OFFSIDE LABS

Flash.Trade

Smart Contract Security Assessment

May 2025

Prepared for:

Flash.Trade

Prepared by:

Offside Labs

Yao Li

Moon Liang

Contents

1 About Offside Labs

Offside Labs stands as a pre-eminent security research team, comprising highly skilled hackers with top - tier talent from both academia and industry.

The team demonstrates extensive and diverse expertise in modern software systems, which encompasses yet are not restricted to *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. Offside Labs is at the forefront of innovative domains such as *cryptocurrencies* and *blockchain technologies*. The team achieved notable accomplishments including the successful execution of remote jailbreaks on devices like the **iPhone** and **PlayStation 4**, as well as the identification and resolution of critical vulnerabilities within the **Tron Network**.

Offside Labs actively involves in and keeps contributing to the security community. The team was the winner and co-organizer for the *DEFCON CTF*, the most renowned CTF competition in Web2. The team also triumphed in the **Paradigm CTF 2023** in Web3. Meanwhile, the team has been conducting responsible disclosure of numerous vulnerabilities to leading technology companies, including *Apple*, *Google*, and *Microsoft*, safeguarding digital assets with an estimated value exceeding **$300 million**.

During the transition to Web3, Offside Labs has attained remarkable success. The team has earned over **$9 million** in bug bounties, and **three** of its innovative techniques were acknowledged as being among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Flash.Trade* smart contracts, starting on March 03, 2025, and concluding on April 22, 2025.

Project Overview

Flash.Trade is a decentralized asset-backed perpetuals and spot exchange on Solana offering up to 100x leverage, low fees, and minimal price impact via a pool-to-peer model. Liquidity providers earn real yield through trading fees, supported by dynamic pricing via Pyth and backup oracles. Initially funded by evolving 3D NFTs, Flash transitioned to the FAF token, unlocking new rewards, governance, and utility for the ecosystem.

Audit Scope

The assessment scope contains mainly the smart contracts of the perpetuals program for the *Flash.Trade* project. The audit is based on the following specific branches and commit hashes of the codebase repositories:

- Flash.Trade
  - Codebase: `https://github.com/flash-trade/flash-contracts-closed`
  - Commit Hash: 017e978b48e7ff0d8b28a5c62ee1ee3d96a924ea

We listed the files we have audited below:

- Flash.Trade
  - programs/perpetuals/**/*.rs

Findings

The security audit revealed:

- 1 critical issues
- 2 high issues
- 7 medium issues
- 2 low issues
- 0 informational issue

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	Unsettled Lock Fees Overwritten in ExecuteLimitOrder Instruction	Critical	Fixed
02	Remove Custody Will Cause Custody ID Misalignment	High	Fixed
03	Incorrect Receive Custody ID Verification for Trigger Order	High	Fixed
04	Lack of Price Slippage Protection in ExecuteLimitOrder Instruction	Medium	Fixed
05	Multiple Permissions Never Checked in Corresponding Instructions	Medium	Fixed
06	Unreasonable Fee Discount and Rebate When Insolvent Position Close	Medium	Fixed
07	Borrow Rate Updating of Collateral Custody May be Skipped in CloseAndSwap Instruction	Medium	Fixed
08	Referee Can Steal Rebate Due to Unverified Remaining Account	Medium	Fixed
09	Remove Instructions Cannot be Executed When Multisig Threshold Exceeds One	Medium	Fixed
10	Reimburse Instruction Cannot be Executed Normally When Multisig Threshold Exceeds One	Medium	Fixed
11	Single Singer Can Bypass Threshold of Multisig in AddInternalOracle Instruction	Low	Fixed
12	Position Profit May Benefit from Precision Loss in Entry Price Calculation	Low	Fixed

4 Key Findings and Recommendations

4.1 Unsettled Lock Fees Overwritten in ExecuteLimitOrder Instruction

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

Description

In the `execute_limit_order` instruction, when a position already exists (`position.size_usd > 0`), the logic is intended to increment the position by merging two positions. However, the value of `unsettled_fees_usd` is overwritten instead of being accumulated. This means the cumulative unsettled lock fees are wiped out rather than added to the existing value.

Here is the relevant code snippet:

```
322     if position.size_usd > 0 {
323         // increment existing position (add collateral + increase size)
324         position.unsettled_fees_usd =
    ↪ collateral_custody.get_lock_fee_usd(&position, curtime)?;
325         position.increment(&collateral_min_price, &delta_position)?;
326         position.cumulative_lock_fee_snapshot =
327             collateral_custody.get_cumulative_lock_fee(curtime)?;
328         position.update_time = curtime;
```

programs/perpetuals/src/instructions/execute_limit_order.rs#L322-L328

Impact

This bug allows users to exploit the system and avoid paying accumulated lock fees. By repeatedly using limit orders, users can reset the `unsettled_fees_usd` field to a lower value or even zero. This could result in significant loss of revenue for the protocol and incentivize exploitative behavior.

Recommendation

Update the logic for `unsettled_fees_usd` to ensure the new lock fee is added to the existing value, rather than overwriting it.

Mitigation Review Log

Updated the logic for `unsettled_fees_usd` in both `execute_limit_order` and `execute_limit_order_with_swap` instructions to account for previously unsettled lock fees.

Fixed in commit 8dd9f3732d4f06e20e97f335841e88fbb53f37ba.

4.2 Remove Custody Will Cause Custody ID Misalignment

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

Description

The `custody_id` represents the index of a `Custody` in the `Pool` 's custody vector. In the `remove_custody` instruction, the corresponding custody is removed from the `Pool` 's custody vector as shown below:

```
114     // remove token from the list
115     let pool = ctx.accounts.pool.as_mut();
116     let custody_id = pool.get_custody_id(&ctx.accounts.custody.key())?;
117     pool.custodies.remove(custody_id);
```

programs/perpetuals/src/instructions/remove_custody.rs#L114-L117

However, this removal causes an unintended issue: any `custody_id` values greater than the removed one are shifted (decremented by 1) due to the re-indexing of the vector. This re-indexing introduces a misalignment of `custody_id` across other contract components that rely on it, such as the `Market` and `Order` accounts.

- Market Account: The `custody_id` stored in the `Market` account is used to fetch backup oracle price data. Misalignment of `custody_id` here results in fetching incorrect price data, which could lead to denial-of-service (DoS) issues or inaccurate price references.
- Order Account: The `custody_id` is used for validation across the `Order` , `Market` , and `Pool` 's custody vector. Misalignment causes these validations to fail, resulting in stuck orders. This issue can even prevent users from canceling orders and withdrawing their reserve tokens.

Impact

- Incorrect Price Data: Misalignment of `custody_id` in the `Market` account leads to fetching incorrect backup oracle prices, which can cause DoS situations or inaccurate price.
- Stuck Orders: Misaligned `custody_id` in `Order` accounts causes orders to become stuck. Users may be unable to cancel orders and withdraw reserve tokens.

Recommendation

To prevent this issue, avoid using the index of a custody vector (`custody_id`) as a dynamic reference. Instead, implement a static, unique identifier for each custody that persists regardless of its position in the vector.

Mitigation Review Log

Introduced a `unique_custody_count` (u8) field in Pool account and `uid` (u8) field in Custody account that persists as a static unique identifier for a given Pool. This system also allows a Custody to be added back again after being removed but with a new uid.

`_custody_id` related fields in Order account are renamed to `_custody_uid` to reference the static uid field of the corresponding custody account.

Fixed in commit `d0c1ae4750ef6568c0008e0efe8d33b4e087b771`.

4.3 Incorrect Receive Custody ID Verification for Trigger Order

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

Description

The `receive_custody_id` is used to indicate the receiving asset after a trigger order is filled.

In the `place_trigger_order` instruction, it is verified with the following logic:

```
154     if params.receive_custody_id > pool.custodies.len() as u8 {
155         return Err(ProgramError::InvalidArgument.into());
156     }
```

programs/perpetuals/src/instructions/place_trigger_order.rs#L154-L156

However, the expected range for `receive_custody_id` should be `[0, pool.custodies.len() - 1]`. The current check allows an invalid `receive_custody_id` equal to `pool.custodies.len()`, which is out of bounds.

Additionally, in the `execute_trigger_order` instruction, the `receive_custody_id` is never validated. This omission allows a user to potentially receive an unexpected asset after the trigger order is filled, leading to unintended asset exposure.

Impact

- Stuck Trigger Orders: Users may create trigger orders that cannot be executed due to the incorrect `receive_custody_id` verification in the `place_trigger_order` instruction.
- Unexpected Asset Exposure: Users may receive unintended assets after the trigger order is executed due to the lack of validation in the `execute_trigger_order` instruction. This could result in exposure to undesired assets and financial losses.

Recommendation

Fix the range check in `place_trigger_order` instruction and add verification in `execute_trigger_order` instruction.

Mitigation Review Log

Eliminated range check and added account and seed validation for receive_custody in the `place_trigger_order` instruction. Also added verification for `receive_custody` uid in `execute_trigger_order` instruction.

Fixed in commit `d0c1ae4750ef6568c0008e0efe8d33b4e087b771`.

4.4 Lack of Price Slippage Protection in ExecuteLimitOrder Instruction

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

Description

In the `execute_limit_order` instruction, the `limit_price` is initially verified, and the `entry_price` is calculated as follows:

```
262     let entry_price = if market.side == Side::Long {
263         require!(
264             target_min_price <= limit_price,
265             PerpetualsError::InvalidLimitPrice
266         );
267         target_max_price
268     } else {
269         require!(
270             target_max_price >= limit_price,
271             PerpetualsError::InvalidLimitPrice
272         );
273         target_min_price
274     };
```

programs/perpetuals/src/instructions/execute_limit_order.rs#L262-L274

While the code checks if the `limit_price` is within the acceptable range (`target_min_price` and `target_max_price`), the final `entry_price` is derived based on the price range and the trade spread. This calculation may result in an `entry_price` that deviates from the user-defined `limit_price` , leading to unintended execution prices.

Impact

Users may receive an unintended `entry_price` when executing a limit order, resulting in unexpected costs or losses due to slippage.

Recommendation

To protect users from unintended price slippage, implement an additional field in the `Order` structure to allow users to specify their maximum acceptable slippage.

Mitigation Review Log

Updated the flow to use limit price as the ultimate execution price instead of ambiguous `target_min_price` or `target_max_price`, and retained the associated size based spread which is static across both `execute_limit_order` and `execute_limit_order_with_swap` instructions.

Fixed in commit `f3a86ce01b8b7fae83ab515133924d896562f4b5`.

4.5 Multiple Permissions Never Checked in Corresponding Instructions

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

Description

The `Permissions` struct in the `Perpetuals` state is designed to determine whether specific actions are allowed within the smart contract. It contains multiple boolean flags to enforce these restrictions. However, certain permissions— `allow_liquidation`, `allow_lp_staking`, `allow_fee_discounts`, and `allow_referral_rebates` —are never validated in their respective instructions.

Impact

This oversight allows these actions to be executed regardless of whether the corresponding permission has been set to `false`.

Recommendation

Add checks for these permissions in corresponding instructions.

Mitigation Review Log

Updated the required permission checks in corresponding instructions. Also refactored and moved the validation to `Pool::fetch_and_update_trading_benefits` function specifi-

cally for `Privilege::Referral` flow.

Fixed in commit `1871026dfd55c68e215f1e51a168e589c8729f7e`.

4.6 Unreasonable Fee Discount and Rebate When Insolvent Position Close

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

Description

In the `get_close_amount` function, when a position is detected as insolvent, the exit fee calculation discounts the fee obligations:

```
518         } else {
519             // Position is insolvent so discount fee obligations
520             let final_fees_usd = assets_usd.saturating_sub(loss_usd);
521             Ok((
522                 0u64,
523                 collateral_min_price
524                     .get_token_amount(final_fees_usd,
    ↪ position.collateral_decimals)?,
525             ))
526         }
```

programs/perpetuals/src/state/pool.rs#L518-L526

However, if the discounted exit fee (`final_fees_usd`) is not zero, the instructions continues processing through the referral logic to calculate further discounts, rebates, and refunds. This behavior creates a logical inconsistency, as insolvent positions should ideally not qualify for additional fee rebates or discounts.

Impact

This issue can result in less fees being allocated to liquidity providers (LPs) because the insolvent position unnecessarily benefits from referral logic discounts and rebates.

Recommendation

Refactor the `get_close_amount` function to introduce a boolean flag that explicitly indicates whether the position is insolvent. This flag can then be used to bypass any further logic (such as referral discounts and rebates) for insolvent positions.

Mitigation Review Log

Updated the function signature of `get_close_amount` to also return a boolean flag `is_solvent` to identify if the current position is solvent post fee obligations and to bypass privileges associated to fee discounts and/or rebates in case of insolvency.

Fixed in commit `fe974713f385c497b7f09269ff22e31837ce3324`.

4.7 Borrow Rate Updating of Collateral Custody May be Skipped in CloseAndSwap Instruction

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

Description

In the `close_and_swap` instruction, the logic checks whether the available token amount in the dispensing custody is sufficient before executing the corresponding actions. The relevant code snippet is as follows:

```
531     if math::checked_sub(
532         dispensing_custody.assets.owned,
533         dispensing_custody.assets.locked,
534     )? >= dispensing_amount
535     {
536         // transfer token
537         msg!("Transfer token");
```

programs/perpetuals/src/instructions/close_and_swap.rs#L531-L537

However, in the second branch of this conditional, the `update_borrow_rate` function for collateral custody is not called, which may lead to unintended consequences.

Impact

Failure to update the borrow rate for collateral custody will result in a higher borrow rate and increased lock fees until the next update occurs. This could adversely affect users relying on the accurate calculation of borrowing costs.

Recommendation

Ensure that the `update_borrow_rate` function for collateral custody is invoked in the second branch of the conditional statement.

Mitigation Review Log

Updated the execution branch identified above to also include `update_borrow_rate` function invocation.

Fixed in commit `5b90876279b3c69ff71175e0edbed66f77c4e68d`.

4.8 Referee Can Steal Rebate Due to Unverified Remaining Account

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

Description

The smart contract utilizes three accounts in remaining accounts to calculate and execute referral discounts and rebates: the referee's referral account, the referrer's trading account, and the referrer's rebate receiving account.

The code responsible for transferring rebates is as follows:

```
403     if rebate > 0 {
404         net_fee_amount = net_fee_amount.saturating_sub(rebate);
405         perpetuals.transfer_tokens(
406             ctx.accounts
407                 .collateral_custody_token_account
408                 .to_account_info(),
409             ctx.remaining_accounts[2].to_account_info(),
410             ctx.accounts.transfer_authority.to_account_info(),
411             ctx.accounts.token_program.to_account_info(),
412             rebate,
413         )?;
414     }
```

programs/perpetuals/src/instructions/close_and_swap.rs#L403-L414

Currently, the rebate receiving account (located at `ctx.remaining_accounts[2]`) is not verified. This lack of verification allows the referee to manipulate the account used for the rebate transfer.

Impact

The referee could potentially use a different account instead of the token account specifically owned by the referrer, leading to unauthorized rebate theft.

Recommendation

Ensure that the rebate receiving account is the associated token account of referrer.

Mitigation Review Log

Added the required validation checks to ensure rebate is always credited to the rightful owner.

Fixed in commit `be40238c76f24eea37167f37d6e33ae58344e4d9`.

4.9 Remove Instructions Cannot be Executed When Multisig Threshold Exceeds One

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

Description

In the `remove_pool`, `remove_market`, and `remove_custody` instructions, the target account is closed, and the size of the parent account is reallocated. Below is an example of the `remove_custody` instruction:

```
42      #[account(
43          mut,
44          realloc = Pool::LEN + (pool.custodies.len() + pool.markets.len() - 1) *
    ↪ std::mem::size_of::<Pubkey>() +
45              (pool.ratios.len() - 1) * std::mem::size_of::<TokenRatios>(),
46          realloc::payer = admin,
47          realloc::zero = false,
48          seeds = [b"pool",
49                   pool.name.as_bytes()],
50          bump = pool.bump
51      )]
52      pub pool: Box<Account<'info, Pool>>,
```

programs/perpetuals/src/instructions/remove_custody.rs#L42-L52

The implementation attempts to remove the related data from the state of the parent account after the threshold is met. However, this logic only functions correctly if the multi-sig threshold is one. If the threshold is greater than one, the size of the upper account is reallocated, but the actual state modification is not executed during the first signer call. This results in an instruction failure.

Impact

If the multi-sig threshold is greater than one, the `remove_*` instructions cannot be executed successfully due to instruction failure.

Recommendation

Ensure that the account closure and reallocation of the parent account size occur only after the multi-sig threshold is met.

Mitigation Review Log

Updated the required instructions accordingly to have account closures and reallocation only after the multi-sig threshold is met. Reallocation of parent account is redundant.

Fixed in commit `c21e87544f6e460f5666b3a24f8d8404c922d307`.

4.10 Reimburse Instruction Cannot be Executed Normally When Multisig Threshold Exceeds One

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

Description

In the multi-sig implementation, it hashes the accounts except admin signer and instruction data to ensure uniqueness. However, in the `reimburse` instruction, the `funding_account` is constrained by the admin signer. The relevant code snippet is as follows:

```
34      #[account(
35          mut,
36          constraint = funding_account.mint == custody.mint
37              && funding_account.owner == admin.key(),
38      )]
39      pub funding_account: Box<Account<'info, TokenAccount>>,
```

programs/perpetuals/src/instructions/reimburse.rs#L34-L39

If the multi-sig threshold exceeds one, the `funding_account` must remain the same, but its authority would differ due to the constraint. This introduces an abnormal execution flow: the only way to execute the instruction successfully is for the admin signer to repeatedly transfer the authority of the token account to the next signer in the multi-sig process. This behavior is both impractical and inconsistent with expected multi-sig functionality.

Impact

The `funding_account` constraint in the `reimburse` instruction is incompatible with the current multi-sig implementation when the threshold exceeds one. This results in abnormal execution flow, making the `reimburse` instruction effectively unusable under these conditions.

Recommendation

Consider using a program-owned token account instead of a user-owned account for the `funding_account` .

Mitigation Review Log

Updated the validation to check for admin control instead of multisig threshold to avoid use of additional program-owned token accounts.

Fixed in commit `88976c103019a8dd2e7d1a72c98dbf9d753d3836`.

4.11 Single Singer Can Bypass Threshold of Multisig in AddInternalOracle Instruction

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

Description

In the `add_internal_oracle` instruction, the `int_oracle_account` is created during the first signer call using the `init_if_needed` constraint. However, there is no subsequent initialization or verification of this account after the threshold is met. This means the creation of the account effectively completes the multi-sig execution for this instruction, allowing a single signer to bypass the multi-sig threshold entirely.

```
27      #[account(
28          init_if_needed,
29          payer = admin,
30          space = CustomOracle::LEN,
31          seeds = [b"oracle_account",
32                   custody_token_mint.key().as_ref()],
33          bump
34      )]
35      pub int_oracle_account: Box<Account<'info, CustomOracle>>,
```

programs/perpetuals/src/instructions/add_internal_oracle.rs#L27-L35

Impact

A single signer can exploit the `init_if_needed` constraint to bypass the multi-sig threshold, enabling unauthorized execution of the `add_internal_oracle` instruction.

Recommendation

Ensure that the `int_oracle_account` is created only after the multi-sig threshold is validated or add a state flag to the account to indicate whether it has been properly initialized.

Mitigation Review Log

Moved to admin control with implicit initialization.

Fixed in commit `049e17ff0d0bf2862deaf02eb66d0c1bb4bb4387`.

4.12 Position Profit May Benefit from Precision Loss in Entry Price Calculation

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

Description

The `increase_size` instruction in the contract calculates the new `entry_price` as follows:

```
386     // Average_Entry_Price = updated_size_usd / updated_size
387     position.entry_price.price = math::checked_as_u64(math::checked_div(
388         size_usd_scaled,
389         updated_size_amount as u128,
390     )?)?;
```

programs/perpetuals/src/instructions/increase_size.rs#L386-L390

However, the use of floor division (`checked_div`) results in precision loss. This behavior allows users to manipulate their position profit under certain conditions.

For example, when a user opens a long position, they can increase a small amount of size at the new price (when the market price rises). Due to the precision loss, the `entry_price` may remain unchanged, which could artificially increase the profit for the newly added size. This issue is limited in scope, as the profit gain is constrained by the precision granularity of the price.

Impact

A user can exploit the precision loss by increasing a small amount of size for a long position before closing it, thereby increasing their profit. However, the impact is minimal due to the limited effect of price precision on the calculation. Under the current configuration and liquidity, no significant attack or exploit appears feasible.

Recommendation

Use ceiling division instead of floor division for the entry price calculation of long position increasing.

Mitigation Review Log

Added conditional flow to increase precision for entry price calculations on position increment.

Fixed in commit `5967dc8599b9690d58d5330dc328909a93743ab5`.

5 Disclaimer

This report reflects the security status of the project as of the date of the audit. It is intended solely for informational purposes and should not be used as investment advice. Despite carrying out a comprehensive review and analysis of the relevant smart contracts, it is important to note that Offside Labs' services do not encompass an exhaustive security assessment. The primary objective of the audit is to identify potential security vulnerabilities to the best of the team's ability; however, this audit does not guarantee that the project is entirely immune to future risks.

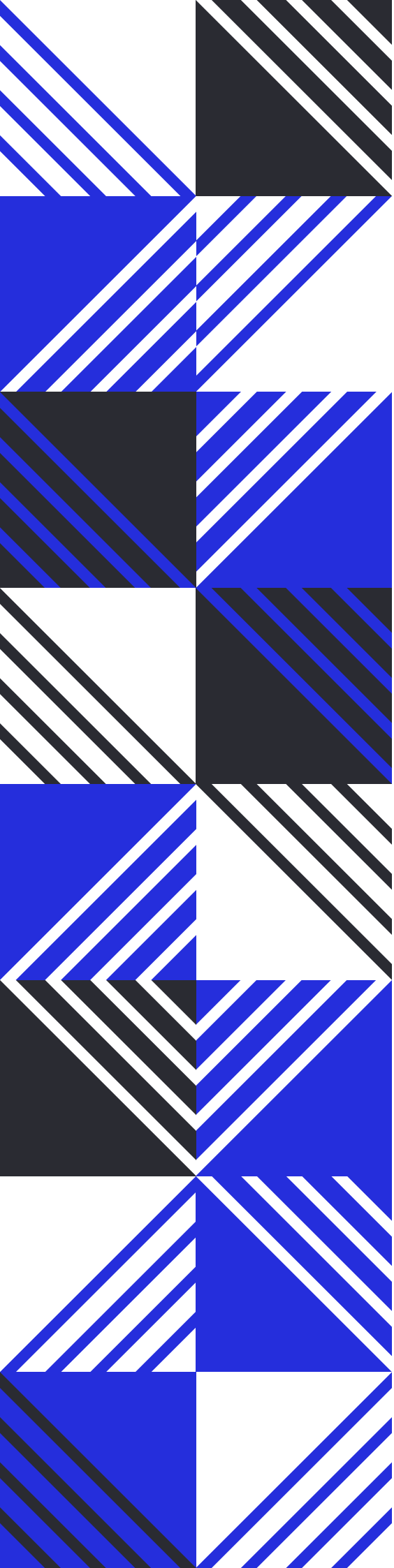
Offside Labs disclaims any liability for losses or damages resulting from the use of this report or from any future security breaches. The team strongly recommends that clients undertake multiple independent audits and implement a public bug bounty program to enhance the security of their smart contracts.

The audit is limited to the specific areas defined in Offside Labs' engagement and does not cover all potential risks or vulnerabilities. Security is an ongoing process, regular audits and monitoring are advised.

Please note: Offside Labs is not responsible for security issues stemming from developer errors or misconfigurations during contract deployment and does not assume liability for centralized governance risks within the project. The team is not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By utilizing this report, the client acknowledges the inherent limitations of the audit process and agrees that the firm shall not be held liable for any incidents that may occur after the completion of this audit.

This report should be considered null and void in case of any alteration.



OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs