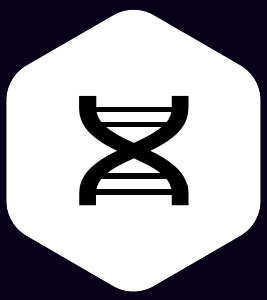


HELIX
CRYPTOCURRENCY

Helix FRC-0046
SMART CONTRACT AUDIT

zokyo

March 29th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

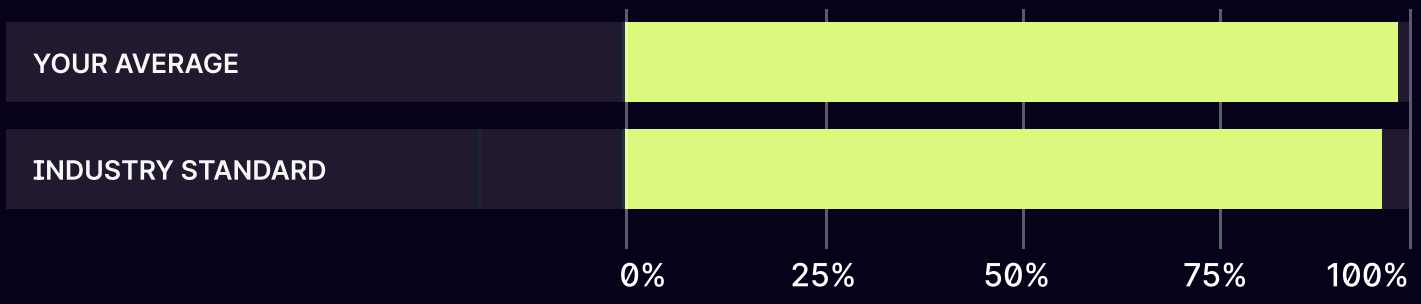
This document outlines the overall security of the Helix smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Helix smart contract's codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

Testable Code



100% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the FVM network's fast-paced and rapidly changing environment, we recommend that the Helix team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Helix repository:
https://github.com/helix-onchain/filecoin

Last commit: https://github.com/helix-onchain/filecoin/pull/204

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- frc42_dispatch
- frc46_token
- fvm_actor_utils

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Helix smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Rust testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contracts logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

Contracts are well written and structured. There was no critical issue found during the audit alongside two with high severity, two with medium severity, one with low severity and some informational issues . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Helix team and the Helix team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Bug in pooling instance allocator	High	Resolved
2	count_balances always return 1	High	Resolved
3	The `Err`-variant returned from this function is very large	Medium	Resolved
4	set_balance doesn't update supply	Medium	Acknowledged
5	Check the "from" address before executing code	Low	Acknowledged
6	ansi_term is Unmaintained	Informational	Acknowledged
7	mapr is Unmaintained	Informational	Acknowledged
8	memmap is unmaintained	Informational	Acknowledged
9	Crate parity-wasm deprecated by the author	Informational	Acknowledged
10	Code duplication	Informational	Resolved
11	Code duplication	Informational	Acknowledged
12	Code duplication	Informational	Acknowledged
13	Code duplication	Informational	Acknowledged
14	Code duplication	Informational	Resolved

HIGH | RESOLVED

Bug in pooling instance allocator

File: Cargo.lock
Details: bug in Wasmtime's implementation of its pooling instance allocator where when a linear memory is reused for another instance, the initial heap snapshot of the prior instance can be visible, erroneously to the next instance.
Mitigations are described here: https://github.com/bytecodealliance/wasmtime/security/advisories/GHSA-wh6w-3828-g9qf

Recommendation:
Update Wasmtime library to the version >=1.0.2, <2.0.0 OR >=2.0.2

HIGH | RESOLVED

count_balances always return 1

File: frc46_token/src/token/state.rs
Details:
The count_balances function always returns 1, regardless of the number of elements in the balance_map.

Recommendation:
modify the function, so it returns the correct count of elements

MEDIUM | RESOLVED

the `Err`-variant returned from this function is very large

File: frc46_token/src/token/state.rs
Details: A Result is at least as large as the Err-variant. While we expect that variant to be seldomly used, the compiler needs to reserve and move that much memory every single time. The `InsufficientAllowance` is the largest variant and it contains at least 192 bytes.

Recommendation:
try reducing the size of `token::state::StateError`, for example by boxing large elements or replacing
it with `Box<token::state::StateError>`. See: https://rust-lang.github.io/rust-clippy/master/index.html#result_large_err

MEDIUM | ACKNOWLEDGED

set_balance doesn't update supply

File: frc46_token/src/token/state.rs
Details:
The set_balance function doesn't update the supply value of the state. Of course, there is an implementation for the token with the set_balance function, which calls the state.set_balance and then updates the supply, but it's not obvious. Anyone, writing the code, could call the state.set_balance and face an issue, when balances aren't correlated with the total supply.

Recommendation:
move the supply update to the state.set_balance function

LOW | ACKNOWLEDGED

Check the "from" address before executing code

File: frc46_token/src/token/mod.rs
Details:
In the function "transfer", right now, the code looks like: `let from_id = self.msg.resolve_or_init(from)?;` and if the address "from" is not existing, a new one would be created. Of course, the "StateError::InsufficientBalance" would be panicked, but excess executions would be done, like "initialize_account", "change_balance_by" (for "to" address), "get_balance_map", etc.

Recommendation:
To prevent excess gas spending, we recommend replacing "resolve_or_init" with "resolve", then using match to find if it exists, and returning "Err" if not. Or, at least, switch the "change_balance_by" function calls so the balance changing for the "from" address goes first, that will at least save gas from excess calling the "change_balance_by" for "to" address.

Comment from Helix team:
It's not necessary that InsufficientBalance will panic. Uninitialized addresses have an implicit `0` balance. A transfer of amount `0` in cases of an uninitialized address (SECP address or Actor Address) should first initialize that address and then succeed in transferring the `0` amount. See test case "transfers_from_uninitialized_addresses" in frc46_token/src/token/mod.rs

INFORMATIONAL | ACKNOWLEDGED

ansi_term is Unmaintained

File: Cargo.lock
Details: The maintainer has advised that this crate is deprecated and will not receive any maintenance. The crate does not have many dependencies and may or may not be ok to use as-is. The last release was three years ago.

Comment from Helix team:
This and the following three unmaintained deps all appear to be used only for testing. The token library depends on the FVM itself for integration tests. Thus, I think we're happy with these dependencies remaining, though the FVM team will eventually work to remove them

INFORMATIONAL | ACKNOWLEDGED

mapr is Unmaintained

File: Cargo.lock
Details:The mapr fork has been merged back into upstream fork memmap2.
The maintainer(s) have advised mapr is deprecated and will not receive any maintenance in favor of using memmap2.

INFORMATIONAL | ACKNOWLEDGED

memmap is unmaintained

File: Cargo.lock
Details: The author of the memmap crate is unresponsive.
Maintained alternatives:
- memmap2

INFORMATIONAL | ACKNOWLEDGED

Code duplication

File: frc46_token/src/token/mod.rs
Details: Both functions "transfer" and "transfer_from" have the same code. That code could be merged and used from both functions. The same is correct for "burn" and "burn_from"

INFORMATIONAL | ACKNOWLEDGED

Code duplication

File: frc46_token/src/token/state.rs
Details: Both functions ""change_balance_by"" and ""set_balance"" have the same code functionality but written in some different manner. Make sure the code is consistent as well as the implementation by moving the duplicated functionality outside those two functions. The same above is correct for ""get_allowance_between"" and ""change_allowance_by"" functions, where getting the existing allowance is implemented in some different way that is inconsistent.

Comment from Helix team:
Acknowledged: a helper function could be introduced to change_balance_by and set_balance to retrieve the map and balance in one line, however, the extra specialized function is probably not worth using the existing primitives with slight duplication.

Note
that using `self.get_balance` here is not suitable as during modification the allowance HAMT would have to be loaded from Cid again, causing unnecessary gas costs (unless transaction/ caching mechanisms were designed)

INFORMATIONAL | ACKNOWLEDGED

Crate parity-wasm deprecated by the author

File: Cargo.lock
Details: This PR (https://github.com/paritytech/parity-wasm/pull/334) explicitly deprecates parity-wasm.
The author recommends switching to wasm-tools (https://github.com/bytecodealliance/wasm-tools).

INFORMATIONAL | RESOLVED

Code duplication

File: frc46_token/src/token/mod.rs
Details: A Token implementation contains such functions as: replace, load_state and load_replace. The last one is does the same functionality as the first two do together. Instead of duplicating the functionality, call both functions from the last one.

INFORMATIONAL | ACKNOWLEDGED

Code duplication

File: frc46_token/src/token/mod.rs
Details: Functions such as balance_of, allowance, revoke_allowance, burn_from and transfer_from have almost the same ""match self.msg.resolve_id..."" functionality which differs only by the ""Err(MessagingError::AddressNotResolved)"" section in the burn_from and transfer_from functions. Those all could be simplified.

INFORMATIONAL | RESOLVED

Code duplication

File: frc46_token/src/token/state.rs
Details: In the function "attempt_use_allowance" there are two if's "current_allowance.is_zero () && operator != owner" and "current_allowance.lt(amount)". Both have the same code inside. Those could be merged into one if-block.

	frc42_dispatch	frc46_token
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	fvm_actor_utils
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Helix in verifying the correctness of their contract's code, our team was responsible for writing integration tests using the Rust testing framework.

The tests were based on the functionality of the code's, as well as a review of the Helix contracts requirements for details about issuance amounts and how the system handles these.

Unit test for: frc42_dispatch
Finished test [unoptimized + debuginfo] target(s) in 0.31s
Running unittests src/lib.rs

running 4 tests
✓ test match_method::tests::handle_constructor ... ok
✓ test match_method::tests::handle_optional_commas ... ok
✓ test match_method::tests::handle_unknown_method ... ok
✓ test match_method::tests::handle_user_method ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

Unit test for: frc46_token
Finished test [unoptimized + debuginfo] target(s) in 0.23s
Running unittests src/lib.rs

running 48 tests
✓ test token::error::test::it_creates_exit_codes ... ok
✓ test token::error::test::error_codes_and_messages ... ok
✓ test token::state::test::it_handles_invalid_data_load ... ok
✓ test token::state::test::it_handles_missing_data_load ... ok
✓ test token::state::test::it_fails_to_decrease_balance_below_zero ... ok
✓ test token::state::test::it_instantiates ... ok
✓ test token::state::test::check_invariants_accumulates_errors ... ok
✓ test token::state::test::it_increases_balance_from_zero ... ok
✓ test token::state::test::check_invariants_balances ... ok
✓ test token::state::test::check_invariants_allowances ... ok
✓ test token::state::test::it_consumes_allowances_atomically ... ok
✓ test token::state::test::it_sets_balances ... ok
✓ test token::state::test::it_makes_transfers ... ok
✓ test token::state::test::it_changes_allowances_between_actors ... ok
✓ test token::state::test::it_sets_allowances_between_actors ... ok
✓ test token::state::test::it_revokes_allowances ... ok
✓ test token::test::it_allows_delegated_burns ... ok

✓ test token::test::
it_doesnt_initialize_accounts_when_default_values_can_be_returned ... ok
✓ test token::test::check_invariants_returns_a_state_summary ... ok
✓ test token::test::it_disallows_delgated_transfer_by_uninitialised_pubkey ... ok
✓ test token::test::it_burns ... ok
✓ test token::test::it_disallows_delegated_burns_by_uninitialised_pubkeys ... ok
✓ test token::test::it_allows_delegated_burns_by_resolvable_pubkeys ... ok
✓ test token::test::it_allows_delegated_transfer_by_resolvable_pubkey ... ok
✓ test token::test::it_fails_to_burn_below_zero ... ok
✓ test token::test::it_fails_to_mint_if_receiver_hook_aborts ... ok
✓ test token::state::test::it_counts_balances ... ok
✓ test token::state::test::it_allows_variable_bit_width ... ok
✓ test token::test::it_fails_to_transfer_when_insufficient_balance ... ok
✓ test token::test::it_instantiates_and_persists ... ok
✓ test token::test::it_provides_atomic_transactions ... ok
✓ test token::test::it_instantiates_with_variable_bit_width ... ok
✓ test token::test::it_enforces_granularity ... ok
✓ test token::test::it_fails_to_transfer_when_receiver_hook_aborts ... ok
✓ test token::test::it_allows_delegated_transfer ... ok
✓ test token::test::it_mutates_externally_loaded_state ... ok
✓ test token::test::it_fails_to_transfer_when_insufficient_allowance ... ok
✓ test token::test::it_doesnt_use_allowance_when_insufficent_balance ... ok
✓ test token::test::it_sets_balances ... ok
✓ test token::test::it_wraps_a_previously_loaded_state_tree ... ok
✓ test token::test::it_transfers_from_uninitialized_addresses ... ok
✓ test token::test::it_transfers_to_self ... ok
✓ test token::test::it_sets_allowances ... ok
✓ test token::test::it_transfers_to_uninitialized_addresses ... ok
✓ test token::test::it_transfers ... ok
✓ test token::test::it_mints ... ok
✓ test token::test::it_tracks_allowances ... ok
✓ test token::test::test_account_combinations ... ok
test result: ok. 48 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s

Unit test for: fvm_actor_utils
Finished test [unoptimized + debuginfo] target(s) in 1.11s
Running unittests src/lib.rs

running 2 tests
✓ test receiver::test::calls_hook ... ok
✓ test receiver::test::panics_if_not_called - should panic ... ok
test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
frc42_dispatch	100	100	100	100	
frc46_token	100	100	100	100	
fvm_actor_utils	100	100	100	100	
All Files	**100**	**100**	**100**	**100**	

We are grateful for the opportunity to work with the Helix team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Helix team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo