Guvenkaya® The Bedrock of Security

Templar

NEAR Smart Contract Security Review

Lead Security Engineer: Michal Bajor
Date of Engagement: 1st April 2025 - 14th April 2025
Visit: www.guvenkaya.co

Contents

About Us

Guvenkaya is a security research firm specializing in Rust security, Web3 security of Non-EVM protocols, and Web2 security. With our expertise, we provide both security auditing services and custom security solutions

About Templar

Templar is the first Cypher Lending Protocol for Bitcoin. Templar lets you borrow any asset against your Bitcoin without trusting centralized institutions.

Audit Results

Guvenkaya conducted a security assessment of the **Templar Single Chain NEAR Smart Contracts** from 1st April 2025 to 14th April 2025. During this engagement, a total of **3 findings** were reported. 1 of the findings were High and the remaining were informational severity. The Templar team has either fixed or acknowledged all the issues.

Project Scope

Files	Link
Accumulator	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/accumulator.rs
Asset	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/asset.rs
Borrow	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/borrow.rs
Chunked Append Only List	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/chunked_append_only_list.rs
Event	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/event.rs
Fee	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/fee.rs
Interest Rate Strategy	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/interest_rate_strategy.rs

Files	Link
Lib	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/lib.rs
Market Balance Oracle Configuration	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/market/balance_oracle_configuration.rs
Market Configuration	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/market/configuration.rs
Market External	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/market/external.rs
Market Implementation	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/market/impl.rs
Market Module	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/market/mod.rs
Number	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/number.rs
Oracle Module	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/oracle/mod.rs
Oracle Pyth	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/oracle/pyth.rs
Snapshot	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/snapshot.rs

Files	Link
Static Yield	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/static_yield.rs
Supply	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/supply.rs
Time Chunk	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/time_chunk.rs
Withdrawal Queue	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/common/src/withdrawal_queue.rs
Market Contract Lib	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/contract/market/src/lib.rs
Market Contract FT Receiver	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/contract/market/src/impl_ft_receiver.rs
Market Contract Helper	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/contract/market/src/impl_helper.rs
Market Contract External	https://github.com/Templar-Protocol/contract-mvp/blob/aeb8a93d8844b6b9bb17187e6d9172b713c611fd/contract/market/src/impl_market_external.rs

Out of Scope

The audit will include reviewing the code for security vulnerabilities. The audit does not include a review of the tests and dependencies.

Timeline

Start of the audit	Draft report	Final report
1st April 2025	14th April 2025	17th April 2025

Methodology

RESEARCH INTO PROJECT ARCHITECTURE

PREPARING ATTACK VECTORS

SETTING UP AN ENVIRONMENT

MANUAL CODE REVIEW OF THE CODE

ASSESSMENT OF RUST SECURITY ISSUES

ASSESSMENT OF NEAR SECURITY ISSUES

ASSESSMENT OF ARITHMETIC ISSUES

BUSINESS LOGIC VULNERABILITY ASSESSMENT

ONCHAIN TESTING USING NEAR WORKSPACES

BEST PRACTICES AND CODE QUALITY

CHECKING FOR CODE REFACTORING/SIMPLIFICATION POSSIBILITIES

ARCHITECTURE IMPROVEMENT SUGGESTIONS

PREPARING POCS AND/OR TESTS FOR EACH CRITICAL/HIGH/MEDIUM ISSUES

Severity Breakdown

01. Likelihood Ratings

Likely: The vulnerability is easily discoverable and not overly complex to exploit.
Possible: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Rare: The vulnerability is either very difcult to discover or complex to exploit, or both.
This matrix provides a nuanced view, taking into account both the ease of discovering a vulnerability and the complexity involved in exploiting it.

02. Impact

Severe: Exploitation could result in critical loss or compromise, such as full system control, substantial financial loss, or severe reputational damage.
Moderate: Exploitation may lead to limited data loss, partial compromise, moderate financial impact, or noticeable degradation of services.
Negligible: Exploitation has minimal impact, such as minor data exposure without significant consequences or slight inconvenience without substantial disruption.

03. Severity Ratings

Critical: Assigned to vulnerabilities with severe impact and a likely likelihood of exploitation.
High: For vulnerabilities with either severe impact but only a possible likelihood, or moderate impact with a likely likelihood.
Medium: Used for vulnerabilities with severe impact but a rare likelihood, moderate impact with a possible likelihood, or negligible impact with a likely likelihood.
Low: For vulnerabilities with moderate impact and rare likelihood, or negligible impact with a possible likelihood.
Informational: The lowest severity rating, typically for vulnerabilities with negligible impact and a rare likelihood of exploitation.

CRITICAL **HIGH** **MEDIUM** Low Informational

Likelihood Matrix:

Attack Complexity \ Discovery Ease	Obvious	Concealed	Hidden
Complex	Possible	Rare	Rare
Moderate	Likely	Possible	Rare
Straightforward	Likely	Possible	Possible

Likelihood/Impact Matrix:

Likelihood \ Impact	Severe	Moderate	Negligible
Likely	**CRITICAL**	**HIGH**	**MEDIUM**
Possible	**HIGH**	**MEDIUM**	Low
Rare	**MEDIUM**	Low	Informational

Findings Summary

01. Remediation Complexity: This measures how difcult it is to fx the vulnerability once it has been identifed.

Simple: Patches or fixes are readily available and easily implemented.

Moderate: Requires some time and resources to remediate, but well within the capabilities of most organizations.

Difficult: Remediation requires significant resources, specialized skills, or substantial changes to systems or architecture.

02. Status: This measures how difcult it is to fx the vulnerability once it has been identifed.

Not Fixed: Indicates that the vulnerability has been identifed but no remedial action has been taken yet. This status is crucial for newly discovered vulnerabilities or those awaiting prioritization.

Fixed: This status is applied when the vulnerability has been successfully remediated. It implies that appropriate measures (like patching, confguration changes, or architectural modifcations) have been implemented to resolve the issue.

Acknowledged: This status is used for vulnerabilities that have been recognized, but for various reasons (such as risk acceptance, cost, or other business decisions), have not been fxed. It indicates that the risk posed by the vulnerability is known and has been consciously accepted.

Scheduled: This status indicates that the vulnerability has been acknowledged and a plan is in place to fix it in the future. It signifies that while remediation hasn't yet occurred, the issue has been prioritized and is part of the planned development roadmap.

Finding	Impact	Likelihood	Severity	Remediation Complexity	Remediation Status
GUV-1: Supply Withdrawal Mechanism DoS	Moderate	Likely	HIGH	Moderate	Fixed
GUV-2: Reliance On Bot Oracle Updates	Negligible	Rare	Informational	Simple	Acknowledged
GUV-3: Possible Self-liquidiation	Negligible	Rare	Informational	Simple	Acknowledged

Findings Details

GUV-1 Supply Withdrawal Mechanism DoS - High

The Supply Withdrawal process operates on a queue mechanism. First user creates a withdrawal request which is added to the queue. Then, separate and permissionless function (**execute_next_supply_withdrawal_request**) is processing those requests in order of their creation. There is a check in callback that verifies if the token transfer was successful. In case it wasn't, it essentially doesn't modify the queue so that next execute_next_supply_withdrawal_request call will try to process the same request. The assumption being that this transfer fails if Market contract doesn't have enough tokens to transfer. There is another possible case, that a malicious user can use to damage the protocol (and also create a ransom-like situation). NEP141 tokens allow users to unregister (withdraw their storage deposit). When that happens, their balance is completely removed from the token contract and transferring tokens to that user fails. A malicious user can:

- Supply tokens
- Unregister from the token contract
- Create a supply withdrawal request

At that point, when withdrawal queue reaches that malicious user's request - the whole functionality stops functioning as no request can be processed at that time. This is a somewhat recoverable state, if token used for supply is following the default storage implementation, which allows anyone to register anyone else. However, that means someone else would need to cover storage fees for a malicious user in a token contract that this malicious user can then withdraw to himself by unregistering again, which can create a "vicious cycle" and essentially means malicious user would be requesting a ransom to allow supply withdrawal to function.

impl_market_external:execute_next_supply_withdrawal_request:market/src/impl_helper.rs

```
PromiseOrValue::Promise(
            self.configuration
              .borrow_asset
             .transfer(
               withdrawal_resolution.account_id.clone(),
               withdrawal_resolution.amount_to_account,
              )
              .then(
               self_ext!(Self::GAS_AFTER_EXECUTE_NEXT_WITHDRAWAL)

.execute_next_supply_withdrawal_request_01_finalize(withdrawal_resolution),
            ), )}
```

impl_helper:execute_next_supply_withdrawal_request_01_finalize:market/src/impl_helper.rs

```
PromiseResult::Failed => {
            // Withdrawal failed: unlock the queue so they can try again.
            // This occurs when the contract does not control enough of
            // the borrow asset to fulfill the withdrawal request. That is
            // to say, it has distributed all of the funds to current
            // borrows.

            env::log_str("The withdrawal request cannot be fulfilled at this time. Please try
again later.");
           self.withdrawal_queue.unlock();
            if let Some(mut supply_position) =
              self.supply_position_guard(withdrawal_resolution.account_id.clone())
            {
              // This should not do very much since we also accumulate
              // yield in the initial function call of execute_next_supply_withdrawal_request()
              let proof = supply_position.accumulate_yield();
              let mut amount = withdrawal_resolution.amount_to_account;
              amount.join(withdrawal_resolution.amount_to_fees);
              supply_position.record_deposit(proof, amount, env::block_timestamp_ms());
            } }
```

Impact:

Denial of Service condition on the supply withdrawal functionality, along with potential funds lost for users or protocol operators when trying to mitigate the Denial of Service.

Recommendation

Implement a mechanism that would utilize internal accounting to estimate whether a token transfer failed due to malicious user unregistering or due to missing token supply. Should the situation be determined as a result of malicious user's action, the Withdrawal Request should be removed from the queue.

Remediation - Fixed

The Templar team has fixed the issue.

GUV-2 Reliance On Bot Oracle Updates - Informational

The contract uses Pyth Oracle's **list_ema_prices_no_older_than** function to retrieve the prices. The functionalities using the price data will fail if the Oracle will not be able to provide prices, which will happen if the most recent available price is older than the configured threshold. It must be noted that currently there is a bot service running on mainnet that seems to be updating the prices, however, that might not always be the case. Should the bot be disabled or otherwise stop working, the core functionalities of the contract will not be possible to execute successfuly.

Impact:

Possible Denial of Service leading to a potential economic repercussions due to inability to execute actions related to the price data.

Recommendation

It is recommended to implement a fail-safe mechanism that would be able to use Pyth's **update_price_feeds** endpoint in order to update the prices manually, should the bot stop being reliable.

Remediation - Acknowledged

The Templar team has acknowledged the issue by stating that there are already measures implemented preventing that scenario.

GUV-3 Possible Self-liquidiation - Informational

It was observed that the contract does not implement any mechanism that would prevent users from self-liquidating. Such an action does not directly damage the protocol and requires a liquidable user to win a race with other liquidators. Should this situation happen, this scenario translates to a repayment with a discount rather than a liquidation.

It must be noted that ultimately this scenario is not completely preventable, as users are free to operate using many different AccountIds.

Impact:

Cheaper loan repayment.

Recommendation

It is recommended to add a check that would at least prevent a self-liquidation.

Remediation - Acknowledged

The Templar team has acknowledged the issue but it won't be fixed.