Beam DAO Security Analysis

by Pessimistic

This report is public.

Published: September 14, 2021

Abstract

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

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 Beam Dao code base. We performed our audit according to the procedure described below.

The audit did not reveal any issues that endanger the security of the project in the current implementation. However, the complexity of the code base, the lack of detailed comments, insufficient documentation, and disregard of good coding practices together might result in bugs and vulnerabilities in the future versions of the code, especially if new members join the development team.

The project is well-designed, and the code base is thoroughly optimized. However, at this stage of the project, the readability of the code is preferable.

General recommendations

We recommend improving documentation (including business and technical documentations, user's guides, etc.), providing detailed comments in the code, adopting a more novice-friendly and defensive code style, and prioritizing code readability and security over low-level efficiency.

We do not recommend using the existing contract for educational purposes or as an example for developers.

Project overview

Project description

For the audit, we were provided with two public GitHub repositories of Beam project:

- dao-core repo, commit c6a13d2dce894d979e8834c7f6413ecbcc712bc0, files **app.cpp**, **contract.cpp**, and **contract.h** in **shaders/** directory.
- beam repo, commit d931d47990c02360b483d98b6fd37ccf338251d4, files **app_common_impl.h**, **contract.cpp**, and **contract.h** in **bvm/Shaders/upgradable/** directory

Tests for the project are located in **dao-core/unittests/shader_test.cpp** file.

Procedure

We perform our audit according to the following procedure:

- Onboarding
  - We set up Beam node and environment.
  - We inspect existing entities and investigate their relationships.
- Automated analysis
  - We compile contracts and deploy them locally.
  - We run provided tests in emulated environment.
  - We run Valgrind on the contracts in emulated environment.
  - We manually verify all the issues reported by PVS studio and Clang Tidy.
- Manual audit
  - We manually review the code and assess its quality.
  - We check the code for known vulnerabilities.
  - We check whether the code logic complies with provided documentation.
  - We suggest possible charge optimizations.
  - We check whether interacting with the contract might degrade user's privacy*.
- Report
  - We reflect all the gathered information in the report.

We are actively looking for:

- Access control issues (incorrect admin or user identification/authorization).
- Lost/stolen assets (assets being stuck on the contract or sent to nowhere or to a wrong person).
- DoS due to logical issues (deadlock, state machine error, etc.).
- DoS due to technical issues (charge, other limitations).
- Blockchain-related (cross-transaction) issues (replay, reorder, race condition).
- Contract interaction issues (reentrancy, insecure calls).
- Arithmetic issues (overflow, underflow, rounding issues).
- Incorrect `ENV::X` usage.
- Other issues.
- * As for now, we do not actively look for privacy issues.

Automated analysis

At this stage, we run automated tools and inspect their results. Valgrind did not report any issues. All the issues from PVS-Studio report and results of Clang Tidy run were manually verified (rejected or confirmed).

All discovered issues are of low severity, i.e. they do not affect the security of the current implementation, but might lead to bugs or vulnerabilities in case of code modification.

The tools have correctly identified four types of issues:

- Not initialized class members.
- Constructors with one parameter are not defined as `explicit`.
- Empty constructor is used instead of `=default` one.
- Unnecessary use of `virtual` modifier in one declaration with `override`.

All confirmed occurrences are listed in Appendix section.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Code quality and style

This list includes general recommendations and best coding practices for C/C++.

- Use `nullptr` instead of `NULL`.
- Use `span<>` for arrays with static and initially defined length.
- Use range-based loops based on span array declaration.
- Make constructors for public classes (structures).
- All class members should be initialized at the moment of the class instantiation.
- Templates and defines are widely used in the source code. They should be well documented and explained.
- “Magic” constants should be explained and declared as `constant`s with appropriate names.
- Current state of source code makes the entry threshold for newcomers pretty high and provides the wide possibility to make avoidable mistakes

Contract improvements

We suggest the following security and code quality improvements of the contracts:

- Failing fast is considered the most secure option for smart contracts. We recommend calling `Env::Halt()` rather than handling errors and unexpected situations.
- `#defines` are widely used in the project, which makes the code harder to read and understand. E.g., the translation of `ON_METHOD(manager, schedule_upgrade)` to `void On_manager_schedule_upgrade(const ContractID& cid, const ContractID& cidVersion, const Height& dh, int unused = 0)` is not evident. As a rule of thumb, we recommend optimize smart contracts for readability, security, and robustness rather than efficiency.
- The code contains multiple structures without implicit variables initialization. This provokes potential bugs. Consider using constructor instead of fields initialization directly in code. E.g., `WalkerUpgradable::VerInfo` struct defined in **beam/bvm/Shaders/upgradabe/app_common_impl.h** is consequently initialized line by line in **dao-core/shaders/app.cpp** at lines 116–120.

Further, we group issues by files where they were discovered.

dao_core/shaders/app.cpp

- **Line 92**: `DemoXdaoRoles_All(THE_ROLE)` expression is translated to a large block of code of 43 lines, which is difficult to follow.
- **Line 100**: a possible replacement in the expression should be explained. Providing comments for each `ON_METHOD()` usage is also a good option.
- **Lines 116–120**: a consequential initialization of a struct is error prone. This might become a bug if a single member is not initialized. Consider adding a constructor.
- **Lines 483**: avoid using “magic constants” directly in the code.

dao_core/shaders/contract.cpp

- **Line 5**: Consider providing an explanation for the macro:

```
#define BEAMX_ALLOCATION_All(macro, macro_seed) \
```

- **Line 77**: no reaction on possible error in `Env::SaveVar_T()` call. The assets amount could be distributed incorrectly.
- **Line 84**: no reaction on possible error in `Env::SaveVar_T()` call. The time of initial distribution might be set incorrectly.
- **Line 97**: function `AllocateAll(s)` might have wrong state or wrong allocation. In such a case, the constructor will not fail.
- **Line 99**: no reaction on possible error in `Env::SaveVar_T((uint8_t) s.s_Key, s)` call.
- **Line 124**: no reaction on possible error in `Env::LoadVar_T((uint8_t) pr.s_Key, pr)` call. In such a case, the calculations below will be incorrect.
- **Line 135**: no reaction on possible error in `Env::SaveVar_T(puk, pu)` call.
- **Line 140**: no reaction on possible error in `Env::LoadVar_T((uint8_t) s.s_Key, s)` call.
- **Line 163**: consider calling `get_EmissionSoFar()` directly instead since this is the main `Method_4` activity.
- **Line 176**: no reaction on possible error in `Env::LoadVar_T((uint8_t) s.s_Key, s)` call. The provided data for `Env::FundsUnlock(s.m_Aid, r.m_WithdrawBeamX)` call at line 178 might be incorrect.
- **Lines 198, 201**: no reaction on possible error in `Env::SaveVar_T()` call.

beam/bvm/Shaders/uprgadable/contract.cpp

- **Line 28**: no reaction on possible error in `Env::LoadVar_T(key, s)` call. Thus, `s` variable might have unpredictable value. As a result, `Env::CallFar()` at line 39 might be called with incorrect contract ID.
- **Line 37**: no reaction on possible error in `Env::SaveVar_T(key, s)` call. In case of error in `LoadVar_T`, `SaveVar_T` will save garbage.
- **Line 52**: no reaction on possible error in `Env::SaveVar_T(key, s)` call.
- **Lines 63, 74**: no reaction on possible error in `Env::LoadVar_T(key, s)` call. In case of error, incorrect `s` value will be passed to the functions that follow.
- **Line 79**: no reaction on possible error in `Env::SaveVar_T(key, s);` call. As a result, a possible transaction might be performed with incorrect data.

beam/bvm/Shaders/uprgadable/contract.h

- **Line 24**: the value of `s_Key` constant should be declared separately as an important part of business logic.
- **Lines 30, 36**: the value of `s_iMethod` constant should be declared separately as an important part of business logic.

beam/bvm/Shaders/uprgadable/app_common_impl.h

- **Line 30**: `Env::Key_T<WalkerContracts::SidCid> key` is not initiated.

Appendix

Results of automated tools

Here, we list confirmed issues discovered by automated tools in either of two formats:

- `<line-number> initial-text -> variant-proposed`
- `<line-number> initial-text - <note>`

dao_core/beam/core/uintBig.h

- `118 uintBig_t() - m_pData not initialized in non-debug run (when _DEBUG is undefined).`
- `125 uintBig_t(Zero_) -> explicit uintBig_t(Zero_)`
- `130 uintBig_t(const uint8_t p[nBytes]) -> explicit uintBig_t(const uint8_t p[nBytes])`
- `140 uintBig_t(const Blob& v) -> explicit uintBig_t(const Blob& v)`
- `146 uintBig_t(T x) -> explicit uintBig_t(T x)`
- `288, 294 void ShiftRight(uint32_t nBits, uintBig_t<nBytesOther_>& res) const - better to rename nBits: it shadows uintBig_t::nBits`
- `321 Threshold(const uintBig_t& val) -> explicit Threshold(const uintBig_t& val)`
- `381 FourCC() {} -> FourCC()=default;`
- `382 FourCC(uint32_t x) -> explicit FourCC(uint32_t x)`
- `387 Text(uint32_t) -> explicit Text(uint32_t)`

dao_core/beam/core/ecc.h

- `38 Scope(Enum e); -> explicit Scope(Enum e);`
- `67 Unary(const X& x_) -> explicit Unary(const X& x_)`
- `103 Scalar() {} -> Scalar()=default;`
- `125 m_Y - not initialized.`
- `127 Point() {} -> Point()=default;`
- `130 Point(const Native& t) -> explicit Point(const Native& t)`

- 132 `Point(const Commitment& t) -> explicit Point(const Commitment& t)`
- 228 `Type() {} -> Type()=default;`
- 249 `m_Idx` - not initialized.
- 251 `m_SubIdx` - not initialized.
- 253 `ID() {} -> ID()=default;`
- 271 `void operator = (const ID&); -> Packed& operator = (const ID&);`
- 275 `void operator = (const Packed&); -> ID& operator = (const Packed&);`
- 300 `virtual void DeriveKey(Scalar::Native&, const Hash::Value&) = 0;` - redefined the same method above on 299.
- 461 `Amount m_Value;` - not initialized.

dao_core/beam/core/ecc_native.h

- 14 `#include <assert.h> -> #include <cassert>`
- 58 `secp256k1_scalar` - not initialized.
- 76 `template <typename T> Native(const T& t) -> template <typename T> explicit Native(const T& t)`
- 79 `return Minus(*this); -> return {*this};`
- 80 `return Plus(*this, y); -> return {*this, y};`
- 81 `return Minus2(*this, y); -> return {*this, y};`
- 82 `return Mul(*this, y); -> return {*this, y};`
- 145 `secp256k1_gej` - not initialized.
- 151 `return Minus(*this); -> return {*this};`
- 152 `return Plus(*this, y_); -> return {*this, y_};`
- 153 `return Minus2(*this, y_); -> return {*this, y_};`
- 154 `return Mul(*this, y_); -> return {*this, y_};`
- 155 `return Double(*this); -> return {*this};`
- 232 `m_pFesBuf` - not initialized.
- 373 `m_Fast` - not initialized. Better to set `m_Fast{};`
- 378 `m_pPt` - not initialized. Better to set `m_pPt[(1 << nBits)]{};`

- 379 `m_Compensation` - not initialized. Better to set `m_Compensation{};`
- 389 `m_pCasual` - not initialized.
- 390 `m_ppPrepared` - not initialized.
- 391 `m_pKPrep` - not initialized.
- 392 `m_pKCasual` - not initialized.
- 393 `m_pWnafPrepared` - not initialized.
- 423 `m_pCasual` - not initialized.
- 424 `m_ppPrepared` - not initialized.
- 427 `m_pWnafPrepared` - not initialized.
- 545 `m_AddPt` - not initialized.
- 629 `secp256k1_hmac_sha256_t` - not initialized.
- 633 `Mac() {} -> Mac()=default`
- 651 `m_bFirstTime` - not initialized.
- 659 `NonceGenerator(const char(&szSalt)[nSalt]) -> explicit NonceGenerator(const char(&szSalt)[nSalt])`
- 691 `HKdf(const HKdf&) = delete;` - deleted constructor better be public.
- 737 `HKdfPub(const HKdfPub&) = delete;` - deleted constructor better be public.
- 792 `H` - not initialized.
- 799 `m_pGet1_Minus` - not initialized.
- 816 `m_Casual` - not initialized.
- 829 `m_Casual` - not initialized.
- 834 `Context() {} -> Context()=default;`
- 853 `Scope(PseudoRandomGenerator* p) -> explicit Scope(PseudoRandomGenerator* p)`
- 875 `Scope(BatchContext& bc) -> explicit Scope(BatchContext& bc)`
- 919 `BatchContext(uint32_t nCasualTotal); -> explicit BatchContext(uint32_t nCasualTotal);`
- 980 `Nonces() {} -> Nonces()=default;`
- 981 `Nonces(const uintBig& seedSk) -> explicit Nonces(const uintBig& seedSk)`

dao-core/beam/core/radixtree.h

- 32 `operator bool() const` -> `explicit operator bool() const`
- 108 `uint16_t m_nBits;` - not initialized.
- 109 `uint16_t m_nPtrs;` - not initialized.
- 110 `uint16_t m_nPosInLastNode;` - not initialized.
- 122 `CursorBase(Node** pp) :m_pp(pp) {}` ->
  `explicit CursorBase(Node** pp) :m_pp(pp) {}`
- 134 `Node* m_ppBuf[nKeyBits + 1];` - not initialized.
- 151 `const uint8_t* m_pBound[2];` - not initialized.
- 154 `:m_pCu(NULL)` -> `:m_pCu(nullptr)`
- 274 `union` - not initialized.
- 290 `return Cast::Up<MyLeaf>(RadixTree::Find(cu, key.V.m_pData, key.s_Bits, bCreate));` ->
  `return Cast::Up<MyLeaf>(RadixTree::Find(cu, key.V.m_pData, beam::UtxoTree::Key::s_Bits, bCreate));` - better do not access static member through instance.
- 321 `uint16_t m_nBitsCommon;` - not initialized.
- 355 `Serializer(Archive& ar) :m_ar(ar) {}` ->
  `explicit Serializer(Archive& ar) :m_ar(ar) {}`

dao-core/beam/core/block_crypt.h

- 61 `Height m_Min;` - not initialized.
- 62 `Height m_Max;` - not initialized.
- 73 `HeightRange(Height h)` -> `explicit HeightRange(Height h)`
- 104 `Height m_Height;` - not initialized.
- 114 `Height m_Height;` - not initialized.
- 115 `uint32_t m_Pos;` - not initialized.
- 117 `HeightPos() {}` -> `HeightPos()=default;`
- 118 `HeightPos(Height h, uint32_t pos = 0)` ->
  `explicit HeightPos(Height h, uint32_t pos = 0)`
- 206 `Asset::ID m_Begin;` - not initialized.
- 220 `Scope(BatchContext& bc)` -> `explicit Scope(BatchContext& bc)`
- 245 `Scope(const Rules&)` -> `explicit Scope(const Rules&)`

- 369 `CoinID() {} -> CoinID()=default;`
- 370 `CoinID(Zero_) -> explicit CoinID(Zero_)`
- 425 `Generator(Asset::ID); -> explicit Generator(Asset::ID);`
- 438 `Worker(const CoinID&); -> explicit Worker(const CoinID&);`
- 458 `uint32_t m_Kernels;` - not initialized.
- 459 `uint32_t m_KernelsNonStd;` - not initialized.
- 460 `uint32_t m_Inputs;` - not initialized.
- 461 `uint32_t m_Outputs;` - not initialized.
- 462 `uint32_t m_InputsShielded;` - not initialized.
- 463 `uint32_t m_OutputsShielded;` - not initialized.
- 464 `uint32_t m_Contract;` - not initialized.
- 465 `uint32_t m_ContractSizeExtra;` - not initialized.
- 524 `Input() {} -> Input()=default;`
- 525 `Input(Input&& v) -> Input(Input&& v) noexcept` - move constructor should be `noexcept`.
- 526 `:TxElement(v) -> :TxElement(std::move(v))` - move condtructor calling a copy constructor from `ECC::Point`.
- 533 `void operator = (const Input&); -> Input& operator = (const Input&);`
- 566 `Amount m_Fee;` - not initialized.
- 567 `Amount m_Amount;` - not initialized.
- 615 `void operator = (const Output&); -> Output& operator = (const Output&);`
- 673 `void operator = (const ShieldedTxo&); -> ShieldedTxo& operator = (const ShieldedTxo&);`
- 682 `Key::Index m_nIdx;` - not initialized.
- 683 `bool m_IsCreatedByViewer;` - not initialized.
- 733 `uint8_t m_MaxPrivacyMinAnonymitySet;` - not initialized.
- 734 `uint64_t m_ReceiverOwnID;` - not initialized.
- 735 `uint8_t m_Padding[sizeof(m_pMessage) - sizeof(TxID) - sizeof(uint8_t) - sizeof(uint64_t)];` - not initialized.
- 761 `Amount m_Value;` - not initialized.
- 848 `virtual ~TxKernel() {} -> virtual ~TxKernel()=default;`

- 851 `virtual bool IsValid(Height hScheme, ECC::Point::Native& exc, const TxKernel* pParent = nullptr) const = 0;` - best practice is to avoid default parameters in virtual functions.
- 917 `Height m_LockHeight;` - not initialized.
- 925 `virtual ~TxKernelStd() {}` -> `~TxKernelStd() override = default;`
- 987 `virtual ~TxKernelAssetCreate() {}` -> `~TxKernelAssetEmit() override =default;`
- 1013 `virtual ~TxKernelAssetCreate() {}` -> `~TxKernelAssetEmit() override =default;`
- 1028 `virtual ~TxKernelAssetCreate() {}` -> `~TxKernelAssetEmit() override =default;`
- 1053 `virtual ~TxKernelAssetCreate() {}` -> `~TxKernelAssetEmit() override =default;`
- 1089 `virtual ~TxKernelContractCreate() {}` -> `~TxKernelContractCreate() override =default;`
- 1105 `virtual ~TxKernelContractInvoke() {}` -> `~TxKernelContractInvoke() override =default;`
- 1124 `const Input* m_pUtxoIn;` - not initialized.
- 1125 `const Output* m_pUtxoOut;` - not initialized.
- 1126 `const TxKernel* m_pKernel;` - not initialized.
- 1128 `virtual ~IReader() {}` -> `virtual ~IReader()=default;`
- 1143 `virtual ~IWriter() {}` -> `virtual ~IWriter()=default;`
- 1174 `size_t m_pIdx[3];` - not initialized.
- 1203 `return Reader(*this, *this);` -> `return {*this, *this};`
- 1298 `Height m_Height;` - not initialized.
- 1328 `Height m_Height;` - not initialized.
- 1543 `uint32_t m_iVerifier;` - not initialized.
- 1545 `Context(const Params& p)` -> `explicit Context(const Params& p)`
- 1590 `bool IsValid(SystemState::Full* pTip = NULL) const;` -> `bool IsValid(SystemState::Full* pTip = nullptr) const;`

dao_core/beam/utility/common.h

- 17 `#include <assert.h>` -> `<cassert>` header is deprecated. Better to use this one.
- 39 `#include <stdint.h>` -> `<cstdint>`
- 40 `#include <string.h>` -> `<cstring>`
- 54 `#endif // verify` -> `#endif // BEAM_VERIFY`
- 180 `Blob() {}` -> `Blob()=default;`
- 182 `Blob(const ByteBuffer& bb);` -> `explicit Blob(const ByteBuffer& bb);`
- 184 `Blob(const std::array<uint8_t, nBytes_>& x)` -> `explicit Blob(const std::array<uint8_t, nBytes_>& x)`
- 188 `Blob(const uintBig_t<nBytes_>& x)` -> `explicit Blob(const uintBig_t<nBytes_>& x)`

dao_core/beam/core/lelantus.h

- 59 `Cfg() {}` -> `Cfg()=default`
- 129 `Prover(CmList& lst, const Cfg& cfg, Proof& proof)` - fields `Prover::m_a`, `Prover::m_Tau`, `Prover::m_p` are not initialized.
- 139, 208 `m_L` not initialized

This analysis was performed by Pessimistic:

Sergey Grigoriev, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

September 14, 2021