SmartDec

Grin++ Security Analysis

This report is public.

Published: March 4, 2020

Abstract

In this report, we consider the security of the Grin++ project. Our task is to find and describe security issues in the code base of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code base. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of Grin++ code base. We performed our audit according to the procedure described below.

The first stage of the audit showed many issues of various severity levels. We worked through these issues with the developers, so in the latest version of the code they fixed most of the issues found during the audit. For the issues that were not fixed the developers provided the comments, which can be found in the report.

General recommendations

- Never use `new` or `delete` explicitly, unless it's required by third party API. Use RAII and smart pointer factory methods.
- Prefer using standard and reliable and well-tested third-party components instead of shipping an own implementation whenever it's possible.
- Make sure making all destructors and thread methods not throwing.
- Do not utilize `std::is_base_of` for dynamic dispatch, use RTTI.
- Enable all and extra warnings and make sure eliminating them.
- Cover the code base with unit and integration tests.
- Do not ignore return values and error codes, process them properly.
- Thoroughly validate data from untrusted sources.
- Avoid using C APIs whenever it's possible, utilize safe alternatives, whenever they are available. Profile first before doing optimizations.
- Prepare the code for the audit properly, i.e. make sure that the implementation is stable and complete, is missing significant `TODO`-s and is not a subject to major changes. The work done by the audit in the other case has much less value for the final product.
- Make sure always utilizing standard methods for working with file paths, instead of treating them as strings.
- Ensure transactionality of important operations, which should be committed/rollbacked atomically.
- Keep dependencies up to date.
- Use `override` to mark virtual overriding methods.
- Use `noexcept` instead of `throw()`.
- Employ rule of three (rule of five) for all classes to ensure correct behavior.
- Use assertions to denote program invariants and catch their violations.
- Consider compiling and testing with sanitizers for debug nodes.
- Always keep third party libraries up to date.
- Determine logically connected instances and ensure synchronized access to data, with enough granularity, i.e. ensure logical synchronization.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- automated analysis
  - we scan project's code base with SmartDec Scanner
  - we run a set of publicly available automated C++ dynamic checkers on project's executables such as sanitizers
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze code base for security vulnerabilities
  - we check protocol logic and compare it with the one described in the documentation
  - we run tests
- report
  - we reflect all the gathered information in the report

Project overview

Project description

In our analysis we consider Mimblewimble protocol specification and Grin++ code, version on commit a04fe64fd8912bb3d6831713e01591b5db401a15.

After the first stage of the audit, many issues were found. To address them, some fixes were applied and the code was updated to version on commit 40ecfa56854882f561e8af2b4f56d2ebee5abdb2. After that, the code base was entirely reanalyzed.

The latest version of the code

After the final audit, some fixes were applied, and the code was updated to the latest version on commit 9b00b73316cbee7726524b1d862610ad33a003aa.

Project architecture

For the audit, we were provided with a CMake project. The project had no tests.

After several fixes, the project successfully compiles on macOS with CMake compile command.

The total LOC of audited C++ sources without dependencies is 26547.

Manual analysis, stage I

The code base on commit a04fe64fd8912bb3d6831713e01591b5db401a15 was completely manually analyzed, its logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

include/Crypto/BigInteger.h

Data corruption

line 75

```
        std::vector<unsigned char> bytes(2);
        memcpy(&bytes, (unsigned char*)&bigEndian, 2);
```

should be

```
        std::vector<unsigned char> bytes(2);
        memcpy(&bytes[0], (unsigned char*)&bigEndian, 2);
```

Array access out of bounds

lines 218, 235:

```
template<size_t NUM_BYTES, class ALLOC>
CBigInteger<NUM_BYTES, ALLOC> CBigInteger<NUM_BYTES, ALLOC>::FromHex(const
std::string& hex)
{
    // TODO: Verify input size
    size_t index = 0;
    if (hex[0] == '0' && hex[1] == 'x')
    {
        index = 2;
    }

    std::string hexNoSpaces = "";
    for (size_t i = index; i < hex.length(); i++)
    {
        if (hex[i] != ' ')
        {
            hexNoSpaces += hex[i];
        }
    }

    std::vector<unsigned char, ALLOC> data(NUM_BYTES);
    for (size_t i = 0; i < hexNoSpaces.length(); i += 2)
    {
        data[i / 2] = (FromHexChar(hexNoSpaces[i]) * 16 +
FromHexChar(hexNoSpaces[i + 1]));
    }
    return CBigInteger<NUM_BYTES, ALLOC>(&data[0]);
}
```

src/PMMR/Common/MMRUtil.cpp

Data loss shifts

Shifting 32 bit value `1` instead of `1ULL` (lines 51, 63, 68, 75):

```
        return mmrIndex + (1 << (height + 1));
```

src/PMMR/Common/PruneList.cpp

Data loss shifts

Shifting 32 bit value `1` instead of `1ULL` (lines 201, 206):

```
        const uint64_t currentShift = 2 * ((1 << height) - 1);
```

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

include/BlockChain/BlockChainServer.h

Polymorphic class `IBlockChainServer` is missing virtual destructor.

include/Common/Secure.h

Unimplemented method logic

line 17:

```
static void LOCK_MEMORY(void* pMemory, size_t size)
{
#if defined(_MSC_VER)
    VirtualLock(pMemory, size);
#else
// TODO:
#endif
}
```

line 26:

```
static void UNLOCK_MEMORY(void* pMemory, size_t size)
{
#if defined(_MSC_VER)
    VirtualUnlock(pMemory, size);
#else
// TODO:
#endif
}
```

Handcrafted `memset_s` implementation

line 47:

```
static void cleanse(void *ptr, size_t len)
{
    std::memset(ptr, 0, len);
    /* As best as we can tell, this is sufficient to break any
optimisations that
       might try to eliminate "superfluous" memsets. If there's an easy way
to
       detect memset_s, it would be better to use that. */
#if defined(_MSC_VER)
    SecureZeroMemory(ptr, len);
#else
    __asm__ __volatile__("" : : "r"(ptr) : "memory");
#endif
}
```

include/Common/Util/FileUtil.h

Invalid logic

`tellg` in not guaranteed to return file size, line 34:

```
    static bool ReadFile(const std::string& filePath, std::vector<unsigned
char>& data)
    {
        if (!fs::exists(fs::path(filePath)))
        {
            return false;
        }

        std::ifstream file(filePath, std::ios::in | std::ios::binary |
std::ios::ate);
        if (!file.is_open())
        {
            return false;
        }
        auto size = file.tellg();
```

Use `std::filesystem::file_size`.

Also see: https://stackoverflow.com/a/22986486.

Race condition

`destinationPath` can be created between `fs::exists` and `fs::rename` function calls, line 49:

```
        if (fs::exists(destinationPath))
        {
            fs::remove(destinationPath);
        }
        std::error_code error;
        const fs::path sourcePath(source);
        fs::rename(sourcePath, destinationPath, error);
```

Using uninitialized memory

Last bytes of `homeDriveBuf` and `homePathBuf` in case if `getenv_s` call writes less bytes than it could because of insufficient buffer size. Written bytes are stored in `homeDriveLen` and `homePathLen` but never checked, line 130:

```
    static std::string GetHomeDirectory()
    {
        #ifdef _WIN32
        char homeDriveBuf[MAX_PATH_LEN];
        size_t homeDriveLen = MAX_PATH_LEN;
        getenv_s(&homeDriveLen, homeDriveBuf, sizeof(homeDriveBuf) - 1,
"HOMEDRIVE");

        char homePathBuf[MAX_PATH_LEN];
        size_t homePathLen = MAX_PATH_LEN;
        getenv_s(&homePathLen, homePathBuf, sizeof(homePathBuf) - 1,
"HOMEPATH");

        return std::string(&homeDriveBuf[0]) +
std::string(&homePathBuf[0]);
```

include/Common/Util/HexUtil.h

Array access out of bounds

Array access out of bounds when `hex.size()` is an odd number in expression `hex[i + 1]`, line 46:

```
static std::vector<unsigned char> FromHex(const std::string& hex)
{
    std::vector<unsigned char> data((hex.size() + 1) / 2);
    for (size_t i = 0; i < hex.length(); i += 2)
    {
        data[i / 2] = (FromHexChar(hex[i]) * 16 + FromHexChar(hex[i + 1]));
    }

    return data;
}
```

include/Common/Util/StringUtil.h

Unsafe format

Taking `std::string` is insecure, avoid dynamically formed format strings, line 18:

```
    template<typename ... Args>
    static std::string Format(const std::string& format, Args ... args)
    {
        size_t size = std::snprintf(nullptr, 0, format.c_str(), args ...) +
1; // Extra space for '\0'
        std::unique_ptr<char[]> buf(new char[size]);
        std::snprintf(buf.get(), size, format.c_str(), args ...);
        return std::string(buf.get(), buf.get() + size - 1); // We don't
want the '\0' inside
    }
```

The project already has the `fmt` library bundled in the code base as a `spdlog` dependency, which is suggested to utilize instead.

See: https://github.com/fmtlib/fmt.

include/Config/Environment.h

Uninitialized fields

`m_publicKeyVersion`, `m_privateKeyVersion`, line 9:

```
    Environment(const EEnvironmentType environmentType, const FullBlock&
block, const std::vector<uint8_t>& magicBytes, const uint16_t port)
        : m_environmentType(environmentType), m_block(std::move(block)),
m_magicBytes(magicBytes), m_port(port)
    {

    }
```

include/Crypto/ProofMessage.h

Switch commitment is not implemented

Switch commitment is not implemented though it seems to be expected, line 43:

```
        bool switchCommits = true;
        size_t length = 3;
        if (bulletproofType == EBulletproofType::ENHANCED)
        {
            byteBuffer.ReadU8(); // RESERVED: Always 0
            byteBuffer.ReadU8(); // Wallet Type
            switchCommits = (byteBuffer.ReadU8() == 1);
            length = byteBuffer.ReadU8();
        }
```

`switchCommits` is never utilized afterwards, i.e. this is a dead store.

include/Crypto/BigInteger.h

Division algorithm implementation is invalid for negative divisors

E.g. `0x00ffffff / -1 = 0x00010101`, line 244:

```
template<size_t NUM_BYTES, class ALLOC>
CBigInteger<NUM_BYTES, ALLOC> CBigInteger<NUM_BYTES,
ALLOC>::operator/(const int divisor) const
{
    std::vector<unsigned char, ALLOC> quotient(NUM_BYTES);
    int remainder = 0;
    for (int i = 0; i < NUM_BYTES; i++)
    {
        remainder = remainder * 256 + m_data[i];
        quotient[i] = (unsigned char)(remainder / divisor);
        remainder -= quotient[i] * divisor;
    }

    return CBigInteger<NUM_BYTES, ALLOC>(&quotient[0]);
}
```

include/Database/BlockDb.h

Polymorphic class `IBlockDB` is missing virtual destructor.

include/Database/Database.h

Polymorphic class `IDatabase` is missing virtual destructor.

include/Database/PeerDB.h

Polymorphic class `IPeerDB` is missing virtual destructor.

include/P2P/IPAddress.h

Uninitialized fiels

m`_family`, line 36:

```
    IPAddress() = default;
```

Use of manual initialization

Use loop instead of error prone manual initialization, line 129:

```
            const uint16_t words[8] = { BitUtil::ConvertToU16(m_address[0],
m_address[1]), BitUtil::ConvertToU16(m_address[2], m_address[3]),
                BitUtil::ConvertToU16(m_address[4], m_address[5]),
BitUtil::ConvertToU16(m_address[6], m_address[7]),
BitUtil::ConvertToU16(m_address[8], m_address[9]),
                BitUtil::ConvertToU16(m_address[10], m_address[11]),
BitUtil::ConvertToU16(m_address[12], m_address[13]),
BitUtil::ConvertToU16(m_address[14], m_address[15]) };
```

Logic should be changed

Logic should be changed in order to get rid of off-by-one `words` array access in `words[i + 1]`, line 141:

```
    if (words[i] == 0 && i < 8 && words[i + 1] == 0)
    {
        while (words[i + 1] == 0)
        {
            ++i;
        }
        continue;
    }
```

Unimplemented `std::hash` specialization for IPv6

line 216:

```
    template<>
    struct hash<IPAddress>
    {
        size_t operator()(const IPAddress& address) const
        {
            const std::vector<unsigned char>& bytes = address.GetAddress();
            if (address.GetFamily() == EAddressFamily::IPv4)
            {
                return BitUtil::ConvertToU32(bytes[0], bytes[1], bytes[2],
bytes[3]);
            }
            else
            {
                // TODO: Implement
                return 0;
            }
        }
    };
```

include/P2P/Peer.h

Uninitialized field

`m_lastTxHashSetRequest`, line 57:

```
    Peer(const Peer& other)
        : m_socketAddress(other.m_socketAddress),
        m_version(other.m_version),
        m_capabilities(other.m_capabilities.load()),
        m_userAgent(other.m_userAgent),
        m_lastContactTime(other.m_lastContactTime.load()),
        m_lastBanTime(other.m_lastBanTime.load()),
        m_banReason(other.m_banReason.load())
    {
    }
```

include/PMMR/HeaderMMR.h

Polymorphic class `IHeaderMMR` is missing virtual destructor.

include/PMMR/TxHashSet.h

Polymorphic class `ITxHashSet` is missing virtual destructor.

include/TxPool/TransactionPool.h

Polymorphic class `ITransactionPool` is missing virtual destructor.

include/Wallet/NodeClient.h

Polymorphic class `INodeClient` is missing virtual destructor.

include/Wallet/WalletDB/WalletDB.h

Polymorphic class `IWalletDB` is missing virtual destructor.

include/Wallet/WalletManager.h

Polymorphic class `IWalletManager` is missing virtual destructor.

src/BlockChain/BlockChainServerImpl.cpp

Uninitialized fields

`m_pBlockStore`, `m_pChainState`, `m_pChainStore`, `m_pHeaderMMR`, line 18:

```
BlockChainServer::BlockChainServer(const Config& config, IDatabase&
database, TxHashSetManager& txHashSetManager, ITransactionPool&
transactionPool)
    : m_config(config), m_database(database),
m_txHashSetManager(txHashSetManager), m_transactionPool(transactionPool)
{
}
```

Throwing destructor

line 26:

```
BlockChainServer::~BlockChainServer()
{
    Shutdown();
}
```

Call to `Shutdown` is throwing, because e.g. it calls `HeaderMMRAPI::CloseHeaderMMR`, which calls `HeaderMMR::Commit`, which calls `MMRUtil::GetNumLeaves`, which can throw on memory allocation.

`src/PMMR/Common/MMRUtil.cpp`, line 160

```
uint64_t MMRUtil::GetNumLeaves(const uint64_t lastMMRIndex)
{
    const uint64_t numNodes = GetNumNodes(lastMMRIndex);
    uint64_t numLeaves = 0;
    std::vector<uint64_t> peakSizes = GetPeakSizes(numNodes);
    for (uint64_t peakSize : peakSizes)
    {
        numLeaves += ((peakSize + 1)/ 2);
    }
    return numLeaves;
}
```

Memory leak

line 32:

```
    const FullBlock& genesisBlock =
m_config.GetEnvironment().GetGenesisBlock();
    BlockIndex* pGenesisIndex = new BlockIndex(genesisBlock.GetHash(), 0,
nullptr);
```

src/BlockChain/Chain.h

Useless member field/complicated non-robust logic

`m_indices.size() - 1 == m_height` should always hold, therefore it is recommended to get rid of `m_height`.

src/Core/File.cpp

Invalid logic

`tellg` in not guaranteed to return file size, line 22:

```
bool File::Load()
{
    std::ifstream file(m_path, std::ios::in | std::ifstream::ate |
std::ifstream::binary);
    if (!file.is_open())
    {
        return false;
    }

    m_fileSize = file.tellg();
```

Use `std::filesystem::file_size`.

Also see: https://stackoverflow.com/a/22986486.

Useless check

line 64:

```
        if (!m_buffer.empty())
        {
            file.write((const char*)&m_buffer[0], m_buffer.size());
        }
```

Invalid logic

First branch is always taken, line 108:

```
    m_bufferIndex = nextPosition;
    if (nextPosition <= m_bufferIndex)
    {
        m_buffer.clear();
        m_bufferIndex = nextPosition;
    }
    else
    {
        m_buffer.erase(m_buffer.begin() + nextPosition - m_bufferIndex,
m_buffer.end());
    }
```

src/Crypto/ThirdParty/aes.cpp

Insecure heap memory clean-up

Lines 17, 32, 47, 62, 161, 178, 189, 205.

src/Database/BlockDBImpl.cpp

Uninitialized fields

`m_pDatabase`, `m_pDefaultHandle`, `m_pBlockHandle`, `m_pHeaderHandle`, `m_pBlockSumsHandle`, `m_pOutputPosHandle`, `m_pInputBitmapHandle`, line 9:

```
BlockDB::BlockDB(const Config& config)
    : m_config(config)
{
}
```

src/Database/PeerDBImpl.cpp

Uninitialized field

`m_pDatabase`, line 10:

```
PeerDB::PeerDB(const Config& config)
    : m_config(config)
{
}
```

src/Infrastructure/LoggerImpl.cpp

Array access out of bounds

Array access out of bounds when `eventText` ends with `\n`, line 37:

```
        std::string eventTextClean = eventText;
        while (eventTextClean.find("\n") != std::string::npos)
        {
            eventTextClean.erase(eventTextClean.find("\n"), 2);
        }
```

src/P2P/Connection.h

Insecure API `std::rand`

line 383:

```
    const int index = std::rand() % mostWorkPeers.size();
```

src/P2P/MessageProcessor.cpp

Invalid logic

`tellg` in not guaranteed to return file size, line 411:

```
    const uint64_t fileSize = file.tellg();
```

Use `std::filesystem::file_size`.

Also see: https://stackoverflow.com/a/22986486.

src/P2P/Sync/BlockSyncer.cpp

Insecure API `std::rand`

line 153:

```
    size_t nextPeer = std::rand() % mostWorkPeers.size();
```

src/PMMR/Common/MMR.h

Polymorphic class `MMR` is missing virtual destructor.

src/PMMR/Zip/ZipFile.cpp

Uninitialized field

`m_unzFile`, line 6:

```
ZipFile::ZipFile(const std::string& zipFilePath)
    : m_zipFilePath(zipFilePath)
{
}
```

src/PMMR/Zip/Zipper.cpp

Invalid logic

`tellg` in not guaranteed to return file size, line 70:

```
        long size = file.tellg();
```

Use `std::filesystem::file_size`.

Also see: https://stackoverflow.com/a/22986486.

Invalid zero fill initialization

This expression only initializes the first field of `zip_fileinfo` structure, line 76:

```
            zip_fileinfo zfi = { 0 };
```

This zero initialization is useless anyway and can be safely removed, as `zfi` is only used as an out parameter.

src/Server/Node/NodeDaemon.cpp

Uninitialized fields

`m_pDatabase`, `m_pTxHashSetManager`, `m_pTransactionPool`, `m_pBlockChainServer`, `m_pP2PServer`, `m_pNodeRestServer`, `m_pNodeClient`, line 18:

```
NodeDaemon::NodeDaemon(const Config& config)
    : m_config(config)
{
}
```

Memory leak

Line 33:

```
    m_pNodeRestServer = new NodeRestServer(m_config, m_pDatabase,
m_pTxHashSetManager, m_pBlockChainServer, m_pP2PServer);
    m_pNodeRestServer->Initialize();
```

Unreliable hardcoded time intervals in tear down code

Lines 46, 48, 50, 52, 54, 56:

```
void NodeDaemon::Shutdown()
{
    try
    {
        delete m_pNodeClient;
        m_pNodeRestServer->Shutdown();
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        P2PAPI::ShutdownP2PServer(m_pP2PServer);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        TxPoolAPI::DestroyTransactionPool(m_pTransactionPool);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        BlockChainAPI::ShutdownBlockChainServer(m_pBlockChainServer);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        delete m_pTxHashSetManager;
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        DatabaseAPI::CloseDatabase(m_pDatabase);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        LoggerAPI::Flush();
```

src/Server/Node/NodeRestServer.cpp

Uninitialized field

`m_pNodeCivetContext`, line 20:

```
NodeRestServer::NodeRestServer(const Config& config, IDatabase* pDatabase,
TxHashSetManager* pTxHashSetManager, IBlockChainServer* pBlockChainServer,
IP2PServer* pP2PServer)
    : m_config(config), m_pDatabase(pDatabase),
m_pTxHashSetManager(pTxHashSetManager),
m_pBlockChainServer(pBlockChainServer), m_pP2PServer(pP2PServer)
{
    m_pNodeContext = new NodeContext(pDatabase, pBlockChainServer,
pP2PServer, pTxHashSetManager);
}
```

src/Server/Wallet/WalletDaemon.cpp

Uninitialized fields

`m_pWalletManager`, `m_pWalletRestServer`, line 6:

```
WalletDaemon::WalletDaemon(const Config& config, INodeClient& nodeClient)
    : m_config(config), m_nodeClient(nodeClient)
{
}
```

Memory leak

Line 16:

```
    m_pWalletRestServer = new WalletRestServer(m_config, *m_pWalletManager,
m_nodeClient);
    m_pWalletRestServer->Initialize();
```

src/Server/Wallet/WalletRestServer.cpp

Uninitialized field

`m_pOwnerCivetContext`, line 12:

```
WalletRestServer::WalletRestServer(const Config& config, IWalletManager&
walletManager, INodeClient& nodeClient)
    : m_config(config), m_walletManager(walletManager)
{
    m_pWalletContext = new WalletContext(&walletManager, &nodeClient);
}
```

src/Wallet/WalletDB/WalletRocksDB.cpp

Uninitialized fields

`m_pDefaultHandle`, `m_pSeedHandle`, `m_pNextChildHandle`, `m_pLogHandle`, `m_pSlateHandle`, `m_pTxHandle`, `m_pOutputHandle`, `m_pUserMetadataHandle`, line 17:

```
WalletRocksDB::WalletRocksDB(const Config& config)
    : m_config(config)
{
}
```

src/Wallet/WalletRefresher.cpp

Heap inspection

Adding `uint32_t` variable to `const char *`, line 218:

```
LoggerAPI::LogDebug("WalletRefresher::RefreshTransactions - Output is
spent. Marking transaction as sent: " + walletTx.GetId());
```

Robustness

Memory management issues

Not utilizing RAII for resource management, leading to multiple resource leaks, dangling pointer, double frees and other memory management issues.

Example:

`include/PMMR/TxHashSetManager.h`, line 18:

```
class TXHASHSET_API TxHashSetManager
{
public:
    TxHashSetManager(const Config& config, IBlockDB& blockDB);
    ~TxHashSetManager() = default;
```

Default `TxHashSetManager` destructor does not utilize any resource cleanup which is performed in `TxHashSetManager::Close` instead.

`src/Server/Node/NodeDaemon.cpp`, line 53:

```
void NodeDaemon::Shutdown()
{
    try
    {
        delete m_pNodeClient;
        m_pNodeRestServer->Shutdown();
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        P2PAPI::ShutdownP2PServer(m_pP2PServer);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        TxPoolAPI::DestroyTransactionPool(m_pTransactionPool);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        BlockChainAPI::ShutdownBlockChainServer(m_pBlockChainServer);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
        delete m_pTxHashSetManager;
```

Instance of `TxHashSetManager` in `NodeDaemon` is not closed.

Moreover, `NodeDaemon::Shutdown` is vulnerable to double free abuses, in case it is called twice.

Also see:
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#Rr-raii.

Exception unsafe code

The code in general tends to be exception unsafe, i.e. exceptional paths are not covered (no clean-up on thrown exceptions, objects can be left in intermediate states).

Example 1:

`src/BlockChain/BlockChainServerImpl.cpp`, line 31:

```
    const FullBlock& genesisBlock =
m_config.GetEnvironment().GetGenesisBlock();
    BlockIndex* pGenesisIndex = new BlockIndex(genesisBlock.GetHash(), 0,
nullptr);

    m_pChainStore = new ChainStore(m_config, pGenesisIndex);
    m_pChainStore->Load();

    m_pHeaderMMR = HeaderMMRAPI::OpenHeaderMMR(m_config);

    m_pBlockStore = new BlockStore(m_config, m_database.GetBlockDB());
    m_pChainState = new ChainState(m_config, *m_pChainStore,
*m_pBlockStore, *m_pHeaderMMR, m_transactionPool, m_txHashSetManager);
    m_pChainState->Initialize(genesisBlock.GetBlockHeader());
```

In this case if any method in initialization method throws, there are multiple memory leaks because `m_initialized` is not set to be `true`, therefore, cleanup won't be performed.

Example 2:

`src/Crypto/Pedersen.cpp`, line 122

```
        secp256k1_pedersen_commitment* pCommitment = new
secp256k1_pedersen_commitment();
        const int parsed = secp256k1_pedersen_commitment_parse(&context,
pCommitment, &commitmentBytes[0]);
        convertedCommitments[i] = pCommitment;
        if (parsed != 1)
        {
            // TODO: Log failue
            CleanupCommitments(convertedCommitments);
            return std::vector<secp256k1_pedersen_commitment*>();
        }
```

If for instance `new secp256k1_pedersen_commitment()` throws, memory pointed by `convertedCommitments` will be unreachable forever, implying that there is a memory leak. Most of such problems could be solved by utilizing smart pointers.

No error handling

No error handling/ignoring errors which will lead to stability and maintenance problems, bugs and vulnerabilities.

Example 1:

`src/Core/File.cpp`, lines 28, 80:

```
        std::error_code error;
        m_mmap = mio::make_mmap_source(m_path, error);
```

Example 2:

`src/P2P/Seed/Seeder.cpp`, line 121

```
                    Connection* pConnection = new
Connection(std::move(socket), seeder.m_nextId++, seeder.m_config,
seeder.m_connectionManager, seeder.m_peerManager,
seeder.m_blockChainServer, ConnectedPeer(Peer(socket.GetSocketAddress()),
EDirection::INBOUND));
                    pConnection->Connect();
```

Example 3:

`src/P2P/MessageProcessor.cpp`, line 193

```
                    const EBlockChainStatus status =
blockChainServer.AddBlockHeaders(blockHeaders); // TODO: Handle failures
                    LOG_DEBUG_F("Headers message from %s finished
processing", formattedIPAddress.c_str());
```

`src/Database/BlockDBImpl.cpp`, line 35

Example 4:

```
    std::vector<std::string> columnFamilies;
    DB::ListColumnFamilies(options, dbPath, &columnFamilies);
```

Utilize smart pointers instead of explicit `new`/`delete`.

Modern C++ code in general should not have explicit `new`/`delete` expressions, except when interacting with old APIs. Smart pointers aid writing exception safe code as well and directly describe pointer semantics (i.e. owning/non-owning/shared).

Also see:
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#r11-avoid-calling-new-and-delete-explicitly.

Avoid implementation of own primitive classes

Avoid implementation of own primitive classes instead of using well-written and tested ones from libraries, e.g. implementing `IPAddress` by hand instead of using `asio::ip::address`.

Matureness

Enable `clang` compilation on macOS.

To do this, change deployment target to `10.14`.

```
set(CMAKE_OSX_DEPLOYMENT_TARGET "10.14" CACHE STRING "Minimum OS X
deployment version")
```

Enable -Wall/-Wextra

This will allow you to get almost 100% TP rate diagnostics which should always be desired to be fixed.

Disable `-Werror` for rocks dependency

`-Werror` produces unstable builds as you can't cover all possible toolchains.

See: http://blog.schmorp.de/2016-02-27-tidbits-for-the-love-of-god-dont-use-werror.html.

Lack of unit tests

Code base almost completely lacks unit tests, which renders security bound code library unreliable.

Existing unit test executor is not cross platform.

`tests/RunAllTests.cpp`, line 3:

```
int main(int argc, char* argv[])
{
    std::cout << "Preparing to run core tests\n";
    system("pause");
    system("Core_Tests.exe");

    std::cout << "Preparing to run crypto tests\n";
    system("pause");
    system("Crypto_Tests.exe");
    std::cout << "Preparing to run PMMR tests\n";
    system("pause");
    system("PMMR_TESTS.exe");

    std::cout << "Preparing to run Wallet tests\n";
    system("pause");
    system("Wallet_Tests.exe");

    system("pause");
    return 0;
}
```

Use assertions for expected invariants

Example:

`src/Server/RestUtil.h`, line 24

```
    static std::string GetURIParam(struct mg_connection* conn, const
std::string& baseURI)
    {
        const struct mg_request_info* req_info = mg_get_request_info(conn);
        const std::string requestURI(req_info->request_uri);

        return requestURI.substr(baseURI.size(), requestURI.size() -
baseURI.size());
    }
```

E.g. in this case, we recommend putting an assertion that `conn` is not `nullptr` and is a request, which ensures that `req_info` in not `nullptr`. After receiving `req_info` we also recommend putting an assertion that `req_info` is not `nullptr`.

Obfuscate/encrypt sensitive data stored on heap

Minimize its lifetime and rewrite used memory just after you don't need the data (make sure you utilize a variant of `memset_s` for this, e.g. `SecureString`).

Example 1:

`src/Server/Wallet/API/OwnerPostAPI.cpp`, line 83:

```
    const std::optional<std::string> passwordOpt =
RestUtil::GetHeaderValue(pConnection, "password");
    if (!passwordOpt.has_value())
    {
        return RestUtil::BuildBadRequestResponse(pConnection, "password
missing.");
    }
```

Here, `passwordOpt` is vulnerable to heap inspection.

Example 2:

`include/Crypto/SecretKey.h`, line 7:

```
class SecretKey
{
public:
    SecretKey(CBigInteger<32>&& seed)
        : m_seed(std::move(seed))
    {
    }

    ~SecretKey()
    {
        m_seed.erase();
    }

    inline const CBigInteger<32>& GetBytes() const { return m_seed; }
    inline const unsigned char* data() const { return
m_seed.GetData().data(); }
    inline size_t size() const { return m_seed.GetData().size(); }
    //
    // Serialization/Deserialization
    //
    void Serialize(Serializer& serializer) const
    {
        serializer.AppendBigInteger<32>(m_seed);
    }

    static SecretKey Deserialize(ByteBuffer& byteBuffer)
    {
        return SecretKey(byteBuffer.ReadBigInteger<32>());
    }

private:
    CBigInteger<32> m_seed;
};
```

Correctness & Style

Using of C standard library headers instead of C++ counterparts

Example:

`include/Core/File.h`, line 11:

```
#include <stdint.h>
```

should be

```
#include <cstdint>
```

Useless `inline`-s for inline class methods

These are already `inline`.

Example:

`src/BlockChain/Chain.h`, line 22:

```
    inline EChainType GetType() const { return m_chainType; }
```

Useless `const`-s for function return types

These are going to be ignored by compiler.

Example:

`src/P2P/Messages/MessageHeader.h`, line 40:

```
    inline const MessageTypes::EMessageType GetMessageType() const { return
m_messageType; }
```

Use of `const`-s for value parameters

We recommend avoid using `const`-s for value parameters, as this provides no new semantic value for API users but clutters the signature.

Example:

`src/BlockChain/Chain.h`, line 16:

```
    BlockIndex* GetByHeight(const uint64_t height);
    const BlockIndex* GetByHeight(const uint64_t height) const;
```

Counterexample:

`src/BlockChain/Chain.h`, line 24:

```
    bool AddBlock(BlockIndex* pBlockIndex);
```

If `const` is supposed to be used consistently over all code base, then there should be a `const` either, i.e.

```
    bool AddBlock(BlockIndex* const pBlockIndex);
```

which provides no additional semantic value but clutters the interface.

Useless assignments of pointers to `nullptr`

Example:

`src/Database/DatabaseImpl.cpp`, line 44:

```
    DATABASE_API void CloseDatabase(IDatabase* pDatabase)
    {
        Database* pDatabaseImpl = (Database*)pDatabase;
        delete pDatabaseImpl;
        pDatabase = nullptr;
    }
```

`pDatabase = nullptr` is useless here. In general this idiom is used to prevent double free when managing memory manually, which is very error prone (imagine you forget to put `= nullptr` in clean up code), and should be avoided in favor of using smart pointers instead, which are invulnerable to double frees.

String manipulation instead of file path API

There are cases of string manipulation instead of file path API for working with file paths.

Example:

`include/Config/Config.h`, line 36:

```
    Config(
        const EClientMode clientMode,
        const Environment& environment,
        const std::string& dataPath,
        const DandelionConfig& dandelionConfig,
        const P2PConfig& p2pConfig,
        const WalletConfig& walletConfig,
        const ServerConfig& serverConfig,
        const std::string& logLevel
    )
        : m_clientMode(clientMode),
        m_environment(environment),
        m_dataPath(dataPath),
        m_dandelionConfig(dandelionConfig),
        m_p2pConfig(p2pConfig),
        m_walletConfig(walletConfig),
        m_serverConfig(serverConfig),
        m_logLevel(logLevel)
    {
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_txHashSetPath);
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_txHashSetPath + "kernel/");
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_txHashSetPath + "output/");
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_txHashSetPath + "rangeproof/");
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_chainPath);
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_databasePath);
        fs::create_directories(m_dataPath + "NODE/" /*+
std::string(SEPARATOR) */+ m_logDirectory);
    }
```

In this example, e.g. if `m_dataPath` or `m_txHashSetPath` is not ending with file path separator, the resulting file path will be invalid.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

CMake

Erroneous logic (always `STATIC`): all usages of `GRINPP_STATIC`.

Example:

`src/BlockChain/CMakeLists.txt`, line 12:

```
if(GRINPP_STATIC)
    add_library(${TARGET_NAME} STATIC ${SOURCE_CODE})
elseif(GRINPP_STATIC)
    add_library(${TARGET_NAME} SHARED ${SOURCE_CODE})
endif(GRINPP_STATIC)
```

should be

```
if(GRINPP_STATIC)
    add_library(${TARGET_NAME} STATIC ${SOURCE_CODE})
else(GRINPP_STATIC)
    add_library(${TARGET_NAME} SHARED ${SOURCE_CODE})
endif(GRINPP_STATIC)
```

src/BlockChain/LockedChainState.h

Public class member fields

line 57:

```
    LockedChainState& operator=(const LockedChainState&) = delete;

    int* m_pReferences;
    std::shared_mutex& m_mutex;
    ChainStore& m_chainStore;
    BlockStore& m_blockStore;
    IHeaderMMR& m_headerMMR;
    OrphanPool& m_orphanPool;
    ITransactionPool& m_transactionPool;
    TxHashSetManager& m_txHashSetManager;
};
```

Manual analysis, stage II

The code base on commit 40ecfa56854882f561e8af2b4f56d2ebee5abdb2 was completely manually analyzed, its logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. We highly recommend fixing them.

deps/secp256k1-zkp/src/modules/schnorrsig/main_impl.h

Integer overflow

This results in invalid argument check, line 319:

```
    ARG_CHECK(n_sigs < (size_t)(1 << 31));
```

include/Common/Util/BitUtil.h

Invalid logic

lines 45-46:

```
        return ((((uint64_t)byte1) << 56) | (((uint64_t)byte2) << 48) |
(((uint64_t)byte3) << 40) | ((uint64_t)byte4) << 32
            | ((uint64_t)byte5) << 24 | ((uint64_t)byte5) << 16 |
((uint64_t)byte5) << 8 | ((uint64_t)byte5));
```

Should be

```
        return ((((uint64_t)byte1) << 56) | (((uint64_t)byte2) << 48) |
(((uint64_t)byte3) << 40) | ((uint64_t)byte4) << 32
            | ((uint64_t)byte5) << 24 | ((uint64_t)byte6) << 16 |
((uint64_t)byte7) << 8 | ((uint64_t)byte8));
```

include/Common/Util/HexUtil.h

Own implementation of hex converting algorithms

Shipping an own implementation of hex converting algorithms is unreliable. It is suggested to utilize a well-tested library instead.

Array access out of bounds

Array access out of bounds when `hex.size()` is an odd number in expression `hex[i + 1]`. This is especially important issue because the call can occur on data received from an untrusted/unvalidated source, i.e. from other nodes and REST API users.

line 46:

```
    static std::vector<unsigned char> FromHex(const std::string& hex)
    {
        std::vector<unsigned char> data((hex.size() + 1) / 2);
        for (size_t i = 0; i < hex.length(); i += 2)
        {
            data[i / 2] = (FromHexChar(hex[i]) * 16 + FromHexChar(hex[i +
1]));
        }

        return data;
    }
```

Typo

Should be `value >= '0'`, line 28:

```
    static unsigned char FromHexChar(const char value)
    {
        if (value <= '9' && value >= 0)
```

Invalid implementation

E.g. `FromHex("1")` results into `16`, line 46:

```
    static std::vector<unsigned char> FromHex(const std::string& hex)
    {
        std::vector<unsigned char> data((hex.size() + 1) / 2);
        for (size_t i = 0; i < hex.length(); i += 2)
        {
            data[i / 2] = (FromHexChar(hex[i]) * 16 + FromHexChar(hex[i +
1]));
```

Data corruption

Writing two bytes into the `bytes` vector itself, instead of writing into its contents, line 75:

```
    static std::string ConvertToHex(const uint16_t value, const bool
abbreviate)
    {
        const uint16_t bigEndian = EndianHelper::GetBigEndian16(value);
        std::vector<unsigned char> bytes(2);
        memcpy(&bytes, (unsigned char*)&bigEndian, 2);
```

include/P2P/IPAddress.h

Own implementation of IP API

Shipping an own implementation of IP API is unreliable. It is suggested to utilize a well-tested library instead.

Invalid read `words[i + 1]`

line 154-156:

```
                    while (words[i + 1] == 0)
                    {
                        ++i;
```

src/Net/Socket.cpp

Data races on `Socket` fields

E.g. on `m_errorCode`. Consider the following access sequence from threads `T1` and `T2`:

`T1`

```
ConnectionManager::PruneConnections
Connection::IsConnectionActive
Socket::IsActive
```

`T2`

```
Connection::Thread_ProcessConnection
MessageRetriever::RetrieveMessage
Socket::HasReceivedData
```

In this case, `m_errorCode` can be modified by `HasReceivedData` call in `T2` thread, while `T1` is validating whether the socket is active, resulting into invalid method call results.

API

API requests often do not validate the input properly

`src/Server/Node/API/ChainAPI.cpp`, line 164:

```
        Commitment commitment(CBigInteger<33>::FromHex(id));
```

Other examples:

```
# crash
127.0.0.1:3413/v1/txhashset/lastkernels?n=123
reading empty file .GrinPP/MAINNET/NODE/TXHASHSET/kernel/pmmr_hash.bin
# unvalidated
127.0.0.1:3413/v1/txhashset/lastoutputs?n=fds
127.0.0.1:3413/v1/txhashset/outputs?start_index=ds&max=
127.0.0.1:3413/v1/txhashset/lastrangeproofs?n=###
```

General

Methods with `Commit`/`Rollback` overrides

Methods that utilize `Commit`/`Rollback` overrides should provide strong guarantee. For example, if an exception is thrown in `m_pHashFile->Rollback()` call during `KernelMMR::Rollback` call, `m_pDataFile->Rollback()` will not be called.

> *Comment from the developers: commit does not offer strong guarantees. Because the Grin protocol relies on storing data in multiple separate files to read/write efficiently, instead of using a standard, transactable database, it's just not feasible to handle every way it could fail. The "right" way would be to copy all files, make the changes, then one by one rename the existing files, and then rename the new ones. But that's hugely inefficient.*

ShutdownManager

`ShutdownManager` is not affecting all stages of the node, e.g. `StateSyncer`.

`TxHashSet` synchronization stuck

In some cases, `TxHashSet` synchronization stucks after reaching designated progress level. Signaling and exiting the work is not working as well.

```
Press Ctrl-C to exit...Time Running: 160s
Status: Syncing TxHashSet 16777216/527428643(3%)
NumConnections: 15
Header Height: 491826
Header Difficulty: 1380796615448119
Block Height: 0
Block Difficulty: 17179869184
Network Height: 491826
Network Difficulty: 1380796615448119

Press Ctrl-C to exit...Time Running: 161s
Status: Syncing TxHashSet 16777216/527428643(3%)
NumConnections: 15
Header Height: 491826
Header Difficulty: 1380796615448119
Block Height: 0
Block Difficulty: 17179869184
Network Height: 491826
Network Difficulty: 1380796615448119
```

This seems to be a consequence of multiple deadlock issues, which can occur around `Locked` object instances. Consider an example with two thread `T1`, which executes `Dandelion::Thread_Monitor` and `T2`, which executes `Connection::Thread_ProcessConnection`.

1. `T2` first enters `BlockProcessor::DetermineBlockStatus` and acquires an exclusive access to an `IBlockDB` instance.
2. `T1` enters `Dandelion::ProcessStemPhase`, acquires shared access to an `ITxHashSet` instance.
3. `T1` then tries to acquire shared access to an `IBlockDB` instance and locks.
4. `T2` tries to acquire exclusive access to an `ITxHashSet` instance and locks.

Other dead lock issues are also possible when using these `Locked` instances. Instead of protecting separate objects, logically connected components should be protected by mutexes or other synchronization primitives.

`std::is_base_of` usage

`std::is_base_of` should only be used for static dispatch, e.g. SFINAE. For runtime type checks RTTI should be used instead.

Shutdown APIs

Shutdown APIs runs tear down code for static objects, in particular for singletons, e.g. `Logger`, which still can be used in other threads.

Old version of Roaring library

An old version of `Roaring` library is used, which contains buggy code. For example, there are buffer overflows in `bitset_container_rank` method which are fixed in the freshest library version.

- https://github.com/RoaringBitmap/CRoaring/issues/208
- https://github.com/RoaringBitmap/CRoaring/commit/a8153c681f21352435b054dff82bf6e21df095af#diff-aa693b9b342e768e9307175295b6a135
- https://github.com/RoaringBitmap/CRoaring/commit/604c5a65452c5770c118302bcbe129d17889c86f#diff-aa693b9b342e768e9307175295b6a135

The buffer overflow occurs on the following execution path.

```
uint64_t PruneList::GetShift(const uint64_t position) const
const uint64_t index = m_prunedRoots.rank(position + 1);
uint64_t rank(uint32_t x) const
return roaring_bitmap_rank(&roaring, x);
return size + container_rank(bm->high_low_container.containers[i],
return bitset_container_rank((const bitset_container_t *)container, x);
uint64_t leftoverword = container->array[k / 64];// k / 64 should be within
scope
```

Empty catch block and ignored error codes

There are empty catch block and ignored error codes, which makes the code unreliable for exceptional execution paths.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

include/Common/Base64.h

Own implementation of Base64 algorithm

Shipping an own implementation of Base64 algorithm is unreliable. It is suggested to utilize a well-tested library instead.

include/Common/Secure.h

Handcrafted `memset_s` implementation

Use standard API instead, line 47:

```
static void cleanse(void *ptr, size_t len)
{
    std::memset(ptr, 0, len);

    /* As best as we can tell, this is sufficient to break any
optimisations that
       might try to eliminate "superfluous" memsets. If there's an easy way
to
       detect memset_s, it would be better to use that. */
#if defined(_MSC_VER)
    SecureZeroMemory(ptr, len);
#else
    __asm__ __volatile__("" : : "r"(ptr) : "memory");
#endif
}
```

include/Common/Util/FileUtil.h

Exceptions

Some of used methods are throwing exceptions, which makes the API inconsistent, making the code unstable. Error code using counterparts should be employed instead.

Using uninitialized memory

Last bytes of `homeDriveBuf` and `homePathBuf` in case if `getenv_s` call writes less bytes than it could because of insufficient buffer size. Written bytes are stored in `homeDriveLen` and `homePathLen` but never checked, line 130:

```
    static std::string GetHomeDirectory()
    {
        #ifdef _WIN32
        char homeDriveBuf[MAX_PATH_LEN];
        size_t homeDriveLen = MAX_PATH_LEN;
        getenv_s(&homeDriveLen, homeDriveBuf, sizeof(homeDriveBuf) - 1,
"HOMEDRIVE");

        char homePathBuf[MAX_PATH_LEN];
        size_t homePathLen = MAX_PATH_LEN;
        getenv_s(&homePathLen, homePathBuf, sizeof(homePathBuf) - 1,
"HOMEPATH");

        return std::string(&homeDriveBuf[0]) +
std::string(&homePathBuf[0]);
```

Race conditions between file existence and removal calls

File existence checks can be safely removed, leaving only file removal calls, lines 47-49:

```
        if (fs::exists(destinationPath))
        {
            fs::remove(destinationPath);
```

Lines 77-79:

```
    static bool RemoveFile(const std::string& filePath)
    {
        std::error_code ec;
        if (fs::exists(filePath, ec))
        {
            const uintmax_t removed = fs::remove_all(filePath, ec);
```

Neglect of error codes and exceptions handling

Error codes are sometimes just ignored, e.g. lines 89-92:

```
        std::error_code ec;
        fs::create_directories(destDir, ec);
        if (!fs::exists(sourceDir, ec) || !fs::exists(destDir, ec))
```

Lines 192-203:

```
                    iter.increment(ec);
                    if (ec)
                    {
                        //std::cerr << "Error While Accessing : " << iter-
>path().string() << " :: " << ec.message() << '\n';
                    }
                }
            }
        }
        catch (std::system_error&)
        {
            //std::cerr << "Exception :: " << e.what();
        }
```

Unix-specific implementation

line 184:

```
    if (includeHidden || !StringUtil::StartsWith(filename, "."))
```

include/Common/Util/StringUtil.h

Own implementation of formatting API

Shipping an own implementation of formatting API is unreliable. It is suggested to utilize a well-tested library instead. The project already has the `fmt` library bundled in the code base as a `spdlog` dependency, which is suggested to utilize instead.

See: https://github.com/fmtlib/fmt.

Unsafe format

Taking `std::string` is insecure, avoid dynamically formed format strings, line 18:

```
    template<typename ... Args>
    static std::string Format(const std::string& format, Args ... args)
    {
        size_t size = std::snprintf(nullptr, 0, format.c_str(), args ...) +
1; // Extra space for '\0'
        std::unique_ptr<char[]> buf(new char[size]);
        std::snprintf(buf.get(), size, format.c_str(), args ...);
        return std::string(buf.get(), buf.get() + size - 1); // We don't
want the '\0' inside
    }
```

Use-after-free

Returning a pointer to memory, allocated by temporary string, resulting in a use-after-free, lines 119-121:

```
    static typename std::enable_if<std::is_base_of<Traits::IPrintable,
T>::value, const char*>::type convert_for_snprintf(const T& x)
    {
        return x.Format().c_str();
```

Moreover, usage `std::is_base_of` for SFINAE is a static check, that is, the `convert_for_snprintf` call is statically dispatched. When using interfaces, i.e. when `T` is inherited from `Traits::IPrintable` and, for instance, other interface `I`, and the calling `StringUtil::Format` code only works with `I` instances, i.e. it knows nothing about that some of these instances can also be inherited from `Traits::IPrintable`, the call will be dispatched to the general overload instead, i.e. to

```
    static typename std::enable_if<!std::is_base_of<Traits::IPrintable,
T>::value, T>::type convert_for_snprintf(const T& x)
    {
        return x;
    }
```

resulting in that `x` will be pushed onto the `snprintf` arguments stack as it is. Thus, `snprintf` will dereference garbage and is a very likely to be exploited, as all memory management issues are.

include/Common/Util/TimeUtil.h

Usage of `strftime`

Consider using `put_time` as a safe alternative for `strftime`.

include/Config/TorConfig.h

Uninitialized value

`m_socksPort` is self-initialized, leading to using of uninitialized value, line 10:

```
m_socksPort(m_socksPort), m_controlPort(controlPort), m_password(password),
m_hashedPassword(hashedPassword)
```

include/Consensus/BlockDifficulty.h

Consider separating sequence points explicitly

Otherwise, the result of the operation can be undefined, line 73:

```
    xpr_edge_bits = xpr_edge_bits -= (std::min)(xpr_edge_bits, 1 + (height
- expiry_height) / WEEK_HEIGHT);
```

include/Core/AppendOnlyFile.h

Incorrect error code value comparison

Error code value is only guaranteed to be non-zero in case there was an error, lines 58-60:

```
            std::error_code error;
            m_mmap = mio::make_mmap_source(m_path, error);
            if (error.value() > 0)
```

Missing a throwing statement

Lines 114-117:

```
    if (error.value() > 0)
    {
        LOG_ERROR_F("Failed to mmap file: %d", error.value());
    }
```

include/Core/BitmapFile.h

Invalid logic

`BitToByte(j) > 0` will be evaluated first, line 130:

```
    if (byte & BitToByte(j) > 0)
```

include/Core/Exceptions/FileException.h

Non-standard macro extension

Non-standard macro extension is used. Consider using standard `__func__` instead, line 6:

```
#define FILE_EXCEPTION(msg) FileException(msg, __FUNCTION__)
```

Zero arguments

The macro won't work with zero arguments `__VA_ARGS__`, line 7:

```
#define FILE_EXCEPTION_F(msg, ...) FileException(StringUtil::Format(msg,
__VA_ARGS__), __FUNCTION__)
```

include/Core/Traits/Lockable.h

Usage of `shared_ptr(new)`

Consider using `make_shared` instead of `shared_ptr(new)`, line 40:

```
    return Reader(std::shared_ptr<InnerReader<T>>(new
InnerReader<T>(pObject, pMutex, lock)));
```

The `InnerWriter` destructor can possibly throw

Lines 108-123:

```
        virtual ~InnerWriter()
        {
            // Using MutexUnlocker in case exception is thrown.
            MutexUnlocker unlocker(m_pMutex);

            if (m_batched)
            {
                Rollback();
            }
            else
            {
                Commit();
            }

            OnEndWrite();
        }
```

Using `std::is_base_of` instead of RTTI

Note that `std::is_base_of` is a static check, it cannot refer runtime type information.

```
            if (std::is_base_of<Traits::IBatchable, U>::value)
```

Putting `virtual InnerWriter` and `Locked` destructors is useless.

include/Crypto/BigInteger.h

Own implementation of long arithmetics API

Shipping an own implementation of long arithmetics API is unreliable. It is suggested to utilize a well-tested library instead.

Unchecked memory accesses

E.g. when the input string is empty or when accessing `i + 1`-th element, lines 216-246:

```
template<size_t NUM_BYTES, class ALLOC>
CBigInteger<NUM_BYTES, ALLOC> CBigInteger<NUM_BYTES, ALLOC>::FromHex(const
std::string& hex)
{
    // TODO: Verify input size
    size_t index = 0;
    if (hex[0] == '0' && hex[1] == 'x')
    {
        index = 2;
    }

    std::string hexNoSpaces = "";
    for (size_t i = index; i < hex.length(); i++)
    {
        if (hex[i] != ' ')
        {
            hexNoSpaces += hex[i];
        }
    }

    std::vector<unsigned char, ALLOC> data(NUM_BYTES);
    for (size_t i = 0; i < hexNoSpaces.length(); i += 2)
    {
        data[i / 2] = (FromHexChar(hexNoSpaces[i]) * 16 +
FromHexChar(hexNoSpaces[i + 1]));
    }
    return CBigInteger<NUM_BYTES, ALLOC>(&data[0]);
}
```

Division algorithm implementation

Division algorithm implementation is invalid for negative divisors, e.g. `0x00ffffff / -1 = 0x00010101`, line 244:

```
template<size_t NUM_BYTES, class ALLOC>
CBigInteger<NUM_BYTES, ALLOC> CBigInteger<NUM_BYTES,
ALLOC>::operator/(const int divisor) const
{
    std::vector<unsigned char, ALLOC> quotient(NUM_BYTES);

    int remainder = 0;
    for (int i = 0; i < NUM_BYTES; i++)
    {
        remainder = remainder * 256 + m_data[i];
        quotient[i] = (unsigned char)(remainder / divisor);
        remainder -= quotient[i] * divisor;
    }

    return CBigInteger<NUM_BYTES, ALLOC>(&quotient[0]);
}
```

src/Database/PeerDBImpl.cpp

Memory leak

```
rocksdb::Iterator* it = m_pDatabase->NewIterator(rocksdb::ReadOptions());
```

src/PMMR/KernelMMR.cpp

Possible division by zero

Lines 90-91:

```
    uint128_t hash128 = (uint128_t)hash64;
    const uint128_t difference = scaled / hash128;
```

src/PMMR/Zip/Zipper.cpp

Incorrect logic when working with paths

Use Path API instead, line 25:

```
    const std::string destinationPath =
paths[i].substr(std::max(paths[i].rfind('\\'), paths[i].rfind('/')) + 1);
```

line 77:

```
std::string fileName = destDir + "/" +
sourceFile.substr((std::max)(sourceFile.rfind('\\'),
sourceFile.rfind('/')) + 1);
```

src/Server/Main.cpp

Use-after-frees

Multiple use-after-frees of `pNode` and `pWallet` pointers, which are immediately destructed on receiving a termination signal, but references to their interals can still persist in threads in non-main threads, e.g. in `Thread_ProcessConnection`.

src/Wallet/SessionManager.cpp

Memory leak

Memory leak may occur if `m_sessionsById[sessionId]` throws, lines 84-85:

```
    LoggedInSession* pSession = new LoggedInSession(wallet,
std::move(encryptedSeedWithCS), std::move(encryptedGrinboxAddress));
    m_sessionsById[sessionId] = std::shared_ptr<LoggedInSession>(pSession);
```

src/Wallet/WalletDB/WalletRocksDB.cpp

Memory leak

```
auto iter = m_pDatabase->NewIterator(ReadOptions(), m_pSeedHandle);
```

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

include/Core/Traits/Lockable.h

Useless virtual inheritance

Line 95:

```
class Writer : virtual public Reader<T>
```

Same applies to lines 127 and further.

src/PMMR/TxHashSetValidator.cpp

Costly operation

Use a thread pool instead, lines 38-42:

```
    std::vector<std::thread> threads;
    std::atomic_bool mmrHashesValidated = true;
    threads.emplace_back(std::thread([this, pKernelMMR,
&mmrHashesValidated] { if (!this->ValidateMMRHashes(pKernelMMR)) {
mmrHashesValidated = false; }}));
    threads.emplace_back(std::thread([this, pOutputPMMR,
&mmrHashesValidated] { if (!this->ValidateMMRHashes(pOutputPMMR)) {
mmrHashesValidated = false; }}));
    threads.emplace_back(std::thread([this, pRangeProofPMMR,
&mmrHashesValidated] { if (!this->ValidateMMRHashes(pRangeProofPMMR)) {
mmrHashesValidated = false; }}));
```

> *Comment from the developers: Thread pool was not used for `src/PMMR/TxHashSetValidator.cpp`. This function should only be called once on an initial sync, so not worth the effort of adding a threadpool.*

src/PoW/DifficultyLoader.cpp

Using a global variable

Line 7:

```
LRU::Cache<Hash, std::shared_ptr<BlockHeader>> BLOCK_HEADERS_CACHE(128);
```

This analysis was performed by SmartDec.

Alexander Seleznev, Chief Business Development Officer
Lenar Safin, Senior Software Engineer
Alexander Chernov, Chief Research Officer

March 4, 2020