Cross - DEX 2

Security Assessment

CertiK Assessed on Nov 3rd, 2025

CertiK Assessed on Nov 3rd, 2025

Cross - DEX 2

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DEX	EVM Compatible	Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 10/28/2025
	Final report published on 11/03/2025

Vulnerability Summary

10	10	0	0	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

0	Centralization		Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1	Minor	1 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	6 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | CROSS - DEX 2

CODEBASE | CROSS - DEX 2

Repository

https://github.com/to-nexus/dex-contracts/tree/d58873720ecbbfda1530ab056da9ed406883f441

https://github.com/to-nexus/dex-contracts/tree/6cca6acc7a617d8080cd99309f67dc6b67beca27/src/

https://github.com/to-nexus/dex-contracts/tree/6520fc25245bf92bd92d88478b6c5e9bc7ef9bf7/src

Commit

d58873720ecbbfda1530ab056da9ed406883f441

6cca6acc7a617d8080cd99309f67dc6b67beca27

6520fc25245bf92bd92d88478b6c5e9bc7ef9bf7

AUDIT SCOPE | CROSS - DEX 2

to-nexus/dex-contracts
src/CrossDexRouterV2.sol
src/MarketImplV2.sol
src/PairImplV2.sol
src/CrossDexImplV2.sol
src/Verse8MarketOwner.sol

APPROACH & METHODS | CROSS - DEX 2

This audit was conducted for Cross to evaluate the security and correctness of the smart contracts associated with the Cross - DEX 2 project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Manual Review and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

FINDINGS | CROSS - DEX 2

10	0	0	1	2	1	6
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Cross to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 10 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
CD2-02	Quote Reserve Inflation Vulnerability In `submitLimitOrder()`	Logical Issue, Incorrect Calculation	Major	Resolved
CD2-03	Incorrect Storage Gap Calculation In Upgradeable Contracts	Coding Issue, Volatile Code	Medium	Resolved
CD2-04	Inconsistent Fee Validation Between Market And Pair Contracts	Logical Issue, Inconsistency	Medium	Resolved
CD2-06	Incorrect `matchedAt` Timestamp Updates In `_setLatest()` Modifier	Logical Issue	Minor	Resolved
CD2-07	Redundant `address()` Cast On Already Typed `pair` Parameter	Code Optimization	Informational	Resolved
CD2-08	Misleading Parameter Name `amount` In `submitBuyMarket()` Actually Represents QUOTE Volume	Coding Style, Inconsistency	Informational	Resolved
CD2-09	Unpredictable Token Approval Requirements In `submitBuyMarket()` Due To Fee Calculation	Inconsistency	Informational	Resolved
CD2-10	Redundant Underscore In Error Message Parameter Name	Coding Style	Informational	Resolved
CD2-11	Incomplete Comment In Pair Contract Modifier	Coding Style	Informational	Resolved
CD2-12	Incorrect Error Type In Reserve Addition Functions	Inconsistency	Informational	Resolved

CD2-02 | Quote Reserve Inflation Vulnerability In `submitLimitOrder()`

Category	Severity	Location	Status
Logical Issue, Incorrect Calculation	Major	src/PairImplV2.sol (base): 298~299	Resolved

Description

A vulnerability exists in the `PairImplV2.submitLimitOrder()` function that allows an attacker to inflate the global `quoteReserve` beyond the actual token balance, effectively freezing all subsequent BUY orders and hijacking previously reserved QUOTE tokens.

Attack Vector:

1. **Setup**: Ensure existing BUY liquidity (`quoteReserve = R`)
2. **Attack**: Place a BUY limit order that doesn't match immediately
3. **Exploitation**: The function incorrectly calculates `receivedAmount` using the full contract balance instead of the actual deposited amount

Vulnerable Code:

```
 295  uint32 feeBps = _buyerMakerFeeBps();
 296  uint256 receivedAmount = QUOTE.balanceOf(address(this)) - mustRemainQuoteAmount
;
 297  _addQuoteReserve(order.owner, receivedAmount);
 298
// ^ This adds the ENTIRE contract balance to reserves, not just the deposited
amount
```

Impact:

- Global `quoteReserve` becomes inflated (`2R + deposit` vs actual balance `R + deposit`)
- All subsequent BUY orders revert due to insufficient balance checks
- Attacker hijacks previously reserved QUOTE tokens from other users

Scenario

1. Ensure there is existing BUY liquidity so `quoteReserve > 0`
2. From an unprivileged address, place a BUY limit order designed not to match immediately
3. The router transfers exactly `quoteAmount + takerFee` to the pair
4. When `_executeBuyOrder()` runs, it computes `skimQuoteAmount = 0` and no matches occur
5. The function incorrectly calculates `receivedAmount = QUOTE.balanceOf(this) - 0 = R + deposit`
6. This inflates the attacker's reserve by `R + deposit` and global `quoteReserve` becomes `2R + deposit`

7. All subsequent BUY orders revert due to insufficient balance checks

Recommendation

Fix the `receivedAmount` calculation to only include the actual deposited and not used QUOTE volume.

Additionally, implement validation to ensure `quoteReserve` never exceeds the actual token balance.

CD2-03 | Incorrect Storage Gap Calculation In Upgradeable Contracts

Category	Severity	Location	Status
Coding Issue, Volatile Code	Medium	src/PairImplV2.sol (base): 100	Resolved

Description

Multiple upgradeable contracts in the codebase contain incorrect storage gap calculations that could lead to storage collisions during upgrades. The gaps are calculated without considering the actual number of storage slots used by the contracts.

For each variable, a storage slot is assigned sequentially. Structs and arrays (except dynamic arrays and mappings) are laid out contiguously, just like individual variables.

Affected Contracts:

- `CrossDexImplV2` : Uses `uint256[44] __gap` but has 8 storage variables (EnumerableMap takes 3 slots)
- `CrossDexRouterV2` : Uses `uint256[44] __gap` but has 7 storage variables (EnumerableSet takes 2 slots)
- `MarketImplV2` : Uses `uint256[42] __gap` but has 10 storage variables (EnumerableMap takes 3 slots)
- `PairImplV2` : Uses `uint256[31] __gap` but has 25 storage variables (List.U256[2] takes 8 slots)

The gaps appear to be arbitrary numbers rather than calculated based on actual storage layout, which creates significant risk for future upgrades.

Moreover, ImplV2s don't comply with Impl. For example, in MarketImplV2 the field `_feeConfig` overwrites `feeBps`. Upgrading contract is expected to expand the existing storage, not overwrite.

Recommendation

Recalculate storage gaps based on actual storage usage:

1. Count all non-constant storage variables in each contract
2. Calculate remaining slots needed to reach 50 slots (standard practice)
3. Use consistent gap calculation methodology across all contracts
4. Consider using automated tools like `slither` to verify storage layout
5. Always expand the storage instead of fields reordering and overwriting

CD2-04 | Inconsistent Fee Validation Between Market And Pair Contracts

Category	Severity	Location	Status
Logical Issue, Inconsistency	Medium	src/MarketImplV2.sol (base): 174~179	Resolved

Description

There are multiple inconsistencies in fee validation between the Market and Pair contracts that could lead to unpredictable fee behavior.

Issues Identified:

1. **Missing validation in Market initialization:**

```
// MarketImplV2.initialize() - no taker >= maker validation
// Only range checks, no logical fee structure validation
```

2. **Unpredictable fee resolution with `NO_FEE_BPS`:**

```
// PairImplV2._resolveEffectiveFees()
feeInfos.sellerMakerFeeBps = feeConfig.sellerMakerFeeBps == NO_FEE_BPS
    ? defaultFeeBps.sellerMakerFeeBps : feeConfig.sellerMakerFeeBps;
```

If both Market and Pair use `NO_FEE_BPS`, the actual fee becomes unpredictable.

3. **Missing validation in Pair's `_setFeeBps()`:**

```
// PairImplV2._setFeeBps() - only checks local values
// Doesn't validate against Market's default fees when using NO_FEE_BPS
```

This creates a situation where fee structures can be logically invalid after resolution, breaking the taker >= maker invariant.

Recommendation

Consider removing `NO_FEE_BPS` ambiguity by requiring explicit fee values at both Market and Pair levels.

CD2-06 Incorrect `matchedAt` Timestamp Updates In `_setLatest()` Modifier

Category	Severity	Location	Status
Logical Issue	Minor	src/PairImplV2.sol (base): 120~123	Resolved

Description

The `setLatest` modifier incorrectly updates the `matchedAt` timestamp even when no actual trades occurred during the function execution. This creates misleading market data and inaccurate trading activity indicators.

Problem Details:

```
modifier setLatest() {
    _;
    _setLatest(); // Always called regardless of matching activity
}

function _setLatest() private {
    uint256 _latestPrice;
    assembly {
        _latestPrice := tload(_matchedPriceSlot)
    }
    if (_latestPrice != 0 && _latestPrice != matchedPrice) matchedPrice =
_latestPrice;
    if (matchedAt != block.timestamp) matchedAt = block.timestamp; // Always updates
}
```

The `_setLatest()` function is called after `_matchSellOrder()` and `_matchBuyOrder()` executions, but it unconditionally updates `matchedAt` to the current block timestamp even if:

- No orders were matched

This creates false signals about market activity and could mislead external systems monitoring trading patterns.

Recommendation

Modify the `setLatest` modifier to only update timestamps when actual trades occurred.

This ensures `matchedAt` accurately reflects actual trading activity rather than just matching attempts.

CD2-07 | Redundant `address()` Cast On Already Typed `pair` Parameter

Category	Severity	Location	Status
Code Optimization	Informational	src/CrossDexRouterV2.sol (base): 147, 166, 167, 187, 188	Resolved

Description

The code contains redundant casting of the `pair` parameter to `address` when it's already an address type. This occurs in multiple locations throughout the router contract.

The `pair` parameter is already declared as `address pair` in the function signatures, making the `address(pair)` cast unnecessary.

Recommendation

Remove the redundant `address()` casts and use the `pair` parameter directly.

CD2-08 Misleading Parameter Name `amount` In `submitBuyMarket()` Actually Represents QUOTE Volume

Category	Severity	Location	Status
Coding Style, Inconsistency	Informational	src/CrossDexRouterV2.sol (base): 174	Resolved

Description

The `submitBuyMarket()` function uses a misleading parameter name `amount` that actually represents the volume in QUOTE tokens rather than the BASE amount.

This naming inconsistency can lead to developer confusion and potential integration errors.

Recommendation

Rename the parameter to clearly indicate it represents QUOTE volume.

Alternatively, consider using more descriptive parameter names throughout the contract for better clarity.

CD2-09 | Unpredictable Token Approval Requirements In `submitBuyMarket()` Due To Fee Calculation

Category	Severity	Location	Status
Inconsistency	Informational	src/CrossDexRouterV2.sol (base): 188	Resolved

Description

The `submitBuyMarket()` function requires users to approve an unpredictable amount of QUOTE tokens because the actual transfer amount includes dynamically calculated fees. This creates user experience issues as users cannot know the exact approval amount beforehand.

The `_calculateRequireBuyVolume()` function adds the buyer taker fee to the base amount, meaning users must either:

- Over-approve significantly to account for maximum possible fees
- Use infinite approvals (security risk)

Recommendation

Implement one of the following solutions:

1. **Add a view function** to calculate required approval amount:

```
function getRequiredBuyVolume(address pair, uint256 quoteAmount) external view
returns (uint256) {
    return _calculateRequireBuyVolume(pair, quoteAmount);
}
```

2. **Document the maximum fee percentage** and implement the approval calculation on the UI.

CD2-10 | Redundant Underscore In Error Message Parameter Name

Category	Severity	Location	Status
Coding Style	Informational	src/CrossDexRouterV2.sol (base): 238	Resolved

Description

The `setMaxMatchCount()` function in `CrossDexRouterV2` uses a redundant underscore prefix in the error message parameter name, creating inconsistency with other validation checks in the same contract.

Recommendation

Remove the redundant underscore from the error message parameter name to maintain consistency.

CD2-11 | Incomplete Comment In Pair Contract Modifier

Category	Severity	Location	Status
Coding Style	Informational	src/PairImplV2.sol (base): 105	Resolved

Description

The comment in the `onlyOwner` modifier of `PairImplV2` is incomplete and potentially misleading. The "owner" is missing.

Recommendation

Complete the comment to clearly explain the ownership relationship.

CD2-12 | Incorrect Error Type In Reserve Addition Functions

Category	Severity	Location	Status
Inconsistency	Informational	src/PairImplV2.sol (base): 654, 672	Resolved

Description

The reserve addition functions in `PairImplV2` use `PairInvalidAccountReserve` error when they should use `PairInvalidReserve`, as these functions deal with global reserve overflow rather than account-specific issues.

The `PairInvalidAccountReserve` error is designed for account-specific reserve issues, but these functions are checking for global reserve overflow which affects the entire contract.

Recommendation

Use the appropriate `PairInvalidReserve` error for global reserve overflow checks.

This ensures consistent error handling where account-specific errors are used for account operations and contract-level errors for global state issues.

OPTIMIZATIONS | CROSS - DEX 2

ID	Title	Category	Severity	Status
CD2-01	Unnecessary `unchecked` Blocks For Loop Increments In Solidity 0.8.22+	Code Optimization	Optimization	Acknowledged

CD2-01 Unnecessary `unchecked` Blocks For Loop Increments In Solidity 0.8.22+

Category	Severity	Location	Status
Code Optimization	Optimization	src/CrossDexRouterV2.sol (base): 254~256	Acknowledged

Description

Multiple contracts contain unnecessary `unchecked` blocks for loop counter increments, which provide no gas savings in Solidity 0.8.22 and later versions.

Recommendation

Remove the unnecessary `unchecked` blocks and use standard for-loop syntax for better code readability.

This maintains the same gas efficiency while improving code clarity and reducing visual noise. The Solidity compiler is smart enough to optimize these increments without explicit `unchecked` blocks.

APPENDIX | CROSS - DEX 2

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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 considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.