Secure Code Review

Technical Report

Texture Labs

12 November 2024
Version: 1.1

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.1
File Name:	Kudelski_Security_Texture_Labs_Secure_Code_Review_v1.1.pdf
Publication Date:	12 November 2024
Confidentiality Level:	Confidential
Document Owner:	Jamshed Memon
Document Authors:	Jamshed Memon, Luca Dolfi
Document Recipients:	Dmitry Potashnikov
Document Status:	Approved

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Texture Labs (“the Client”) engaged Kudelski Security (“Kudelski”, “we”) to perform the Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

The review took place between 09 September 2024 and 18 October 2024, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

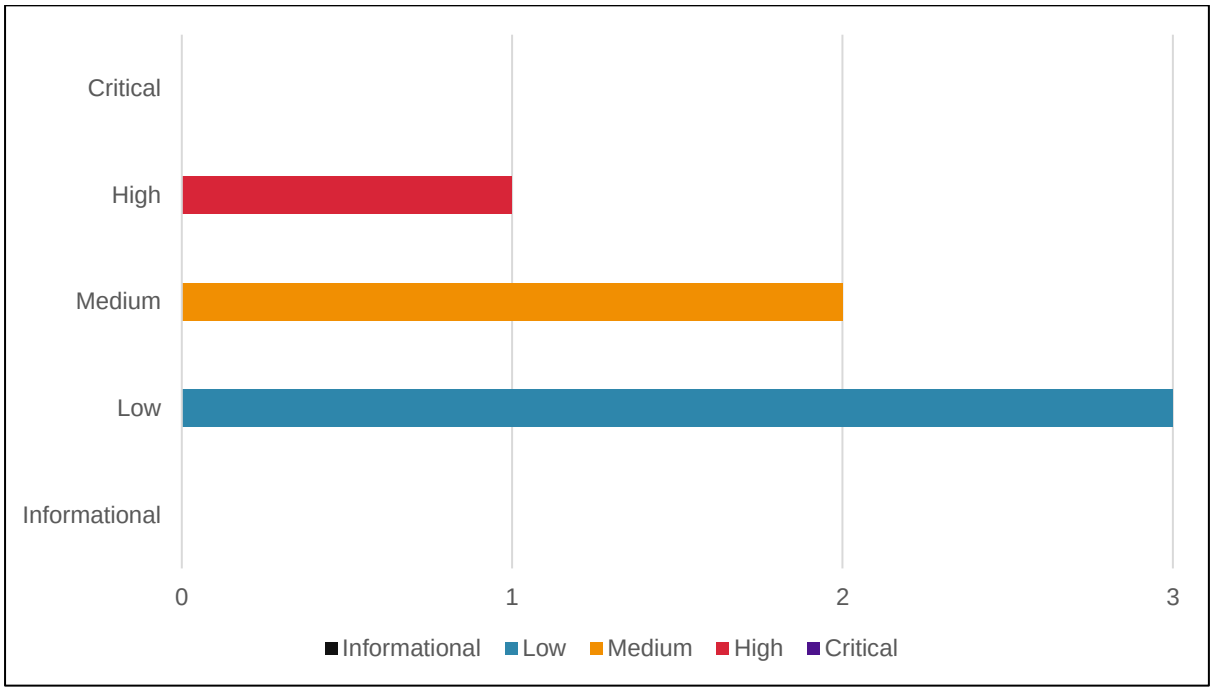
After discussion with the client, a second round of re-review was performed on November 12th based on an updated version of the code.

Key Findings

The following are the major themes and issues identified during the testing period.

These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Lack of input validation in IRM model
- Lack of input validation in ReserveConfig
- Risk of Centralization



Findings ranked by severity.

1. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

1.1 Context

The Texture Labs offers a flexible platform for lending and borrowing of cryptocurrencies. Key features include multi-currency pools, cross-pool collateral, price flexibility, yield curve customization, partial liquidations, vaults, P2P integration, protected collateral, and rewards.

1.2 Scope

The scope consisted in specific Rust files and folders located at:

- `source_code/audit-pack-v2.7/curvy/program`
- `source_code/audit-pack-v2.7/price-proxy/program`
- `source_code/audit-pack-v2.7/superlendy/program`

Additionally, a newer version of the repositories `curvy-0.1.2`, `superlendy-0.1.4` and `superlendy-0.1.5` were provided by The Client for a re-review of the code on November 7th ,November 8th and November 12th .

1.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The developers have made a careful and in-depth analysis of their project.
- Tests were also provided as part of the project, which is convenient for better understanding how the library works and useful for elaborating scenarios and validating findings.
- Finally, we had regular and very enriching technical exchanges on various topics.

1.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in Chapter Vulnerability Scoring System of this document.

2. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	SEVERITY	TITLE	STATUS
KS–TF–F–01	High	Insufficient Input Validation in ReserveConfig Parameters	Resolved
KS–TF–F–02	Medium	Potential Input Validation Vulnerability in check_params Function	Resolved
KS–TF–F–03	Medium	Risk of Centralization	Acknowledged
KS–TF–F–04	Low	Unchecked init for structs	Acknowledged
KS–TF–F–05	Low	Zero-Amount Operations in Withdrawal and Fee Claim Functions	Resolved
KS-TF-F-06	Low	Missing Minimum Value Validation for ReserveConfig Parameters	Acknowledged

Findings overview.

2.1 KS–TF–F–01 Insufficient Input Validation in ReserveConfig Parameters

Severity	Impact	Likelihood	Status
High	High	Medium	Resolved

Description

The validate function in the `ReserveConfig` struct does not implement lower bound checks for several critical parameters. This allows these parameters to be set to 0 or extremely low values, which could severely impact the protocol's functionality and economics. The affected parameters include:

- `partly_unhealthy_ltv_bps`
- `fully_unhealthy_ltv_bps`

Impact

Setting these parameters to 0 or extremely low values could lead to the following issues:

1. `partly_unhealthy_ltv_bps`: Trigger unnecessary liquidations for safe positions
2. `fully_unhealthy_ltv_bps`: Allow full liquidations of relatively healthy positions

Evidence

```rust
pub fn validate(&self) -> LendyResult<()> {

    if self.partly_unhealthy_ltv_bps > 10_000 {

        msg!("partly_unhealthy_ltv must be in range [0, 100] %");

        return Err(SuperLendyError::InvalidConfig);
    }
    // ... other checks ...

}
```

state/reserve.rs

Affected Resources

- `superlendy/program/src/state/reserve.rs/` lines 939

Recommendation

Implement additional checks in the `validate` function to ensure that these fields are not zero or very small. Consider setting a minimum value for these fields to prevent them from being set to zero or a very small value.

Client Feedback

Code ensures that `partly_unhealthy_ltv_bps & fully_unhealthy_ltv_bps` values are not `0`. Input validation of parameters is the responsibility of the curators. However, Texture Finance has implemented a timelock for delayed parameter changes in the reserves.

2.2 KS–TF–F–02 Potential Input Validation Vulnerability in check_params Function

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The `check_params` function in the curve.rs file checks the parameters of the curve, but it doesn't ensure that `x_step` and `y_count` are non-zero and that `max_x` is greater than `x0`. This could lead to unintended behavior such as a flat curve or a curve with insufficient points.

Impact

- `x_step`: This parameter defines the step size along the x-axis of the curve. If `x_step` is zero, it would result in a flat curve. This means that regardless of the utilization rate, the interest rate would remain constant. This could lead to an imbalance in the lending pool as the interest rate would not increase with higher utilization, potentially leading to a lack of liquidity.
- `y_count`: This parameter defines the number of points along the y-axis (usually representing the interest rate) of the curve. If y_count is zero, it would result in a curve with no points beyond x0, meaning that the interest rate would not change regardless of the utilization rate. This could also lead to an imbalance in the lending pool as the interest rate would not adjust to the demand for loans.
- `max_x`: This parameter defines the maximum value on the x-axis of the curve. If `max_x` is not greater than `x0`, it would result in a curve that doesn't progress forward. This means that the interest rate would not increase even with a high utilization rate, potentially leading to a lack of liquidity in the lending pool.

Evidence

```
pub fn check_params(params: &CurveParams) -> CurvyResult<()> {
    //...otherchecks...
    let max_x = Decimal::from_i128_with_scale(params.x0 as i128, params.decimals as
u32)?
        .checked_add(
            Decimal::from_i128_with_scale(params.x_step as i128, params.decimals as
u32)?
                .checked_mul(Decimal::from(params.y_count as u64))?,
        )?;
    let u32_max = Decimal::from_i128_with_scale(u32::MAX as i128, params.decimals
as u32)?;
    if max_x > u32_max {
        msg!("Provided x0, x_step and y_count results in too big maximum X value
{}. It should not exceed {}", max_x, u32_max);
        return Err(CurvyError::InvalidParams);
```

curvy/program/src/state/curve.rs

Affected Resources

- `curvy/program/src/state/curve.rs` line 135

Recommendation

Implement additional checks in the `check_params` function to ensure that `x_step` and `y_count` are non-zero and that `max_x` is greater than `x0`. This will prevent the creation of a flat curve, a curve with insufficient points, and a curve that doesn't progress forward.

2.3 KS–TF–F–03 Risk of Centralization

Severity	Impact	Likelihood	Status
Medium	High	Low	Acknowledged

Description

The curator structure in this code plays a critical role in managing and overseeing various aspects of the system. Its primary purposes include:

1. Pool Management: The curator has the authority to create and configure lending pools, setting key parameters that govern lending and borrowing activities.
2. Reserve Oversight: Curators manage reserves, which are the backbone of the lending protocol, controlling the assets available for lending and borrowing.
3. Vault Administration: Curators have the power to create and manage vaults, which may be used for additional yield generation or other protocol-specific functions.
4. Fee Management: The curator structure includes control over fee collection and distribution, a crucial aspect of the protocol's economic model.

Given these significant responsibilities, the curator holds substantial influence over the protocol's operation and user funds.

However, the current implementation of the curator structure allows for potentially centralization of control. While the structure provides separate fields for different authorities (`pools_authority`, `vaults_authority`, `fees_authority`, `owner`), these can all be set to the same address. Importantly, the curator account is not implemented as a multisig, meaning each authority is controlled by a single key.

Impact

1. Single Point of Failure: If all authorities are set to the same address, compromising this single account could grant an attacker full control over all aspects of the curator's responsibilities.
2. Governance Manipulation: A centralized curator could unilaterally make critical decisions affecting the entire protocol, bypassing expected checks and balances.
3. Asset Risk: Centralized control over pools, vaults, and fees puts user funds at heightened risk if the controlling account is compromised or acts maliciously.

Evidence

```rust
pub struct Curator {
    pub owner: Pubkey,
    pub pools_authority: Pubkey,
    pub vaults_authority: Pubkey,
    pub fees_authority: Pubkey,
    // ... other fields ...
}

pub(super) fn alter_curator(&self, params: CuratorParams) -> LendyResult<()> {
    // ... verification code ...
    unpacked_curator.owner = params.owner;
```

```
    unpacked_curator.pools_authority = params.pools_authority;
    unpacked_curator.vaults_authority = params.vaults_authority;
    unpacked_curator.fees_authority = params.fees_authority;
    // ... other fields ...
    Ok(())
}
```

superlendy/program/src/state/curator.rs

Affected Resources

- `superlendy/program/src/state/curator.rs` line 15
- `superlendy/program/src/processor/curator.rs` line 28

Recommendation

Replace single-address authorities with multi-sig structures, requiring multiple parties for critical actions.

Client Feedback

Decentralization of the curator is the responsibility of the curator admin. Texture Finance recommends using a multisig wallet for the curator, but it is not enforced.

2.4 KS–TF–F–04 Unchecked init for structs

Severity	Impact	Likelihood	Status
Low	Medium	Low	Acknowledged

Description

The `init_unckecked` method of trait `PodAccount` is implemented for several structs in the code to perform initialization of the contract. However, all of the implementations do not perform input validation. This method directly sets fields from user-provided parameters without ensuring the correctness or uniqueness of the input data. Examples of these fields include names, addresses, URLs, symbols, and configuration values. The absence of validation opens the system to potential inconsistencies, duplication, and invalid states. For example:

- In `Pool::init_unckecked`, multiple pools can be created with the same name since there is no check for name uniqueness.
- In `Reserve::init_unckecked`, parameters like liquidity, collateral, and configuration could be set inconsistently, leading to invalid states.
- Other structures such as `Curator`, `Curve`, and `Position` also set various critical fields from the user-provided parameters, such as public keys, names, and addresses, without ensuring their validity.

Impact

The lack of input validation in `init_unckecked` allows the creation of duplicate entries (e.g., pools with the same name) and inconsistent internal states (e.g., mismatched liquidity and collateral in Reserve). This can lead to operational confusion, misrouting of funds, and inaccurate financial calculations, potentially causing system instability or financial loss.

Evidence

```
    fn init_unckecked(
         &mut self,
         (params, curator_key): Self::InitParams,
     ) -> Result<(), Self::InitError> {
         let Self {
             discriminator,
             version,
             visible,
             _flags,
             curator,
             name,
             market_price_currency_symbol,
             _padding,
         } = self;

        *discriminator = *POOL_DISCRIMINATOR;
         *version = Self::VERSION;
         *name = params.name;
```

```
        *market_price_currency_symbol = params.market_price_currency_symbol;
        *curator = curator_key;
        *visible = 0;
        *_padding = Zeroable::zeroed();
        *_flags = Zeroable::zeroed();

        Ok(())
    }
```

superlendy/program/src/state/pool.rs line 72

```
    fn init_unckecked(&mut self, params: Self::InitParams) -> Result<(),
Self::InitError> {
        let Self {
            discriminator,
            version,
            reserve_type,
            mode,
            _flags,
            last_update,
            pool,
            liquidity,
            collateral,
            config,
            reward_rules,
            _padding,
        } = self;

        *discriminator = *RESERVE_DISCRIMINATOR;
        *version = Self::VERSION;
        *last_update = params.1;
        *reserve_type = params.0.reserve_type;
        *mode = params.0.mode;
        *pool = params.0.pool;
        *liquidity = params.0.liquidity;
        *collateral = params.0.collateral;
        *config = params.0.config;

        *reward_rules = Zeroable::zeroed();
        *_padding = Zeroable::zeroed();
        *_flags = Zeroable::zeroed();

        Ok(())
    }
```

superlendy/program/src/state/reserve.rs line 134

Affected Resources

- `curvy/program/src/state/curve.rs` line 178
- `price-proxy/program/src/state/price_feed.rs` line 295
- `superlendy/program/src/state/curator.rs` line 72
- `superlendy/program/src/state/pool.rs` line 72
- `superlendy/program/src/state/position.rs` line 96
- `superlendy/program/src/state/reserve.rs` line 134
- `superlendy/program/src/state/texture_config.rs` line 71

Recommendation

Implement strict input validation to ensure uniqueness (e.g., pool names) and consistency across parameters like liquidity and collateral. Replace unchecked initialization with safer, validated methods, and provide clear error messages to help users correct invalid inputs.

Client Feedback

After discussion with The Client, we agreed that the security impact on Pool and Reserve is further reduced by implementing additional checks in the UI. Moreover, presenting the data more clearly will prevent possible user confusion. It was recognised that modifying the contract functionality to limit struct initialization would negatively affect the user experience.

2.5 KS–TF–F–05 Zero-Amount Operations in Withdrawal and Fee Claim Functions

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

In the `reserve.rs` file, the functions `withdraw_liquidity`, `claim_curator_performance_fees`, and `claim_texture_performance_fees` currently allow operations with a zero amount. This means users or automated systems can initiate withdrawals or fee claims for 0 tokens, which results in state changes and emits events without any actual value transfer. The relevant functions are:

1. `withdraw_liquidity`: Allows withdrawing liquidity from a reserve.
2. `claim_curator_performance_fees`: Enables claiming of performance fees for curators.
3. `claim_texture_performance_fees`: Permits claiming of performance fees for the protocol (texture).

Impact

1. Misleading Event Emissions: Zero-amount operations in these functions still emit events, potentially confusing monitoring systems or analytics platforms that track reserve activity.
2. State Bloat: Repeated zero-amount operations could lead to unnecessary state changes and bloat in the reserve contract's storage
3. Unnecessary Gas Consumption: Users or automated systems might unknowingly or maliciously create transactions that consume gas without any meaningful operation in the `reserve.rs` file.

Evidence

```
pub fn withdraw_liquidity(&self, lp_amount: u64) -> LendyResult<()> {
    msg!("withdraw_liquidity ix");
    // ... [code omitted for brevity]
    let liquidity_amount = unpacked_reserve.withdraw_liquidity(lp_amount)?;
    // ... [rest of the function]

    Ok(())
}
 pub fn claim_curator_performance_fees(&self) -> LendyResult<()> {
    msg!("claim_curator_performance_fees ix");
    // ... [code omitted for brevity]
    let fee_amount = unpacked_reserve.liquidity.claim_curator_performance_fee()?;
    // ... [rest of the function]
    Ok(())
}

pub fn claim_curator_performance_fees(&self) -> LendyResult<()> {
    msg!("claim_curator_performance_fees ix");
```

```
    // ... [code omitted for brevity]
    let fee_amount = unpacked_reserve.liquidity.claim_curator_performance_fee()?;
    // ... [rest of the function]
    Ok(())
}
```

/superlendy/program/src/processor/reserve.rs

Affected Resources

- `/superlendy/program/src/processor/reserve.rs` line 348

Recommendation

Implement checks in all three functions within reserve.rs to ensure that the amount being withdrawn or claimed is greater than zero.

2.6 KS–TF–F–06 Missing Minimum Value Validation for ReserveConfig Parameters

Severity	Impact	Likelihood	Status
Low	Low	Low	Acknowledged

Description

The validate function in the ReserveConfig struct does not implement minimum value checks for the following parameters:

- liquidation_bonus_bps
- curator_borrow_fee_rate_bps
- curator_performance_fee_rate_bps
- max_borrow_utilization_bps
- max_withdraw_utilization_bps

Impact

Without minimum value validation, these parameters could potentially be set to zero or very low values (where not explicitly checked), leading to unexpected behavior or vulnerabilities in the system.

1. `liquidation_bonus_bps`: No incentive for liquidators, leading to accumulation of bad debt
2. `curator_borrow_fee_rate_bps`: Loss of protocol revenue from borrowing activity
3. `curator_performance_fee_rate_bps`: Loss of protocol revenue from yield generation
4. `max_borrow_utilization_bps`: Severe restriction or complete prevention of borrowing activity
5. `max_withdraw_utilization_bps`: Prevention of withdrawals, effectively locking user funds

Evidence

```
if self.liquidation_bonus_bps < 0 || self.liquidation_bonus_bps > 5_000 {
    msg!("Liquidation bonus must be in range [0, 50] %");
    return Err(SuperLendyError::InvalidConfig);
}

if self.fees.curator_borrow_fee_rate_bps < 0 ||
self.fees.curator_borrow_fee_rate_bps >= 200 {
    msg!("curator_borrow_fee must be in range [0, 2] %");
    return Err(SuperLendyError::InvalidConfig);
}
```

```
if self.fees.curator_performance_fee_rate_bps < 0 ||
self.fees.curator_performance_fee_rate_bps > 3000 {
    msg!("curator_performance_fee_rate_bps must be in range [0, 30] %");
    return Err(SuperLendyError::InvalidConfig);
}
```

state/reserve.rs

Affected Resources

- `superlendy/program/src/state/reserve.rs/` lines 939

Recommendation

Implement additional checks in the `validate` function to ensure that these fields are not zero or very small. Consider setting a minimum value for these fields to prevent them from being set to zero or a very small value.

Client Feedback

Input validation of parameters is the responsibility of the curators. However, Texture Finance has implemented a timelock for delayed parameter changes in the reserves.

3. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.

3.1 Kickoff

The Kudelski Security Team set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

3.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

3.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)

- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- assessment of smart contract admin centralization
- reentrancy attacks and external contracts interactions
- verification of compliance with existing standards such as ERC20 or PSP34
- unsafe arithmetic operations such as overflow and underflow
- verification dependance on timestamp
- access control verification to ensure that only authorized users can call sensitive functions.

3.4 Reporting

Kudelski Security delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

3.5 Verify

After the preliminary findings have been delivered, we verify the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

4. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT / LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	Medium	High	High
MEDIUM	Low	Medium	High
LOW	Low	Low	Medium

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

5. CONCLUSION

The objective of this Secure Code Review was to evaluate whether there were any vulnerabilities that would put the Texture Labs or its customers at risk.

The Kudelski Security Team identified 6 security issues: 1 high risk, 2 medium risks and 3 lower risk. On average, the effort needed to mitigate these risks is estimated as low.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Implement the additional validation checks in `ReserveConfig`, `check_param` and `init_unckecked`
- Reduce the risk of centralization for the curator account.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Texture Labs for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

DOCUMENT RECIPIENTS

NAME	POSITION	CONTACT INFORMATION
Igor Burlakov	Director	igor@texture.finance
Dmitry Potashnikov	Manager	dima@texture.finance

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Jean-Sebastien Nahon	Application and Blockchain Security Practice Manager	jean-sebastien.nahon@kudelskisecurity.com
Ana Acero	Project Manager/ Operations Coordinator	ana.acero@kudelskisecurity.com

DOCUMENT HISTORY

VERSION	DATE	STATUS/ COMMENTS
V1.0	18 October 2024	Draft
V1.1	12 November 2024	Final