H-X

Report on Security Assessment of Smart Contracts for Tacans Labs

Version 1.0

Audit overview

Veax is a decentralized exchange (DEX) that operates on the NEAR Protocol. It is designed to provide an advanced trading experience that combines the best features of traditional centralized exchanges (TradeFi) and decentralized exchanges (DeFi). As a single-sided liquidity DEX, Veax allows traders to provide liquidity to the exchange using a single token. The platform also incorporates advanced features such as adaptable exchange pools with smart routing, concentrated liquidity, and dynamic fee levels that guarantee the best swap price for traders. Veax is built on NEAR Protocol, which is a blockchain platform designed to provide a scalable and secure infrastructure for decentralized applications. By leveraging the NEAR Protocol, Veax can offer fast transaction speeds and low fees, making it accessible to a wide range of traders.

At the request dated January 25, 2023, a security audit of Veax smart contracts was conducted.

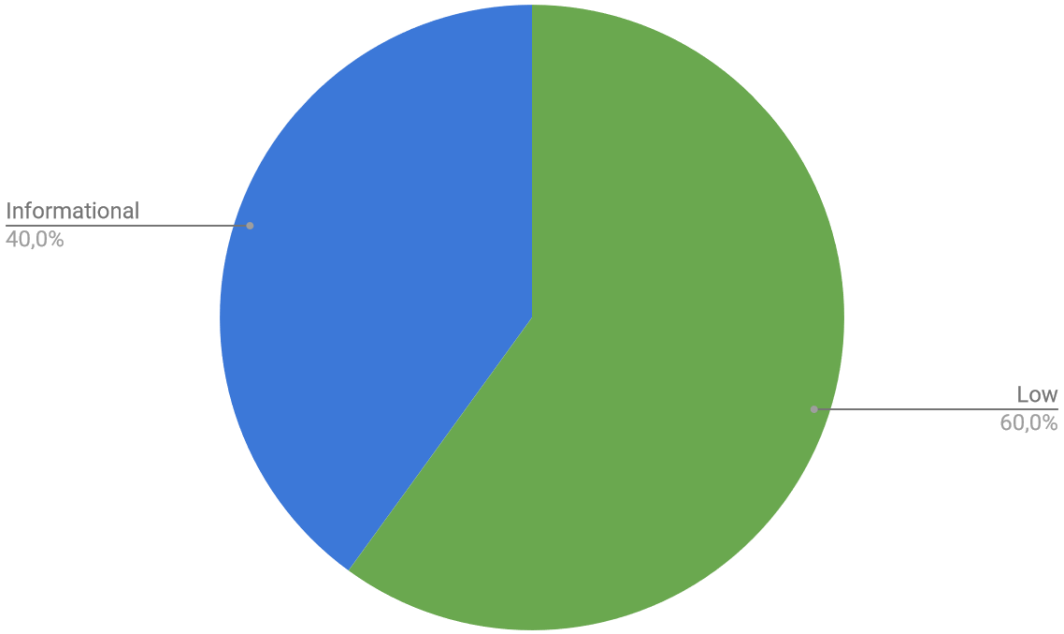
The audit of a smart contract involved a comprehensive review and evaluation of its code and associated processes to identify any vulnerabilities or weaknesses that could potentially compromise the security, functionality, or performance of the contract. The purpose of the audit was to ensure that the smart contract operates as intended, meets the specified requirements, and complies with industry standards and best practices.

As a result of the audit, it was established that an attacker could not abuse the smart contract or violate business requirements. A few low-level issues and some informational issues were identified during our assessment, and we are pleased to report that the developers have been very responsive and have taken appropriate measures to further improve the security level of the protocol. Additionally, we provided recommendations for improving certain mechanics of the project, which the developers have taken into account.

The Veax smart contract was found to be of high quality and meets industry standards and best practices. Its design and implementation demonstrate a strong commitment to security and reliability, and the audit results provide confidence in the contract's ability to operate as intended.

Diagram of the findings

0	0	0	6	4
Critical	High	Medium	Low	Informational



ID	Findings	Risk level	Status
F-1	Payable API state	Low	*Fixed*
F-2	Contract suspension check missing	Low	*Fixed*
F-3	Contract suspension check missing	Low	*Fixed*
F-4	Contract suspension check missing	Low	*Fixed*
F-5	Unnecessary check	Low	*Fixed*
F-6	Unnecessary check	Low	*Fixed*
F-7	Possible occurrence of an unwanted event	Informational	*Noted*
F-8	Missing cargo overflow checks	Informational	*Fixed*
F-9	Elastic supply problem	Informational	*Noted*
F-10	Unnecessary storage of data on-chain	Informational	*Noted*

Table of contents

1. Introduction

By request of Tacans Labs (Customer, Company), and according to Purchase Order dated 25 Jan 2023, H-X Technologies (H-X, Provider or pen testers) has delivered the professional information security services, namely, security assessment of the Customer's smart contracts (target object).

After reviewing the implementation of Veax's smart contracts, this audit report has been prepared to discover potential issues and vulnerabilities in their source code. We have outlined our approach to evaluate the potential security risks. Advice on how to improve security and performance has also been given in the report.

Veax is a decentralized exchange (DEX) that operates on the NEAR Protocol. It is designed to provide an advanced trading experience that combines the best features of traditional centralized exchanges (TradeFi) and decentralized exchanges (DeFi).

As a single-sided liquidity DEX, Veax allows traders to provide liquidity to the exchange using a single token. This means that traders can easily add liquidity to the platform without having to provide both tokens in a trading pair.

Veax is built on NEAR Protocol, which is a blockchain platform designed to provide a scalable and secure infrastructure for decentralized applications. By leveraging the NEAR Protocol, Veax can offer fast transaction speeds and low fees, making it accessible to a wide range of traders.

The platform also incorporates advanced features such as adaptable exchange pools with smart routing, concentrated liquidity, and dynamic fee levels that guarantee the best swap price for traders. This provides traders with greater control over their trades and helps to mitigate risks.

Overall, Veax aims to provide a user-friendly and seamless trading experience that combines the best aspects of both TradeFi and DeFi. By doing so, it hopes to become a leading platform for decentralized trading on the NEAR Protocol.

2. What is a Smart Contract Audit

Smart contracts for Near are self-executing digital programs that run on the Near blockchain. They allow developers to create and deploy decentralized applications (dApps) that can perform a variety of functions, such as managing digital assets, running automated transactions, enforcing rules, and more.

Smart contracts for Near are written in Rust programming language, which is known for its security and performance. The smart contracts run on the Near Virtual Machine (VM), which is a lightweight and efficient execution environment designed for the fast and secure execution of smart contracts.

Writing smart contracts is a relatively new field, without many security standards, documentation, or best practices. It is also the ultimate test of defensive software engineering. Smart contracts can end up controlling tens of millions of dollars, making them a target for attackers.

Audit of Smart Contracts is focused on finding logic flaws and security vulnerabilities, especially, which could let an attacker misuse the Smart Contract, violate the customer's business requirements, or cause any other harm to the customer or its clients or partners. The goal of the audit is to model and verify the target object compromise, sensitive information theft, weak conditions, or other ways or prerequisites for the realization

of fraud or security incidents. To achieve this goal, tools and techniques very similar to those that an attacker would use are typically required.

3. Disclaimer

This audit report is solely intended to assess the security and functionality of the smart contract for the decentralized exchange project on the Near blockchain. It is not intended to provide investment advice or personal recommendations, nor does it take into account any potential economic implications of tokens, token sales, or other assets. Under no circumstances should any entity rely on this report to make investment decisions, buy or sell any tokens, products, services, or other assets.

It should be noted that this audit report does not endorse any particular project or team, nor does it guarantee the security of the project. The evaluation result does not ensure the absence of any further security issues, as a single audit cannot be comprehensive. Therefore, it is highly recommended that the project undergoes additional independent audits and a public bug bounty program to ensure the security of smart contracts.

Furthermore, this audit report is subject to certain limitations, including but not limited to the fact that the evaluation is based on the information provided by the project team, and no independent verification of the information was conducted. Additionally, the audit is limited to the specific smart contract codebase reviewed and does not cover any associated web or mobile applications or third-party integrations. Any changes to the smart contract codebase after the completion of the audit are not covered by this report. Finally, this audit report does not provide any warranties regarding the discovery of all security issues of the smart contract.

4. Audit Summary

According to the assessment, the Customer's smart contracts are well protected. According to the Tarpaulin code coverage reporting tool, the code is 77% covered, but it does not cover the integration tests used in the project. Given this, the actual coverage percentage could be much higher. The following is an overview of the project, containing the scope of the security review. It then provides an overall summary of the audit findings, followed by a detailed review of the vulnerabilities found, with each vulnerability given a severity rating (critical, high, medium, or low) and a possible solution. Conclusions that are not directly related to security are marked as informational. Based on the findings, we recommend resolving all low, medium, and high findings, and further reviewing the information level findings. This will help ensure confidentiality, integrity, and availability.

During the audit process, each vulnerability is also assigned a status:

- **Open:** the issue has not been presented to the project development team;
- **Fixed:** the issue has been fixed;
- **Noted:** the issue has been acknowledged by the project team, but no further action has been taken because a compelling case has been made.

5. Recommendations

Auditors recommend mitigating all the vulnerabilities described in this report. It is also highly recommended to complete all todo!, FIXME, and TODO statements.

The nature of information threats involves the uncertainty of penetration paths that may be used by an attacker. In addition, the set of known technical vulnerabilities in libraries, components, and hosting environments is constantly increasing. Therefore, the results of this audit cannot guarantee to uncover all possible compromise or penetration ways and security problems and only show the weakest points in the security of the target object.

Besides smart contract audits, to enhance the customer's security effectively and to reduce the customer's business risks, other appropriate security management processes and security solutions should be designed and implemented. These security measures include but are not limited to the following: a secure development lifecycle, regular security audits by an independent party, security event monitoring, and incident response.

Using internationally recognized standards and best practices such as ISO 27000, PCI DSS, and NIST is recommended. We can help in the implementation of these processes and solutions.

6. Methodology

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contract-related security issues, using automatic verification tools and manual review. We test some discovered issues on our local network to reproduce the issue and prove our findings.

In this audit, we considered the following important features of the code.

Common issues:

- Behavior flow management—evaluation of possible scenarios of unexpected or undesirable behavior:
  - *Front running*
  - *Reentrancy*
  - *Cross contact calls:*
    - *Exploitable state between the call and the callback*
    - *Rollback any changes to the state in the callback if the external call failed*
    - *Fefund*
  - *Transactions or events order dependencies*
  - *Assert violation*

    - *Unnecessary checks*
    - *Unexpected balance*
- Access control—presence of an access control check for privileged actions:
    - *Public methods or variables that should be private*
    - *Unencrypted private data on-chain*
    - *Make sure it's the call made by a user*
    - *Overview of administrative roles and trust model*
- Improper initialization of the smart contract:
    - *No validation of passed arguments*
- Denial of service:
    - *Storage staking (NEAR uses storage staking which means that a contract account must have sufficient balance to cover all storage added over time.)*
    - *Insufficient balance to withdraw or transfer tokens*
- Unused code:
    - *Code with no effects*
    - *Unused variables*
    - *Unused functions*
- Typographical issues:
    - *Misspelling*
- Requirement violation
- Arithmetic issues
- Cryptographic issues
- Weak randomness
- Shadowing variables
- Centralization related issues
- Secure oracles usage

Additional:

- Issues caused by the token smart contracts and their operators themselves:
    - *Fee on transfer tokens*
    - *Changing the balance of tokens (token supply manipulation)*
      However, these issues can interfere with the proper operation of a DEX.
- Check the correct operation of the tokens stored by the smart contract that belong to users
- Possibility of obtaining more tokens than expected
- Incorrect commission (fee) calculation
- Ability to drain or steal funds from the pool
- Arithmetic
- Carefully checking the implementation to make sure that functions and methods return correct or expected results or report an error or return if incorrect.
    - *Correctness*
    - *Rounding*
    - *Type-safe casts*
    - *Swap math*
    - *Liquidity math*
    - *Fee math*

Cargo issues:

- *Vulnerable dependencies*
- *Dependency versions*
- *Profile settings:*
    - *Overflow-checks*
    - *Optimization*

Automated analysis:

- Scanning the project's codebase with Rustle and others.
- Manual check of all problems found by the tools.

After reviewing the documentation and any available tests, the smart contract undergoes testing and fuzzing to verify its functionality. Fuzzing is a technique used in software testing that involves providing invalid, unexpected, or random inputs to a program to observe its behavior and identify any vulnerabilities or bugs. In the context of smart contracts for the Near blockchain, fuzzing can be used to test the contract's resilience to unexpected or malicious inputs.

Fuzzing involves creating an automated testing framework that generates random or unexpected inputs to a smart contract and monitors its behavior. The goal is to find any edge cases, invalid inputs, or unexpected behavior that could lead to security vulnerabilities or bugs.

By testing smart contracts with fuzzing techniques, developers can identify and fix any potential issues before deploying the contract to the blockchain. This can help prevent attacks and ensure the proper functioning of the contract once it is live.

7. Project Scope

The scope of the project is a smart contract for a decentralized exchange written in the Rust programming language for the Near blockchain. The project allows users to trade, provide liquidity, and earn rewards. Currently supports deposits, withdrawals, and exchanges in NEP-141 tokens.

Here is an overview of the features available to different actors:

Function	User Role	Tested
view metadata	User	True
get_deposits	User	True
get_deposit	User	True
get_verified_tokens	User, Owner	True

get_user_tokens	User	True
get_pool_info	User	True
get_user_storage_state	User	True
get_version	User	True
get_owner	User	True
get_position_info	User	True
storage_deposit	User	True
storage_withdraw	User	True
storage_unregister	User	True
storage_balance_bounds	User	True
storage_balance_of	User	True
extend_verified_tokens	Owner	True
remove_verified_tokens	Owner	True
extend_guard_accounts	Owner	True
remove_guard_accounts	Owner	True
set_protocol_fee_fraction	Owner	True
suspend_payable_api	Owner, Guard, Liquidity Provider	True

resume_payable_api	Owner, Guard, Liquidity Provider	True
open_position	Liquidity Provider	True
withdraw_fee	Liquidity Provider	True
close_position	Liquidity Provider	True
exact-in swap	Trader	True
exact-out swap	Trader	True

Note: The audit report only covers certain modules in the repository and not all of them. Specifically, the audit report only includes the contract located in the `veax/dex/src` folder. To be more specific, the audit report focuses on the following files:

`./`

File	keccak256
lib.rs	ee3ce39365784ede353ac48e47f83e78164b525dd8417f105c3c6e01359f088c

`./chain`

File	keccak256
account.rs	5f2a02689219862a1467eee67191fbd3f846910c092faac045e947eb7aa601cb
mod.rs	8b0f11cb5aa5af76aab15526719be25ee5fbcadeb71f725a7ddcea1728bf01d0
pairs.rs	a5c2bbd3a058791c12e91863b78fabade145d891bf87fd8135e4106a03f70bfb
storage_key.rs	85a149458ac63859b0e926063d46471dea803630ae2b30b5908ed543f29e612f
types.rs	e9c01544b8bfcf126740ed7a57907e7ad1e61d3a41f380d6423a7a65e9e19064
utils.rs	914ced279a992f0572d42b3bd51db6d2add15bc23f9918e95c5cd19b61abd271
wasm.rs	3a851cd5e3dcc68e7c2d1bc1080d8dab97f8d7e27de389e3e47f0f5b66677693

./chain/events

File	keccak256
mod.rs	deb428759867785dad83ff92d75659b30f8df0edaa35fb39c1e67493bbb3f3fc
tests.rs	136efffdd8d1d204a300386bc357787e6bd65a8189d5cf4626720e3c1602c215

./chain/log

File	keccak256
mod.rs	f317d636142f06eeecf3a2996d6a8f6325dae8dc08aa3f6b8bcd851b5f92ba0d
non_wasm.rs	840dc5e59970fa8cb8d9b08db4c919a0c8fe804cb6c4f8aa0752fb953d8d9663
wasm.rs	313671a93ba824e990c4937bec40df09d1d703e27ade986c98e022a996a5e3ad

./dex

File	keccak256
dex_impl.rs	b23acaa46e81eed6f99a2bd24ebae966f3526e3192fb10d40ef573c94bbb74b4
dex_tests.rs	8f2eceec1fd1f480c91035014af990764473b769bd61777809c3a296e660f6c3
errors.rs	0603a220b967b985057d3dcf91640c86c0971b24f3ffb65f4d5b1674c8f159b7
mod.rs	ec81aa87c6eba64134b551f271ea42080d2662da3812e109758e122357882b43
primitives.rs	4020762da09d5c0f79ce017eeccf20903ad169b5c481812fae661b9fc2181f9e
state_types.rs	afa326a777d3a68c8b18a0f48bc6c658669c0226cd562a7af9bf71490284ec82
test_utils.rs	e301afabad6ab1ea7b1a90f5d1887f454af8b148d8fb823e7bbfd851f32a49ac
tick.rs	bc9a784ea11dbb263ea94423809cc35e5712679ae89aab6d871cbbd7e09a4a21
tick_state_ex.rs	0c63df847913f9c77a9f0567f2488a9cfc94c71638c36f5328a4f327eefd9994
traits.rs	da637e72a0796025409e0540a352528a6a9fa108caa3ae498f914ef20e02aa0a
utils.rs	8c5d2db7955a5af5d21f7e3887164045c6f8a3b8fb757a11675c70c707c94314
util_types.rs	57068743c398e404b75e93f197c8efd4895dcaf4c7b3e166aefe3e55b85b11fe

./dex/v0

File	keccak256
account_state_ex.rs	51fb65f96a4f317b938c1679c4a74f90d0a899c9fbb556e8301e59f5b4d9f533
mod.rs	63e00845ca7e2896ce7d211479af9a6711bf05ce95d73244ed4fcaf6550752d7
pool_state_ex.rs	0bd654059fa8b2c8f0daa36bc834a381256a3a4d4c76059adbb959c1ed30c6e5
util_types.rs	050623c66ad6c3f22d8287ad378e339d1de9f41e884c65c2bcba76b262055b7d

./fp

File	keccak256
display.rs	e6b3370485f411ac72bcb73042b2d9ad40ce766c39b98efe5cb655540fcc1676
error.rs	eac11376e0b8feccfb7592a222df1336640a316d6ae40b7b9f6a23eaf5af12ab
i128x128.rs	249bb47c56303ad820733458cb75039497892926916f728b43e020caa9587b6b
i192x192.rs	77edad3a99578a450596833ff34ec88f0812718ba38614bcb4d9bc8718bdc583
i192x64.rs	8937e999e35e90d751289db74c0b69a073fa70f270809d3867e189f1a4f8b7a8
i256x128.rs	2826039e6c01367a27510a689d6fc0e4408c8ffebb9d14d06ae8c080a139bc72
i320x64.rs	82e8c55e5448dcfe3519a63b019247733d5cca44502037c29d259e4661734689
mod.rs	6a4702536a63d9d7f815e3f2a7442e22bb8c98b62945d53391ba120c7ae5e7ee
signed.rs	2f4e21440ce30a6b06fb6c897a412203f1165cd09a2c5016a5033ee1270b4c86
traits.rs	82eebdf7a46ca7a73d1f9829d462bcd6363b295ade07266d07f32cb2d719ece4
try_float_to_ufp.rs	ed24b8fa98d30009f7aefa6ad90ca3322542086edd1e7f6132acd4507789b78a
types.rs	d02bb85571123584c3935822e07d79eafb99c570abdd1e7b9e063264fce2a665
u128x128.rs	528485cefc470eefd4da41569f463dc78f91a394966e7c30a2d07c48a57ee90c
u192x192.rs	64624cb48758cd68fe439dcd4668e9718d3aa126ee4d5820d5c58c9f920bf4a8
u192x64.rs	c58802451c04237036ca91ba6ff94534dab6ce3aee331c299ac22270a4203699
u256x128.rs	ef93387d354e90a3275a0e821864ef103da27297d26f232082310954184c4486
u256x256.rs	2b1dac90e4d76ad3a3e4a7fccb55e4ec3142618c8a79953869048d3cf4e783d8
u320x64.rs	d4ffea2c3cf7190233b245953ed622d63200b74903e57357e440897e55f471e2

ufp_to_float.rs	b959daa6ee3ea30898593a795cb6bbf47f4cf5f670a36ab3f16adf44bde280aa
unsigned_symmetric.rs	a76223240f3d74921cf9364fa296ef0783912ec68644e34ad7fa50a120080844

Used dependencies:

- `uint = { version = "0.9.3", default-features = false }`
- `serde = "1.0.138"`
- `serde_json = "1.0.82"`
- `thiserror = "1.0.31"`
- `near-sdk = { version = "=4.0.0" }`
- `near-contract-standards = { version = "=4.0.0" }`
- `itertools = "0.10.4"`
- `num-traits = "0.2.15"`
- `typed-index-collections = "3.1.0"`
- `bitvec = "1.0.1"`
- `paste = "1.0.9"`
- `strum = "0.24.1"`
- `strum_macros = "0.24.3"`
- `static_assertions = "1.1.0"`

8. The Severity Level of the Issues

Severity	Description
Critical	Issues that could result in an unlimited loss of funds or completely disrupt the contract's workflow. Malicious code (including malicious modification of libraries) is also considered a critical issue.
High	Issues that may lead to limited loss of funds, breach of user experience, or other contracts under certain conditions. In addition, smart contract issues that allow a privileged account to steal or block other users' funds.
Medium	Issues that do not lead to the loss of funds directly, but violate the logic of the contract. May lead to contract failure or denial of service.
Low	Issues that are not optimal coding, such as gas optimization hints, or unused variables.
Informational	Issues that do not affect the operation of the contract. Usually, information severity issues are related to code best practices—for example, a style guide.

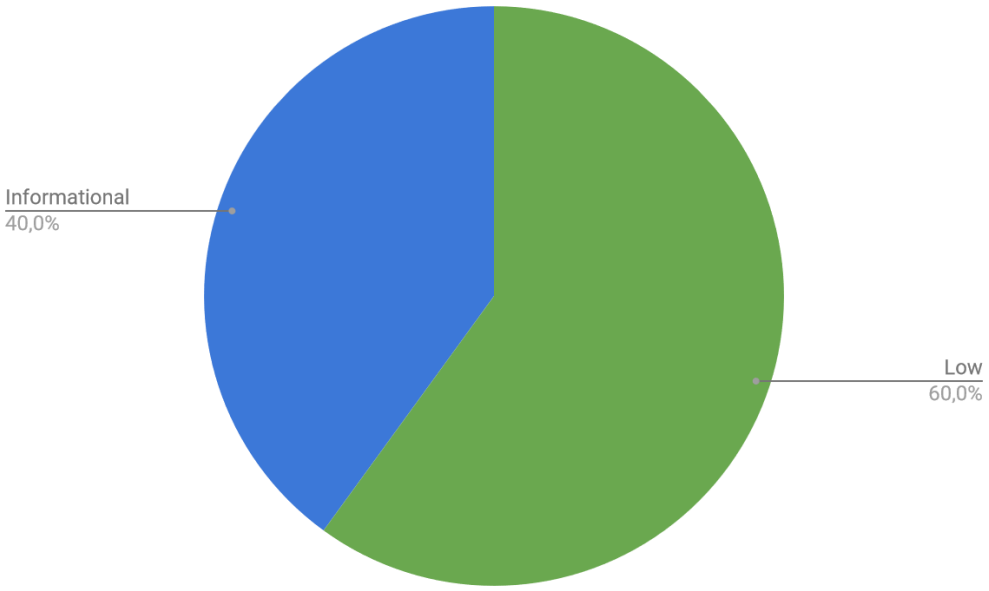
9. Findings and Risk Levels

In the table below you can find a list of issues found during manual code analysis.

ID	Findings	Risk level	Status
F-1	Payable API state	Low	*Fixed*
F-2	Contract suspension check missing	Low	*Fixed*
F-3	Contract suspension check missing	Low	*Fixed*
F-4	Contract suspension check missing	Low	*Fixed*
F-5	Unnecessary check	Low	*Fixed*
F-6	Unnecessary check	Low	*Fixed*
F-7	Possible occurrence of an unwanted event	Informational	*Noted*
F-8	Missing cargo overflow checks	Informational	*Fixed*
F-9	Elastic supply problem	Informational	*Noted*
F-10	Unnecessary storage of data on-chain	Informational	*Noted*

10. Diagram of the Findings

Critical	High	Medium	Low	Informational
0	0	0	6	4



10.1 CVSSV3 Score

Vulnerability ID	Vulnerability Description	CVSS Score	Severity
11.2	F-2 Contract suspension check missing	**4.9**	**Low**
11.3	F-3 Contract suspension check missing	**4.2**	**Low**
11.4	F-4 Contract suspension check missing	**4.1**	**Low**
11.5	F-7 Unnecessary check	**2.2**	**Low**
11.6	F-8 Unnecessary check	**2.2**	**Low**
11.7	F-9 Possible occurrence of an unwanted event	**2.0**	**Informational**
11.8	F-10 Missing cargo overflow checks	**2.0**	**Informational**
11.9	F-11 Elastic supply problem	**0**	**Informational**

11.10	F-12 Unnecessary storage of data on-chain	**0**	**Informational**

11. Results from Manual Analysis

11.1 F-1 Payable API State

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: There is no state check before changes to the `suspend_payable_api` and `resume_payable_api` methods. This lets you repeatedly call the suspend or resume payable API function, which can lead to the disruption of the project because an event is generated during the call.

Risk: Low

Location:

- `./veax/dex/src/dex/dex_impl.rs`: 233-243
- `./veax/dex/src/dex/dex_impl.rs`: 245-254

Code section:

```
impl<T: Types, S: StateMut<T>, SS: BorrowMut<S>> Dex<T, S, SS> {
...
	pub fn suspend_payable_api(&mut self) -> Result<()> {
		self.ensure_caller_is_guard()?;

		let Contract::V0(ref mut contract) = self.contract_mut();
		contract.suspended = true;

		let caller_id = self.get_caller_id();
		self.logger_mut().log_suspend_payable_api_event(&caller_id);

		Ok(())
	}

	pub fn resume_payable_api(&mut self) -> Result<()> { // todo: check it
		self.ensure_caller_is_guard()?;
		let Contract::V0(ref mut contract) = self.contract_mut();
		contract.suspended = false;

		let caller_id = self.get_caller_id();
		self.logger_mut().log_resume_payable_api_event(&caller_id);

		Ok(())
	}
...
}
```

11.1.1 Improvement Recommendation

Implement a state check for the Payable API before assigning a new state to avoid re-raising the event.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Methods `suspend_payable_api` and `resume_payable_api` are idempotent, and repeated execution doesn't cause unwanted smart contract state changes. The only impact is doubling log occurrences that have an informational character.

Business impact: In the context of a smart contract on the NEAR blockchain, the impact of the possible occurrence of an undesirable event can be significant for a business.

11.2 F-2 Contract Suspension Check Missing

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Risk: Low

Location: `./veax/dex/src/chain/wasm.rs`: 397-448

Code section:

```
#[near_bindgen]
impl StorageManagement for State {
...
	#[payable]
	fn storage_deposit(
	&mut self,
	account_id: Option<AccountId>,
	registration_only: Option<bool>,
	) -> StorageBalance {
		let amount = env::attached_deposit();
		let account_id = account_id.unwrap_or_else(env::predecessor_account_id);
		let registration_only = registration_only.unwrap_or(false);
		let min_balance = self.storage_balance_bounds().min.0;

		let mut dex = self.as_dex_mut();
		let dex::StateMembersMut {
			contract: Contract::V0(ref mut contract),
			item_factory,
			..
		} = dex.members_mut();
		contract
			.accounts
			.update_or_insert(
			&account_id,
			|| item_factory.new_account(&account_id),
			|Account::V0(ref mut account), already_registered| {
				ensure_here!(
					amount >= min_balance || already_registered,
					Error::DepositLessThanMinStorage
				);
				// Just add amount to account's NEAR balance
				if !registration_only {
					account.extra.near_amount += amount;
					return Ok(account.storage_balance_of());
				}
				// Registration only setups the account but doesn't leave
	space for tokens.
				if already_registered {
					log_str("ERR_ACC_REGISTERED");
					if amount > 0 {

		Promise::new(env::predecessor_account_id()).transfer(amount);
					}
					return Ok(account.storage_balance_of());
```

```
                            }
                            // Supply min balance to account and refund rest
                            account.extra.near_amount = min_balance;

                            let refund = amount - min_balance;
                            if refund > 0 {

                Promise::new(env::predecessor_account_id()).transfer(refund);
                            }

                            Ok(account.storage_balance_of())
                        },
                        )
                        .near_unwrap()
        }
...
}
```

11.2.1 Improvement Recommendation

Add the `ensure_payable_api_resumed` check to the `storage_deposit` method.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Initially, pausing of payable API was designed as a safeguard only for DEX operations—swapping, opening, and closing positions. After internal discussion, following the auditor's recommendation, the team decided to change requirements and expand behavior on other public methods that change smart contract state, including `storage_deposit`, `storage_withdraw`, and `storage_unregister`.

Business impact: The absence of the `ensure_payable_api_resumed` check in the smart contract when calling the `storage_deposit` method can break the business logic.

11.3 F-3 Contract Suspension Check Missing

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: Missing check when calling `storage_withdraw` method. The `storage_withdraw` method refers to the payable API and the `ensure_payable_api_resumed` check is needed.

Risk: **Low**

Location: `./veax/dex/src/chain/wasm.rs`: 451-472

Code section:

```
#[near_bindgen]
impl StorageManagement for State {

    #[payable]
    fn storage_withdraw(&mut self, amount: Option<U128>) -> StorageBalance {
        assert_one_yocto();
        let account_id = env::predecessor_account_id();
        let amount = amount.unwrap_or(U128(0)).0;
        let mut dex = self.as_dex_mut();
        let Contract::V0(ref mut contract) = dex.contract_mut();
        let (withdraw_amount, storage_balance) = contract
            .accounts
```

```
            .update(&account_id, |Account::V0(ref mut account)| {
                let available = account.storage_available();
                ensure_here!(available > 0, Error::NoStorageCanWithdraw);
                let withdraw_amount = if amount == 0 { available } else { amount };
                ensure_here!(withdraw_amount <= available,
Error::StorageWithdrawTooMuch);
                account.extra.near_amount -= withdraw_amount;
                Ok((withdraw_amount, account.storage_balance_of()))
            })
            .ok_or(dex::ErrorKind::AccountNotRegistered)
            .near_unwrap()
            .near_unwrap();
        Promise::new(account_id).transfer(withdraw_amount);
        storage_balance
    }

}
```

11.3.1 Improvement Recommendation

Add the `ensure_payable_api_resumed` check to the `storage_withdraw` method.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Initially, pausing of payable API was designed as a safeguard only for DEX operations (swapping, opening, and closing positions). After internal discussion, following the auditor's recommendation, the team decided to change requirements and expand behavior on other public methods that change smart contract state, including `storage_deposit`, `storage_withdraw`, and `storage_unregister`.

Business impact: The absence of `the ensure_payable_api_resumed` check in the smart contract when calling the `storage_withdraw` method can break the business logic.

11.4 F-4 Contract Suspension Check Missing

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: Missing check when calling the `storage_unregister` method. The `storage_unregister` method refers to the payable API, and the `ensure_payable_api_resumed` check is needed.

Risk: **Low**

Location: `./veax/dex/src/chain/wasm.rs:` 476-493

Code section:

```
#[near_bindgen]
impl StorageManagement for State {
...
       #[allow(unused_variables)]
       #[payable]
       fn storage_unregister(&mut self, force: Option<bool>) -> bool {
              assert_one_yocto();
              let account_id = env::predecessor_account_id();
              let Contract::V0(ref mut contract) = &mut self.0;
              let account = match contract.accounts.get(&account_id) {
```

```
                        None => return false,
                        Some(Account::V0(account)) => account,
                };
                assert!(
                        account.token_balances.is_empty(),
                        "{}",
                        dex::ErrorKind::TokensStorageNotEmpty
                );
                let balance = account.extra.near_amount;
                contract.accounts.remove(&account_id);
                Promise::new(account_id).transfer(balance);
                true
        }
...
}
```

11.4.1 Improvement Recommendation

Add the `ensure_payable_api_resumed` check to the `storage_unregister` method.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Initially, pausing of payable API was designed as a safeguard only for DEX operations—swapping, opening, and closing positions. After internal discussion, following the auditor's recommendation, the team decided to change requirements and expand behavior on other public methods that change smart contract state, including storage_`deposit`, `storage_withdraw`, and `storage_unregister`.

Business impact: The absence of the `ensure_payable_api_resumed` check in the smart contract when calling the `storage_unregister` method can break the business logic.

11.5 F-5 Unnecessary Check

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: The `open_position_full` method contains the `ensure_payable_api_resumed` check and a call to the `open_position` method. In turn, the `open_position` method also contains the `ensure_payable_api_resumed` check. There is no need to double-check.

Risk: Low

Location: `./veax/dex/src/dex/dex_impl.rs`: 526-552

Code section:

```
impl<T: Types, S: StateMut<T>, SS: BorrowMut<S>> Dex<T, S, SS> {
...
        pub fn open_position_full(
                &mut self,
                token_a: &TokenId,
                token_b: &TokenId,
                fee_rate: BasisPoints,
                amount_a: Amount,
                amount_b: Amount,
        ) -> Result<(PositionId, Amount, Amount, Liquidity)> {
                self.ensure_payable_api_resumed()?;
                self.open_position(
```

```
                    token_a,
                    token_b,
                    fee_rate,
                    PositionInit::FullRange {
                    amount_ranges: (
                           Range {
                                  min: Amount::one(),
                                  max: amount_a,
                           },
                           Range {
                                  min: Amount::one(),
                                  max: amount_b,
                           },
                    ),
                    },
             )
      }
...
}
```

11.5.1 Improvement Recommendation

Remove the call to the `ensure_payable_api_resumed` method from the `open_position_full` method.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Unnecessary checks were removed.

Business impact: The presence of unnecessary double-checks in a smart contract on the NEAR blockchain can lead to increased gas fees, longer execution times, and potential security risks, all of which can have a negative impact on the success of a decentralized exchange and its associated business.

11.6 F-6 Unnecessary Check

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: The `swap` method contains the `ensure_payable_api_resumed` check, however the swap method call only occurs in two methods (`swap_exact_in`, `swap_exact_out`) that also contain the `ensure_payable_api_resumed` check before calling the swap method.

Risk: Low

Location: `./veax/dex/src/dex/dex_impl.rs`: 786-809

Code section:

```
impl<T: Types, S: StateMut<T>, SS: BorrowMut<S>> Dex<T, S, SS> {
...
pub fn swap(
       &mut self,
       token_in: &TokenId,
       token_out: &TokenId,
       exact_in_or_out: Exact,
       amount: Amount,
       ) -> Result<Amount> {
              self.ensure_payable_api_resumed()?;
              let (pool_id, swapped) = PoolId::try_from_pair((token_in.clone(),
       token_out.clone()))
```

```
                .map_err(|e| error_here!(e))?; // todo: avoid .clone()
            let direction = if swapped { Side::Right } else { Side::Left };

            let Contract::V0(ref mut contract) = self.contract_mut();
            let amount = contract
                .pools
                .update(&pool_id, |Pool::V0(ref mut pool)| {
                pool.swap(direction, exact_in_or_out, amount)
                })
                .ok_or(error_here!(ErrorKind::PoolNotRegistered))??;

            self.log_pool_state(&pool_id, PoolUpdateReason::Swap);

            Ok(amount)
        }
...
}
```

11.6.1 Improvement Recommendation

Remove the call to the `ensure_payable_api_resumed` method from the `open_position_full` method.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Unnecessary checks were removed.

Business impact: The presence of unnecessary double-checks in a smart contract on the NEAR blockchain can lead to increased gas fees, longer execution times, and potential security risks, all of which can have a negative impact on the success of a decentralized exchange and its associated business.

11.7 F-7 Possible Occurrence of an Unwanted Event

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: The `log_withdraw_event` method is triggered before the `send_tokens` function is called, which may fail. This can lead to an unwanted sequence of logs.

Risk: Informational

Location: `./veax/dex/src/dex/dex_impl.rs`: 346-384

Code section:

```
impl<T: Types, S: StateMut<T>, SS: BorrowMut<S>> Dex<T, S, SS> {
...
pub fn withdraw(
        &mut self,
        account_id: &AccountId,
        token_id: &TokenId,
        amount: Amount,
        unregister: bool,
        extra: S::SendTokensExtraParam,
        ) -> Result<S::SendTokensResult> {
        self.ensure_payable_api_resumed()?;
        let Contract::V0(ref mut contract) = self.contract_mut();
        let (amount, balance) = contract
                .accounts
                .update(account_id, |Account::V0(ref mut account)| {
```

```
                let balance = account
                        .token_balances
                        .inspect(token_id, |balance| *balance)
                        .ok_or(error_here!(ErrorKind::TokenNotRegistered))?;

                // get full amount if amount param is 0
                let amount = if amount == Amount::zero() {
                        balance
                } else {
                        amount
                };

                ensure_here!(amount > Amount::zero(), ErrorKind::IllegalWithdrawAmount);
                account.withdraw(token_id, amount)?;
                if unregister {
                        account.unregister_token(token_id)?;
                }
                Ok((amount, balance - amount))
                })
                .ok_or(error_here!(ErrorKind::AccountNotRegistered))??;

        self.logger_mut()
                .log_withdraw_event(account_id, token_id, &amount, &balance);

        Ok(self.send_tokens(account_id, token_id, amount, extra))
        }
...
}
```

11.7.1 Improvement Recommendation

Perhaps you should call the `log_withdraw_event` method after the successful completion of the `send_tokens` method.

Status: *Noted*

Comment (developers): Noted, won't fix. Due to the asynchronous nature of Near cross-contract calls, it's impossible to guarantee consistency of withdraw event generated by the DEX smart contract and transferring tokens to the user balance performed by a token smart contract. The user balance in DEX smart contract decreasing before the method `send_tokens` of the token smart contract is called to prevent a double spending attack. In case of tokens smart contract misbehaving or lack of gas, the callback execution cannot be guaranteed. Deposit and withdraw events produced by DEX smart contract signal about increasing or decreasing internal user balance on DEX itself. To ensure that tokens were actually transferred to the user account, only events produced by tokens smart contracts should be used.

Business impact: In the context of a smart contract on the NEAR blockchain, the impact of the possible occurrence of an undesirable event can be significant for a business.

11.8 F-8 Missing Cargo Overflow Checks

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: It was observed that there are no `overflow-checks=true` in `Cargo.toml`. By default, overflow checks are disabled in optimized release builds. Hence, if there is an overflow in release builds, it will be silenced, leading to unexpected behavior of an application.

Risk: Informational

Location: `./veax/dex/Cargo.toml`

11.8.1 Improvement Recommendation

It is recommended to add `overflow-checks=true` under your release profile in `Cargo.toml`.

Status: *Fixed*

Comment (developers): Fixed since version 1.0.11. Most arithmetic operations executed by DEX smart contract calculated on floating point or fixed point types, for which native 32- and 64-bit integer types are building blocks and silent overflowing wrapped around at the boundary of the type is wanted behavior. Nevertheless, after internal discussion, the team decided to enable overflow checks for release build to improve code transparency and increase security guarantees of final builds.

Business impact: Overflow and underflow checks are important in Rust programming, particularly in smart contract development on the NEAR blockchain. In the context of a smart contract, overflow and underflow can occur when arithmetic operations are performed on numbers that are outside the valid range of the data type being used. The impact of missing overflow checks in the `Cargo.toml` file can be significant for businesses.

11.9 F-9 Elastic Supply Problem

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: The potential issue with elastic supply tokens is that their price, supply, and user balances can dynamically adjust. Examples of elastic supply tokens include inflation tokens, deflation tokens, and rebasing tokens. However, the current implementation of the protocol does not support elastic supply tokens. If the token being used is a deflation token, there could be a discrepancy between the recorded amount of transferred tokens to the smart contract and the actual number of transferred tokens due to a small number of tokens being burned by the token smart contract. This inconsistency can have security implications for operations that rely on the transferred amount of tokens.

Risk: **Informational**

Status: *Noted*

Comment (developers): Noted, won't fix. As most of the DEXes, VEAX doesn't support inflation tokens, deflation tokens, and rebasing tokens. Unfortunately, identification of such tokens is not specified in any NEAR standard, so creation of pools with such tokens and performing swaps remains possible. However, withdrawing such tokens from DEX will cause insufficient fund issues due to inconsistency of balance representation. That inconsistency doesn't have security implications rather then inability to withdraw misbehaved token that we can't prevent.

Business impact: An elastic token supply problem can have significant business impacts in a smart contract on the NEAR blockchain.

11.10 F-10 Unnecessary Storage of Data On-chain

Commit: *40a784900bb14085625ac2902b436840606550fa*

Branch: *concentrated-liquidity*

Description: The smart contract contains the logic for creating, extending, and removing verified tokens. In the current implementation of a smart contract, storing a list of verified tokens on the blockchain is potentially redundant. The problem with unnecessary storage of data that can be placed off-chain is that it can lead to higher storage costs and slower performance.

To address this issue, it is important to consider which data really needs to be stored on-chain and which data can be placed off-chain. Off-chain data storage can be much cheaper and faster than on-chain storage. In some cases, it may be possible to use existing data storage solutions, such as cloud storage services, to store data off-chain. However, it is important to ensure that any off-chain data storage solutions are secure and reliable, and that they do not compromise the security of the smart contract or the blockchain as a whole.

Risk: Informational

Status: *Noted*

Comment (developers): Noted, won't fix. The list of verified tokens stored in the smart contract state doesn't imply changing the behavior of the smart contract itself because VEAX has a decentralized nature and allows operations for all compatible tokens, including unverified ones. However, token verification affects the user interface of veax.com, so we maintain the list of verified tokens on blockchain to provide publicity and accountability of the process.

Business impact: When it comes to smart contracts on the NEAR blockchain, storing unnecessary data on-chain can result in a range of negative impacts for businesses. This is because storing data on-chain incurs storage costs that are paid in NEAR tokens, which can affect the profitability of the business using the smart contract. Additionally, retrieving unnecessary data from the blockchain can increase the time required to process transactions, ultimately reducing the efficiency of the smart contract and the associated business processes.

12. Results from Semi-Automatic Scans

12.1 Rustle

Rustle is a static analyzer designed to automatically analyze **Rust-based** smart contracts on the **NEAR blockchain**. It is a tool that helps developers identify potential bugs and vulnerabilities in their code before it is deployed, allowing them to fix issues early in the development cycle.

Static analysis involves examining code without executing it, with the goal of finding potential issues that could cause problems during execution. Rustle performs static analysis by analyzing the code and looking for potential issues such as *uninitialized variables, integer overflow, and invalid pointer usage.*

By identifying these issues early, Rustle can help developers avoid common mistakes and improve the overall security and stability of their smart contracts. Rustle can also help developers write more efficient code by pointing out areas where optimizations can be made.

Overall, Rustle is a powerful tool for developers working on Rust-based smart contracts for the NEAR blockchain, helping them write more secure, efficient, and reliable code.

Below you can see the full list of vulnerabilities that Rustle found in the project.

Not detected: The detector found no problems. However, that doesn't mean they don't exist.

Detected: The detector has detected a potential problem.

Title	Description	Status
Unhandled promise	Find Promises that are not handled.	Not detected
Non private callback	Missing macro `#[private]` for callback functions.	Not detected
Reentrancy	Find functions that are vulnerable to reentrancy attack.	Not detected
Unsafe math	Lack of overflow check for arithmetic operation.	Not detected
Self-transfer	Missing check of `sender != receiver`	Not detected
Incorrect json type	Incorrect type used in parameters or return values.	Not detected
Unsaved changes	Changes to collections are not saved.	Not detected
NFT approval check	Find `nft_transfer` without check of approval ID.	Not detected
NFT owner check	Find approve or revoke functions without owner check.	Not detected
Div before mul	Precision loss due to incorrect operation order.	Detected
Round	Rounding without specifying ceil or floor.	Not detected
Lock callback	Panic in callback function may lock contract.	Not detected
Yocto attach	No `assert_one_yocto` in privileged function.	Not detected
Duplicate collection ID	Duplicate id uses in collections.	Not detected
Unregistered receiver	No panic on unregistered transfer receivers.	Not detected
NEP ***$id*** interface	Find all unimplemented NEP interface.	Not detected
Prepaid gas	Missing check of prepaid gas in `ft_transfer_call`.	Not detected
Non callback private	Macro `#[private]` used in non-callback function.	Not detected
Unused return value	Function result not used or checked.	Not detected
Upgrade function	No upgrade function in contract.	Not detected
Tautology	Tautology used in conditional branch.	Not detected
Storage gas	Missing balance check for storage expansion.	Not detected
Unclaimed storage fee	Missing balance check before storage unregister.	Not detected
Inconsistency	Use of similar but slightly different symbol.	Not detected

Timestamp	Find all uses of timestamp.	Not detected
Complex loop	Find all loops with complex logic which may lead to DoS.	Detected
External call	Find all cross-contract invocations.	Not detected
Promise result	Find all uses of promise result.	Detected
Transfer	Find all transfer actions.	Detected
Public interface	Find all public interfaces.	Detected

12.2 RustSec: Cargo Audit

RustSec is a project focused on improving the security of Rust software. One of its tools is cargo-audit, a command-line tool that scans Rust dependencies for issues reported to the National Vulnerability Database (NVD) and the RustSec Advisory Database.

Crate	Version	Title	Date	ID	URL	Solution	Dependency Tree
chrono	0.4.19	Potential segfault in localtime_r invocations	2020-11-10	RUSTSEC-2020-0159	https://rustsec.org/advisories/RUSTSEC-2020-0159	Upgrade to >=0.4.20	chrono 0.4.19, near-primitives 0.13.0, near-vm-logic 0.13.0, near-sdk 4.0.0, veax-dex 0.1.1, near-contract-standards 4.0.0
libgit2-sys	0.14.1+1.5.0	git2 does not verify SSH keys by default	2023-01-20	RUSTSEC-2023-0003	https://rustsec.org/advisories/RUSTSEC-2023-0003	Upgrade to >=0.13.5, <0.14.0 OR >=0.14.2	libgit2-sys 0.14.1+1.5.0, git2 0.16.0, ver-from-git 0.1.0, veax-dex 0.1.1

time	0.1.44	Potential segfault in the time crate	2020-11-18	RUSTSEC-2020-0071	https://rustsec.org/advisories/RUSTSEC-2020-0071	Upgrade to >=0.2.23	time 0.1.44, chrono 0.4.19, near-primitives 0.13.0, near-vm-logic 0.13.0, near-sdk 4.0.0, veax-dex 0.1.1, near-contract-standards 4.0.0

The static code analyzer and dependency analysis scans were completed successfully, and the identified errors, bugs, and issues were carefully reviewed. In the tested version, we did not discover any significant security concerns in the codebase.

The majority of the reported issues were deemed irrelevant, and related to naming conventions, visibility, or access control to methods that might be unwanted. The successful completion of the static code analyzer and dependency analysis scans means that our codebase has been thoroughly scrutinized, and we are confident that it is free of any significant security concerns.

12.3 Fuzzing Results

Function	Fuzzing Characters	Fuzzing Special Characters	Fuzzing Big Amounts	Fuzzing Low Amounts	Fuzzing Negative Numbers	Fuzzing Large Arrays	Passed All
open_position	alphanumeric	!@#$%^&*()_+[]{}	1000000000, 100000000.0	1-100	-100, -1000	N/a	Passed
withdraw_fee	alphanumeric	!@#$%^&*()_+[]{}	1000000000, 100000000.0	1-100	-100, -1000	N/a	Passed
exact-in swap	alphanumeric	!@#$%^&*()_+[]{}	1000000000, 100000000.0	1-100	-100, -1000	N/a	Passed

exact-out swap	alphanumeric	!@#$%^&*()_+[]{}	1000000000, 100000000.0	1-100	-100, -1000	N/a	Passed

Note: Our team worked with Cargo-fuzz. Cargo-fuzz is a fuzzing tool that can be used to test smart contracts for issues and potential bugs. The process involves defining the input space for the smart contract, using Cargo-fuzz to generate test cases, executing the test cases against the smart contract, analyzing the results, and fixing any issues that are identified.

The benefit of using Cargo-fuzz is that it can generate a large number of test cases that cover a wide range of input values, which can help identify potential issues that may not be discovered through manual testing. Cargo-fuzz also uses coverage-guided fuzzing to optimize the testing process and identify areas that require more testing. In the context of testing a smart contract, Cargo-fuzz can be used to identify issues such as buffer overflows, integer overflows, and other common issues that can occur in smart contracts.

By identifying and fixing these issues, the overall security and reliability of the smart contract can be improved, which can help prevent potential attacks and other security risks.

Overall, using Cargofuzz for fuzzing a smart contract involves a comprehensive approach that includes defining the input space, using Cargofuzz to generate test cases, executing the test cases, analyzing the results, and fixing any issues that are identified. By following this process, the smart contract can be thoroughly tested for potential issues and bugs, which can help improve its overall security and reliability.

- "Alphanumeric" includes all letters (both uppercase and lowercase) and digits.
- "Fuzzing Special Characters" include various special characters.
- "Fuzzing Big Amounts" and "Fuzzing Low Amounts" refer to large and small values, respectively, that can be used as input parameters for the function.
- "Fuzzing Negative Numbers" refers to negative values that can be used as input parameters for the function.
- "Fuzzing Large Arrays" refers to an array with a large number of elements that can be used as input parameters for the function.
- The "Passed All" column indicates whether the function has passed all the fuzzing tests or not.

13. Conclusion

The auditors carried out a comprehensive security audit of the Client's smart contracts with the specific aim of ascertaining whether the protection of the smart contract could be compromised by an attacker. The ultimate goal of the audit was to ensure that the Customer's smart contract was secure from external threats.

As a result of the audit, it was established that an attacker could not completely abuse the smart contract or directly violate the Customer's business requirements. However, the audit did reveal the presence of six low issues and four informational issues, which should be addressed to enhance the overall security of the smart contract.

To mitigate these issues, we recommend that the Customer takes steps to address the issues identified. In addition, implementing thorough documentation and unit and functional tests for all contracts will help to prevent future issues and ensure the overall security of the smart contract.

The customer, having familiarized themself with the identified and analyzed issues. They have shown that they have a deep understanding of the issues and have taken the necessary steps to address them. The customer's proactive approach to identifying potential problems and implementing remedial actions has further enhanced the already high quality of the project in terms of security.

It is commendable that the customer has taken a responsible approach to ensuring the security of the project. By being proactive, they have not only ensured that the project meets the required security standards, but they have also increased the overall quality of the project. It is always reassuring to work with a customer who takes security seriously, and this level of dedication sets an excellent example for others to follow.

Overall, the customer's commitment to addressing potential issues and implementing remedial actions demonstrates their strong sense of responsibility and dedication to ensuring that the project is of the highest quality in terms of security. This level of attention to detail and proactive approach is vital in today's environment where security threats are becoming increasingly prevalent, and it is always reassuring to have a customer who takes security seriously.

Appendix/Test Functions

After conducting a thorough analysis of the provided table of tests, it can be concluded that all tests have passed without any issues. This is an extremely positive finding, as it indicates that there were no critical, high, or medium issues present in the system. However, six low issues were discovered, as well as four informational issues.

It is important to take these findings seriously and investigate them further. Low issues may not pose an immediate threat, but they should still be addressed to prevent potential future issues. Informational issues may not necessarily be security-related, but they can still provide valuable insights into areas of the system that could be improved.

Overall, the fact that no critical and high issues were found is a very positive result. It is important to continue to monitor the system for any potential issues and address them promptly to ensure the ongoing security and stability of the system.

Test Description	Test Result
dex::dex_tests::add_remove_guards	ok
dex::dex_tests::deposit_fails_account_not_registered	ok
dex::dex_tests::deposit_fails_token_not_registered	ok
dex::dex_tests::deposit_successful	ok
dex::dex_tests::create_instance	ok
chain::events::tests::test_deposit_event	ok
dex::dex_tests::open_two_positions	ok
chain::events::tests::test_guards_events	ok
dex::dex_tests::open_non_first_position_signle_sided_fails	ok
dex::dex_tests::open_first_position_signle_sided_fails	ok
dex::dex_tests::swap_exact_in_failure	ok
dex::dex_tests::open_close_position	ok
chain::events::tests::test_verified_tokens_events	ok
dex::dex_tests::swap_exact_out_failure	ok
dex::dex_tests::test_reserves_consistency	ignored
dex::dex_tests::swap_exact_in_success	ok

dex::dex_tests::version	ignored
dex::dex_tests::withdraw_failure_token_not_registered	ok
dex::dex_tests::withdraw_failure_account_not_registered	ok
dex::dex_tests::withdraw_failure_not_enough_tokens	ok
dex::dex_tests::withdraw_failure_zero_amount_zero_balance	ok
dex::dex_tests::withdraw_success_whole_balance	ok
dex::errors::tests::error_desc_roundtrip	ok
chain::events::tests::test_close_position_event	ok
dex::test_utils::test_add_account	ok
dex::test_utils::test_new_state	ok
dex::test_utils::test_root_key	ok
dex::tick::tests::create_tick_with_limited_range_of_value::case_1_success_zero	ok
dex::test_utils::test_ser_de_loop	ok
dex::tick::tests::create_tick_with_limited_range_of_value::case_2_success_min	ok
dex::dex_tests::swap_exact_out_success	ok
dex::tick::tests::create_tick_with_limited_range_of_value::case_3_success_max	ok
dex::test_utils::test_ordered_map	ok
chain::events::tests::test_withdraw_event	ok
chain::events::tests::test_open_position_event	ok
dex::tick::tests::eff_sqrtprices_on_different_levels_match::case_3	ok
dex::tick::tests::eff_sqrtprices_on_different_levels_match::case_5	ok
dex::tick::tests::eff_sqrtprices_on_different_levels_match::case_6	ok
dex::tick::tests::eff_sqrtprices_on_different_levels_match::case_2	ok
dex::tick::tests::eff_sqrtprices_on_different_levels_match::case_1	ok
dex::tick::tests::max_bit_index_for_price_tick	ok
dex::tick::tests::scale_back_and_forth::case_2	ok

dex::v0::pool_state_ex::pool_tests::empty_pool_default	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_1_0	ok
dex::dex_tests::withdraw_success_arbitrary	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_2_1	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_3_2	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_4_3	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_5_4	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_6_5	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_7_6	ok
dex::v0::pool_state_ex::pool_tests::fee_rate::fee_level_8_7	ok
dex::v0::pool_state_ex::pool_tests::liquidities	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_1_0	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_3_2	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_4_3	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_2_1	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_5_4	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_6_5	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_7_6	ok
dex::v0::pool_state_ex::pool_tests::liquidity::fee_level_8_7	ok
dex::v0::pool_state_ex::pool_tests::lp_fee_fraction	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_1_0	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_2_1	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_3_2	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_4_3	ok

dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_5_4	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_6_5	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price_shift::side_1_Side__Left::fee_level_7_6	ok
dex::v0::pool_state_ex::pool_tests::max_effective_sqrt_price	
fp::i192x64::test::test_div	ok
fp::i192x64::test::test_i192x64_to_f64	ok
fp::i192x64::test::test_mul	ok
fp::i192x64::test::test_mul_large	ok
fp::i192x64::test::test_sub	ok
fp::i192x64::test::test_sum	ok
fp::i192x64::test::test_try_f64_to_i192x64_large	ok
fp::i192x64::test::test_try_f64_to_i192x64_overflow	ok
fp::i192x64::test::test_try_f64_to_i192x64_prec_loss	ok
fp::i192x64::test::test_try_f64_to_i192x64_tiny	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_from_leading_and_trailing_parts	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_large	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_negative	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_overflow	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_prec_loss	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_tiny	ok
fp::try_float_to_ufp::test::test_try_f64_to_u128x128_zero	ok
fp::try_float_to_ufp::test::test_try_f64_to_u192x192	ok
fp::try_float_to_ufp::test::test_try_f64_to_u192x192_zero	ok
fp::u128x128::test::test_div	ok

fp::u128x128::test::test_fract	ok
fp::u128x128::test::test_integer_sqrt	ok
fp::u128x128::test::test_floor	ok
fp::u128x128::test::test_mul	ok
fp::u128x128::test::test_mul_large	ok
fp::u128x128::test::test_sub	ok
fp::u128x128::test::test_sum	ok
fp::u128x128::test::test_u128x128_to_f64	ok
fp::u192x192::test::test_ceil	ok
fp::u192x192::test::test_div	ok
fp::u192x192::test::test_fract	ok
fp::u192x192::test::test_floor	ok
fp::u192x192::test::test_integer_sqrt	ok
fp::u192x192::test::test_mul	ok