Unich

OTC Contracts
Solana & EVM

SMART CONTRACT AUDIT

11.03.2025

Made in Germany by Softstack.io

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of UNICH GLOBAL PTE. LTD. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (19.02.2025)	Layout
0.4 (21.02.2025)	Automated Security Testing Manual Security Testing
0.5 (24.02.2025)	Verify Claims
0.9 (26.02.2025)	Summary and Recommendation
1.0 (12.03.2025)	Final document

2. About the Project and Company

Company address:

UNICH GLOBAL PTE. LTD.
10 Anson Road #13-09 International Plaza
Singapore

Website: https://unich.com

Twitter (X): https://twitter.com/unich_com

Telegram: http://t.me/unich_com

Discord: https://discord.gg/unich

Facebook: https://www.facebook.com/Unich.official

YouTube: https://www.youtube.com/@Unichcom

Instagram: https://www.instagram.com/unich_com

TikTok: https://www.tiktok.com/@unichotc

2.1 Project Overview

Unich is a decentralized OTC trading platform designed to revolutionize peer-to-peer asset exchanges by providing secure, efficient, and cost-effective trading solutions.

At its core, Unich leverages blockchain technology and smart contracts to eliminate intermediaries, ensuring transparency and trustless execution for traders. The Unich platform introduces a seamless trading experience across multiple OTC markets, including Pre-Market OTC for early-stage token sales, Point-Market OTC for tokenized credit exchanges, and Options OTC-Market for decentralized derivatives trading. By integrating Web3 functionalities directly into its platform, Unich enhances accessibility and usability, positioning itself as a leader in decentralized trading solutions.

Unich operates through its native utility token, which plays a critical role in facilitating transactions, incentivizing liquidity providers, and ensuring economic security within the ecosystem. By implementing staking mechanisms and governance participation, the platform fosters community-driven development and sustainable growth.

Beyond its trading capabilities, Unich is committed to expanding its ecosystem across various blockchain networks, ensuring interoperability and cross-chain trading. Future developments will focus on integrating Unich within mobile applications, social media platforms, and institutional trading tools to drive mainstream adoption. Unich's long-term vision is to become the leading decentralized OTC trading hub, bridging the gap between traditional finance and blockchain technology. By prioritizing security, efficiency, and user empowerment, Unich aims to establish a global standard for OTC trading while maintaining full decentralization and user control over assets.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert auditors and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

EVM Contract

File	Fingerprint (MD5)
./contracts/diamond/facets/Test2Facet.sol	de378098a7ee8e5d17f38f329b335739
./contracts/diamond/facets/DiamondCutFacet.sol	646a7670b2177935080520c3bd5084cd
./contracts/diamond/facets/OwnershipFacet.sol	3ecfa87738e59d567e6c44c1708b9a7d
./contracts/diamond/facets/DiamondLoupeFacet.sol	3b52f9432f8f55361a1785228f77fe9f
./contracts/diamond/facets/Test1Facet.sol	0748c631993903370052c649ec6e0dc4
./contracts/diamond/upgradeInitializers/DiamondInit.sol	9719f08d9857ace8a598fae3720cea0b
./contracts/diamond/upgradeInitializers/DiamondMultiInit.sol	0d76218688249ca94ee155df617c0dfb
./contracts/diamond/libraries/LibDiamond.sol	bed93ccf9210e77ea95276c13d6a7514
./contracts/diamond/Diamond.sol	56ce1c343a15448c164669177a33f345
./contracts/diamond/interfaces/IDiamondCut.sol	764b64e115c5859c3676c3e990df12fb
./contracts/diamond/interfaces/IDiamondLoupe.sol	3e7c5880a6349d66b10477df5c66dc22
./contracts/diamond/interfaces/IDiamond.sol	a436ec95c32c361b7f802e87d7113dde
./contracts/diamond/interfaces/IERC165.sol	f8f9ee2c76e1678bfead7a3fad131934
./contracts/diamond/interfaces/IERC173.sol	65e379207bc221335d0eedfe7b077af6
./contracts/impl/OTCGetter.sol	e0a20f133f184a66c55c21cc2915d222
./contracts/impl/OTCMatchEngine.sol	cf3f066a36e9cfbc7743bad7c011b6ff

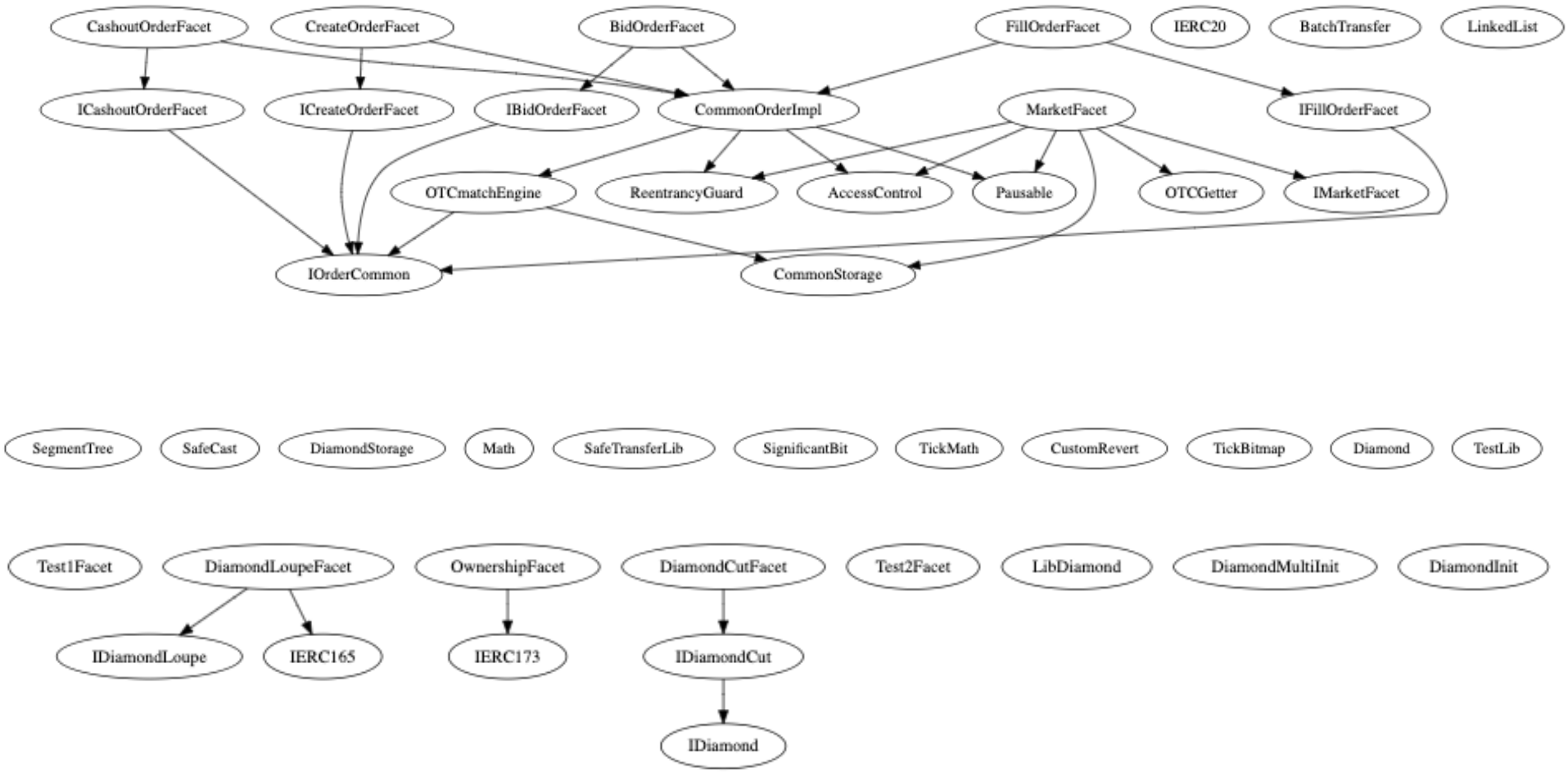
./contracts/impl/CommonOrderImpl.sol	36d0d57739614b226e73d8a894e2c514
./contracts/impl/CommonStorage.sol	20939f3bfbd0830ea549f08fd3dd6917
./contracts/MarketFacet.sol	00feb6db8d1b8c96d6d927da1c80f122
./contracts/BidOrderFacet.sol	84257b56d2e92b0f330019d37399fe5f
./contracts/FillOrderFacet.sol	5c82b7ef9cb6da30ed90fb3c4cf8ce35
./contracts/CashoutOrderFacet.sol	7aefba5dbdc922981ff09a66199a80a2
./contracts/libraries/TickBitmap.sol	c08eeb6f760e21b23d56a2b93b0dcbcd
./contracts/libraries/CustomRevert.sol	f325914772c97e25c851372a4349c656
./contracts/libraries/TickMath.sol	2c0b60a64be7892531a4dae5ff479b23
./contracts/libraries/SignificantBit.sol	0a0832c027b7548b5046847017295bc3
./contracts/libraries/SafeTransfer.sol	0832cfb4ecc7633bc59f170d423370f8
./contracts/libraries/Math.sol	d84a6c8a724bc6ab679280901397f1b4
./contracts/libraries/LinkedList.sol	a1745940ab01d00d2d4aacd9293d25ce
./contracts/libraries/DiamondStorage.sol	f5208f24448604e4b7bc993f90b27f88
./contracts/libraries/SafeCast.sol	05042967e61a8584078eb413b98abf0f
./contracts/libraries/SegmentTree.sol	7f75408aee331806c8e59ea0f6b27642
./contracts/BatchTransfer.sol	8ecda4638a65ac28073bd0a449419c85
./contracts/CreateOrderFacet.sol	67a8e9ee2b43641b1476914235c51937
./contracts/interfaces/IFillOrderFacet.sol	b6d51f9aae84a628f8f8673e54408dcc
./contracts/interfaces/ICreateOrderFacet.sol	e15c24a2c4f25323a00c01e4d1da3661
./contracts/interfaces/IMarketFacet.sol	f7af38ac648d0c3daaf2bc711e6c1140
./contracts/interfaces/IBidOrderFacet.sol	04441ddb84cbe67dc836c2f32abec39e
./contracts/interfaces/IOrderCommon.sol	5df771999dd0a5e4364a19ec4b7f224e
./contracts/interfaces/ICashoutOrderFacet.sol	621676d3cf6bafd405f35074fc3eb624

Solana Program

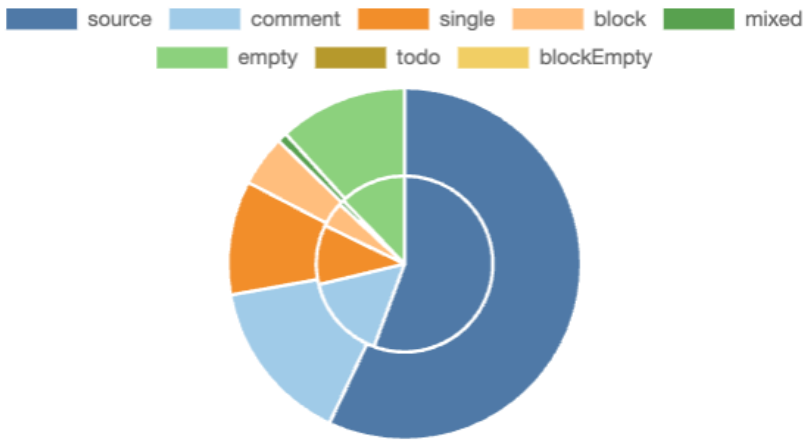
File	Fingerprint (MD5)
./programs/otc/src/instructions/system_instructions/update_wallets.rs	e85ff9dede72b0371e30027af16709c0
./programs/otc/src/instructions/system_instructions/new_market.rs	7f022bdd0322214b17102f55d8b3bfbc
./programs/otc/src/instructions/system_instructions/force_close_cashout_order.rs	a6a76faf212fd49febe6625f92c429d7
./programs/otc/src/instructions/system_instructions/remove_role.rs	466ccf848c5ac1b084d23c41ac858ac5
./programs/otc/src/instructions/system_instructions/initialize.rs	47d38de53ffd08e0ee1a04fbc9a694f5
./programs/otc/src/instructions/system_instructions/admin/init_owner.rs	478c2076a29dee8fbdd304bdadca298c
./programs/otc/src/instructions/system_instructions/admin/unpause_system.rs	a813682490d9aca0cd171b3092d42650
./programs/otc/src/instructions/system_instructions/admin/pause_system.rs	03319f218e48dfabe208c264ec28da1b
./programs/otc/src/instructions/system_instructions/admin/remove_owner.rs	8a0f2dc980826bf5c69c24e7f5af80e2
./programs/otc/src/instructions/system_instructions/admin/set_owner_role.rs	90e1873f55434ce103252c0fccccaea5
./programs/otc/src/instructions/system_instructions/admin/mod.rs	4f716e4ac058131d68ab1585792a8045
./programs/otc/src/instructions/system_instructions/mod.rs	1f844d590efb512e52615bf90e6fd79b
./programs/otc/src/instructions/system_instructions/settle_market.rs	a681c4a7f8a26696cc65c6a6ef8e59bb
./programs/otc/src/instructions/system_instructions/new_native_market.rs	2b25d816df1a5f2814dff147df25b99d
./programs/otc/src/instructions/system_instructions/update_market.rs	efb20ed0932bd341ca5ac44f76d6c1bd
./programs/otc/src/instructions/system_instructions/force_close_trade_native.rs	56b6f3ec0fc0c50adf3a41551f0cbbc3
./programs/otc/src/instructions/system_instructions/force_close_trade.rs	75be3d4df7b88d18f7a95c34d11318d3
./programs/otc/src/instructions/system_instructions/set_role.rs	117c35a0f6a26791af992881ee16eb59
./programs/otc/src/instructions/system_instructions/force_close_order.rs	0d40fc8750399c748ac7cc74f7e53677
./programs/otc/src/instructions/system_instructions/update_config.rs	cdd6c6419211e653e59417538202d610
./programs/otc/src/instructions/system_instructions/force_cancel_settle_phase.rs	b63cbe49e02c1462928aa71233c29689
./programs/otc/src/instructions/user_instructions/cashout_trade.rs	4810a15a968b44db932b541f11cdf343
./programs/otc/src/instructions/user_instructions/settle_filled.rs	2ee6a217c75c3d93de290949f2bc051d
./programs/otc/src/instructions/user_instructions/match_bid_cashout_order_native.rs	b9d120ed7f4529c4ea4a3e791e74ff2e
./programs/otc/src/instructions/user_instructions/fill_cashout_order_native.rs	4fe1364f08840ebd71287faf3f3fa22c
./programs/otc/src/instructions/user_instructions/match_cashout_order_native.rs	9fa9e94c80ebba85aa1d38a5ddd38af4
./programs/otc/src/instructions/user_instructions/match_order.rs	58979c9598f90b952813702cc6b13f25
./programs/otc/src/instructions/user_instructions/create_order.rs	e69fddf6b02e4685f5c3982434c646ba
./programs/otc/src/instructions/user_instructions/settle_canceled_native.rs	9f28be167f831de3f988eab440475041
./programs/otc/src/instructions/user_instructions/match_order_native.rs	8c2c79fc6989729b6e731f6e794e3100

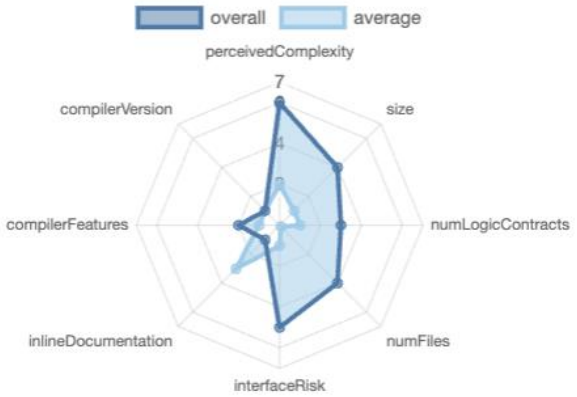
./programs/otc/src/instructions/user_instructions/fill_cashout_order.rs	be03021bae027b3d6a10bef90df161fd
./programs/otc/src/instructions/user_instructions/cancel_order_native.rs	1b685e56fffe349d05631d400a3a9a12
./programs/otc/src/instructions/user_instructions/cancel_order.rs	3977152bc552252aadff54a6ff592c02
./programs/otc/src/instructions/user_instructions/mod.rs	e6fa0cb57c9186f5761715bad133adfe
./programs/otc/src/instructions/user_instructions/settle_canceled.rs	d2bf725d9a4b40acc7cc00e90401cbd2
./programs/otc/src/instructions/user_instructions/cancel_cashout_order.rs	13743f702f75f102f24e518590eabc93
./programs/otc/src/instructions/user_instructions/create_order_native.rs	967c65d1823fecbf6e85afddeb0d03ba
./programs/otc/src/instructions/user_instructions/fill_order.rs	cc0ff7149bd50aaf3a4beb06397a88f
./programs/otc/src/instructions/user_instructions/match_cashout_order.rs	1ddab342998d7a2b7cfd8370a41b52e1
./programs/otc/src/instructions/user_instructions/fill_order_native.rs	be1bab64dc66c8f45e7e396194c55f0
./programs/otc/src/instructions/user_instructions/match_bid_order_native.rs	c6908c091b5c9297527ff548a9c1465a
./programs/otc/src/instructions/user_instructions/match_bid_cashout_order.rs	411dd0eee5d8ac33f7d22e2ccebf9fd9
./programs/otc/src/instructions/user_instructions/cashout_trade_native.rs	41dc0c505297216f08c34d6c623ecee7
./programs/otc/src/instructions/user_instructions/match_bid_order.rs	f207bf69c57d2ed88c12b85db3c0ab96
./programs/otc/src/instructions/user_instructions/settle_filled_native.rs	b43c318edc5aca3e2eb549fc8976d76b
./programs/otc/src/instructions/mod.rs	eb5f4ccf6289a7777b14f22df78dd08
./programs/otc/src/events.rs	2130b329cf800292ffd87cbd4074c8c8
./programs/otc/src/constants.rs	d85452dba8600ba954c5ceacd7621d9
./programs/otc/src/lib.rs	4b78f9a9409c0e9a2e5228462ef7b807
./programs/otc/src/utils/mod.rs	7af5dabe2b3efd083d3cbbb90630eaec
./programs/otc/src/utils/spl.rs	3a172ebf1a56eacb8b81de59463ca178
./programs/otc/src/utils/math.rs	fb0c3b778aec805ba5ea9a36b1e38c50
./programs/otc/src/utils/utils.rs	39c93707fc7d9bb1acd30feb8c6b85e5
./programs/otc/src/errors.rs	13887ee9e8699ae5c41e292820fa09f3
./programs/otc/src/states/market_account.rs	c4f6176c91ab6189de0659ea636f926c
./programs/otc/src/states/role_account.rs	7097a60c80a99f3a014c7a5a80e04175
./programs/otc/src/states/config_account.rs	1064ad2964f9ac7c143bb36ed8216b59
./programs/otc/src/states/mod.rs	d1aa9ba5d47afe6035de0eeea51e7dd8
./programs/otc/src/states/order_account.rs	230e646d32a6c1c6775d52b7d4907619
./programs/otc/src/states/trade_account.rs	265c8f06a8ec2ab7d3471bb9921b7b91
./programs/otc/src/states/vault_account.rs	e28fe06308607230f822506993f2f623

5.2 Inheritance Graph (EVM)



5.3 Source Lines & Risk (EVM)





5.4 Capabilities (EVM)

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
^0.8.24 ^0.8.0 >=0.8.0		yes	yes (62 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
		yes	yes		

Public	Payable
120	14

External	Internal	Private	Pure	View
115	261	6	40	59

5.5 Source Unites in Scope

EVM Contracts

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines	Complex. Score
evm-contracts-prod/contracts/impl/CommonStorage.sol	1		118	85	11	68
evm-contracts-prod/contracts/impl/CommonOrderImpl.sol	1		102	68	5	36
evm-contracts-prod/contracts/impl/OTCMatchEngine.sol	1		850	538	142	231
evm-contracts-prod/contracts/impl/OTCGetter.sol	1		38	29	1	19
evm-contracts-prod/contracts/CreateOrderFacet.sol	1		122	67	26	56
evm-contracts-prod/contracts/BatchTransfer.sol	1	1	53	30	1	43
evm-contracts-prod/contracts/CashoutOrderFacet.sol	1		189	122	16	63
evm-contracts-prod/contracts/FillOrderFacet.sol	1		133	101	15	47
evm-contracts-prod/contracts/BidOrderFacet.sol	1		274	211	18	90
evm-contracts-prod/contracts/MarketFacet.sol	1		781	435	169	329
evm-contracts-prod/contracts/libraries/LinkedList.sol	1		88	65	13	7
evm-contracts-prod/contracts/libraries/SegmentTree.sol	1		57	37	13	38
evm-contracts-prod/contracts/libraries/SafeCast.sol	1		68	34	24	23
evm-contracts-prod/contracts/libraries/DiamondStorage.sol	1		125	82	27	8
evm-contracts-prod/contracts/libraries/Math.sol	1		219	118	91	363

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines	Complex. Score
evm-contracts-prod/contracts/libraries/SafeTransfer.sol	1		113	53	47	181
evm-contracts-prod/contracts/libraries/SignificantBit.sol	1		53	42	8	162
evm-contracts-prod/contracts/libraries/TickMath.sol	1		273	208	43	641
evm-contracts-prod/contracts/libraries/CustomRevert.sol	1		95	68	17	189
evm-contracts-prod/contracts/libraries/TickBitmap.sol	1		113	91	5	68
evm-contracts-prod/contracts/interfaces/ICashoutOrderFacet.sol		1	8	4	1	5
evm-contracts-prod/contracts/interfaces/IOrderCommon.sol		1	52	43	3	1
evm-contracts-prod/contracts/interfaces/IBidOrderFacet.sol		1	8	4	1	5
evm-contracts-prod/contracts/interfaces/IMarketFacet.sol		1	84	23	19	32
evm-contracts-prod/contracts/interfaces/ICreateOrderFacet.sol		1	18	4	1	12
evm-contracts-prod/contracts/interfaces/IFillOrderFacet.sol		1	8	4	1	8
evm-contracts-prod/contracts/diamond/Diamond.sol	1		71	42	21	55
evm-contracts-prod/contracts/diamond/facets/Test1Facet.sol	2		80	50	2	56
evm-contracts-prod/contracts/diamond/facets/DiamondLoupeFacet.sol	1		147	96	44	142

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines	Complex. Score
evm-contracts-prod/contracts/diamond/facets/OwnershipFacet.sol	1		16	12	1	10
evm-contracts-prod/contracts/diamond/facets/DiamondCutFacet.sol	1		30	9	13	7
evm-contracts-prod/contracts/diamond/facets/Test2Facet.sol	1		44	23	1	41
evm-contracts-prod/contracts/diamond/interfaces/IERC165.sol		1	12	3	7	3
evm-contracts-prod/contracts/diamond/interfaces/IERC173.sol		1	19	4	10	5
evm-contracts-prod/contracts/diamond/interfaces/IDiamond.sol		1	20	10	6	1
evm-contracts-prod/contracts/diamond/interfaces/IDiamondLoupe.sol		1	38	7	20	9
evm-contracts-prod/contracts/diamond/interfaces/IDiamondCut.sol		1	24	4	11	5
evm-contracts-prod/contracts/diamond/libraries/LibDiamond.sol	1		233	185	17	130
evm-contracts-prod/contracts/diamond/upgradeInitializers/DiamondMultiInit.sol	1		28	13	9	11
evm-contracts-prod/contracts/diamond/upgradeInitializers/DiamondInit.sol	1		43	30	6	8

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines	Complex. Score
Totals	30	12	4847	3054	886	3208

Solana Programs

File	Code Lines	Comment Lines	Blank Lines	Total
solana-program-prod/programs/otc/src/constants.rs	49	0	3	52
solana-program-prod/programs/otc/src/errors.rs	92	0	2	94
solana-program-prod/programs/otc/src/events.rs	200	0	26	226
solana-program-prod/programs/otc/src/instructions/mod.rs	4	0	2	6
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/init_owner.rs	83	7	17	107
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/mod.rs	10	0	1	11
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/pause_system.rs	51	0	12	63
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/remove_owner.rs	54	2	11	67
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/set_owner_role.rs	55	2	12	69
solana-program-prod/programs/otc/src/instructions/system_instructions/admin/unpause_system.rs	53	0	12	65
solana-program-prod/programs/otc/src/instructions/system_instructions/force_cancel_settle_phase.rs	45	1	11	57

solana-program-prod/programs/otc/src/instructions/system_instructions/force_close_cashout_order.rs	105	1	16	122
solana-program-prod/programs/otc/src/instructions/system_instructions/force_close_order.rs	107	1	18	126
solana-program-prod/programs/otc/src/instructions/system_instructions/force_close_trade.rs	188	4	27	219
solana-program-prod/programs/otc/src/instructions/system_instructions/force_close_trade_native.rs	164	1	26	191
solana-program-prod/programs/otc/src/instructions/system_instructions/initialize.rs	60	2	10	72
solana-program-prod/programs/otc/src/instructions/system_instructions/mod.rs	30	0	1	31
solana-program-prod/programs/otc/src/instructions/system_instructions/new_market.rs	82	1	15	98
solana-program-prod/programs/otc/src/instructions/system_instructions/new_native_market.rs	69	1	13	83
solana-program-prod/programs/otc/src/instructions/system_instructions/remove_role.rs	62	1	13	76
solana-program-prod/programs/otc/src/instructions/system_instructions/set_role.rs	66	1	14	81
solana-program-prod/programs/otc/src/instructions/system_instructions/settle_market.rs	76	1	17	94
solana-program-prod/programs/otc/src/instructions/system_instructions/update_config.rs	41	0	10	51
solana-program-prod/programs/otc/src/instructions/system_instructions/update_market.rs	72	1	15	88
solana-program-prod/programs/otc/src/instructions/system_instructions/update_wallets.rs	62	0	12	74
solana-program-prod/programs/otc/src/instructions/user_instructions/cancel_cashout_order.rs	85	1	13	99

solana-program-prod/programs/otc/src/instructions/user_instructions/cancel_order.rs	106	1	16	123
solana-program-prod/programs/otc/src/instructions/user_instructions/cancel_order_native.rs	74	1	13	88
solana-program-prod/programs/otc/src/instructions/user_instructions/cashout_trade.rs	309	6	34	349
solana-program-prod/programs/otc/src/instructions/user_instructions/cashout_trade_native.rs	277	6	35	318
solana-program-prod/programs/otc/src/instructions/user_instructions/create_order.rs	138	2	20	160
solana-program-prod/programs/otc/src/instructions/user_instructions/create_order_native.rs	115	1	17	133
solana-program-prod/programs/otc/src/instructions/user_instructions/fill_cashout_order.rs	339	41	37	417
solana-program-prod/programs/otc/src/instructions/user_instructions/fill_cashout_order_native.rs	289	40	35	364
solana-program-prod/programs/otc/src/instructions/user_instructions/fill_order.rs	205	4	28	237
solana-program-prod/programs/otc/src/instructions/user_instructions/fill_order_native.rs	152	3	24	179
solana-program-prod/programs/otc/src/instructions/user_instructions/match_bid_cashout_order.rs	424	38	62	524
solana-program-prod/programs/otc/src/instructions/user_instructions/match_bid_cashout_order_native.rs	373	37	61	471
solana-program-prod/programs/otc/src/instructions/user_instructions/match_bid_order.rs	302	10	45	357
solana-program-prod/programs/otc/src/instructions/user_instructions/match_bid_order_native.rs	263	10	43	316
solana-program-prod/programs/otc/src/instructions/user_instructions/match_cashout_order.rs	370	37	49	456

solana-program-prod/programs/otc/src/instructions/user_instructions/match_cashout_order_native.rs	330	35	49	414
solana-program-prod/programs/otc/src/instructions/user_instructions/match_order.rs	256	3	33	292
solana-program-prod/programs/otc/src/instructions/user_instructions/match_order_native.rs	228	3	32	263
solana-program-prod/programs/otc/src/instructions/user_instructions/mod.rs	46	0	3	49
solana-program-prod/programs/otc/src/instructions/user_instructions/settle_canceled.rs	267	6	36	309
solana-program-prod/programs/otc/src/instructions/user_instructions/settle_canceled_native.rs	219	7	35	261
solana-program-prod/programs/otc/src/instructions/user_instructions/settle_filled.rs	246	22	35	303
solana-program-prod/programs/otc/src/instructions/user_instructions/settle_filled_native.rs	220	8	34	262
solana-program-prod/programs/otc/src/lib.rs	399	1	51	451
solana-program-prod/programs/otc/src/states/config_account.rs	12	0	2	14
solana-program-prod/programs/otc/src/states/market_account.rs	32	1	6	39
solana-program-prod/programs/otc/src/states/mod.rs	12	0	2	14
solana-program-prod/programs/otc/src/states/order_account.rs	39	1	4	44
solana-program-prod/programs/otc/src/states/role_account.rs	15	0	2	17
solana-program-prod/programs/otc/src/states/trade_account.rs	65	3	12	80
solana-program-prod/programs/otc/src/states/vault_account.rs	3	0	2	5
solana-program-prod/programs/otc/src/utils/math.rs	7	0	3	10
solana-program-prod/programs/otc/src/utils/mod.rs	6	0	2	8
solana-program-prod/programs/otc/src/utils/spl.rs	120	0	6	126
solana-program-prod/programs/otc/src/utils/utils.rs	128	41	23	192

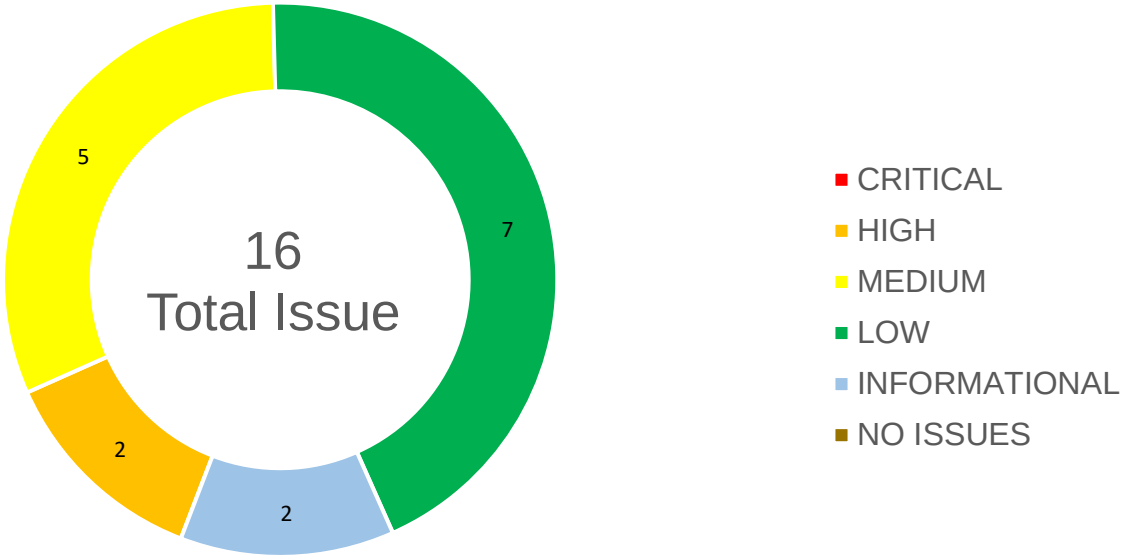
6. Scope of Work

The Unich team has provided the necessary smart contract files for EVM and Solana programs for auditing. The audit focuses on verifying the security, efficiency, and correctness of the OTC trading mechanisms, ensuring they are robust against potential vulnerabilities. The team has outlined the following security and functionality assumptions for the contract:

1. **Order Execution and Settlement Integrity** The audit should verify that the contract enforces proper order creation, matching, and settlement logic. This includes ensuring price and amount constraints are followed, preventing unauthorized cancellations, and confirming that trades cannot be settled multiple times.
2. **Secure Collateral and Fund Management** The contract must securely manage escrowed funds, ensuring that assets are locked and released only according to the defined contract logic. The audit will check for reentrancy vulnerabilities, fund drain risks, and ensure secure handling of token transfers.
3. **Robust Access Control and Administrative Privileges** The audit should verify that only designated roles can update critical contract parameters, execute emergency halts, or modify order book data. The security review will ensure that admin actions do not compromise user funds or market fairness.
4. **Gas Optimization and Computational Efficiency** The audit will review the contract's logic to ensure efficient execution, minimizing unnecessary gas costs for traders. This includes optimizing settlement transactions, reducing storage operations, and preventing excessive computational overhead.
5. **Handling of Edge Cases and Attack Scenarios** The contract should be resilient against unexpected inputs, including zero-value trades, maximum integer values, and potential timestamp manipulation. The audit will simulate various attack scenarios, including front-running risks, oracle dependencies, and settlement exploits.

The main goal of this audit will be to verify these claims and ensure that the contracts are secure, efficient, and reliable. Upon the client's request, the audit team can provide further feedback on specific areas of the contract.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Unchecked Delegatecall in diamondCut	HIGH	FIXED
6.2.2	Reinitialization Vulnerability	HIGH	FIXED
6.2.3	Fee/Pledge Rate Inconsistency	MEDIUM	FIXED
6.2.4	Gas Limit Issues in Order Matching	MEDIUM	ACKNOWLEDGED
6.2.5	Inconsistent PDA Seed Construction	MED	ACKNOWLEDGED
6.2.6	Unsafe Use of init_if_needed Exposing Vault Accounts to Re-initialization Attacks	MED	FIXED
6.2.7	Silent Error Swallowing in Cost Accounting Logic	MED	FIXED
6.2.8	Order State Management	LOW	FIXED
6.2.9	Unsafe transferFrom in BatchTransfer	LOW	FIXED

6.2.10	Inconsistent Tick Boundaries	LOW	FIXED
6.2.11	Inconsistent Signer Mutability Requirements in Multi-Signature Governance	LOW	FIXED
6.2.12	Logical Inconsistency in Owner Removal Process	LOW	FIXED
6.2.13	Potential Panic in Event Emission Due to Unwrap on Optional Value	LOW	FIXED
6.2.14	Counterintuitive Price Matching Validation Logic	LOW	FIXED
6.2.15	Redundant Fee Calculation in Cashout	INFORMATIONAL	FIXED
6.2.16	Typos and Naming Inconsistencies in Codebase	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack‘s experts found **two High issues** in the code of the smart contract.

6.2.1 Unchecked Delegatecall in diamondCut

Severity: HIGH
Status: FIXED
File(s) affected: DiamondCutFacet.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The diamondCut function in the Diamond Standard implementation allows unrestricted delegatecall to an arbitrary _init address with user-provided _calldata. This occurs during contract upgrades,

where the LibDiamond.diamondCut method executes _init.delegatecall(_calldata) without validating the target contract's code or intent.

Attack Vector:

- A compromised or malicious contract owner can specify a rogue _init contract.
- The delegatecall executes arbitrary code in the context of the diamond contract's storage.

Root Cause:

- Lack of validation on the _init address and _calldata allows arbitrary code execution.

Impact

- Full Contract Control: Attacker can overwrite storage (e.g., change contractOwner, drain funds).
- Permanent Damage: Malicious initialization could irreversibly corrupt the diamond's state.

Proof of Concept (PoC)

```
// Malicious Initializer Contract
contract ExploitInit {
    function pwn() external {
        LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();
        ds.contractOwner = msg.sender; // Take ownership
    }
}
```

Attack Steps:

1. Attacker deploys ExploitInit.
2. Owner calls diamondCut(..., address(ExploitInit), abi.encodeWithSignature("pwn()")).

	3. Attacker becomes the contract owner.

Code	Line 22 – 30 (DiamondCutFacet.sol):
	
	`function diamondCut(`
	`  FacetCut[] calldata _diamondCut,`
	`  address _init,`
	`  bytes calldata _calldata`
	`) external override {`
	`  LibDiamond.enforceIsContractOwner();`
	`  LibDiamond.diamondCut(_diamondCut, _init, _calldata);`
	`}`
	`}`

Result/Recommendation	Restrict Initialization Contracts:
	
	`address immutable trustedInitializer; // Set during deployment`
	
	`function diamondCut(...) external override {`
	`  LibDiamond.enforceIsContractOwner();`
	`  require(_init == address(0) || _init == trustedInitializer, "Untrusted initializer");`
	`  LibDiamond.diamondCut(_diamondCut, _init, _calldata);`
	`}`
	
	Further Mitigations
	1. Time-Locked Upgrades: Implement a delay between proposal and execution of upgrades.

	2. Multi-Sig Ownership: Require multiple signatures for critical operations like diamondCut.

6.2.2 Reinitialization Vulnerability

Severity: HIGH
Status: FIXED
File(s) affected: MarketFacet.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The initialize function in the MarketFacet contract lacks an initializer modifier, allowing the contract owner to reinitialize critical protocol configurations (e.g., feeWallet, teamWallet) multiple times. This violates the initialization best practices for upgradeable contracts. **Impact** ● Fund Theft: A malicious/compromised owner can reset feeWallet to a controlled address, diverting protocol fees. ● Governance Takeover: Critical addresses like teamWallet can be hijacked, disrupting protocol operations. ● Trust Collapse: Users lose confidence in the protocol's security model. **Root Cause** ● Missing Initialization Guard: The initialize function is unprotected and callable multiple times. ● Privilege Escalation: The onlyOwner modifier allows unrestricted reinitialization. **Proof of Concept (PoC)** Attack Scenario: • Owner calls initialize → Sets feeWallet to Address A. • Later, owner calls initialize again → Resets feeWallet to Address B (attacker-controlled). • Result: All future fees flow to Address B.

Code	Line 33 – 34 (MarketFacet.sol):
	
	`function initialize(address feeWallet, address teamWallet) external onlyOwner {`
	`// __AccessControl_init();`
	`// __ReentrancyGuard_init();`
	`_grantRole(DEFAULT_ADMIN_ROLE, msg.sender);`

Result/Recommendation	Add OpenZeppelin's initializer modifier to restrict the function to a single call:
	
	`import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";`
	
	`contract MarketFacet is Initializable {`
	`function initialize(...) external initializer onlyOwner {`
	`// ...`
	`}`
	`}`
	
	Further Mitigations
	● Use Upgradeable Contracts Pattern: Inherit from Initializable and follow UUPS upgradeable standards.
	
	● Role-Based Access Control: Restrict critical functions to a secure multi-sig wallet.

MEDIUM ISSUES

During the audit, softstack‘s experts found **one Medium issue** in the code of the smart contract.

6.2.3 Fee/Pledge Rate Inconsistency

Severity: MEDIUM
Status: FIXED
File(s) affected: MarketFacet.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The updateConfig function updates the global feeSettle parameter without validating it against existing markets’ pledgeRate. This allows feeSettle to exceed individual market pledge rates, risking underflow in critical calculations (e.g., surplus distribution). **Impact** ● Transaction Reverts: Surplus calculations (e.g., surplusPool in _cancelSellerSettledTrade) underflow, failing trades. ● Fund Mismanagement: Incorrect surplus distribution due to negative values wrapped via underflow (unlikely but possible in older Solidity versions). ● Protocol Instability: Markets become unusable if feeSettle > pledgeRate. **Root Cause** ● Missing Validation: updateConfig does not ensure feeSettle ≤ pledgeRate for all active markets. ● Economic Assumption Violation: The protocol assumes feeSettle ≤ pledgeRate for all markets, but this is not enforced. **Proof of Concept (PoC)** **Scenario**: 1. Market A has pledgeRate = 2% (20000 in WEI6 terms). 2. Admin updates feeSettle to 3% (30000).

	3. Surplus calculation in _cancelSellerSettledTrade: uint256 surplusPool = ... - (feeSettle - pledgeRate); // 3% - 2% = -1% → Underflow **Result**: Transaction reverts, breaking the market.

Code	Line 190 - 220 (MarketFacet.sol):
	
	`function updateConfig(`
	`address feeWallet,`
	`address teamWallet,`
	`uint24 feeSettle,`
	`uint24 feeRefund`
	`) external onlyRole(ADMIN_ROLE) {`
	`DiamondStorage.OTCState storage ds = DiamondStorage.diamondStorage();`
	
	`// Check for invalid addresses`
	`require(feeWallet != address(0) && teamWallet != address(0), "Invalid Address");`
	
	`// Check that the fees are between 1% and 10%`
	`require(feeSettle <= WEI6 / 10, "Settle Fee <= 10%");`
	`require(feeRefund <= WEI6 / 10, "Cancel Fee <= 10%");`
	
	`// Update the config`
	`ds.config.feeWallet = feeWallet;`

	ds.config.teamWallet = teamWallet;
	ds.config.feeSettle = feeSettle;
	ds.config.feeRefund = feeRefund;
	
	// Emit an event
	emit UpdateConfig(
	ds.config.feeWallet,
	ds.config.feeSettle,
	ds.config.feeRefund,
	feeWallet,
	feeSettle,
	feeRefund
	);
	}

Result/Recommendation	Add validation in updateConfig to ensure feeSettle does not exceed any market's pledgeRate:
	
	function updateConfig(...) external onlyRole(ADMIN_ROLE) {
	// ... existing checks ...
	for (uint256 i = 0; i < allMarketIds.length; i++) {
	require(
	feeSettle <= ds.markets[allMarketIds[i]].pledgeRate,
	"feeSettle exceeds market pledgeRate"
	);
	}

	// ... update feeSettle ... } **Further Mitigations** 1. Market-Specific Fees: Allow per-market feeSettle to avoid global constraints. 2. Circuit Breakers: Halt markets if feeSettle > pledgeRate post-update.

6.2.4 Gas Limit Issues in Order Matching

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: OTCmatchEngine.sol
Update: the stop condition is the buyOrderId = 0 or sellOrderId = 0, that's not based on availableAmountBuy and availableAmountSell. if 2 orders are matched and fulfilled its amount, those ordersId will be removed from the orders books, thereby if there is not order in the order books, it will stop the loop noted: if a new order is huge, it will match with large number of opposite order, it may cause the error out-of-gas.

Attack / Description	The _matchOrders function uses a while (availableAmountBuy > 0 && availableAmountSell > 0) loop that relies on cached initial values of availableAmountBuy and availableAmountSell. These values are not refreshed after each iteration, even though the _fillOrder function modifies the actual on-chain available amounts. **Impact** 1. Gas Overconsumption: The loop may process orders that no longer exist, wasting gas. 2. Out-of-Gas Failures: Large order books could exceed block gas limits due to redundant iterations. 3. Market Halts: Failed transactions disrupt order matching, causing system instability. **Root Cause**

	• Stale State Reliance: availableAmountBuy and availableAmountSell are cached at loop start. • Unchecked Storage Changes: _fillOrder updates storage (e.g., reduces available amounts), but loop variables are not refreshed. **Proof of Concept (PoC)** **Scenario:** • Initial availableAmountBuy = 100, availableAmountSell = 100. • First iteration fills 100 units, setting actual storage values to 0. • Loop continues for 99 more unnecessary iterations (using stale availableAmountBuy = 100). **Result**: • Gas wasted on 99 invalid iterations. • Transaction fails if total gas exceeds block limit.

Code	Line 38 - 72 (OTCmatchEngine.sol):
	
	`function _matchOrders(`
	`bytes32 marketId,`
	`int24 highestTickBuy,`
	`int24 lowestTickSell,`
	`bool isOrderCashoutBuy,`
	`bool isOrderCashoutSell`
	`) internal {`
	`DiamondStorage.OTCState storage ds = DiamondStorage.diamondStorage();`

```
uint availableAmountBuy = 0;
uint availableAmountSell = 0;

if (isOrderCashoutBuy) {
    availableAmountBuy = ds.cashoutTicks[marketId][highestTickBuy][ORDER_BUY].availableAmount;
    if (availableAmountBuy == 0) {
        ds.cashoutTickBitmap[marketId][ORDER_BUY].clear(highestTickBuy);
    }
} else {
    availableAmountBuy = ds.orderTicks[marketId][highestTickBuy][ORDER_BUY].availableAmount;
    if (availableAmountBuy == 0) {
        ds.tickBitmap[marketId][ORDER_BUY].clear(highestTickBuy);
    }
}

if (isOrderCashoutSell) {
    availableAmountSell = ds.cashoutTicks[marketId][lowestTickSell][ORDER_SELL].availableAmount;
    if (availableAmountSell == 0) {
        ds.cashoutTickBitmap[marketId][ORDER_SELL].clear(lowestTickSell);
    }
} else {
    availableAmountSell = ds.orderTicks[marketId][lowestTickSell][ORDER_SELL].availableAmount;
    if (availableAmountSell == 0) {
        ds.tickBitmap[marketId][ORDER_SELL].clear(lowestTickSell);
```

	} }

Result/Recommendation	Refresh availableAmountBuy and availableAmountSell from storage within the loop:
	
	while (true) {
	// Refresh values from storage each iteration
	uint currentBuy = isCashoutBuy ?
	ds.cashoutTicks[...].availableAmount :
	ds.orderTicks[...].availableAmount;
	uint currentSell = isCashoutSell ?
	ds.cashoutTicks[...].availableAmount :
	ds.orderTicks[...].availableAmount;
	
	if (currentBuy == 0 || currentSell == 0) break;
	
	// ... process orders ...
	}
	
	Further Mitigations
	1. Batch Processing: Limit iterations per transaction (e.g., process 100 orders per call).
	2. Gas Monitoring: Track gas usage mid-loop and exit before hitting limits.

6.2.5 Inconsistent PDA Seed Construction

Severity: MEDIUM
Status: ACKNOWLEGED
File(s) affected: e.g., init_owner.rs, remove_owner.rs
Update: This design was intentional to reflect the different scopes of authority in our system:

- Owners have program-wide authority and are not tied to a specific config account
- Admins and operators have permissions only within the scope of a specific config account

Attack / Description	The codebase exhibits inconsistency in how Program Derived Addresses (PDAs) are constructed for role-related accounts. PDAs are fundamental to Solana's programming model and provide deterministic addresses derived from seeds and the program ID. In Anchor, consistent seed patterns are essential for proper account validation and access control. Two different patterns have been identified: 1. Some role account PDAs are derived using minimal seeds: seeds = [ROLE_PDA_SEED, owner_1.key().as_ref()] 2. Other role account PDAs include the config account key as an additional seed: seeds = [ROLE_PDA_SEED, config_account.key().as_ref(), admin.key().as_ref()] This inconsistency creates several problems: ● It violates the principle of deterministic derivation, where the same entity type should use the same seed pattern ● It complicates client-side PDA derivation, as different patterns must be used for different operations ● It may lead to validation failures when interacting with multiple parts of the program ● It makes the code harder to maintain and understand, especially for new developers

Code	In owner management modules (e.g., init_owner.rs, remove_owner.rs):
	
	#[account(
	  init,
	  payer = authority,
	  seeds = [ROLE_PDA_SEED, owner_1.key().as_ref()],
	  bump,
	  space = 8 + RoleAccount::INIT_SPACE
	)]
	
	In admin-related modules:
	
	#[account(
	  seeds = [ROLE_PDA_SEED, config_account.key().as_ref(), admin.key().as_ref()],
	  bump = role_account.bump,
	  constraint = role_account.role == Role::Admin && role_account.banned == false
	)]

Result/Recommendation	Standardize the PDA seed construction pattern across all role-related accounts. Choose one consistent approach: **Option A** (recommended): Include the config account in all role PDAs to ensure proper namespace isolation: seeds = [ROLE_PDA_SEED, config_account.key().as_ref(), user.key().as_ref()] **Option B:** Use minimal seeds consistently, only if namespace isolation is not required: seeds = [ROLE_PDA_SEED, user.key().as_ref()] 1. Update all account constraint definitions to use the standardized pattern.

	2. Update any client-side code that derives these PDAs to match the standardized pattern.
	
	3. Store the bump value in each account during initialization to avoid expensive re-derivation, as recommended in Solana's documentation:
	
	// During initialization
	account.bump = ctx.bumps.account_name;
	
	// During validation
	#[account(
	seeds = [...],
	bump = account.bump
	)]
	
	4. Add comprehensive documentation explaining the PDA derivation pattern and how it's used throughout the program.

6.2.6 Unsafe Use of init_if_needed Exposing Vault Accounts to Re-initialization Attacks

Severity: MEDIUM
Status: FIXED
File(s) affected: new_market.rs, settle_market.rs
Update: https://github.com/unich-labs/solana-program/commit/603fb241e376551e39b93317aade2732c9672f8f

Attack / Description	The current implementation uses Anchor's init_if_needed constraint for token vault accounts without implementing proper safeguards against re-initialization attacks. This constraint initializes an account only if it hasn't been initialized yet, but allows the instruction to proceed if the account already exists. According to Solana's security best practices, when using init_if_needed, additional checks must be included to ensure the initialized account cannot be reset to its initial settings after the first

	initialization. The current implementation lacks these essential validation checks, creating a vulnerability where an attacker could potentially reset critical token vault accounts. This vulnerability is especially concerning for vault accounts that may hold significant token amounts. An attacker could manipulate the program to re-initialize a vault account, potentially leading to fund loss or unauthorized access to tokens.
Code	The issue appears in both NewMarket and SettleMarket struct definitions: #[account(init_if_needed, payer = operator, seeds = [VAULT_EX_TOKEN_PDA_SEED, config_account.key().as_ref(), ex_token.key().as_ref()], bump, token::mint = ex_token, token::authority = config_account, token::token_program = token_program)] pub vault_ex_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

Result/Recommendation	Implement one of the following solutions to protect against re-initialization attacks:
	
	1. Add explicit verification in the instruction handler:
	
	pub fn new_market_handler(ctx: Context<NewMarket>, ...) -> Result<()> {
	// Check if vault account already exists and verify its state
	if !ctx.accounts.vault_ex_token_account.data_is_empty() {
	// Verify expected state (authority, mint, etc.)
	require!(
	ctx.accounts.vault_ex_token_account.mint == ctx.accounts.ex_token.key() &&
	ctx.accounts.vault_ex_token_account.owner == ctx.accounts.config_account.key(),

```
            CustomError::InvalidVaultState
        );
    }

    // Rest of the handler logic
    // ...
}
```

2. Use separate instruction paths for new and existing accounts (recommended):

- Create separate instruction handlers for initializing new markets vs. working with existing markets
- Use the standard init constraint for new vaults instead of init_if_needed
- Explicitly verify account state in the existing market handler

Consider account owner checks:

```
// Add this check in your handler
if ctx.accounts.vault_ex_token_account.owner != ctx.program_id {
    return Err(ProgramError::IncorrectProgramId.into());
}
```

6.2.7 Silent Error Swallowing in Cost Accounting Logic

Severity: MEDIUM
Status: FIXED
File(s) affected: cashout_trade.rs, cashout_trade_native.rs, create_order.rs, create_order_native.rs, fill_cashout_order.rs, fill_cashout_order_native.rs, fill_order.rs, fill_order_native.rs, match_bid_cashout_order.rs, match_bid_cashout_order_native.rs, match_cashout_order.rs, match_cashout_order_native.rs, match_order.rs, match_order_native.rs
Update: https://github.com/unich-labs/solana-program/commit/dc5755310a87aed6f1a045320669ddabac89f602

Attack / Description	The codebase contains multiple instances where the sponsor_creation_cost function result is explicitly discarded using the let _ = ... pattern. This pattern deliberately suppresses any error information that might be returned from the function, preventing both the application and developers from being aware when errors occur during cost accounting operations. When using let _ = ... with a function that returns Result<T, E>, the code is explicitly telling the compiler to discard both successful results and errors. This practice violates fundamental principles of error handling in Rust, where errors should be either properly handled or propagated to the caller. This pattern is particularly concerning for cost accounting functionality, which likely manages financial or computational resources. If this function fails silently, the application might continue execution with incorrect assumptions about resource allocation, potentially leading to: 1. Inconsistent financial or computational accounting 2. Resource leaks 3. Incorrect billing or resource attribution 4. Cascading failures in downstream operations that depend on proper cost accounting
Code	This anti-pattern appears in multiple handler functions throughout the codebase, including: let _ = sponsor_creation_cost(config_account.to_account_info(), ctx.accounts.user.to_account_info(), TradeAccount::INIT_SPACE,);
Result/Recommendation	Several alternatives exist to properly handle the result of sponsor_creation_cost: 1. Propagate errors using the ? operator (preferred approach):

```
sponsor_creation_cost(
    config_account.to_account_info(),
    ctx.accounts.user.to_account_info(),
    TradeAccount::INIT_SPACE,
)?;
```

2. Explicitly handle both success and failure cases:

```
match sponsor_creation_cost(
    config_account.to_account_info(),
    ctx.accounts.user.to_account_info(),
    TradeAccount::INIT_SPACE,
) {
    Ok(_) => { /* Success path */ },
    Err(e) => {
        // Log the error or take appropriate action
        log::error!("Cost accounting failed: {}", e);
        // Decide whether to continue or propagate the error
    }
}
```

3. If errors are truly ignorable (which is rarely the case), use explicit notation:

```
// Method 1: Use .ok() to convert to Option and discard
sponsor_creation_cost(
    config_account.to_account_info(),
    ctx.accounts.user.to_account_info(),
    TradeAccount::INIT_SPACE,
).ok();

// Method 2: Type annotation makes intent clearer
let _: Result<_, _> = sponsor_creation_cost(
    config_account.to_account_info(),
```

	ctx.accounts.user.to_account_info(), TradeAccount::INIT_SPACE,);

LOW ISSUES

During the audit, softstack's experts found **one Low issue** in the code of the smart contract

6.2.8 Order State Management

Severity: LOW
Status: FIXED
File(s) affected: OTCmatchEngine.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The order state management system lacks explicit validation checks for extreme edge cases during partial fills, potentially leading to incorrect order states. While SafeCast prevents integer overflows, scenarios involving near-limit fills or improper input validation could leave orders in inconsistent states. **Impact** 1. Inconsistent Filled Amounts: If filledAmount exceeds the remaining order quantity, order.filledAmount may be incorrectly updated. 2. Premature Order Closure: Orders could close before reaching the actual filled threshold due to rounding/precision errors. 3. Storage Corruption: Extreme edge cases might corrupt tick availability tracking (ds.orderTicks[...].availableAmount). **Root Cause Analysis** 1. Missing Input Validation: No check that filledAmount ≤ remainingAmount before updating

	state. 2. Concurrency Assumption: Relies on EVM's sequential execution but lacks reentrancy guards for internal state transitions. 3. Precision Edge Cases: The market.minPrecision check may not account for cumulative floating-point-like errors in high-frequency trading. **Proof of Concept (PoC)** **Scenario**: ● Order Total: 100 units ● filledAmount Input: 95 units (remaining: 5) ● Subsequent filledAmount Input: 6 units **Current Behavior:** ● Fails (100 - 0 - 95) ≤ minPrecision → Marks as fully filled (100/100). ● Next call attempts to fill 6/100, bypassing validation. **Risk**: Protocol may credit excess fills due to improper state transition.

Code	Line 382 - 422 (OTCmatchEngine.sol):
	
	function _fillOrder(uint256 orderId, uint256 filledAmount) internal {
	// ...
	if ((order.amount - order.filledAmount - filledAmount) <= market.minPrecision) {
	_filledAmount = order.amount - order.filledAmount;
	}

	// No check for filledAmount > remaining amount
	}

Result/Recommendation	1. Explicit Input Validation:
	
	function _fillOrder(...) internal {
	uint256 remaining = order.amount - order.filledAmount;
	require(filledAmount <= remaining, "Overfill attempt");
	// ...
	}
	
	2. Reentrancy Guard:
	modifier nonReentrantFill() {
	require(!locked, "Reentrant call");
	locked = true;
	_;
	locked = false;
	}
	
	Further Mitigations
	1. Precision-Aware Arithmetic: Use fixed-point libraries for fractional fills.
	2. Event-Driven State Tracking: Emit granular events (OrderPartiallyFilled, OrderOverfilledAttempt) for off-chain monitoring.

6.2.9 Unsafe transferFrom in BatchTransfer

Severity: LOW
Status: FIXED
File(s) affected: BatchTransfer.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The batchTransfer function uses msg.sender as the from address in transferFrom, requiring users to pre-approve the contract. Tokens like USDT require resetting allowance to zero before approval, which the code doesn't handle. **Impact** ● Transfers Fail: Non-compliant ERC20 tokens (e.g., USDT) will revert due to improper allowance management. ● Functionality Halt: Contract operations dependant on token transfers will break.
Code	IERC20(tokenAddress).transferFrom(msg.sender, recipients[i], amounts[i]) directly uses msg.sender without allowance checks. USDT requires allowance resetting (e.g., https://consensys.net/diligence/blog/2019/09/stop-using-solidity s-transfer-now/).
Result/Recommendation	We recommend the following: 1. Use safeIncreaseAllowance: Replace transferFrom with OpenZeppelin's safeIncreaseAllowance for compliant allowance handling. 2. Documentation: Explicitly state that users must pre-approve the contract for ERC20

	transfers.

6.2.10 Inconsistent Tick Boundaries

Severity: LOW
Status: FIXED
File(s) affected: OTCMatchEngine.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The contract defines ZERO_TICK and MAX_TICK using the absolute minimum/maximum values of int24 (-8,388,608 and 8,388,607), which conflict with the standardized Uniswap V3 tick range of -887,272 to 887,272. This mismatch causes invalid price calculations and order matching failures for ticks outside the expected range. **Impact** 1. Price Inaccuracy: Ticks beyond ±887,272 produce astronomically high/low prices (0 or ∞), breaking price calculations. 2. Protocol Incompatibility: Orders with out-of-bounds ticks fail to interact with Uniswap V3 pools or third-party tools. 3. Liquidity Mismanagement: Liquidity positions may be created with invalid/unusable price ranges. **Root Cause** ● Uniswap V3 Standard: Uniswap restricts ticks to ±887,272 to prevent overflow in sqrtPriceX96 calculations (https://ethereum.stackexchange.com/a/144796). ● Arbitrary Boundaries: Using int24 min/max ignores protocol-specific constraints, violating interoperability. **Proof of Concept (PoC)** **Scenario**:

	• User creates an order with tick = 1,000,000 (valid per current code but invalid per Uniswap).
	• **Result**:
	  ○ Price calculation overflows or returns 0/∞.
	  ○ Order fails to match with Uniswap pools, causing liquidity silos.

Code	Line 187 – 188 (OTCMatchEngine.sol):
	
	`int24 ZERO_TICK = int24(type(int24).min);`
	`int24 MAX_TICK = int24(type(int24).max);`

Result/Recommendation	Adopt Uniswap V3-compatible tick boundaries:
	
	`int24 constant ZERO_TICK = -887272;`
	`int24 constant MAX_TICK = 887272;`
	
	Validation
	Add tick range checks in order creation:
	
	`function _createOrder(...) internal {`
	`  require(tick >= ZERO_TICK && tick <= MAX_TICK, "Invalid tick");`
	`  // ...`
	`}`

6.2.11 Inconsistent Signer Mutability Requirements in Multi-Signature Governance

Severity: LOW

Status: FIXED
File(s) affected: pause_system.rs, remove_owner.rs, set_owner_role.rs, unpause_system.rs
Update: https://github.com/unich-labs/solana-program/commit/ac19eabf455fc722911f0baed5d95f07c4ae9692

Attack / Description	In multiple instruction handlers requiring multi-signature operations, there are inconsistencies in how signers are marked as mutable. According to Anchor documentation and framework requirements, accounts that undergo state changes (including balance changes) must be marked as mutable with the mut attribute. In the current implementation, only the first owner (owner_1) is consistently marked as mutable, while the other owners participating in the same governance actions are not, despite all owners supposedly having equal authority in the governance system. This inconsistency creates confusion about whether all owners are intended to be treated equally and may lead to unexpected behavior if any of the non-mutable signers need to pay for transaction fees or undergo state changes. According to Anchor documentation, "Checks the given account is mutable. Makes anchor persist any state changes". Furthermore, "You only need to mark accounts that have changes as mut. Changes include balance changes as a result of paying the rent for an account or token balance changes".
Code	The issue appears in multiple instruction handlers, including: // In PauseSystem struct #[account(mut)] pub owner_1: Signer<'info>, #[account()] pub owner_2: Signer<'info>, #[account()] pub owner_3: Signer<'info>, Similar patterns appear in UnPauseSystem, RemoveOwner, and SetOwnerRole structs.
Result/Recommendation	Determine whether all owners should be treated equally in the governance system: ● If all owners should be treated equally, make mutability consistent for all signers:

	#[account(mut)] pub owner_1: Signer<'info>, #[account(mut)] pub owner_2: Signer<'info>, #[account(mut)] pub owner_3: Signer<'info>, ● If only one owner is intended to pay for transaction fees or undergo state changes, this should be clearly documented in comments to explain the design decision. 1. Confirm the execution flow to ensure that any account that might undergo state changes (including balance changes for transaction fees) is properly marked as mutable. 2. Update all similar multi-signature instruction handlers to maintain consistency throughout the codebase. 3. Add explanatory comments to clarify the role of each signer in multi-signature operations, particularly explaining why some signers require mutability while others don't (if this is an intentional design decision).

6.2.12 Logical Inconsistency in Owner Removal Process

Severity: LOW
Status: FIXED
File(s) affected: remove_role.rs, remove_owner.rs
Update: https://github.com/unich-labs/solana-program/commit/3aac6342a03d9964ceb0cec50961156a72a8835d

Attack / Description	In the remove_owner_handler function, there is a logical inconsistency between the account constraint definition and the handler implementation. The owner role account is marked to be

	closed (with funds transferred to owner_1) in the account constraints, but the handler also sets the banned flag to true on this account before closure. This creates two contradictory operations: 1. Setting banned = true modifies the account state, suggesting the account should continue to exist but in a banned state 2. The close = owner_1 constraint instructs Anchor to close the account entirely, zeroing out its data and reclaiming rent When an account is closed in Solana using Anchor's close constraint, the account's data is zeroed out and its lamports are transferred to the specified recipient. Therefore, setting any state data (like the banned flag) immediately before closure is redundant and misleading, as this data will be discarded during the close operation. This inconsistency suggests confusion about the intended behavior for removing owners - whether they should be banned (disabled but retained in the system) or completely removed (account closed). It also introduces unnecessary computation costs by modifying data that will be immediately discarded.
Code	In remove_owner.rs: // In account constraints: #[account(mut, close = owner_1, seeds = [ROLE_PDA_SEED, owner.key().as_ref()], bump = owner_role_account.bump, constraint = owner_role_account.banned == false @ CustomError::OwnerBanned)] pub owner_role_account: Box<Account<'info, RoleAccount>>,

	// In handler implementation:
	pub fn remove_owner_handler(ctx: Context<RemoveOwner>) -> Result<()> {
	  let role_account = &mut ctx.accounts.owner_role_account;
	  role_account.banned = true; // Redundant operation before account closure
	
	  emit!(RemoveRoleOwnerEvent {
	  owner_1: ctx.accounts.owner_1.key(),
	  owner_2: ctx.accounts.owner_2.key(),
	  owner_3: ctx.accounts.owner_3.key(),
	  user: ctx.accounts.owner.key()
	  });
	
	  Ok(())
	}

Result/Recommendation	Choose one of the following two approaches based on the intended behavior:
	
	Option 1: Complete removal (recommended)
	If the intention is to completely remove the owner from the system, eliminate the redundant state change:
	
	pub fn remove_owner_handler(ctx: Context<RemoveOwner>) -> Result<()> {
	  // No need to set banned flag as account will be closed
	
	  emit!(RemoveRoleOwnerEvent {
	  owner_1: ctx.accounts.owner_1.key(),
	  owner_2: ctx.accounts.owner_2.key(),
	  owner_3: ctx.accounts.owner_3.key(),
	  user: ctx.accounts.owner.key()
	  });

	Ok(())
	}
	
	Option 2: Logical ban without account closure
	If the intention is to ban but retain the owner's history in the system, remove the close constraint:
	
	rust
	#[account(
	mut,
	seeds = [ROLE_PDA_SEED, owner.key().as_ref()],
	bump = owner_role_account.bump,
	constraint = owner_role_account.banned == false @ CustomError::OwnerBanned
	)]
	pub owner_role_account: Box<Account<'info, RoleAccount>>,
	
	// Then in handler
	pub fn remove_owner_handler(ctx: Context<RemoveOwner>) -> Result<()> {
	let role_account = &mut ctx.accounts.owner_role_account;
	role_account.banned = true;
	
	// Rest of the code
	
	Ok(())
	}

6.2.13 Potential Panic in Event Emission Due to Unwrap on Optional Value
Severity: LOW
Status: FIXED
File(s) affected: update_market.rs
Update: https://github.com/unich-labs/solana-program/commit/ce8a934f4f536d8588342534a23400937f8498ed

Attack / Description	The update_market_handler function contains a potentially unsafe use of unwrap() when emitting the UpdateMarketEvent. The code attempts to unwrap an Option<u64> value without first verifying that the value exists. In Rust, calling unwrap() on a None value causes a runtime panic, which in the context of a Solana program, would cause the entire transaction to fail. When a transaction fails due to a panic, users experience an unexpected error, the transaction is rejected, and they lose the transaction fees they paid. This creates a poor user experience and could lead to user frustration and distrust in the platform. The specific issue occurs when constructing the event payload for UpdateMarketEvent. The code unconditionally calls unwrap() on market_account.min_precision, which is defined as an Option<u64>. If this field is None at the time of event emission, the program will panic.

Code	The issue is found in the update_market_handler function, specifically in the event emission:
	
	emit!(UpdateMarketEvent {
	config_account: ctx.accounts.config_account.key(),
	market_id,
	status: market_account.status,
	settle_time: market_account.settle_time,
	settle_duration: market_account.settle_duration,
	settle_rate: market_account.settle_rate,
	min_trade: market_account.min_trade,
	min_precision: market_account.min_precision.unwrap() // Could panic if None
	});

Result/Recommendation	Replace the unsafe unwrap() call with a safer alternative that handles the None case gracefully. Several options are available: 1. Use unwrap_or() to provide a default value: min_precision: market_account.min_precision.unwrap_or(DEFAULT_PRECISION)

	2. Use unwrap_or_default() to use the type's default value (0 for u64):
	
	min_precision: market_account.min_precision.unwrap_or_default()
	
	3. Add a validation check before event emission:
	
	// At the beginning of the handler, ensure min_precision is always set
	if market_account.min_precision.is_none() {
	market_account.min_precision = Some(DEFAULT_PRECISION);
	}
	
	// Then in event emission
	min_precision: market_account.min_precision.unwrap()
	
	4. Modify the event structure to accept an Option (best solution if event consumers can handle it):
	
	// Change the event definition to accept Option<u64>
	min_precision: market_account.min_precision

6.2.14 Counterintuitive Price Matching Validation Logic

Severity: LOW
Status: FIXED
File(s) affected: match_bid_cashout_order.rs, match_bid_cashout_order_native.rs, match_bid_order.rs, match_bid_order_native.rs, match_cashout_order.rs, match_cashout_order_native.rs, match_order.rs, match_order_native.rs, order_account.rs, utils.rs
Update: https://github.com/unich-labs/solana-program/commit/874d7c191786ac4287bf1ac758da647bd13c3b09

Attack / Description	The smart contract contains counterintuitive validation logic in multiple order matching functions. Specifically, the code validates that orders do NOT match in price before proceeding with order

	matching operations, which creates a logical contradiction between the function's purpose and its validation criteria. In the match_bid_cashout_order_handler and similar functions, the code requires that check_price_match returns false to continue execution: require!(!order_account.check_price_match(cashout_account), CustomError::InvalidMatchingPrice); This validation is confusing and counterintuitive for the following reasons: 1. The function name check_price_match suggests it should return true when prices match, but the code requires it to return false to proceed with matching. 2. The error message InvalidMatchingPrice implies the price matching is invalid when the function actually wants prices to match. 3. Future developers maintaining this code will likely misinterpret the intention, potentially introducing bugs during modifications. The confusion likely stems from one of these issues: ● The check_price_match function actually checks for price incompatibility but is named to suggest the opposite ● The validation logic is inverted from what was intended ● The function has specific trading logic where "matching" means prices must differ by a certain amount
Code	This issue appears in multiple handler functions including match_bid_cashout_order_handler: // From match_bid_cashout_order_handler

	require!(!order_account.check_price_match(cashout_account), CustomError::InvalidMatchingPrice);

Result/Recommendation	To resolve this issue, one of the following changes should be implemented:
	
	1. Rename the function to accurately reflect its purpose:
	
	// If the function checks for prices that shouldn't match
	require!(
	order_account.check_price_incompatibility(cashout_account),
	CustomError::InvalidMatchingPrice
	);
	
	2. Invert the logic inside the function and keep the current name:
	
	// If we want to maintain current behavior but with more intuitive naming
	require!(
	order_account.check_price_match(cashout_account),
	CustomError::InvalidMatchingPrice
	);
	
	3. Add thorough documentation explaining the actual purpose of this validation and why the negation is necessary:
	
	// Add clear documentation above this check
	// In our order matching system, prices must NOT be equal but must be within an acceptable range
	// The check_price_match function returns true when prices are identical, which we don't want
	require!(
	!order_account.check_price_match(cashout_account),

	CustomError::InvalidMatchingPrice);

INFORMATIONAL ISSUES

During the audit, softstack's experts found **one Informational issue** in the code of the smart contract.

6.2.15 Redundant Fee Calculation in Cashout

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: CashoutOrderFacet.sol
Update: https://github.com/unich-labs/evm-contracts/commit/6ef4bd743e5fdc0656fda08080d6b2cca0121601

Attack / Description	The outputValue calculation in _takeCashoutFeeWithPrice redundantly adjusts fees (e.g., outputValue = (outputValue - fees) + fees), introducing unnecessary complexity.
Code	outputValueWithoutFee = (outputValueWithoutFee - settleFee - cashoutFee) + (cashoutFee + settleFee); This simplifies to outputValueWithoutFee = outputValueWithoutFee, rendering the operation redundant.

Result/Recommendation	Simplify the logic:
	
	outputValueWithoutFee = outputValueWithoutFee - settleFee - cashoutFee;
	// No need to add fees back

6.2.16 Typos and Naming Inconsistencies in Codebase

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: pause_system.rs, unpause_system.rs, set_owner_role.rs, and remove_owner.rs.
Update: https://github.com/unich-labs/solana-program/commit/9285ac529abec9fc997e28052b5d1fc6b44d1719

Attack / Description	Multiple typographical errors have been identified in struct names and error message identifiers across the smart contract code. While these don't impact runtime functionality, they create inconsistency in naming conventions and make code maintenance more challenging. Such inconsistencies can lead to confusion during development, particularly for new team members, and may result in propagating these errors to new code. Additionally, error messages with typos make debugging more difficult and appear unprofessional to developers consuming the API.
Code	In the owner management module: #[account()] #[derive(InitSpace)] pub struct FlagInitOnwer {} // Incorrect spelling In multiple constraints across owner management functions: constraint = owner_1_config.banned == false && owner_1_config.role == Role::Owner @ CustomError::InvalidOnwer This appears in multiple files including pause_system.rs, unpause_system.rs, set_owner_role.rs, and remove_owner.rs.
Result/Recommendation	Rename the struct to correct the spelling: #[account()] #[derive(InitSpace)] pub struct FlagInitOwner {}

	Update the custom error enum to correct the spelling:
	
	#[error_code]
	pub enum CustomError {
	  // Other errors...
	  InvalidOwner,
	  // Other errors...
	}
	
	Update all references to the corrected error in constraint definitions:
	
	constraint = owner_1_config.banned == false && owner_1_config.role == Role::Owner @ CustomError::InvalidOwner
	Implement a naming convention guideline for the project to ensure consistency in future development.
	
	Consider adding a spell-checking step to the CI/CD pipeline to catch similar issues early in the development process.

6.3 Verify Claims

6.3.1 **Order Execution and Settlement Integrity** The audit should verify that the contract enforces proper order creation, matching, and settlement logic. This includes ensuring price and amount constraints are followed, preventing unauthorized cancellations, and confirming that trades cannot be settled multiple times.
Status: tested and verified ✅

6.3.2 **Secure Collateral and Fund Management** The contract must securely manage escrowed funds, ensuring that assets are locked and released only according to the defined contract logic. The audit will check for reentrancy vulnerabilities, fund drain risks, and ensure secure handling of token transfers.
Status: tested and verified ✅

6.3.3 **Robust Access Control and Administrative Privileges** The audit should verify that only designated roles can update critical contract parameters, execute emergency halts, or modify order book data. The security review will ensure that admin actions do not compromise user funds or market fairness.
Status: tested and verified ✅

6.3.4 **Gas Optimization and Computational Efficiency** The audit will review the contract's logic to ensure efficient execution, minimizing unnecessary gas costs for traders. This includes optimizing settlement transactions, reducing storage operations, and preventing excessive computational overhead.
Status: tested and verified ✅

6.3.5 **Handling of Edge Cases and Attack Scenarios** The contract should be resilient against unexpected inputs, including zero-value trades, maximum integer values, and potential timestamp manipulation. The audit will simulate various attack scenarios, including front-running risks, oracle dependencies, and settlement exploits.
Status: tested and verified ✅

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the EVM smart contract and Solana program codebase provided by the Unich team. The main objective of the audit was to verify the security and functionality claims of the EVM smart contract and Solana program. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of one issue, classified as follows:
- No critical issues were found.
- 2 high severity issues were found.
- 5 medium severity issues were found.
- 7 low severity issues were discovered
- 2 informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, proof of concepts and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. We recommend that the Unich team review the suggestions.

Update (12.03.2025): The Unich team has successfully addressed all identified issues from the audit. All high, medium, and low-severity vulnerabilities have been mitigated based on the recommendations provided in the report. A follow-up review confirms that the fixes have been implemented effectively, ensuring the security and functionality of the EVM smart contract and Solana program. The updated codebase reflects the necessary improvements, and no further critical concerns remain.

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over $100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: https://softstack.io