Code security review

Pirate wallet

20211201

Contents

1 Summary **3**

2 Security observations **3**

2.1 wallet/wallet.cpp . . . 3
2.1.1 1: Lock instead of Unlock? . . . 3
2.1.2 2: Null pointer checks . . . 3
2.1.3 3: Missing length check? . . . 4
2.1.4 4: Int overflow . . . 4
2.1.5 5: Optimization . . . 4
2.1.6 6: Log value typo . . . 4
2.2 wallet/rpcdump.cpp . . . 4
2.2.1 7: Missing failure check . . . 4
2.3 wallet/rpcpiratewallet.cpp . . . 5
2.3.1 8: Possible underflow? . . . 5
2.3.2 9: Decryption failures not handled . . . 5
2.4 wallet/rpcwallet.cpp . . . 5
2.4.1 10: Missing address check . . . 5
2.4.2 11: Integer validation . . . 5
2.4.3 12: Lossy cast . . . 5
2.4.4 13: Possible int over/underflows . . . 5
2.5 wallet/crypter.cpp . . . 6
2.5.1 14: Key derivation issues . . . 6
2.5.2 15: Encryption optimization and risks . . . 6
2.5.3 16: Offline seed search attacks possible . . . 6
2.5.4 17: Sensitive data cleansing . . . 6
2.6 init.cpp . . . 6
2.6.1 18: Make encryption a default . . . 6
2.6.2 19: No failure on critical errors . . . 7
2.7 wallet/asyncrpcoperation_saplingconsolidation.cpp . . . 7
2.7.1 20: Possible minQuantity > maxQuantity . . . 7

1 Summary

We reviewed the critical components of PirateNetwork's wallet, looking for bugs in terms of cryptography and software safety. Most of the work consisted in manual code review, but we also use some automated tools (cppcheck and clang analyzers), which, unsurprisingly, did not report anything of interest. We focused on the Pirate-specific code, as opposed to the Zcash and Komodo basis.

Note that we did not cover side-channel risks (timing attacks, oracle attacks, heap/stack cleansing). For example, we noted some usage of `memory_cleanse()`, but not everywhere where it might be needed to erase sensitive values from the program's memory. For example, `CWallet::EncryptWallet()` does not cleanse the sensitive values (master key, encrypted values, etc.).

Furthermore, the goal of the audit was not to match the code against specifications, or to assess the Pirate protocol design's soundness.

We also did not review third party dependencies, but noted that the Rust part used 9 outdated dependencies with known vulnerabilities, as reported by `cargo audit`. This can easily be fixed by updating the concerned dependencies.

2 Security observations

We break down our reporting in subsections corresponding to each source file analyzed, since the files are relatively long. A number of the points reported are not outright bugs, but often proposals of improvements, suggestions, or points that where unclear to use an potentially bugs.

2.1 wallet/wallet.cpp

2.1.1 1: Lock instead of Unlock?

In `CWallet::EncryptWallet()`, shouldn't this `Lock()` be an `Unlock()`?

2.1.2 2: Null pointer checks

Most functions taking pointers as arguments don't check that they are non-NULL; not a bug in itself if the pre-condition is satisfied, but may be leveraged to amplify an insecure state where such pointers happen to be NULLified by an attacker.

2.1.3 3: Missing length check?

In `CWallet::GetSproutNoteWitnesses()` the vector `notes` and `witnesses` should have the same length, but it's not checked, see wallet/wallet.cpp#L3731.

Also, in the same function, this part looks off to me: https://github.com/PirateNetwork/pirate/blob/469f7b07c2ad9b2f5280cd6546775049e0c6a331/src/wallet/wallet.cpp#L3736-L3739. Same comments apply to the Sapling version in https://github.com/PirateNetwork/pirate/blob/469f7b07c2ad9b2f5280cd6546775049e0c6a331/src/wallet/wallet.cpp#L3750.

2.1.4 4: Int overflow

There is a potential int overflow here if `GetCredit()` results can be manipulated, also in `GetChange()`, `GetDebit()`, `GetAccountAmounts()`, later in wallet/wallet.cpp#L5306, and wallet/wallet.cpp#L5337, too, in wallet/wallet.cpp#L5402, and wallet/wallet.cpp#L5884.

2.1.5 5: Optimization

Why not checking directly that `txWrite != 0` here?, (instead of checking it at the end after doing potential `updateArxTxPt`).

2.1.6 6: Log value typo

Typo "Positon" in a log message, might prevent searches from retrieving all the results.

2.2 wallet/rpcdump.cpp

2.2.1 7: Missing failure check

In wallet/rpcdump.cpp#L137, `DecodeSecret()` might have failed (by returning an empty string), which will cause a failure of `tempkey.GetPubKey()`, but it might be cleaner to just check if `tempkey` is empty or not.

2.3 wallet/rpcpiratewallet.cpp

2.3.1 8: Possible underflow?

Is it guaranteed that `transparentValue -= parentOut.nValue;` will not underflow? Same comment here.

2.3.2 9: Decryption failures not handled

Decryption failures are not processed as errors but just discarded in `getSaplingSends()`, in its other definition too), and in `getSaplingReceives()`. Failures are not detected at all in 2 Sprout processing decryptions. (Decryption would return `boost::none` if decryption fails, because the authenticity tag could be invalid.)

2.4 wallet/rpcwallet.cpp

2.4.1 10: Missing address check

In `SendMoney()`, the address doesn't seem checked to be correct (if incorrect the tx may be rejected).

2.4.2 11: Integer validation

In several places, the integer taken from the parameters received via `get_int()` is not validated to be an acceptable value (mainly a valid min depth). For example here.

2.4.3 12: Lossy cast

Potentially lossy 64-to-32 cast, unless total guarantee of no overflow, better check that `p[1]` is smaller than `UINT32_MAX`.

2.4.4 13: Possible int over/underflows

Prevent int over/underflows in: L1084-1095, L1271-1281, L1457, L2059-2068, L3965, L5837, and potentially others (I tried to single out the riskiest ones).

2.5 wallet/crypter.cpp

2.5.1 14: Key derivation issues

In `CCrypter::SetKeyFromPassphrase()`:

- Validate `strKeyData.size()` to be greater than some minimal length (done somewhere else already maybe?).
- "Unless you have a specific need, you should not use `OPENSSL_EVP_BytesToKey`. Rather, you should use a key derivation function like HKDF or Scrypt." from https://www.cryptopp.com/wiki/OPENSSL_EVP_BytesToKey.

2.5.2 15: Encryption optimization and risks

In `CCrypter::Encrypt()`:

- Safer to use authenticated encryption (as AES-GCM), which will directly detect corrupted ciphertext and protect their authenticity.
- The same IV is used for all encryptions, which is fine if a given key+salt is used only once (or always for the same plaintext/ciphertext). Looks like it's the case, but be careful otherwise.

2.5.3 16: Offline seed search attacks possible

In wallet/crypter.cpp#151, using the seed fingerprint/hash as a (non-secret) IV is *in theory* a problem, because it leaks info about the seed (its hash), making it trivial to test if a given HDSeed is the secret one. But here an attacker can verify a seed's guess by computing addresses/pubkeys via BIP32 anyway.

2.5.4 17: Sensitive data cleansing

Here you zeroize sensitive values, but for consistency is should also be done at L147, L160, (and `vchSecret` again in subsequent decryption functions), in L256.

2.6 init.cpp

2.6.1 18: Make encryption a default

Currently "Wallet encryption requires -experimentalfeatures.", as supported via that the `-developerencryptwallet` option. We recommend to make wallet encryption the default behavior,

if deemed mature enough.

2.6.2 19: No failure on critical errors

I suspect this is a design choice, but I don't see why the application does not return and abort upon critical errors here, when error messages are written to `strErrors`.

2.7 wallet/asyncrpcoperation_saplingconsolidation.cpp

2.7.1 20: Possible minQuantity > maxQuantity

In the loop going over Sapling addresses, the bounds on the number of notes can be (randomly) picked as inconsistent:

```
            int maxQuantity = rand() % 35 + 10;

(...)


                if (fromNotes.size() >= maxQuantity)
                  break;

            }

            //random minimum 2 - 12 required
            int minQuantity = rand() % 10 + 2;
            if (fromNotes.size() < minQuantity)
              continue;
```

So we can for example have `minQuantity == 11` and `maxQuantity == 10`. However in this case, the `continue` will be hit. It is unclear to us whether such a state is safe or not.