Maximal Extractable Value (MEV) attacks. Check if the timescale of the project occurs across years, days, and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

ACKNOWLEDGED: The deBridge team acknowledged the issue.

3.8 AUTOMATED SECURITY SCAN

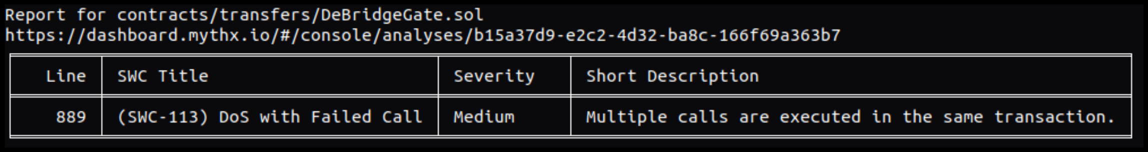
Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, only security findings are considered as in-scope.

Results:

All the findings were either detected during manual code review or false positive. Only results with findings are shown below.

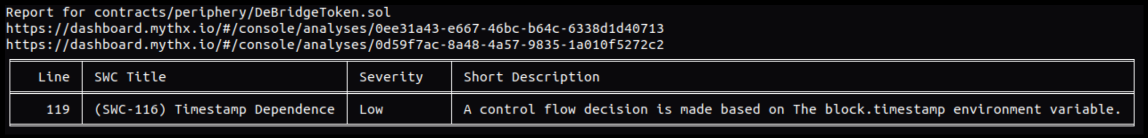
- DeBridgeGate.sol



```
Report for contracts/transfers/DeBridgeGate.sol
https://dashboard.mythx.io/#/console/analyses/b15a37d9-e2c2-4d32-ba8c-166f69a363b7
```

Line	SWC Title	Severity	Short Description
889	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.

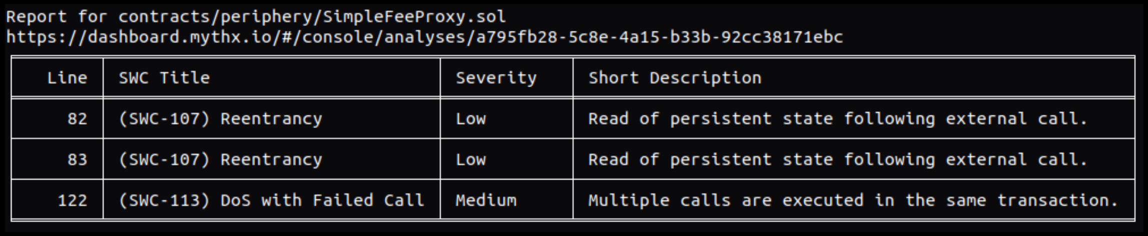
- DeBridgeToken.sol



```
Report for contracts/periphery/DeBridgeToken.sol
https://dashboard.mythx.io/#/console/analyses/0ee31a43-e667-46bc-b64c-6338d1d40713
https://dashboard.mythx.io/#/console/analyses/0d59f7ac-8a48-4a57-9835-1a010f5272c2
```

Line	SWC Title	Severity	Short Description
119	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

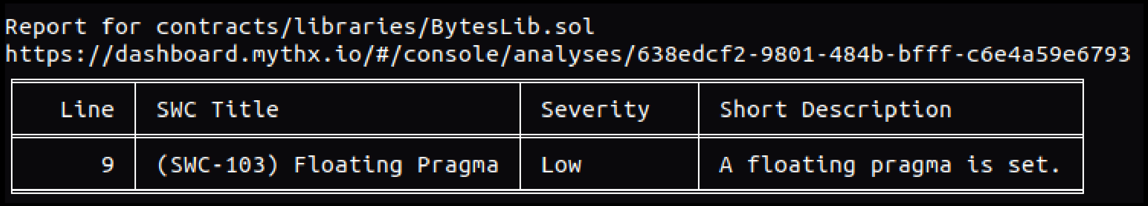
- SimpleFeeProxy.sol



```
Report for contracts/periphery/SimpleFeeProxy.sol
https://dashboard.mythx.io/#/console/analyses/a795fb28-5c8e-4a15-b33b-92cc38171ebc
```

Line	SWC Title	Severity	Short Description
82	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
83	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
122	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.

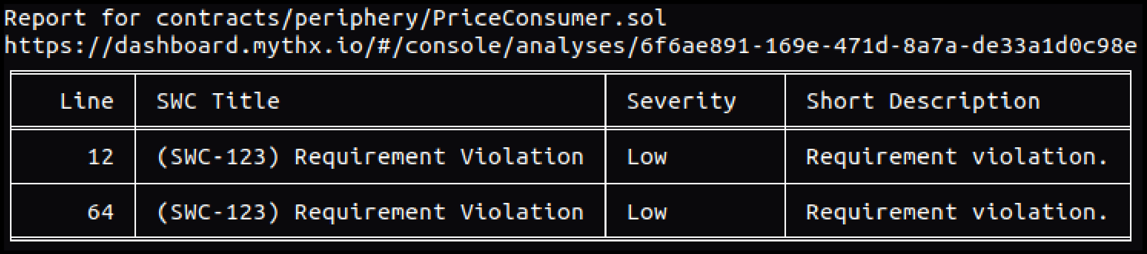
- BytesLib.sol



```
Report for contracts/libraries/BytesLib.sol
https://dashboard.mythx.io/#/console/analyses/638edcf2-9801-484b-bfff-c6e4a59e6793
```

Line	SWC Title	Severity	Short Description
9	(SWC-103) Floating Pragma	Low	A floating pragma is set.

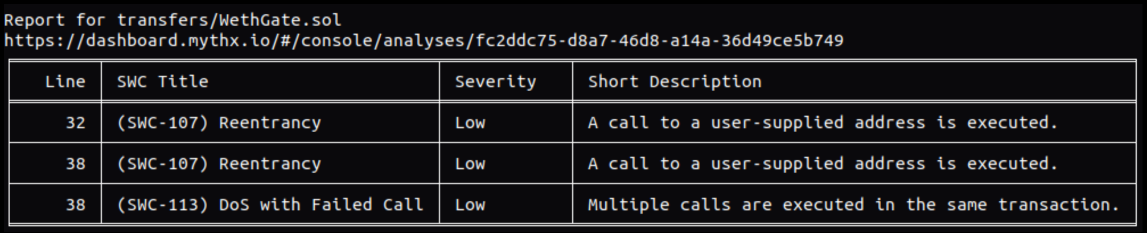
- PriceConsumer.sol



Report for contracts/periphery/PriceConsumer.sol
https://dashboard.mythx.io/#/console/analyses/6f6ae891-169e-471d-8a7a-de33a1d0c98e

Line	SWC Title	Severity	Short Description
12	(SWC-123) Requirement Violation	Low	Requirement violation.
64	(SWC-123) Requirement Violation	Low	Requirement violation.

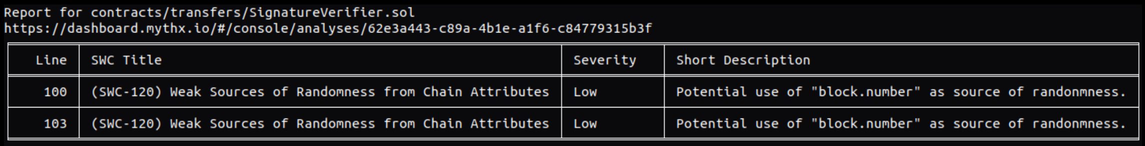
- WethGate.sol



Report for transfers/WethGate.sol
https://dashboard.mythx.io/#/console/analyses/fc2ddc75-d8a7-46d8-a14a-36d49ce5b749

Line	SWC Title	Severity	Short Description
32	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
38	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
38	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

- SignatureVerifier.sol



Report for contracts/transfers/SignatureVerifier.sol
https://dashboard.mythx.io/#/console/analyses/62e3a443-c89a-4b1e-a1f6-c84779315b3f

Line	SWC Title	Severity	Short Description
100	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
103	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

THANK YOU FOR CHOOSING

HALBORN