Cyberscope

Audit Report

Dynex

January 2024

Repository	https://github.com/dynexcoin/Dynex/tree/dynex_d30c774
Commit	d30c7745a4fbad4f00cc1000f7e405105b3cf302
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/dynexcoin/Dynex/tree/dynex_d30c774
Commit	d30c7745a4fbad4f00cc1000f7e405105b3cf302

Audit Updates

Initial Audit	17 Jan 2024

Source Files

Filename	SHA256
/WalletUtils.h	10bd68a4ed4a55a0ec6edd5b8e332f2aedccaae11b53046b5f7439643b553413
/WalletUtils.cpp	8eb76e597bfd58e0ca5c003b579d70b2ead8b174bd9fe5f352088e94527c67d1
/WalletSerializationV2.h	b2112c2789eae146329f45f6dc86884699a70b4c41d3d1b4625c34886b04a160
/WalletSerializationV2.cpp	745fcd7b00fe7c8f54a0660d84de756f45b8411b0c774b828d7fd3649257e89a
/WalletSerializationV1.h	86c1be1762f7430b50f7f371a85666ca2e448be1de21486e5264f8724a8bb688
/WalletSerializationV1.cpp	47584402b447a3a952570af6aa572d92e49f05ee7ab0f91a6efb9c1fe20c4a0d
/WalletRpcServerErrorCodes.h	1aef8eb45fa173df0da90dbfddac23c98e4cae7d8472cfd8b761adeef1c10910
/WalletRpcServerCommandsDefinitions.h	d6f41e6dac9c99fcc2f0e7fb8d80b08e572a33346a2f31d2f8e857fd58905074
/WalletRpcServer.h	66abca7677b45c2fab27eff2528936a78f732fbb3e0a202977f74921a1861061
/WalletRpcServer.cpp	b72bfc5a09391dd9cd69fe934567b2d2feabce79f27e5837c0eb6aef5bd2f565
/WalletIndices.h	dfdd2d9d873d8e9c109d66de7b750c7422bdd2580fdea1cbd5e53d3c6f2792c7
/WalletGreen.h	18807b750a075e2c66552f7b1d1cf6ac97ad4a4a9a5a844e284e3478557cfedc
/WalletGreen.cpp	be6a0c1fe98cc07b03a4bcc0c2b05489493bf04fe678252bfc4fc6b05b9b6733

/WalletErrors.h	243ee5cc3b0152adea59bad3519dcaea85868f190b0191a3cc94fea663cc86c0
/WalletErrors.cpp	8d0bf4cc7dd97fd2b8cc89ea140dc088f36555ad801d91d91b079bb12cce340a
/WalletAsyncContextCounter.h	13c3a262c7e28e9663499fb3b50ba057a0819da18ce9a0ec41820e1f4ad85fde
/WalletAsyncContextCounter.cpp	e63a35b3deffa1d6653d605b6fcc7021595c8accdc8ba77fe83cac1a16b99c38
/LegacyKeysImporter.h	1147933c055017613c7456af9203aabb96f3cf8239ecbf8cc7be8c0d7de8d3c4
/LegacyKeysImporter.cpp	079bbaf239a363e8cbd810033ef5db1d4c16c9c8613acc04f32ff7016e6b9a68
/IFusionManager.h	27a6df6413a01321184729ece602eff733b2c6ad4cdf4f0e40e40f9d9041b5fe

Overview

Dynex, as described on its GitHub repository, is a next-generation platform for neuromorphic computing based on a new flexible blockchain protocol. This technology involves a network of nodes contributing to a massive neuromorphic computing network, aiming for high-speed and efficient computations. The codebase is primarily in C++, supplemented by some C components.

Audit Scope

The audit scope of the C++ code audit focuses on the `Wallet` folder of the release `dynex_d30c774` of the repository. This includes a thorough examination of key areas crucial for functionality and security. The audit primarily concentrates on security vulnerabilities, potential optimizations for performance, code maintainability, and adherence to best coding practices. Additionally, the audit evaluates the robustness of error handling mechanisms, the efficiency of network communication protocols, and the proper use of dependencies and external libraries. Each of these aspects is assessed to ensure the overall reliability and security of the wallet functionality.

Architecture

The Dynex blockchain protocol, according to the repository, is designed for neuromorphic computing, which suggests a unique architectural approach. This architecture might involve specialized data structures and algorithms optimized for neuromorphic processing. However, without direct access to the architectural documentation, the specifics of this architecture remain unclear from the GitHub overview.

Wallet Subsystem

The focus of this audit is the Wallet component of the Dynex project. In blockchain technology, a wallet typically handles key management, transaction creation, and sometimes node interactions. Understanding its implementation, security measures, and integration with the broader system is crucial.

Security and Performance Considerations

Given the Wallet's role in managing keys and transactions, security is paramount. This includes scrutinizing cryptographic implementations, key management, and transaction signing mechanisms. Additionally, performance aspects, especially in transaction processing and network communication, should be evaluated.

Code Quality and Maintainability

The Wallet code is reviewed for clarity, maintainability, and adherence to C++ best practices. This includes evaluating the use of modern C++ features, code modularity, and documentation.

Further Insight

A deeper dive into specific components or algorithms used in the Wallet, possibly comparing them with industry standards or similar implementations in other blockchain projects, would be beneficial.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

Diagnostics

Critical Medium Minor / Informative

everity	Code	Description	Status
	AMV	Accessing Moved Variable	Unresolved
	CAI	Constructor Assignment Inefficiency	Unresolved
	FPO	Function Parameter Optimization	Unresolved
	ISAC	Implicit Single Argument Constructors	Unresolved
	LVS	Local Variable Shadowing	Unresolved
	RCC	Redundant Condition Check	Unresolved
	SAE	STL Algorithm Efficiency	Unresolved
	UMV	Uninitialized Member Variable	Unresolved
	UAV	Unutilized Assigned Variable	Unresolved

AMV - Accessing Moved Variable

Criticality	Minor / Informative
Location	WalletGreen.cpp:2873,2875
Status	Unresolved

Description

The code in `WalletGreen.cpp` accesses the variable wallets after it has been moved. Specifically, the `wallets[0].wallet->spendSecretKey` and `wallets[0].wallet->spendPublicKey` are accessed. Accessing a moved-from object can lead to undefined behavior, as the state of the object after being moved is indeterminate.

Recommendation

It is recommended to avoid accessing variables after they have been moved. If the content of wallets is still needed after the move operation, consider making a copy before moving, or restructure the code to eliminate the need to access a moved-from object.

CAI - Constructor Assignment Inefficiency

Criticality	Minor / Informative
Location	WalletSerializationV1.cpp:93, 96, 101 WalletSerializationV2.cpp:62, 65, 70, 95
Status	Unresolved

Description

Several variables in different classes are assigned in the constructor body instead of using an initialization list. This is less efficient as it involves extra assignments. Examples include the `state`, `hash`, and `extra` variables in `WalletSerializationV1.cpp` and `WalletSerializationV2.cpp`.

Recommendation

It is recommended to refactor these constructors to use initialization lists for member variables. This approach improves performance by avoiding unnecessary default initializations and enhances code clarity.

FPO - Function Parameter Optimization

Criticality	Minor / Informative
Location	WalletSerializationV1.cpp:134,142,150,155,182,410 WalletSerializationV2.cpp:105,133,225,269,282,310,342,356,364,373,396
Status	Unresolved

Description

In the `WalletSerializationV1.cpp` and `WalletSerializationV2.cpp` files, there are multiple instances where function parameters are not optimized for efficiency. Parameters such as `address`, `serializer`, and others are frequently passed by value or as non-const references, leading to unnecessary data copying. This inefficiency is particularly pronounced in serialization functions and when handling large objects. Additionally, variables like `tx`, `walletsIndex`, and `index`, which are not modified within their respective scopes, could be more efficiently declared as references to const to avoid redundant copying.

Recommendation

To enhance performance and reduce overhead, it is recommended to revise the parameter passing strategy. Wherever possible, particularly in cases where parameters are not being modified, objects should be passed by const reference. This change will optimize the code by reducing unnecessary copying and improving efficiency, especially in the context of serialization functions and when dealing with large objects. The recommended modifications should be applied to `WalletSerializationV1.cpp`, `WalletSerializationV2.cpp`, and other similar instances throughout the codebase to ensure consistency and adherence to best coding practices.

ISAC - Implicit Single Argument Constructors

Criticality	Minor / Informative
Location	WalletSerializationV1.cpp:92, 120 WalletSerializationV2.cpp:61, 94
Status	Unresolved

Description

The code, particularly in `WalletSerializationV1.cpp` and `WalletSerializationV2.cpp`, includes single argument constructors that are not explicitly marked as explicit. This oversight can lead to unintentional implicit conversions, potentially causing subtle bugs.

Recommendation

It is recommended to mark these single argument constructors as explicit to prevent the compiler from using them in implicit conversions. This practice enhances code safety and clarity, reducing the risk of unintended type conversions.

LVS - Local Variable Shadowing

Criticality	Minor / Informative
Location	WalletGreen.cpp:1671, 1691, 2245, 2278 WalletGreen.cpp:2719,2929,2914
Status	Unresolved

Description

The contract uses the local variables `transfer`, `start`, `ephemeral`, and `i` that shadow other variables or functions. This shadowing can lead to confusion about which variable is being accessed and potentially lead to bugs, especially if the shadowed variable has a different scope or lifetime.

Recommendation

It is recommended to rename the local variables to avoid shadowing and ensure clear and maintainable code.

RCC - Redundant Condition Check

Criticality	Minor / Informative
Location	WalletGreen.cpp:1616,2505
Status	Unresolved

Description

The contract contains conditions that are always false, such as `mixIn != 0` and `dust`. These conditions are redundant due to prior assignments (mixIn = 0, dust = false) which make these checks always `false`. This redundancy can lead to confusion and may mask logical flaws in the code.

Recommendation

It is recommended to review and refactor these conditions to ensure they accurately reflect the intended logic and do not result in redundant checks.

SAE - STL Algorithm Efficiency

Criticality	Minor / Informative
Location	WalletGreen.cpp:1754, 2325, 2460, 2764, 2854, 2869
Status	Unresolved

Description

The contract could improve efficiency and readability by using Standard Template Library (STL) algorithms. For instance, raw loops in WalletGreen.cpp could be replaced with STL algorithms like `std::transform`, `std::find_if`, etc. These changes can make the code more concise and potentially optimize performance.

Recommendation

It is recommended to refactor the code to use STL algorithms where appropriate. This enhances readability and maintainability, and potentially performance as well.

UMV - Uninitialized Member Variable

Criticality	Minor / Informative
Location	WalletSerializationV1.cpp:120
Status	Unresolved

Description

The contract includes instances where member variables like `amount` and `type` in `WalletTransferDto` are not initialized in the constructor. This omission can lead to undefined behavior if these members are accessed before being explicitly set.

Recommendation

It is recommended to ensure all member variables are initialized in constructors, either through an initialization list or by setting default values within the class definition, to maintain predictable behavior and robustness.

UAV - Unutilized Assigned Variable

Criticality	Minor / Informative
Location	WalletGreen.cpp:781
Status	Unresolved

Description

The contract assigns a value to the variable `updatedTransactions` in `WalletGreen.cpp`, but this variable is never read or used afterward. Such unused variables can lead to unnecessary memory usage and can potentially confuse developers about the code's intent.

Recommendation

It is recommended to remove or utilize the `updatedTransactions` variable appropriately to ensure efficient use of resources and maintain code clarity.

SHA256 Codebase

SHA256	Filename
10bd68a4ed4a55a0ec6edd5b8e332f2aedccaae11b53046b5f7439643b553413	WalletUtils.h
b39727f870feb0f54b438abcba8d4c11fff8530b9ccbe5a35db08c3bcc325dd2	WalletUtils.cpp
b2112c2789eae146329f45f6dc86884699a70b4c41d3d1b4625c34886b04a160	WalletSerializationV2.h
745fcd7b00fe7c8f54a0660d84de756f45b8411b0c774b828d7fd3649257e89a	WalletSerializationV2.cpp
86c1be1762f7430b50f7f371a85666ca2e448be1de21486e5264f8724a8bb688	WalletSerializationV1.h
47584402b447a3a952570af6aa572d92e49f05ee7ab0f91a6efb9c1fe20c4a0d	WalletSerializationV1.cpp
1aef8eb45fa173df0da90dbfddac23c98e4cae7d8472cfd8b761adeef1c10910	WalletRpcServerErrorCodes.h
d6f41e6dac9c99fcc2f0e7fb8d80b08e572a33346a2f31d2f8e857fd58905074	WalletRpcServerCommandsDefinitions.h
66abca7677b45c2fab27eff2528936a78f732fbb3e0a202977f74921a1861061	WalletRpcServer.h
b72bfc5a09391dd9cd69fe934567b2d2feabce79f27e5837c0eb6aef5bd2f565	WalletRpcServer.cpp
dfdd2d9d873d8e9c109d66de7b750c7422bdd2580fdea1cbd5e53d3c6f2792c7	WalletIndices.h
18807b750a075e2c66552f7b1d1cf6ac97ad4a4a9a5a844e284e3478557cfedc	WalletGreen.h
be6a0c1fe98cc07b03a4bcc0c2b05489493bf04fe678252bfc4fc6b05b9b6733	WalletGreen.cpp
243ee5cc3b0152adea59bad3519dcaea85868f190b0191a3cc94fea663cc86c0	WalletErrors.h
8d0bf4cc7dd97fd2b8cc89ea140dc088f36555ad801d91d91b079bb12cce340a	WalletErrors.cpp
13c3a262c7e28e9663499fb3b50ba057a0819da18ce9a0ec41820e1f4ad85fde	WalletAsyncContextCounter.h
e63a35b3deffa1d6653d605b6fcc7021595c8accdc8ba77fe83cac1a16b99c38	WalletAsyncContextCounter.cpp
1147933c055017613c7456af9203aabb96f3cf8239ecbf8cc7be8c0d7de8d3c4	LegacyKeysImporter.h
079bbaf239a363e8cbd810033ef5db1d4c16c9c8613acc04f32ff7016e6b9a68	LegacyKeysImporter.cpp
27a6df6413a01321184729ece602eff733b2c6ad4cdf4f0e40e40f9d9041b5fe	IFusionManager.h

Summary

The Dynex code implements a comprehensive digital wallet system in the release. This audit investigates security issues, business logic concerns, potential improvements in performance, and adherence to best coding practices. The code review and auditing process have uncovered some key areas for improvement within the C++ codebase.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io