OFFSIDE LABS

RateX Mooncake

Smart Contract Security Assessment

October 2025

Prepared for:

Rate Labs Limited

Prepared by:

Offside Labs

Yao Li

Ripples Wen

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.


https://offside.io/


https://github.com/offsidelabs


https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Mooncake* smart contracts, starting on August 21th, 2025 and concluding on October 14th, 2025.

Project Overview

RateX V2 is a cutting-edge leveraged trading protocol on Solana, offering yield-bearing and leveraged tokens through a dual-token system of Funding Tokens and Leverage Tokens. It features real-time funding rate settlement, multi-oracle integration, and a hybrid rebalancing system. The protocol incorporates advanced risk management tools like emergency pauses and circuit breakers, alongside a differentiated fee structure for swaps and management.

Audit Scope

The assessment scope contains mainly the smart contracts of the mooncake program for the *Mooncake* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- Mooncake
  - Codebase: https://github.com/RateX-Protocol/mooncake-contracts
  - Commit Hash:
    - audit: 0cf6d89a9780f4900a5c7ee30d1df817f8169417
    - main-vault-token: ce10e6206148c6159980bf9192e20f785b7d579f

We listed the files we have audited below:

- Mooncake
  - programs/ratex-mooncake/src/*.rs

Findings

The security audit revealed:

- 1 critical issue
- 3 high issues
- 6 medium issues
- 4 low issues
- 7 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Risk Free Oracle Arbitrage in Direct Mode	Critical	Fixed
02	Last Funding Rate Should be Updated After Operations	High	Fixed
03	Oracle Price Should be Calculated After Funding Settlement	High	Fixed
04	Misaligned Management Fee Calculation	High	Fixed
05	Incorrect Rebalance Fee Update	Medium	Fixed
06	Funding Settlement Should be Performed in UpdateFundingRateBounds Instruction	Medium	Fixed
07	Funding Settlement Should be Performed in Final Epoch	Medium	Fixed
08	ManualPriceAccount Vulnerable to Unauthorized Control	Medium	Fixed
09	Oracle Price Calculated by Uninitialized Value	Medium	Fixed
10	Lack of Oracle Validation and Accuracy	Medium	Acknowledged
11	Incorrect Rounding Direction When Insufficient Pool Balance	Low	Fixed
12	Inconsistent Order of HandleSwapFeeDistribution	Low	Fixed
13	Incorrect Division Math Library	Low	Fixed
14	Incorrect Auto Compounding Factor Calculation	Low	Fixed
15	Lack of Fee Charged Events	Informational	Fixed
16	Hardcoded Value	Informational	Fixed
17	Unchecked Value in Market Initialization	Informational	Fixed
18	Precision Loss by Pre Calculated Ratio	Informational	Fixed
19	Avoid Using the as Operator	Informational	Acknowledged
20	Authority Can Not Be Updated for Some Accounts	Informational	Acknowledged
21	Token2022 Support	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Risk Free Oracle Arbitrage in Direct Mode

Severity: Critical	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The protocol operates in two modes: Direct Mode and Request Mode.

1. Direct Mode: Users can interact with the protocol without additional limitations.
2. Request Mode: Transactions must be signed and sent by protocol keepers for execution.

In Direct Mode, the lack of permission checks for oracle updates allows an attacker to execute the following sequence in a single transaction:

1. Buy LV.
2. Update Oracle.
3. Sell LV.

If the profit from the oracle price difference exceeds the swap fees, the attacker can achieve risk-free profit, resulting in a loss for liquidity providers.

Additionally, even in Request Mode, there is a possibility for similar arbitrage due to delayed oracle updates, which could allow attackers to exploit price discrepancies non-atomically.

Impact

1. An attacker can exploit Direct Mode to gain risk-free profit whenever the oracle price difference covers the swap fees.
2. Liquidity providers bear the losses caused by this arbitrage.
3. In Request Mode, delayed oracle updates could still enable attackers to execute similar exploits, albeit in a non-atomic manner.

Recommendation

1. Implement a delay for claiming FV after a purchase, preventing immediate selling and reducing arbitrage opportunities.
2. Ensure protocol keepers validate user-side transaction bundles and reject unexpected or malicious actions.

Mitigation Review Log

RateX Team:

Direct mode will not be used in production.

For Request mode, users first sign the transaction, then it is co-signed by a keeper controlled by the backend. Before signing, the backend compares the transaction's updated oracle price with the current on-chain state. If the backend detects that the trade would yield profits greater than the fees, the keeper will refuse to sign.

Offside Labs:

Recommend to add a conditional compilation attribute like `#cfg(not(mainnet))` for direct mode.

RateX Team:

Decide to remove direct mode.

Fixed in commit 109309e29cfc2baf2602af30b0a2659bbe25e297.

4.2 Last Funding Rate Should be Updated After Operations

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `last_funding_rate` field represents the funding rate after the most recent protocol action. However, it is currently updated in the `funding_settlement` function, which is executed at the beginning of each instruction.

This implementation results in the `last_funding_rate` being calculated based on the state prior to the last action, rather than the state following the most recent operation.

Impact

The misalignment of the `last_funding_rate` can lead to incorrect pricing of FT and LT.

Recommendation

Update the `last_funding_rate` at the end of each instruction instead of at the beginning.

Mitigation Review Log

Fixed in commit 358c7143e779a5f52b2be320e74bfe1a2263099c.

4.3 Oracle Price Should be Calculated After Funding Settlement

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `add_liquidity` and `remove_liquidity` instructions, the `get_oracle_prices` function is called before the `funding_settlement` process. The oracle prices calculated by `get_oracle_prices` are then used in subsequent calculations.

However, the `get_oracle_prices` function does not account for the management fee applied during `funding_settlement` for LT, resulting in a slightly inflated price for LT.

Impact

The price of LT returned by the `get_oracle_prices` function in the `add_liquidity` and `remove_liquidity` instructions will be slightly higher than it should be, as it does not factor in the management fee deducted during the `funding_settlement` .

Recommendation

Ensure that the `get_oracle_prices` function is called after the `funding_settlement` process

Mitigation Review Log

Fixed in commit ba86bc18d3307628816ad18aebbb5a97717415da.

4.4 Misaligned Management Fee Calculation

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `funding_settlement` function charges a management fee. Below is the relevant code snippet:

```
511         let deduct_unit_amount = mul_div_u128(
512             ONE_U64.cast::<u128>()?,  // Use ONE_U64 for standard
    ↪ precision
513             fee_adjustment.cast::<u128>()?,
514             ONE_U128
515         )?.cast::<u64>()?;
516
517         // Calculate management fee based on total assets under
              ↪ management
518         // Management fee = fee_adjustment * asset_vault_balance / 1e9
519         // The asset_vault_balance is already in native underlying token
              ↪ decimals
520         let management_fee_native = mul_div_u64(
521             fee_adjustment as u64,
522             asset_vault_balance,
523             ONE_U64
524         )?;
```

programs/ratex-mooncake/src/state/market.rs#L511-L524

The `deduct_unit_amount` is calculated using `rate * ONE` while the `unit_underlying_amount`, representing the actual value per share, is typically less than one. This mismatch causes the deducted ratio to be higher than the intended management fee rate, leading to overcharging. Furthermore, the management fee for FT is based on their net value, while LT are charged based on `unit_underlying_amount` and FT. As a result, LT holders end up paying more management fees than they should.

Another issue is seen in the `management_fee_native` calculation. The function uses the entire `asset_vault_balance`, which includes the `protocol_fee`. Since the `protocol_fee` is part of the protocol's own balance, it should be excluded from the fee calculation. Including it leads to unintentional overcharging, as fees are being applied to amounts that should not be subject to the management fee.

Impact

The higher deducted ratio associated with the misaligned `deduct_unit_amount` calculation causes LT holders to pay more than the expected management fee rate.

Similarly, by including the `protocol_fee` in the `management_fee_native` calculation, the total fee charged becomes inflated.

Recommendation

First, replace the use of `ONE_U64` with `unit_underlying_amount` when calculating `deduct_unit_amount`.

Second, modify the `management_fee_native` calculation to exclude the `protocol_fee` from the `asset_vault_balance`.

Mitigation Review Log

Fixed in commit 91f35965b7ede5fc31fd99b23ad0e318e90ba763 and ec5404b114f5f7b53f587efb3ae4b4d093b87290.

4.5 Incorrect Rebalance Fee Update

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `collect_rebalance_fee` function, the rebalance fee is charged as follows:

```
14      // Calculate rebalance fee based on total assets under management
15      let rebalance_fee_native = mul_div_u64(
16          REBALANCE_FEE_RATE as u64,
17          asset_vault_balance,
18          ONE_U64
19      )?;
20
21      // Convert to standard precision (1e9) and add to protocol fee
22      let rebalance_fee_standard =
            ↪ market_state.scale_to_standard(rebalance_fee_native)?;
23      market_state.protocol_fee =
    ↪ market_state.protocol_fee.safe_add(rebalance_fee_standard)?;
```

programs/ratex-mooncake/src/controller/rebalance_trading.rs#L14-L23

The `asset_vault_balance` includes the unclaimed protocol fee, which should not be considered in the rebalance fee calculation.

Additionally, the function does not update the `unit_underlying_amount` and the prices of FT and LT, which leads to fees not being deducted from net value accounting.

Impact

Including the `protocol_fee` in the `rebalance_fee_native` calculation inflates the total fee charged. Failing to adjust net value accounting results in double counting of fees.

Recommendation

Modify the fee calculation to exclude unclaimed protocol fees from `asset_vault_balance`. Ensure that net value accounting for `unit_underlying_amount`, FT, and LT prices is updated to reflect the deducted fees accurately.

Mitigation Review Log

RateX Team:

Fixed in commit 91f35965b7ede5fc31fd99b23ad0e318e90ba763.

Offside Labs:

After `collect_rebalance_fee` , only LT price drops, leading to incorrect prices.

Fixed in commit 7d947b7ed3fdcdd66133886b0190d03ec2431d16.

4.6 Funding Settlement Should be Performed in UpdateFundingRateBounds Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `update_funding_rate_bounds` instruction is responsible for updating the parameters of the funding rate bounds, which can influence the funding rate. However, the instruction does not perform a `funding_settlement` or update the `last_funding_rate` , leading to potential inconsistencies.

Impact

If the `funding_settlement` is not performed and the `last_funding_rate` is not updated, the `last_funding_rate` may become stale. This can result in incorrect calculations or outdated values until the next refresh.

Recommendation

Include `funding_settlement` and updating `last_funding_rate` in the instruction.

Mitigation Review Log

Fixed in commit 44c4ede091f489b91f7e0459901d38d1b34a698c.

4.7 Funding Settlement Should be Performed in Final Epoch

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the final epoch, the funding settlement is not executed due to concerns that oracle prices may be stale. However, the management fee is still applied based on the FT price calculation.

```
394     // Calculate net funding rate: funding_rate - management_fee_rate
395     let net_funding_rate = if self.status == MarketStatus::FinalEpoch {
396             -management_fee_rate
397     } else {
398             self.last_funding_rate.safe_sub(management_fee_rate)?
399     };
```

programs/ratex-mooncake/src/state/market.rs#L394-L399

Impact

The missing funding settlement in final epoch will lead to inconsistencies and inaccuracies in management fee calculation.

Recommendation

Perform funding settlement in final epoch.

Mitigation Review Log

Fixed in commit 6fb8a0f8981f0126f7b29dad53d73d1fce373b3d.

4.8 ManualPriceAccount Vulnerable to Unauthorized Control

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: DoS Risk

Description

The `ManualPriceAccount` and `ManualTwapAccount` can be created without proper permission validation. Furthermore, these accounts can only be updated by their respective creators.

This lack of control allows a malicious attacker to create and manage these accounts without oversight. Once created, the protocol maintainer is unable to regain control or update them, effectively locking the protocol out of its own functionality.

Impact

A malicious attacker could exploit this vulnerability by creating `ManualPriceAccount` and `ManualTwapAccount` instances. The protocol maintainer would lose the ability to manage these accounts, potentially causing disruptions or DoS within the protocol.

Recommendation

Implement stricter permission validation during both the initialization and update processes.

Mitigation Review Log

Fixed in commit 05393097d0979e17a4a25b556826469e11b6d63b.

4.9 Oracle Price Calculated by Uninitialized Value

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `initialize_market` instruction, the oracle price is calculated using an uninitialized market state.

```
270     let market_state = &mut ctx.accounts.market_state.load_init()?;
271     // Get oracle prices
272     let oracle_prices = get_oracle_prices(&market_state,
          ↪ &ctx.accounts.oracle)?;
```

programs/ratex-mooncake/src/instructions/market.rs#L270-L272

Impact

The `get_oracle_prices` function relies on `market_state` to provide accurate data. Since `market_state` is uninitialized at this stage, the function will return unexpected or incorrect prices.

Recommendation

Call `get_oracle_prices` only after `market_state` has been fully initialized.

Mitigation Review Log

Fixed in commit ec5404b114f5f7b53f587efb3ae4b4d093b87290.

4.10 Lack of Oracle Validation and Accuracy

Severity: Medium	Status: Acknowledged
Target: Smart Contract	Category: Data Validation

Description

There are multiple issues for oracle module:

1. Confidence not checked.
2. Not check validation level for Pyth oracle.
3. Not check staleness for manual oracle.
4. Not check if the last update time increases.
5. Not check if the twap account is valid in `read_twap_from_source`.
6. The account owner check for Pyth twap account is redundant, while it should be checked in switchboard and meteora oracle.
7. The `last_updated` should be replaced with time calculated by slot in `get_switchboard_price`.

Impact

The oversight could result in invalid or older data updated in oracle account.

Recommendation

Enhance the oracle validation mentioned above.

Mitigation Review Log

RateX Team:

We are aware of these issues and will address them in a future update.

4.11 Incorrect Rounding Direction When Insufficient Pool Balance

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Precision

Description

When the pool balance is insufficient to fulfill a swap request, it recalculates the actual swap input required based on the available output using the following logic:

```
212     // Calculate actual swap input based on actual output
213     swap_input = if desired_swap_output > 0 {
214         mul_div_u64(optimal_swap_input, actual_swap_output,
    ↪ desired_swap_output)?
215     } else {
216         0
217     };
```

programs/ratex-mooncake/src/controller/token_operations.rs#L212-L217

Additionally, the pattern also appears in `try_reset_vault` when partial swap is needed.

In this implementation, the division rounds down by default. However, in cases where rounding is required, the direction should be ceiling instead of floor.

Impact

Using floor division in this scenario results in an underestimation of the swap input. This precision loss is absorbed by liquidity providers.

Recommendation

Replace the floor division with a ceiling division.

Mitigation Review Log

Fixed in commit b6e2d9f20272120671183a20ab7d520c7197fccd and d9207606fdf23a7dfa7360f4e0af7a26a2af1fc4.

4.12 Inconsistent Order of HandleSwapFeeDistribution

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `swap_underlying_to_token` and `swap_token_to_underlying` functions, the `handle_swap_fee_distribution` function is called. However, the execution order varies between the two:

- In `swap_underlying_to_token`, the fee distribution is executed before the swap.
- In `swap_token_to_underlying`, the fee distribution is executed after the swap.

Impact

This inconsistency can lead to slight differences in the swap results between the two instructions.

Recommendation

Align the execution order of `handle_swap_fee_distribution` in both instructions to ensure consistency.

Mitigation Review Log

Fixed in commit 8030e13558195e10e5762bde0d5270db48f9a46d.

4.13 Incorrect Division Math Library

Severity: Low	Status: Fixed
Target: Library	Category: Math Error

Description

The math library introduces two division methods: `ceil_div` and `floor_div`. However, the implementation overlooks whether the quotient is positive or negative, which results in incorrect calculations in certain cases.

```
15      fn checked_ceil_div(&self, rhs: $t) -> Option<$t> {
16          let quotient = self.checked_div(rhs)?;
17
18          let remainder = self.checked_rem(rhs)?;
19
20          if remainder > <$t>::zero() {
21              quotient.checked_add(<$t>::one())
22          } else {
23              Some(quotient)
24          }
25      }
```

programs/ratex-mooncake/src/utils/math/ceil_div.rs#L15-L25

For the input `8.checked_ceil_div(-3)`, the expected result is `-2`, but the function incorrectly calculates `-1`.

As a baseline, the following tests in examples only have 50% pass rate.

https://doc.rust-lang.org/std/primitive.i32.html#method.div_ceil

https://doc.rust-lang.org/std/primitive.i32.html#method.div_floor

Impact

This flawed division implementation can lead to incorrect calculations. While the current codebase may not trigger these cases, the issue leaves room for potential future bugs and logic errors, especially when extended to other contexts.

Recommendation

Update the implementation to consider the sign of the quotient during the calculation.

Mitigation Review Log

Fixed in commit b6e2d9f20272120671183a20ab7d520c7197fccd.

4.14 Incorrect Auto Compounding Factor Calculation

Severity: Low	Status: Fixed
Target: Library	Category: Logic Error

Description

In the `distribute_lp_fee_to_pool` function, it will call `liquidity_pool.update_accumulatd_fees` to update accumulated fees and auto compounding factor. However, fees has been added into `pool_virtual_funding` and `pool_virtual_leverage` before `update_accumulatd_fees`. It means that `lp_tvl` includes fees, which will affect the calculation of `auto_compounding_factor`.

Impact

The `auto_compounding_factor` will be less than expected.

Recommendation

Excludes fees when calculating `auto_compounding_factor`.

Mitigation Review Log

Fixed in commit 4fd9617f2ca539d4622468dbbaab5754ba0e59df.

4.15 Informational and Undetermined Issues

Lack of Fee Charged Events

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic Error

In the `rebalance_execute` function, it charges protocol fees for redemption but misses to emit fee charged events.

Hardcoded Value

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Code QA

In the `rebalance_execute` function, it uses hardcoded value 10000, which could be replaced by using `calculate_fee` function or `BASIS_POINTS_DIVISOR` .

Unchecked Value in Market Initialization

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Validation

The `target_utilization` and `target_rate` are checked in `update_funding_rate_bounds` function. However, when initializing market, these two parameters are unchecked.

Precision Loss by Pre Calculated Ratio

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Precision

There are some code snippets which calculates a ratio and then use it in following calculations.

Here are some examples:

programs/ratex-mooncake/src/controller/vault_operations.rs#L250-L254

programs/ratex-mooncake/src/controller/rebalance_trading.rs#L101-L115

The pattern may introduce precision loss, and it can be simply optimized by merging two formulas.

Avoid Using the as Operator

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Code QA

The codebase predominantly employs the `as` operator, which is not deemed safe. It is recommended to utilize `try_into()?` instead of `as`. Specifically, for the `abs()` function, it is advisable to use `unsigned_abs()` rather than `abs() as`.

Authority Can Not Be Updated for Some Accounts

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

For the State account, the admin authority can be updated by `update_admin` instruction. However, the authority of MarketState, ManualPriceAccount and ManualTwapAccount can not be updated.

Token2022 Support

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Compatibility Issue

When transferring token2022 tokens, the `token_2022::transfer` is used, but it is deprecated. Moreover, current implementation can not support token2022 mints with extensions like transfer fee extension.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

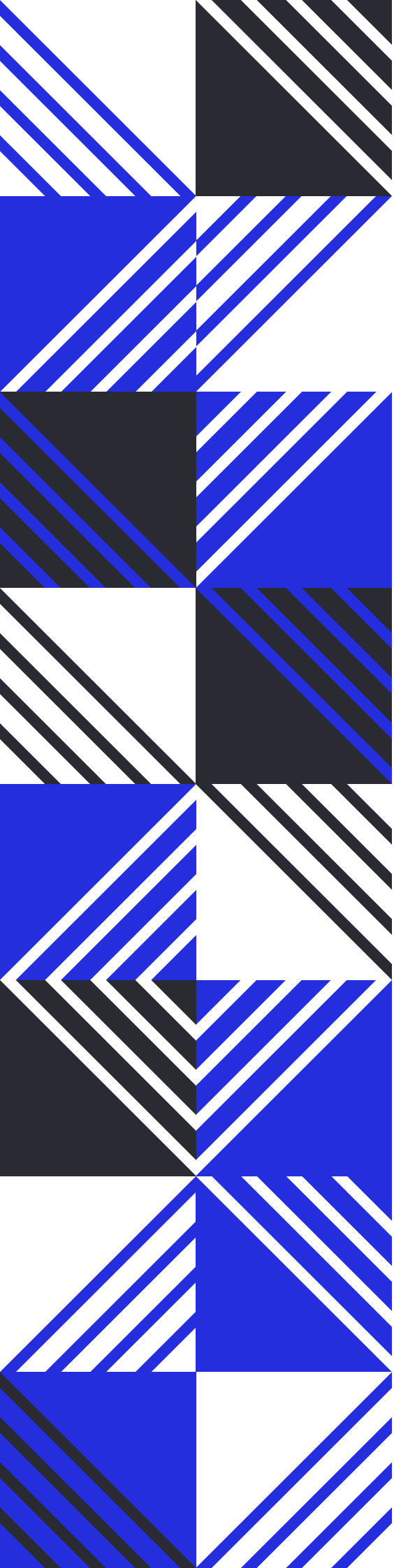
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.



OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs