Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

GrantPicks

Veridise Inc.

April 30, 2025

► **Prepared For:**

PotLock
https://www.potlock.org

► **Prepared By:**

Alberto Gonzalez
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Apr. 30, 2025	V3
Mar. 24, 2025	V2
Feb. 24, 2025	V1
Feb. 19, 2025	Initial Draft

Contents

1 Executive Summary

From Jan. 13, 2025 to Feb. 11, 2025, PotLock engaged Veridise to perform a security assessment of their GrantPicks protocol. The GrantPicks protocol aims to help fund projects by using on-chain voting. Veridise conducted the assessment over 8 person-weeks, with 2 security analysts reviewing the project over 4 weeks on commit `902900a`. The review strategy involved an extensive manual code review performed by Veridise security analysts.

Project Summary. The GrantPicks protocol is a fundraising platform designed to support projects through community-driven decision-making.

All the fundraising activity is organized around *rounds*, which are time-bound events managed by a round owner and a team of administrators. Each round progresses through several phases:

- **Projects application phase.** Interested projects submit applications containing the necessary information for assessment. The round owner selects a set of approved projects that will proceed to the voting phase.
- **Voting phase.** Users vote for projects they support. Instead of voting on individual projects, the protocol employs a randomized pairwise voting approach, where users compare randomly selected pairs of projects.
- **Payouts assignment phase.** Based on the voting results, the administrators of a round determine the distribution of funds and assign payouts to the approved projects.
- **Compliance phase.** Approved projects must undergo Know Your Customer (KYC) verification to qualify for payouts.
- **Cooldown phase.** Before payouts are processed, users can submit challenges if they identify any issues or concerns regarding the assigned payouts. Round administrators then review and resolve these disputes.
- **Payout processing phase.** After the cooldown period, the assigned payouts are processed, distributing funds to the approved projects. Any remaining funds in the round's vault can be returned to the round creator.

During some of these phases, motivated users contribute funds to the round's vault, which will be distributed among eligible projects based on the number of votes they receive.

The protocol is built on three smart contracts that work together to facilitate funds distribution process:

- **Project-Registry**. This contract manages project registrations and maintains essential project data, including project names, repository links, descriptions, ownership details, and payout information. It facilitates the indexing of projects and streamlines their application and approval process.
- **Round**. As the core of the protocol, this contract oversees the entire life-cycle of a funding round. It manages round creation, project applications, the selection of approved projects, the voting process for approved projects, and the distribution of funds after the voting phase.

- **Lists**. This contract enables the creation and management of structures called *lists*, which function as access control mechanisms within the protocol. When a list is created, users can apply for membership, but their registration must be reviewed and approved by the list owner. Once approved, users gain access to specific actions during a funding round that are restricted to members of designated lists.

It should be noted that the funding decision-making process takes place off-chain, where round managers manually assign payouts to projects - this is outside of the control of the smart contracts' logic.

Code Assessment. The GrantPicks developers provided the source code of the contracts for review. The source code appears to be mostly original code written by the GrantPicks developers. To assist Veridise analysts in comprehending the code, the developers provided detailed documentation outlining the project's intended business logic and design decisions. They also conducted a walk-through with the Veridise team, highlighting potential areas of concern and maintaining active communication throughout the code review process.

Summary of Issues Detected. The security assessment uncovered a total of 51 issues, 7 of which were assessed to be of high or critical-severity. Specifically, the analysts discovered that any user could approve their own registration to the voting whitelist or remove legitimate users from it (V-GRP-VUL-001). Additionally, they found that the voting process was vulnerable to Sybil attacks, enabling malicious users to vote multiple times for the same project pair (V-GRP-VUL-002). Another issue involved a missing compliance period check when processing project payments, which could have resulted in legitimate projects losing their grant funds (V-GRP-VUL-005). The analysts also flagged two storage layout issues that could cause denial-of-service scenarios even under normal protocol operations (V-GRP-VUL-006, V-GRP-VUL-007). Lastly, the Veridise team identified two issues related to missing validation checks: One allowed users to apply for list registrations when this action should have been restricted to admins (V-GRP-VUL-004). The other permitted users to submit project applications to a round even during the voting phase (V-GRP-VUL-003).

Furthermore, the Veridise team identified 9 medium-severity issues. Notably, they highlighted a potential incorrect trust assumption regarding round admins, which could allow them to drain all funds from a round (V-GRP-VUL-016). They also found that the challenge mechanism could be exploited to block the round owner from distributing payout funds to winning projects (V-GRP-VUL-010). Additionally, they identified that the referrer fee would be lost if no referrer was specified when calling the deposit function (V-GRP-VUL-014). Lastly, the team discovered a flaw in the ownership transfer process, where ownership changes were not properly written back to storage, making the transfer ineffective (V-GRP-VUL-008).

Lastly, Veridise analysts identified 21 low-severity and 14 warning-severity findings. These included missing checks on the timing of certain actions, which could allow operations to occur outside their intended time-frames (V-GRP-VUL-018, V-GRP-VUL-023, V-GRP-VUL-025). Additionally, they identified potential incorrect trust assumptions regarding certain roles in the system, which could lead to unintended privileges issues (V-GRP-VUL-016, V-GRP-VUL-022, V-GRP-VUL-036). Furthermore, the team noted a lack of input validation checks in some functions, increasing the risk of misconfigurations during setup and operation (V-GRP-VUL-030, V-GRP-VUL-026, V-GRP-VUL-031, V-GRP-VUL-034).

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the GrantPicks protocol security before deployment.

Additional testing. The source code includes test suites for the three contracts that make up the GrantPicks protocol, covering both positive and negative test cases. However, the Veridise team think these test suites are insufficient to fully prevent the introduction of bugs and to ensure correct execution under common conditions:

- The Veridise analysts believe that some of the identified issues could have been prevented with happy path execution tests, such as: Ensuring users can be correctly blacklisted (V-GRP-VUL-009), verifying the successful execution of ownership transfers (V-GRP-VUL-008), confirming challenges are correctly resolved and cleared from storage (V-GRP-VUL-021), validating that payout funds are correctly sent to the intended recipients (V-GRP-VUL-035).
- Additionally, the Veridise analysts believe that the negative execution paths test suite is insufficient, as some expected reverts were not tested. Strengthening these tests could have prevented issues such as: Ensuring that a user cannot register a project when the configuration does not allow it (V-GRP-VUL-004). Checking that a project registration correctly reverts if attempted after the voting phase has started (V-GRP-VUL-003).
- Finally, the Veridise team recommends adding test cases to verify that the fixes for the issues outlined in this report function as intended.

Storage layout. The design of the storage layout for Soroban contracts is important due to known Denial-of-Service risks* associated with the ledger entry size limit when ledger keys are not chosen properly. These issues were observed in the reviewed codebase: V-GRP-VUL-007, V-GRP-VUL-006, V-GRP-VUL-015, V-GRP-VUL-011. To mitigate these risks, the Veridise team recommends that the GrantPicks developers carefully consider the size limitations of ledger entries, ensuring that the expected amount of stored information remains within safe bounds throughout the protocol's lifecycle. Proper restrictions should also be enforced to prevent users from unboundedly populating storage spaces. Additionally, when designing the storage layout, it is essential to ensure that storage for users and rounds does not collide with that of different users or rounds, thereby reducing the attack surface for this category of issues. Finally, it is advisable to thoroughly test these design assumptions and validate the expected limits on ledger entry sizes.

Phases timing constraints. The lifecycle of a funding round consists of multiple phases, starting with the project application phase and ending with the payout processing phase. These phases are described in more detail in the project summary [1]. The Veridise analysts recommend that the GrantPicks developers implement stricter segmentation between phases by enforcing explicit timing constraints on allowed actions. The absence of certain validation checks led to issues such as: The ability to modify project and round details while the voting phase is ongoing, potentially influencing the outcome (V-GRP-VUL-010, V-GRP-VUL-023, V-GRP-VUL-018), and the possibility of triggering payout assignments and fund redistribution before the voting phase has even begun, which could result in misallocated or premature payouts (V-GRP-VUL-049, V-GRP-VUL-025). In this way, the Veridise analysts recommend revisiting the design of the phase transitions to ensure that each stage enforces clear boundaries on permitted actions.

* https://medium.com/veridise/how-to-develop-securely-on-soroban-storage-types-with-unbounded-data-0318d691bb9a

Fuzzing/Property-based Testing. The Soroban platform provides tools for fuzzing Soroban smart contracts[†]. Fuzzing is a valuable technique for randomly executing the protocol while ensuring that key invariants are consistently upheld. While time constraints or other factors may prevent immediate implementation, the Veridise analysts recommend considering the development of a fuzzing test suite in the future to enhance the protocol's resilience against unexpected edge cases and vulnerabilities.

Documentation. The Grantpicks protocol uses the *Budget Boxing Algorithm* to distribute rewards to eligible winning projects based on the pairwise voting results. Unfortunately, Veridise analysts couldn't find any implementation of this algorithm, and it seems unclear which implementation will be used in practice. Given the significance of this aspect of the protocol, which is reward distribution, and the fact that it is implemented off-chain, it is recommended to include this information in the official documentation.

[†] https://developers.stellar.org/docs/build/smart-contracts/example-contracts/fuzzing

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
GrantPicks	`902900a`	Rust	Stellar

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 13–Feb. 11, 2025	Manual	2	8 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	2	2	1
High-Severity Issues	5	5	5
Medium-Severity Issues	9	9	9
Low-Severity Issues	21	21	18
Warning-Severity Issues	14	14	14
Informational-Severity Issues	0	0	0
TOTAL	51	51	47

Table 2.4: Category Breakdown.

Name	Number
Data Validation	18
Logic Error	15
Denial of Service	7
Authorization	5
Maintainability	5
Usability Issue	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The Security Assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the code base for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

3.2 Security Assessment Goals

The engagement was scoped to provide a security assessment of the GrantPicks smart contracts developed in Soroban programming language. During the assessment, the security analysts aimed to answer questions such as:

- Is it possible for an ineligible user to withdraw the round's deposit?
- Is there a risk of funds being permanently locked in the Round's Vault?
- Under what conditions might the Vault mistakenly reject a deposit or incorrectly accept a deposit?
- What could cause a valid payout to fail to transfer to the project recipient's address?
- Is it possible for someone to impersonate a payout assigned to someone else?
- What mechanisms are in place to prevent malicious actors from interfering with or blocking another project's payout?
- How does the system ensure that leftover funds in the Vault are not permanently locked and can be recovered or reallocated?
- Is there a risk of Fake Votes Injection?
- Is it possible to remove votes of legitimate users?

In addition to higher-level protocol security risks, the specific code implementation vulnerabilities of Soroban were also considered:

- Is there any possibility of arithmetic overflow or underflow occurring?
- Are the blockchain storage limits of the Soroban respected?
- Is there a risk of importing outdated code that could lead to the deployment of vulnerable contracts?*

* https://medium.com/veridise/veridise-enhances-soroban-security-breaking-our-teeth-on-stale-dependencies-db5f9c7130ba

3.3 Scope

The scope of this security assessment was limited to the below folders of the source code provided by the GrantPicks developers on commit `902900a` from the *developer* branch:

1. `stellar/contract/lists/src/*`
2. `stellar/contract/project-registry/src/*`
3. `stellar/contract/round/src/*`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-GRP-VUL-001	Missing list ID validation allows . . .	Critical	Fixed
V-GRP-VUL-002	Unrestricted voting can lead to project . . .	Critical	Partially Fixed
V-GRP-VUL-003	Incorrect validation allows projects to . . .	High	Fixed
V-GRP-VUL-004	Lack of validation for . . .	High	Fixed
V-GRP-VUL-005	Premature payout nullification due to . . .	High	Fixed
V-GRP-VUL-006	Unbounded growth of project payouts . . .	High	Fixed
V-GRP-VUL-007	Storing all project registrations in one . . .	High	Fixed
V-GRP-VUL-008	Ownership change is not persisted to storage	Medium	Fixed
V-GRP-VUL-009	Rounds' owners cannot blacklist users	Medium	Fixed
V-GRP-VUL-010	Challenges mechanism can be used for . . .	Medium	Fixed
V-GRP-VUL-011	Function register_batch is vulnerable to . . .	Medium	Fixed
V-GRP-VUL-012	Collection of approved projects can be . . .	Medium	Fixed
V-GRP-VUL-013	Function deposit_to_round is vulnerable . . .	Medium	Fixed
V-GRP-VUL-014	Referrer fee deduction without recipient . . .	Medium	Fixed
V-GRP-VUL-015	Storage limitations . . .	Medium	Fixed
V-GRP-VUL-016	Admins can steal rounds' funds	Medium	Fixed
V-GRP-VUL-017	Applying to round in batch mode doesn't . . .	Low	Fixed
V-GRP-VUL-018	Round details can be updated during voting	Low	Fixed
V-GRP-VUL-019	Function apply_to_round_batch doesn't . . .	Low	Fixed
V-GRP-VUL-020	Function upvote is vulnerable to griefing . . .	Low	Acknowledged
V-GRP-VUL-021	Function remove_resolved_challenges . . .	Low	Fixed
V-GRP-VUL-022	Admins can maliciously redirect project . . .	Low	Fixed
V-GRP-VUL-023	Project details can be changed any time	Low	Acknowledged
V-GRP-VUL-024	Registration status persists after list deletion	Low	Fixed
V-GRP-VUL-025	Vault redistribution can be triggered before . . .	Low	Fixed
V-GRP-VUL-026	Function review_application can approve . . .	Low	Fixed
V-GRP-VUL-027	Incomplete and redundant parameter . . .	Low	Fixed
V-GRP-VUL-028	Unrestricted deposits can lead to issues	Low	Fixed
V-GRP-VUL-029	Centralization risk	Low	Acknowledged
V-GRP-VUL-030	Lack of upper bound check over protocol . . .	Low	Fixed
V-GRP-VUL-031	Function apply_to_round doesn't check . . .	Low	Fixed
V-GRP-VUL-032	Insufficient input validation in . . .	Low	Fixed
V-GRP-VUL-033	Owner transfer mistake is irreversible	Low	Fixed
V-GRP-VUL-034	Risk of exceeding deposited funds in payouts	Low	Fixed
V-GRP-VUL-035	Incorrect recipient address used for payouts	Low	Fixed
V-GRP-VUL-036	Potential unauthorized deletion of . . .	Low	Fixed

4.1 Detailed Description of Issues

4.1.1 V-GRP-VUL-001: Missing list ID validation allows unauthorized registration changes

Severity	Critical	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	unregister(), update_registration()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/242`, c7b7292		

The `unregister()` and `update_registration()` functions are used to modify a specific registration associated with a given `list_id`. The `unregister()` function allows the owner or admin of a list to delete a particular registration record, while the `update_registration()` function enables them to change the status of a registration.

Although both functions verify that the caller is authorized as the owner or admin of the provided `list_id`, they do not check whether the specified `list_id` matches the `registration.list_id` in the registration record. This missing validation allows a malicious user to create their own list and then use their ownership privileges to manipulate registrations belonging to a different list.

Impact In the context of the `round` contract, this issue allows a malicious user to approve their own registration to the voting whitelist or remove a legitimate user's registration from the whitelist, compromising the integrity of the voting process.

Recommendation Both functions should include a validation step to ensure that the `list_id` used for ownership verification matches the `list_id` stored in the registration record.

Developer Response The developers implemented the suggested fix in both functions: `unregister()` and `update_registration()`.

4.1.2 V-GRP-VUL-002: Unrestricted voting can lead to project votes stuffing

Severity	Critical	**Commit**	902900a
Type	Data Validation	**Status**	Partially Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	vote()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/243`, c9ea4e1		

The `vote()` function of the `round` contract allows voters to submit votes for projects in the round. This function uses two voting eligibility checks: blacklisting and whitelisting, to ensure that only eligible voters can submit their votes. The function also ensures that each voter can only submit their vote once. However, beyond these restrictions, there are no additional measures in place to prevent vote duplication or vote stuffing, potentially leading to manipulation of the voting outcome.

Impact The absence of safeguards against vote duplication creates two attack vectors:

- If a round does not enforce a whitelist, an attacker can generate multiple addresses and cast numerous votes for their preferred project.
- Even if a whitelist is enforced, a whitelisted voter can submit multiple voting entries, each containing the same project and pair index, effectively stuffing votes for a specific project over another project.

Since project funding is dependent on the number of votes received, the attack could directly lead to a loss of value for the protocol.

Recommendation To mitigate vote stuffing and ensure fair voting, the following improvements should be implemented:

- Implement a check to ensure that the votes vector contains only unique entries.
- Ensure that whitelisting is always mandatory to prevent unauthorized addresses from submitting multiple votes.

Developer Response The developers implemented a check to prevent votes for the same pair of projects, however the enforcement of a whitelist remains as an option for the round creator.

4.1.3 V-GRP-VUL-003: Incorrect validation allows projects to apply even after voting has started

Severity	High	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/validation.rs`		
Location(s)	validate_voting_not_started()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/235`, eb8a8d5		

The functions `apply_to_round()` and `unapply_from_round()` in the round contract are responsible for allowing projects to apply or unapply to/from a round. These functions include a validation step intended to ensure that these applications are only processed before the voting period begins. However, as can be observed in the below code snippet, the current implementation incorrectly checks if the current time is less than or equal to `round.voting_end_ms` instead of `round.voting_start_ms`.

```
pub fn validate_voting_not_started(env: &Env, round: &RoundDetail) {
    let current_time = get_ledger_second_as_millis(env);

    if current_time > round.voting_end_ms {
        panic_with_error!(env, VoteError::VotingAlreadyStarted);
    }
}
```

Snippet 4.1: Function `validate_voting_not_started()`.

This logic flaw allows projects to apply or unapply even after the voting has started, as long as the voting period has not ended.

Impact This issue impacts every functionality that assumes that `validate_voting_not_started` validates that the voting period of a round has not started. Particularly, in the current code, this logic flaw allows projects to apply/unapply for the round even after the voting has started, as long as the voting period has not ended.

Recommendation To address this issue, the validation logic in the `validate_voting_not_started` function should be corrected to compare the current time against `round.voting_start_ms` instead of `round.voting_end_ms`.

Developer Response The developers implemented the suggested fix.

4.1.4 V-GRP-VUL-004: Lack of validation for admin_only_registrations flag allows unauthorized registrations

Severity	High	**Commit**	902900a
Type	Authorization	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	register_batch()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/237`, a20b748		

The `register_batch()` function in the `ListsContract` is responsible for handling batch registrations to a list. This function is designed to allow either the list owner or its admins to submit registrations. However, it also permits non-admin users to register if the list is configured to allow public registrations. This configuration is controlled by the `admin_only_registrations` boolean flag within the list's settings.

The issue arises in the logic that handles registration requests from non-admin users. The function does not check the `admin_only_registrations` flag before allowing a non-admin user to register. This oversight means that even if a list is configured to restrict registrations to admins only, non-admin users can still register, bypassing the intended access control.

Impact The lack of validation for the `admin_only_registrations` flag can lead to unauthorized registrations on lists that are meant to be restricted to admin-only access.

Recommendation To address this issue, the `register_batch()` function should include a validation step that checks the `admin_only_registrations` flag before processing a registration request from a non-admin user. If the flag is set to true, the function should reject the registration attempt.

Developer Response The developers implemented the suggested fix.

4.1.5 V-GRP-VUL-005: Premature payout nullification due to unenforced compliance period

Severity	High	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	process_payouts()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/244`, e01b84c		

The `process_payout()` function is responsible for processing and executing payouts for winning projects.

Some rounds require winning projects to complete a KYC verification process before becoming eligible for payment. However, the current implementation does not enforce that the compliance period has ended before executing payouts. If a project has not passed the KYC check at the time the function is executed, its payout is effectively nullified, regardless of whether it could still complete verification within the intended timeframe.

```
1 if round.compliance_period_ms.is_some() {
2   let is_kcy_passed = list_client.is_registered(&config.voting_wl_list_id, &payout.
      recipient_id, &Some(RegistrationStatus::Approved));
3   if !is_kcy_passed {
4       payout.paid_at_ms = Some(get_ledger_second_as_millis(env));
5       write_payout_info(env, payout_id, &payout);
6       return;
7   }
8 }
```

Snippet 4.2: Code snippet from the `process_payout()` function.

Impact Because the round owner can call `process_payout()` at any time, a project that has not yet completed the KYC verification process by the time of execution may lose its payout permanently. This flaw can result in eligible projects being denied funding due to premature execution rather than actual KYC failure. Without proper enforcement of the compliance period, the protocol may inadvertently withhold funds from valid recipients, disrupting the fair distribution of payouts.

Recommendation The function should enforce that the compliance period has ended before executing the payout process. This can be achieved by using the function `assert_compliance_period_complete()`.

Developer Response The developers implemented the suggested fix.

4.1.6 V-GRP-VUL-006: Unbounded growth of project payouts mapping leading to payout deletion

Severity	High	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	ContractKey::ProjectPayoutIds		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/245`, da7350d		

The protocol defines a mapping that associates a `projectID` with a vector of `payoutIDs`. This mapping is stored under the contract key `ProjectPayoutIds`, which represents a Ledger Key in the Soroban storage system. Each Ledger Entry corresponding to this key has a maximum size limit of 128 KB, as per the Soroban resource limits.

The issue arises because any round owner can assign payouts to any project. Since the `ProjectPayoutIds` key does not differentiate between rounds, all payouts for a project - across multiple rounds - are stored in the same Ledger Entry. This enables a malicious round owner to assign an excessive number of dummy payouts to a project, potentially filling up the storage limit.

Impact Not differentiating between round's payouts have two impacts:

1. **Payout Data Loss When Clearing Payouts:**
   When `set_payouts()` is called with `clear_existing = true`, it attempts to remove existing payouts before assigning new ones. However, because all payouts for a project are stored under the same Ledger Entry regardless of round, clearing payouts for a single round unintentionally removes payouts from **all rounds**.

```
if clear_existing {
   clear_payouts(env, round_id);

   approved_project.iter().for_each(|project_id| {
      let payout_ids = read_project_payout_ids_for_project(env, project_id);

       payout_ids.iter().for_each(|payout_id| {
          remove_payout_info(env, payout_id);
       });

       clear_project_payout_ids(env, project_id);
   });
}
```

Snippet 4.3: Code snippet from the `set_payouts()` function.

   This can be abused by a malicious user by creating a dummy round, assign payouts to a legitimate project with assigned payouts of a legitimate round and then calling `set_payouts` and clearing the existing ones.
2. **Denial of Service via Unbounded Iteration in** `process_payouts()`:
   The `process_payouts()` function iterates over all stored payout IDs for a project, even those from other rounds. This is problematic because a malicious actor can artificially

inflate the payout vector to its maximum storage capacity, making the loop unbounded and exceeding Soroban's resource limits.

```
approved_projects.iter().for_each(|project_id| {
     let project_payout_ids = read_project_payout_ids_for_project(env,
    project_id);
     // @audit-issue Unbounded for loop.
     project_payout_ids.iter().for_each(|payout_id| {
        let payout_exist_on_round = round_payouts.contains(payout_id);

        if !payout_exist_on_round {
           return;
        }

 // .., Omitted ....
```

Snippet 4.4: Code snippet from the `process_payouts` function.

Recommendation To mitigate these issues, the protocol should introduce round-specific storage keys for project payouts. Instead of using a single Ledger Key (`ProjectPayoutIds`), it is recommended to modify the contract key structure to include the `round_id`.

The above can be achieved by modifying the Ledger Key from `ProjectPayoutIds` to `ProjectPayoutIds(u128)` where `u128` represents the `round_id`.

This change ensures that each round maintains its own separate payout records, preventing unintended data deletion and limiting the impact of storage abuse.

Developer Response The developers implemented the suggested fix.

4.1.7 V-GRP-VUL-007: Storing all project registrations in one ledger entry causes storage exhaustion

Severity	High	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/project-registry/src/storage_key.rs`		
Location(s)	apply()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/246`, ff528fb		

The protocol defines a mapping that associates an applicant (`address` type) with a project ID (`u128` type). This mapping is stored under the contract key `ApplicantToProjectID`, which represents a Ledger Key in the Soroban storage system. Each Ledger Entry associated with a Ledger Key has a maximum size limit of 128 KB, as specified in the Soroban resource limits.

The issue arises because any user can register a project using the `apply()` function. However, the `ApplicantToProjectID` key does not differentiate between applicants, meaning that all project registrations -regardless of the applicant - are stored within the same Ledger Entry. This design allows a malicious user to register a large number of dummy projects, filling up the storage limit and disrupting the protocol's ability to handle new applications.

Impact If the storage limit is reached, new applicants will be unable to register their projects, leading to a denial-of-service condition for legitimate users.

Recommendation To prevent this issue, the protocol should introduce applicant-specific storage ****keys for project registrations. Instead of using a single Ledger Key (`ApplicantToProjectID`), it is recommended to modify the contract key structure to include the applicant address.

This can be achieved by changing the Ledger Key from:

- `ApplicantToProjectID`

To:

- `ApplicantToProjectID(Address)`

Developer Response The developers implemented the suggested fix.

4.1.8 V-GRP-VUL-008: Ownership change is not persisted to storage

Severity	Medium	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	transfer_ownership()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/239`, 21c9664		

In the `round` contract, there is an issue where the `updated_config` is not written back to storage during the ownership transfer in the `transfer_ownership()` function. The `write_config()` function in the `round/src/config_writer.rs` file is designed to persist configuration changes, including the ownership of the contract, using the `ContractKey::Config` to identify the configuration data in storage.

However, the failure to call `write_config()` after updating the ownership means that the new owner is not set in the storage, leading to a discrepancy between the intended and actual ownership state.

Impact The failure to persist the ownership transfer will result in the system not recognizing the intended new owner which becomes a problem in scenarios where ownership changes are expected to be immediately effective.

Recommendation To resolve this issue, ensure that after updating the ownership in the configuration, the `write_config()` function is called to persist these changes to storage.

Developer Response The developers implemented the suggested fix.

4.1.9 V-GRP-VUL-009: Rounds' owners cannot blacklist users

Severity	Medium	Commit	902900a
Type	Logic Error	Status	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	flag_voters()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/241, ea4d64d		

The `flag_voters` function is designed to add voters to the round's blacklist, preventing them from participating in applying or voting to projects within a given round. The function is expected to validate whether a voter is already blacklisted before attempting to add them to the blacklist. However, the current implementation uses the `validate_not_blacklist()` function, which checks if a voter is blacklisted and reverts if they are not:

```
pub fn validate_not_blacklist(env: &Env, round_id: u128, voter: &Address) {
    let is_blacklisted = is_blacklisted(env, round_id, voter.clone());

    if !is_blacklisted {
        panic_with_error!(env, RoundError::BlacklistNotFound);
    }
}
```

Snippet 4.5: Function `validate_not_blacklist()`.

This logic is incorrect because it causes the function to revert when trying to blacklist a voter who is not already blacklisted, which is the intended action.

Impact The impact of this issue prevents the contract from effectively managing round restrictions. The inability to blacklist users who are not blacklisted undermines the round's owner ability to enforce project applications and voting rules.

Recommendation To resolve this issue, the `flag_voters` function should use instead the function `validate_blacklist`.

Developer Response The developers implemented the suggested fix.

4.1.10 V-GRP-VUL-010: Challenges mechanism can be used for griefing attacks

Severity	Medium	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	challenge_payouts()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/240`, 1ff9edf		

The Grantpicks project implements a challenge mechanism that allows users to dispute assigned payouts before they are finalized. These challenges act as service tickets that must be resolved by round administrators before any payouts can proceed.

If even a single challenge remains unresolved, the following functions cannot be executed:

- `process_payout()`
- `redistribute_vault()`

To regulate when challenges can be made, the protocol defines a cooldown period. This period is intended to ensure that challenges are submitted within a specific timeframe, preventing indefinite disruptions. However, the implementation of `challenge_payouts()` does not enforce this cooldown period, meaning challenges can still be submitted even after the period has elapsed.

Additionally, the function lacks access control, allowing anyone to call it at any time, further increasing the risk of abuse.

Impact This issue introduces a griefing attack vector, where an attacker can continuously call `challenge_payouts()` to create unresolved challenges. Since payouts are blocked until all challenges are resolved, this can result in a complete disruption of the payout process, preventing funds from being distributed as intended.

Recommendation To prevent abuse of the challenge mechanism, it is recommended to enforce stricter conditions on when and by whom challenges can be submitted:

- **Enforce the cooldown period**: Ensure that challenges can only be created within the defined cooldown window. If the cooldown period has expired, the function should reject new challenge submissions.
- **Restrict challenge creation to eligible participants**: Only applicants who are actively participating in the round should be allowed to submit challenges.

Developer Response The developers implemented the suggested fix.

4.1.11 V-GRP-VUL-011: Function register_batch is vulnerable to griefing attacks

Severity	Medium	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	register_batch()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/237`, a922649		

The list contract maintains three mappings related to registrants and registrations:

- A mapping that links a registrant (`address` type) to a vector of list IDs (`Vec<u128>` type), representing the lists to which the account has registered. This mapping is stored under the ledger key `RegistrantList(Address)`.
- A mapping that links a registrant (`address` type) to a vector of registration IDs (`Vec<u128>` type), representing the registrations associated with a particular registrant. This mapping is stored under the ledger key `RegistrationIDs(Address)`.
- A mapping that links a list ID (`u128` type) to a vector of registration IDs (`Vec<u128>` type), representing the registrations attached to a particular list. This mapping is stored under the ledger key `ListRegistration(u128)`.

Each Ledger Entry associated with a Ledger Key is subject to a maximum size limit of 128 KB, as outlined in the Soroban resource limits.

The `register_batch()` function, which can be called by arbitrary users, list owners or admins (on behalf of users), allows accounts to register to specific lists. The registration data is stored in the mappings mentioned above.

The issue arises because this function can be used to perform two types of storage exhaustion attacks:

1. **List-Specific Exhaustion:**
   a) A user can repeatedly register to the same list, filling up the `ListRegistration(list_id)` vector. Since this vector is shared among all users of the list, once it reaches its storage limit, no new registrations can be added by legitimate users.
2. **User-Specific Exhaustion:**
   a) A malicious user can create multiple lists and use `register_batch()` to generate excessive registrations on behalf of a targeted user (victim). This fills up the `RegistrantList(Address)` and `RegistrationIDs(Address)` vectors, preventing the victim from registering for any new lists.

Impact If an attacker exploits this issue, it can result in:

- **Denial of Service for List Owners:** If a critical list (e.g., a whitelisting list) reaches its storage limit, new users will be unable to register, disrupting the intended functionality of the protocol.
- **Denial of Service for Individual Users:** If a targeted user's registration vectors become full, they will be unable to register for new lists, restricting their access to protocol features.

Recommendation

- To mitigate the first attack, it is recommended to implement the fix outlined in the issue: V-GRP-VUL-004. This fix would enable list owners to pause the attack and remove unauthorized registrations.
- To prevent the second attack, arbitrary list owners should not be allowed to create registrations on behalf of users without their explicit consent.

Developer Response The developers fixed the issue by allowing users to remove specific registrations. However, the Veridise team noted that the `submitter` should be removed from this privileged.

4.1.12 V-GRP-VUL-012: Collection of approved projects can be altered during voting

Severity	Medium	Commit	902900a
Type	Data Validation	Status	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	add_approved_project(), remove_approved_project()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/247, 64ab233		

The functions `add_approved_project()` and `remove_approved_project()` allow round admins to add or remove a project that has been approved for participation in the voting phase of the round.

However, the function does not enforce any restrictions on when it can be called. This means projects can be added or removed even during the voting period, potentially disrupting the voting process.

Impact Since votes are submitted as project IDs paired with the pair identifier they belong to in the approved projects vector, adding new projects during the voting phase can alter the pairing structure. This can result in vote rejections or, worse, votes being counted incorrectly due to changes in pair indexing.

1. Initial state is as follows:

```
approved_projects: [1, 2, 3, 4]
all pairs : (1,2) (1,3) (1,4) (2,3) (2, 4) (3, 4)
get_pair_by_index(0) = (1,2)
get_pair_by_index(1) = (1,3)
get_pair_by_index(2) = (1,4)
get_pair_by_index(3) = (2,3)
get_pair_by_index(4) = (2,4)
get_pair_by_index(5) = (3,4)
```

Snippet 4.6: Initial state of the `approved_projects` array.

2. Then someone calls the `add_approved_projects()` and adds another project id = 5. Next state:

```

approved_projects: [1, 2, 3, 4, 5]
all pairs :
  (1,2) (1,3) (1,4) (1, 5) (2,3) (2,4) (2, 5) (3,4) (3, 5) (4, 5)

get_pair_by_index(0) = (1,2)
get_pair_by_index(1) = (1,3)
get_pair_by_index(2) = (1,4)
get_pair_by_index(3) = (1, 5)
get_pair_by_index(4) = (2, 3)
```

Snippet 4.7: State of `approved_projects` after adding a new project.

3. As shown, index `4` now refers to a different pair. If a voter originally cast a vote for project `4` at index `4`, their vote would have been for `(2,4)`. However, after the update, the same index now corresponds to `(2,3)`, leading to an unintended change in vote interpretation.

Recommendation Restrict modifications to the approved projects collection once the voting phase has started to ensure consistency and prevent vote misalignment.

Developer Response The developers implemented the suggested fix.

4.1.13 V-GRP-VUL-013: Function deposit_to_round is vulnerable to grief attacks

Severity	Medium	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	deposit_to_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/268`, 5f6215c		

The round contract maintains a mapping that links a round ID (`u128` type) to a vector of deposit IDs (`u128` type). This mapping is stored under the contract key `Deposit(round_id)`, which serves as a Ledger Key in Soroban's storage system for each round. Each Ledger Entry associated with a Ledger Key is subject to a maximum size limit of 128 KB, as outlined in the Soroban resource limits.

The issue arises in the `deposit_to_round()` function, which facilitates token deposits into a round's vault. These deposits contribute to funding winning projects. However, the function does not impose a minimum deposit amount and allows unrestricted deposits from any address. This opens up a griefing attack vector where a malicious user could repeatedly make zero-value deposits, causing the `Deposit(round_id)` ledger entry to reach its storage limit. Once the ledger entry becomes full, further deposits are blocked, disrupting the normal operation of the protocol.

```
fn deposit_to_round(
   env: &Env,
   round_id: u128,
   caller: Address,
   amount: u128,
   memo: Option<String>,
   referrer_id: Option<Address>,) {
   // ... skipped ...
   write_deposit_id_to_round(env, round_id, deposit_id);
   // ... skipped ...
}
```

Snippet 4.8: Code snippet from the `deposit_to_round()` function.

Impact If the `Deposit(round_id)` ledger entry reaches its maximum storage limit, it will no longer be possible to process additional deposits. This could prevent legitimate users from contributing funds to the round, effectively disrupting the funding process.

Recommendation To mitigate this issue, it is recommended to:

- Enforce the minimum deposit amount to prevent zero-value deposits.
- Implement a limit on the total number of deposits per round to avoid exceeding storage constraints.

Developer Response The developers fixed the issue by implementing a minimum deposit amount variable managed by the round creator.

4.1.14 V-GRP-VUL-014: Referrer fee deduction without recipient causes loss of funds

Severity	Medium	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	deposit_to_round()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/248, 676c3d4		

The `deposit_to_round()` function in the `RoundContract` is responsible for handling deposits into a round. This function calculates a `referrer_fee` based on the deposit amount and deducts it from the total deposit. However, if the `referrer_id` is not provided, the `referrer_fee` is not sent but is still deducted.

```
if referrer_id.is_some() {
   let referrer = referrer_id.unwrap();
   token_client.transfer(
       &env.current_contract_address(),
       &referrer,
       &deposit.referrer_fee,
   );
}
```

Snippet 4.9: Code snippet from the `deposit_to_round()` function.

This results in the `referrer_fee` being effectively lost, as it is not transferred to any address. The logic should ensure that if no `referrer_id` is provided, the `referrer_fee` should not be deducted from the total deposit amount.

Impact The current implementation can lead to a loss of funds, as the `referrer_fee` is deducted from the deposit amount without being transferred to a valid recipient when `referrer_id` is not provided.

Recommendation To address this issue, the contract should include a check to ensure that the `referrer_fee` is only deducted if a valid `referrer_id` is provided. If `referrer_id` is `None`, the `referrer_fee` should not be subtracted from the total deposit amount.

Developer Response The developers implemented the suggested fix.

4.1.15 V-GRP-VUL-015: Storage limitations in votedRoundIds mapping

Severity	Medium	**Commit**	902900a
Type	Denial of Service	**Status**	Fixed
File(s)	`stellar/contract/round/src/storage_key.rs`		
Location(s)	ContractKey::VotedRoundIds		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/249`, 9fca1f2		

The protocol defines a mapping that associates a voter (`Address` type) with a vector of round IDs (`Vec<u128>` type). This mapping is stored under the contract key `VotedRoundIds`, which represents a Ledger Key in the Soroban storage system. Each Ledger Entry associated with a Ledger Key has a maximum size limit of 128 KB, as specified in the Soroban resource limits.

The issue arises because all voting records are stored under a single Ledger Entry, meaning that as more voters participate in different rounds, the storage size continues to grow. This design does not differentiate between individual voters, leading to an eventual exhaustion of storage capacity.

Impact Once the storage limit is reached, new voting records can no longer be added, preventing voters from casting votes in new rounds. This can lead to a denial-of-service condition, disrupting the voting process and impacting the protocol's functionality.

Recommendation To mitigate this issue, the protocol should introduce voter-specific storage keys for voting records. Instead of using a single Ledger Key (`VotedRoundIds`), it is recommended to modify the contract key structure to include the voter's address.

This can be achieved by changing the Ledger Key from:

- `VotedRoundIds`

To:

- `VotedRoundIds(Address)`

This change ensures that each voter's voting history is stored separately, preventing the accumulation of excessive data under a single Ledger Entry and maintaining the protocol's scalability.

Given the required changes to the storage layout, each voter must now track their respective TTL for (`VotedRoundIds(Address)`) and restore it if necessary. It is advisable to document this requirement in the project documentation to ensure clarity and proper communication.

Developer Response The developers implemented the suggested fix.

4.1.16 V-GRP-VUL-016: Admins can steal rounds' funds

Severity	Medium	**Commit**	902900a
Type	Authorization	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_redistribution_config()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/274`, c47c5c4		

The `set_redistribution_config()` function in the `RoundContract` allows administrators to redirect the remaining distribution of funds to any address. This functionality is intended to manage the redistribution of leftover funds after a round's payouts have been processed. However, the current implementation permits administrators to change the redistribution recipient **and** execute this redistribution before the payouts are processed. This creates a potential vulnerability where administrators, if not equally trusted as the round owner, could redirect all the funds to an arbitrary address, effectively misappropriating the funds.

Impact The ability for administrators to redirect funds before payouts are processed can lead to significant financial loss. If an administrator is not as trustworthy as the round owner, they could exploit this function to steal all the funds from a round.

Recommendation To mitigate this risk, it is recommended to restrict the `set_redistribution_config()` function to be callable only by the round owner.

Developer Response The developers implemented the suggested fix.

4.1.17 V-GRP-VUL-017: Applying to round in batch mode doesn't handle video check properly

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	apply_to_round_batch()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/236`, 9aaa0e9		

The `apply_to_round_batch()` function allows admins and the round owner to add multiple projects to a round in a single operation.

As part of its validation process, the function checks whether a round requires projects to include a video link and ensures that all added projects comply with this requirement. However, this check is implemented incorrectly:

```
1 if round.is_video_required {
2    if uproject.has_video {
3      panic_with_error!(env, ApplicationError::VideoUrlNotValid);
4    }
5 }
```

Snippet 4.10: Code snippet from the `apply_to_round_batch()` function.

The function incorrectly panics when a project includes a video, whereas it should revert when a project lacks the required video.

Impact Due to this incorrect validation, round administrators cannot add projects with videos to rounds that require them when using batch processing.

Recommendation The condition should be corrected to ensure that projects are only rejected if they do not include a video when one is required.

Developer Response The developers implemented the suggested fix.

4.1.18 V-GRP-VUL-018: Round details can be updated during voting

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	update_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/272`, 69b7417		

The current implementation of the `round` contract allows modifications to key round parameters, such as the application period, voting whitelist, and expected amount, even after the voting phase has started.

```
fn update_round(env: &Env,
        caller: Address,
        round_id: u128,
        round_detail: UpdateRoundParams,
 ) -> RoundDetail {
    caller.require_auth();
    let mut round = read_round_info(env, round_id);
    validate_owner_or_admin(env, &caller, &round);
    validate_round_detail_update(env, &round_detail);
    // ... skipped ...
}
```

Snippet 4.11: Code snippet from the `update_round()` function.

Impact Allowing modifications to critical round details during the voting phase creates a risk that voters may base their decisions on outdated or inconsistent information.

Recommendation To address this issue, consider to:

- Restrict updates to round details so that they can only be modified before the voting phase begins.
- If certain updates must be allowed during voting, ensure that they do not alter key parameters that could mislead participants or disrupt the voting process.

It is also important to consider the following issue identified in the same function: V-GRP-VUL-027.

Developer Response The developers implemented the suggested fix.

4.1.19 V-GRP-VUL-019: Function apply_to_round_batch doesn't check black- and white- lists

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	apply_to_round_batch()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/256`, cd92497		

The `apply_to_round_batch()` function allows admins to add multiple applications to a round in a single operation. However, unlike the `apply_to_round()` function, it does not enforce whitelisting and blacklisting checks on all applications.

Impact This omission allows admins to add applicants that are blacklisted or that have not been whitelisted, violating the intended protocol rules.

Recommendation Ensure `apply_to_round_batch()` enforces the same whitelisting and blacklisting checks as `apply_to_round()` to maintain consistency and enforce the intended restrictions.

Developer Response The developers implemented the suggested fix.

4.1.20 V-GRP-VUL-020: Function upvote is vulnerable to griefing attacks

Severity	Low	**Commit**	902900a
Type	Denial of Service	**Status**	Acknowledged
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	upvote()		
Confirmed Fix At	N/A		

The `upvote` function in the `lists` contract allows users to "remember" lists they favor. This function is publicly accessible and requires the voter's address and the list identifier they wish to vote for.

A voter can only upvote a list once. However, there are no mechanisms in place to prevent a malicious actor from repeatedly calling this function from multiple different addresses, even if they have no genuine association with the lists being upvoted.

```
fn upvote(env: &Env, voter: Address, list_id: u128) {
   voter.require_auth();
   // ... skipped ...
   validate_upvotes_status(env, &voter, list_id);
   add_upvote_to_list(env, list_id, voter.clone());
   // ... skipped ...
}
```

Snippet 4.12: Code snippet from the `upvote()` function.

Impact Repeated execution of this function for a specific list ID can cause the upvote collection for that list to grow indefinitely. Eventually, this will result in reaching the ledger entry size, preventing legitimate users from interacting with the contract.

Recommendation To prevent this issue, it is recommended to restructure the data storage model to eliminate the possibility of unbounded growth. Specifically, avoid using vectors that can expand indefinitely and consider implementing a more scalable storage mechanism.

Developer Response The developers have reviewed and acknowledged the issue. After careful consideration, they have decided not to implement code changes at this time and have accepted the associated risk.

4.1.21 V-GRP-VUL-021: Function remove_resolved_challenges removes all challenges instead of resolved ones

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	remove_resolved_challenges()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/275`, 9e3dfaf		

The `remove_resolved_challenges()` function is intended to allow round admins to clean up the challenge collection when it becomes bloated with resolved challenges.

However, the current implementation does not properly remove resolved challenges from the existing collection. Instead, it attempts to remove them from an empty temporary collection. As a result, the function replaces the original challenge collection with an empty one, effectively deleting all challenges rather than selectively removing only the resolved ones.

```
fn remove_resolved_challenges(env: &Env, round_id: u128, caller: Address) {
  // ... skipped ...
  validate_owner_or_admin(env, &caller, &round);
  // ... skipped ...
  let mut challenges_internal: Map<Address, PayoutsChallenge> = Map::new(env);
  challenges.keys().iter().for_each(|challenger_id| {
    let challenge = challenges.get(challenger_id.clone()).unwrap();
    if challenge.resolved {
      challenges_internal.remove(challenger_id);
    }
  });
  write_payout_challenges(env, round_id, &challenges_internal);
  extend_instance(env);
  extend_round(env, round_id);
}
```

Snippet 4.13: Code snippet from the `remove_resolved_challenges()` function.

Impact This flaw results in the unintended deletion of all challenges, not just the resolved ones. As a consequence, important unresolved challenges may be lost, potentially leading to disputes or unresolved issues in the payout process.

Recommendation Initialize the `challenges_internal` collection with the existing challenges before attempting to remove resolved entries. This ensures that only resolved challenges are removed while retaining unresolved ones.

Developer Response The developers implemented the suggested fix.

4.1.22 V-GRP-VUL-022: Admins can maliciously redirect project payout funds

Severity	Low	**Commit**	902900a
Type	Authorization	**Status**	Fixed
File(s)	`stellar/contract/project-registry/src/internal.rs`		
Location(s)	update_project()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/276`, 7a48c4f		

The `update_project` function in the `project-registry` contract allows for updating a project's configuration if the action is authorized by either the project owner or one of the registered admins. One of the configurable parameters is the `payout_address`, which specifies the address intended to receive payouts allocated to the project.

Impact Malicious project admins can exploit this functionality to modify the `payout_address`, redirecting funds to an unintended address and potentially misappropriating the project's payouts.

Recommendation If project admins are not held to the same trust standard as the project owner, they should be restricted from modifying the `payout_address`.

When planning a fix for this issue, it is also advisable to consider the issue V-GRP-VUL-035 which recommends getting rid of the `payout_address`.

Developer Response The developers removed the `payout_address`, fixing the issue.

4.1.23 V-GRP-VUL-023: Project details can be changed any time

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Acknowledged
File(s)	`stellar/contract/project-registry/src/internal.rs`		
Location(s)	update_project()		
Confirmed Fix At	N/A		

The `update_project` function in the `project-registry` contract allows changes to various configuration fields of a project, such as `name`, `image_url`, `video_url`, `overview`, `contracts`, `team_members` and `repositories`, at any stage of the project's lifecycle. Since many of these fields may influence voter decisions when selecting between projects, this functionality enables the project owner or admins to modify the project details after voting has occurred, potentially altering the information voters originally relied upon.

Impact A malicious project owner or admin could update the project's description fields to values that differ significantly from the original information presented to voters, undermining the integrity of the voting process.

Recommendation Project's configuration fields intended to inform voter decisions should be immutable.

Developer Response The developers have reviewed and acknowledged the issue. After careful consideration, they have decided not to implement code changes at this time and have accepted the associated risk.

4.1.24 V-GRP-VUL-024: Registration status persists after list deletion

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	is_registered()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/265`, 87cc3a1		

The `is_registered()` function in the `ListContract` is designed to verify if a specific account has a registration for a particular list with a specified status. If the condition is met, the function returns true; otherwise, it returns false.

This function is utilized in the `Round` contract to implement whitelist mechanisms, such as restricting voting rights to certain individuals.

However, the issue arises because the `is_registered()` function continues to return true for an account that was previously registered, even if the underlying list has been deleted using the `delete_list()` function.

Impact The persistence of a true return value from the `is_registered()` function for deleted lists can potentially lead to unauthorized access or actions.

Recommendation To address this issue, modify the `is_registered()` function to check the existence of the underlying list before confirming a registration status. For example:

```
let list = get_list_by_id(env, list_id);
if list.is_none(){
  panic_with_error!(env, Error::ListNotFound);
}
```

Snippet 4.14: Code snippet that verifies the existence of a particular list identified by `list_id`.

Developer Response The developers implemented the suggested fix.

4.1.25 V-GRP-VUL-025: Vault redistribution can be triggered before voting

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	redistribute_vault()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/266`, 27ff829		

The protocol implements the `redistribute_vault()` function to send all the round's fund leftovers to a distinguished address.

The fund leftovers may arise for various reasons, such as when some eligible projects fail to complete KYC in a timely manner or when no projects apply for a particular round.

This function is expected to execute only after the voting, compliance, and cooldown periods have been completed. However, there is a flaw in the current implementation: the function does not check for the completion of the voting period.

```
fn redistribute_vault(env: &Env, round_id: u128, caller: Address, memo: Option<String
    >) {
  caller.require_auth();
  // ... skipped ...
  round.assert_cooldown_period_complete(env);
  round.assert_compliance_period_complete(env);
  round.assert_all_payouts_challenges_resolved(env);
  // ... skipped ...
}
```

Snippet 4.15: Code snippet from the `redistribute_vault()` function.

Impact If neither the cooldown period nor the compliance period is set for the round, all deposited funds can be sent even before voting starts. If this ever happens, the entire funding campaign will come to a halt.

Recommendation It is recommended to enforce the voting period completion check.

Developer Response The developers implemented the suggested fix.

4.1.26 V-GRP-VUL-026: Function review_application can approve the same project more than once

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	review_application()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/259, c559c8c		

The `review_application()` function is responsible for approving submitted applications, making their associated projects eligible for voting during the round's voting phase.

However, the function does not prevent an already approved project from being approved again. If an admin mistakenly approves a project multiple times, its project ID will be added multiple times to the list of approved projects.

```
fn review_application(env: &Env,
   round_id: u128,
   caller: Address,
   applicant: Address,
   status: ApplicationStatus,
   note: Option<String>, ) -> RoundApplication {
   // ... skipped ...
   let mut approved_projects = read_approved_projects(env, round_id);
   if updated_application.status == ApplicationStatus::Approved {
      validate_max_participant(env, &round);
      approved_projects.push_back(updated_application.project_id);
   } else {
   // ... skipped ...
   }
   // ... skipped ...
}
```

Snippet 4.16: Code snippet from the `review_application()` function.

Impact If a project is approved multiple times, its project ID will be duplicated in the approved projects list. This duplication can cause:

- **Issues during the voting phase**: The duplicated project ID may lead to incorrect pairing of projects, affecting the integrity of the vote tallying.
- **Incorrect payout distribution**: Since the payout process relies on the list of approved projects, a project appearing multiple times could receive multiple payouts, leading to an unintended allocation of funds.

Recommendation To prevent duplicate approvals, consider implementing one of the following solutions:

- Check that the new status is different to the actual one, and revert in case it is not
- Add the project identifier to the approved projects list only in case there is none yet. This option gives an opportunity to change the `review_note` of the previously approved project.

Developer Response The developers implemented the second recommendation.

4.1.27 V-GRP-VUL-027: Incomplete and redundant parameter updates in update_round

Severity	Low	**Commit**	902900a
Type	Usability Issue	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	update_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/262`, c5166cf		

The `update_round()` function allows the round owner or its admins to modify several key parameters of an ongoing round. However, certain parameters are not being updated as expected, while others are updated redundantly despite having designated functions for modification.

Missing Updates:

- The `use_vault` parameter from `UpdateRoundParams` is ignored and does not get applied.
- The round's `application_wl_list_id` is missing from `UpdateRoundParams` and is not updated, even though it should be included.
- The `referrer_fee_basis_points` parameter is also absent from `UpdateRoundParams`. Unlike other parameters, it does not have a designated function for updates, making its omission a potential oversight.

Parameters that are updated, even though they have designated update functions:

- The `allow_applications` parameter is updated within `update_round()`, but it should only be modified via the `set_application_config()` function, which includes necessary validation checks.
- The `expected_amount` parameter is also updated within `update_round()`, even though it should be modified through the `set_expected_amount()` function, which ensures proper validation.

Impact Failing to update certain parameters could lead to significant usability issues in a live environment, potentially disrupting the intended round configuration. On the other hand, updating parameters that have dedicated functions bypasses crucial validation checks, increasing the risk of misconfiguration.

Recommendation To resolve these issues:

- Ensure that all essential parameters, such as `use_vault`, `application_wl_list_id`, and `referrer_fee_basis_points`, are properly included and updated in `update_round()`.
- Remove updates to parameters that have dedicated functions (`allow_applications` and `expected_amount`). These should only be modified via their respective functions to maintain validation integrity and prevent unintended misconfigurations.

When planning a fix for this issue, is advisable to consider one of the minor issues highlighted in Typos, redundant code and other minor issues which recommend:

- Getting rid of the `application_wl_list` as this is never used in the contract.

Developer Response The developers implemented the suggested fixes.

4.1.28 V-GRP-VUL-028: Unrestricted deposits can lead to issues

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	deposit_to_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/268`, 0006ee2		

The `deposit_to_round()` function allows users to contribute tokens to a round's vault. However, the function does not impose any restrictions on the time period during which deposits can be made, except when the round is deleted.

Impact

- If a deposit is made after the redistribution event, the deposited funds will remain locked in the contract's balance indefinitely, as redistribution can only occur once.
- A completed round cannot be deleted if it has a remaining balance.
- Allowing deposits during the voting phase may affect voter decision-making, as the distribution of funds continues to change during the process.

Recommendation It is recommended that the time for deposits be restricted before the voting period starts.

Developer Response The developers only allow deposits before payouts are assigned.

4.1.29 V-GRP-VUL-029: Centralization risk

Severity	Low	**Commit**	902900a
Type	Authorization	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

The Grantpicks protocol aims to ensure transparent, efficient, and fair distribution of grants to qualified projects. However, in this current iteration, it was identified that several design choices introduce a centralization aspect which gives too much control to a small group of influential users.

Round contract owner is able execute the following privileged operations:

- Change all the contracts code logic.
- Control KYC process for projects applying to any round.

Round owner and its administrators are able to execute the following privileged operations :

- Assign payouts without a clear correlation to voting results - there is no direct connection between the voting process and the assigned payments.
- Resolve challenges opened by users in case of potential disputes.

Impact

- If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project.

Recommendation Privileged operations should be operated by a multi-sig contract or decentralized governance system. Non-emergency privileged operations should be guarded by a timelock to ensure there is enough time for incident response. The risks in this issue may be partially mitigated by validating that the protocol is deployed with the appropriate roles granted to timelock and multi-sig contracts.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

1. Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
2. Using separate keys for each separate function.
3. Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
4. Enabling 2FA for key management accounts. SMS should **not** be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
5. Validating that no party has control over multiple multi-sig keys.
6. Performing regularly scheduled key rotations for high-frequency operations.
7. Securely storing physical, non-digital backups for critical keys.

8. Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
9. Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

Developer Response The development team has reviewed and acknowledged the issue. After careful consideration, they have decided not to implement code changes at this time and have accepted the associated risk.

4.1.30 V-GRP-VUL-030: Lack of upper bound check over protocol and referrer fees

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	initialize(), owner_set_protocol_fee_config(), create_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/260`, adf5705		

The `initialize()` and `owner_set_protocol_fee_config()` functions allow setting the `protocol_fee_basis_points`, which determines the fee percentage applied to deposits allocated to the GrantPicks protocol. However, there is no upper bound check to ensure that this value does not exceed a reasonable maximum, such as 10,000 basis points (100%).

A similar issue exists in the `create_round()` function, where `referrer_fee_basis_points` determines the fee percentage allocated to the specified referrer. This value also lacks an upper bound validation.

The absence of this check means that an excessively high fee could be set, either accidentally or maliciously, which could lead to financial losses for users interacting with the contract. This oversight could undermine the trust and usability of the contract, as users may be subjected to unexpectedly high fees.

Impact Without an upper limit on `protocol_fee_basis_points` and `referrer_fee_basis_points`, the contract allows configurations where fees could be set to extremely high levels. This could result in users losing a significant portion-or even the entirety-of their deposits to fees.

Recommended Mitigation Steps To mitigate this issue, implement a validation step in the `initialize()`, `owner_set_protocol_fee_config()`, `create_round()` functions to ensure that the `protocol_fee_basis_points` and `referrer_fee_basis_points` do not exceed a predefined maximum value.

Developer Response The developers implemented the suggested fix.

4.1.31 V-GRP-VUL-031: Function apply_to_round doesn't check application period

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	apply_to_round()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/257`, 32263f8		

The `apply_to_round()` function of the `round` contract does not ensure that the time of invocation is consistent with the application period, in the event that it is invoked by administrators or the round owner.

```
if is_owner_or_admin {
  validate_voting_not_started(env, &round);
} else {
  validate_application_period(env, &round);
  validate_blacklist(env, round_id, &caller);
}
```

Snippet 4.17: Code snippet from the `apply_to_round()` function.

Impact This issue introduces inconsistency with the `review_application()` function which does enforce the owner and admins to invoke it only during the application period.

Recommendation It is recommended that the function be allowed to execute only during the specified application period.

Developer Response The developers implemented a different fix than recommended. Instead of adding the application period check on the `apply_to_round()` function, they removed the check in the `review_application()` function and added a new check to ensure that the voting period has not started.

The Veridise team confirms that this fixed the inconsistency between both of the functions.

4.1.32 V-GRP-VUL-032: Insufficient input validation in set_voting_period function

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_voting_period()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/258`, 5c5e47b		

The `set_voting_period()` function in the `round` contract allows users to modify the voting period by setting new start and end timestamps. However, the function currently performs only a single validation check, which ensures that the voting start time is earlier than the voting end time:

```
1 if start_ms > end_ms {
2   panic_with_error!(env, RoundError::VotingStartGreaterThanVotingEnd);
3 }
```

Snippet 4.18: Code snippet from the `set_voting_period()` function.

While this validation prevents an obviously incorrect configuration, it does not account for other cases that could negatively impact the voting process.

Impact An improperly configured voting period can disrupt the voting process. Specifically, the following issues may arise:

- **Overlapping Periods**: If an application period exists, the voting period could be set to start before applications have ended, leading to inconsistencies in participation.
- **Unreasonably Short Voting Windows**: If no minimum duration is enforced, an admin could configure a voting period that is too short for meaningful participation.
- **Setting Voting Periods in the Past**: Without a check to ensure timestamps are in the future, an admin could mistakenly or maliciously set the voting period to an already expired time, rendering the round invalid.

Recommendation In addition to the current check, it is recommended to introduce the following additional checks:

- Ensure compatibility with the application period. If an application period is defined, enforce that the new voting period start is greater than the current application period end.
- Define a minimum required duration for the voting period to ensure that participants have sufficient time to vote.
- Ensure that the provided timestamps are in the future and are both greater than the current block's timestamp.

Developer Response The developers implemented the suggested fixes.

4.1.33 V-GRP-VUL-033: Owner transfer mistake is irreversible

Severity	Low	**Commit**	902900a
Type	Maintainability	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	transfer_ownership()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/277`, 8a77790		

The `transfer_ownership()` function directly transfers owner rights without any confirmation or acceptance from the proposed new owner. This means that if the current owner mistakenly sets the wrong address, the transfer is immediate and irreversible.

Impact The impact of this issue could be significant, as an error in transferring owner rights will lead to a loss of control over the contract's administrative functions including potential upgrades.

Recommendation To address this issue, it is recommended to implement a two-step transfer process for owner rights. This process should include an initial request to transfer owner rights, followed by a requirement for the proposed new owner to accept the transfer.

Developer Response The developers implemented the suggested fix.

4.1.34 V-GRP-VUL-034: Risk of exceeding deposited funds in payouts

Severity	Low	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_payouts()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/261`, bec2e22		

The `set_payouts()` function in the `RoundContract` is responsible for assigning payouts to approved projects within a round. This function allows for the assignment of multiple payouts, which are then aggregated to ensure they do not exceed the total funds deposited for that round. However, the current implementation only checks the total of newly assigned payouts against the vault balance, without considering previously assigned payouts.

The relevant logic is found in the `set_payouts()` function, where the `running_total` of payouts is compared to the `vault_balance`. The check is intended to prevent assigning more funds than deposited, but it does not account for the cumulative effect of multiple payout assignments over time.

```
1 if running_total > vault_balance {
2    panic_with_error!(env, RoundError::InsufficientFunds);
3 }
```

Snippet 4.19: Code snippet from the `set_payouts()` function.

Impact While the security analysts did not find a direct way to exploit this issue, it presents a risk of over-allocating funds, which could lead to financial discrepancies and potential loss of funds if upgrades are made to the protocol.

Recommendation To mitigate this issue, it is recommended to modify the logic in `set_payouts()` to include the total of all existing payouts when performing the check against the `vault_balance`. This would ensure that the sum of all payouts, both new and existing, does not exceed the available funds.

Developer Response The developers implemented the suggested fix.

4.1.35 V-GRP-VUL-035: Incorrect recipient address used for payouts

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_payouts()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/276`, 7a48c4f		

The `set_payouts()` function in the `RoundContract` is responsible for setting up payouts for approved projects. The function currently assigns the payout recipient to `payout_input.recipient_id`, which corresponds to the applicant of the project. However, when an applicant registers a project, they specify a payout address intended to receive funds from payouts. This discrepancy means that the funds may not be directed to the intended recipient, potentially leading to misallocation of funds.

The function `process_payouts` also uses `payout_input.recipient_id` for KYC checks, which raises the question of whether the KYC check should be performed on the payout address specified by the project or the applicant of the project. This ambiguity could lead to compliance issues if the wrong entity is verified.

Impact The current implementation could result in funds being sent to the wrong address.

Recommended Mitigation Steps There are two possible approaches to address this issue:

- Remove the `payout_address` field if it is redundant and does not serve a necessary function in the protocol. Eliminating this field would prevent confusion and ensure consistency in fund distribution.
- Modify the `set_payouts()` function to use the payout address specified by the project for fund distribution. If this approach is taken, it is essential to determine whether the KYC check should be performed on the payout address or the applicant.

Developer Response The developers removed the `payout_address` field and implemented the KYC check against the project owner.

4.1.36 V-GRP-VUL-036: Potential unauthorized deletion of registrations by untrusted account

Severity	Low	**Commit**	902900a
Type	Authorization	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	unregister()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/269`, c99d1ff		

The `unregister()` function in the `ListsContract` allows the `registered_by` account to delete a registration. This functionality is part of a broader registration management system where list owners or admins can register users on their behalf. The issue arises because the `registered_by` account retains the ability to delete the registration even if they are no longer a trusted account within the list's configuration. This is problematic because the trust assumptions can change over time, such as when admins are removed or ownership changes. The logic should ensure that the `registered_by` account is still a trusted entity before allowing them to delete a registration.

```
if uregistration.registrant_id != submitter && !is_admin_or_owner {
      if uregistration.registered_by != submitter {
         panic_with_error!(env, Error::AdminOrOwnerOnly);
      }
 }
```

Snippet 4.20: Code snippet from the `unregister()` function.

Impact This vulnerability allows potentially unauthorized accounts to delete registrations. The assumption here is that the `registered_by` account should only have deletion rights if they are still a trusted entity within the list's current configuration.

Recommended Mitigation Steps To mitigate this issue, there a two options:

- Remove the possibility for the account `registered_by` to unregister an applicant.
- Or, the code should include a verification step to ensure that the `registered_by` account is still a trusted entity within the list's configuration before allowing them to delete a registration. This could involve checking the current list of admins or the current owner to confirm that the `registered_by` account is still authorized to perform such actions.

Developer Response The developers implemented the first suggested fix.

4.1.37 V-GRP-VUL-037: Identical pairs in get_pairs_to_vote function

Severity	Low	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	get_pairs_to_vote()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/270, b2d9200		

The `get_pairs_to_vote()` function in the `round` contract generates a set of project pairs that users must compare before submitting their votes. This function relies on Soroban's pseudo-random number generator to create these pairs.

However, since the same number can be generated multiple times, the function may return duplicate pairs within the list.

```
fn get_pairs_to_vote(env: &Env, round_id: u128) -> Vec<Pair> {
  let round = read_round_info(env, round_id);
  let pairs = get_random_pairs(env, round_id, round.num_picks_per_voter);
  pairs
}
```

Snippet 4.21: Code snippet from the `get_pairs_to_vote()` function.

Impact If duplicate pairs are included in the list:

- **Users may be required to compare the same projects pair multiple times**, affecting the efficiency of the voting process.
- **The voting process may be skewed**, as certain projects could appear more frequently than others, leading to an imbalance in vote distribution.
- **Potential inconsistencies in downstream processes** that rely on these comparisons for final vote aggregation.

Recommendation It is recommended to ensure that the returned list doesn't contain duplicate pairs.

Developer Response The developers implemented the suggested fix.

4.1.38 V-GRP-VUL-038: Round can be completed several times

Severity	Warning	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_round_complete()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/238`, f6faf3b		

The `set_round_complete()` function is used by round admins to mark a round as complete, preventing further protocol operations on it.

However, the function does not check whether the round has already been completed before marking it as complete again.

Impact Multiple admins can call `set_round_complete()` on the same round, leading to redundant executions, unnecessary gas costs, and potentially inconsistent timestamps.

Recommendation Modify the function to ensure that a round can only be marked as complete once. If the round is already completed, the function should reject further calls.

Developer Response The developers implemented the suggested fix.

4.1.39 V-GRP-VUL-039: Predicate can_vote isn't consistent with vote function

Severity	Warning	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	can_vote()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/250`, a006f0e		

The view function `can_vote()` is used by voters to check their eligibility before calling the `vote()` function.

However, the current implementation of `can_vote()` is inconsistent with the `vote()` function. Specifically, `can_vote()`returns `true` if the account is on the whitelist, without considering whether the account is also on the blacklist. In contrast, the `vote()` function reverts if an account appears in both the whitelist and the blacklist.

Impact If an account is present in both the whitelist and the blacklist, `can_vote()` will incorrectly indicate that the account is eligible to vote, while `vote()` will fail when called. This inconsistency can lead to confusion for users and unnecessary failed transactions, resulting in wasted gas fees.

Recommendation This issue can be resolved in several ways:

- Make the `can_vote()` function checks are consistent with the `vote()` function checks, or
- Implement checks that would ensure that no account is able to reside on the whitelist and the blacklist simultaneously, or
- Make the whitelisting and the blacklisting mechanisms exclusive, i.e. only one of them has to used at a time.

Developer Response The developers changed the `can_vote()` function to be consistent with the behavior of the `vote()` function.

4.1.40 V-GRP-VUL-040: Predicate can_vote wrongly returns true in corner cases

Severity	Warning	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	can_vote()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/251`, bf2e4cc		

The `can_vote()` function allows potential voters to check their voting eligibility. However, in a specific case, the function provides misleading results. If a round is set to require whitelisting but no whitelist identifier is provided, the function incorrectly returns `true`, even though the `vote()` function will revert in this scenario.

```
fn can_vote(env: &Env, round_id: u128, voter: Address) -> bool {
  let round = read_round_info(env, round_id);
  let current_ms = get_ledger_second_as_millis(env);
  if round.voting_start_ms <= current_ms && current_ms <= round.voting_end_ms {
    if round.use_whitelist_voting {
      if round.voting_wl_list_id.is_none() {
        return true;
      }
      // ... skipped ...
     }
     // ... skipped ...
   }
   false
}
```

Snippet 4.22: Code snippet from `can_vote()` function.

Impact If no whitelist identifier is provided for a round that requires whitelisting, potential voters may falsely assume they are eligible to vote. However, any attempt to call `vote()` will result in a transaction failure, leading to confusion and wasted gas fees.

Recommendation Ensure that when a round is configured to require whitelisting, a whitelist identifier is also provided. If no whitelist identifier exists, `can_vote()` should return `false` to align with the actual behavior of `vote()`.

Developer Response The developers implemented the suggested fix.

4.1.41 V-GRP-VUL-041: Failure to update the updated_ms field

Severity	Warning	**Commit**	902900a
Type	Maintainability	**Status**	Fixed
File(s)	`stellar/contract/project-registry/src/internal.rsstellar/contract/lists/src/internal.rs`		
Location(s)	update_project(), update_list(), update_registration()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/252`, e18c1bb		

The functions `update_project()`, `update_list()`, and `update_registration()` in the `ProjectRegistry` and `ListsContract` contracts are responsible for updating various parameters of projects, lists, and registrations. However, these functions fail to update the `updated_ms` field, which is intended to track the last modification timestamp of these entities.

Impact The failure to update the `updated_ms` field can lead to a lack of transparency and accountability in projects and lists management. For example, any logic that relies on the `updated_ms` field to trigger actions based on the last update time will be compromised, potentially leading to missed updates or incorrect data processing.

Recommendation To address this issue, ensure that the `updated_ms` field is updated with the current timestamp whenever the `update_project()`, `update_list()` and `update_registration()` functions are called.

Developer Response The developers implemented the suggested fix.

4.1.42 V-GRP-VUL-042: Incorrect cover_img_url assignment in list getters

Severity	Warning	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/lists/src/internal.rs`		
Location(s)	get_list(), get_lists(), get_lists_for_owner(), get_lists_for_registrant(), get_upvoted_lists_for_account()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/253`, de483a6		

The list getter functions in the `ListContract` such as: `get_list()`, `get_lists()`, `get_lists_for_owner()`, `get_lists_for_registrant()` and `get_upvoted_lists_for_account()`, incorrectly set the `cover_img_url` field to the value of the `description` field when constructing the list's fields to return.

Impact The incorrect assignment of the `cover_img_url` field in the list getter functions results in the `cover_img_url` being set to the value of the `description` field. This leads to incorrect data being returned to the caller, which can cause confusion and potential errors in any application logic that relies on the correct URL for the cover image.

Recommendation To resolve this issue, update the list getter functions to correctly assign the `cover_img_url` field with the `cover_image_url` value.

Developer Response The developers implemented the suggested fix.

4.1.43 V-GRP-VUL-043: Function set_number_of_votes isn't restricted enough

Severity	Warning	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	set_number_of_votes()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/254, b60a08c		

The `set_number_of_votes()` function allows an admin to set the required number of votes per submission. However, the current implementation has some issues that could lead to inconsistencies in the voting process:

- The new required number of votes is not validated to be smaller than the number of possible pairs for the approved projects.
- Unlike in `create_round()`, there is no restriction ensuring that the value does not exceed 10, which is enforced through the `validate_pick_per_votes` function.

```
fn set_number_of_votes(env: &Env, round_id: u128, admin: Address, num_picks_per_voter
    : u32) {
  admin.require_auth();
  let mut round = read_round_info(env, round_id);
  validate_owner_or_admin(env, &admin, &round);
  if num_picks_per_voter < 1 {
    panic_with_error!(env, VoteError::EmptyVote);
  }
  let states = read_voting_state(env, round_id);
  let votes = states.len();
  if votes > 0 {
    panic_with_error!(env, RoundError::VotesAlreadyCast);
  }
  // ... skipped ...
}
```

Snippet 4.23: Code snippet from the `set_number_of_votes()` function.

Impact

- If the required number of votes is set to a value greater than the number of possible project pairs, voters may be unable to cast a valid vote, leading to rejected submissions.
- The absence of an upper bound check (as enforced in `create_round()`) allows an excessively high number of votes per submission.

Recommendation To prevent these issues:

- Ensure that the new number of required votes does not exceed the number of possible project pairs for the approved projects.
- Apply the `validate_pick_per_votes()` function within `set_number_of_votes()` to maintain consistency with the constraints enforced during round creation.

Developer Response The developers implemented the suggested fix.

4.1.44 V-GRP-VUL-044: Typos, redundant code and other minor issues

Severity	Warning	**Commit**	902900a
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

Unused Functions and Variables

- In the `process_payout()` function, the variable `total_amount_paid` is calculated but never used.
- In the `initialize()` function, the parameter `application_wl_list_id` is assigned a value but is not utilized anywhere in the protocol.
- The function `read_old_projects()` in `project_writer.rs` is not referenced in the `project registry` contract.
- The function `assert_cooldown_period_in_process()` in `data_types.rs` is not used in the `round` contract.
- The `RoundError` enum contains the unused constants: `UserAlreadyBlacklisted`, `AdminNotFound`, `UserWhitelisted`, and `ZeroValutBalance`.

Duplicate Functions

- The `validate_round_detail_update()` function is identical to `validate_round_detaul()`, making one of them redundant.

Misleading Function and Parameter Names

- The function `validate_has_voted()` should be renamed to `validate_has_not_voted()` to better reflect its purpose.
- The function `validate_upvotes_status()` should be renamed to `validate_has_not_upvoted_list()` for clarity.
- The parameter `voting_wl_list_id` in the `initialize()` function should be renamed to `kyc_list_id` to better represent its purpose.

Commented-Out Code That Should Be Removed

- The `add_voting_result()` function.
- The `migrate()` function.
- The `validate_application()` function contains commented-out code inside its body.
- The `validate_update_project()` function contains commented-out code inside its body.

Redundant Code

- Line 546 in the `vote()` function is unnecessary and should be removed.

Potential Panic Without Clear Reason

- In the `process_payout()` function, if the `voting_wl_list_id` variable does not contain a valid ID, the call on line 693 will revert without providing a specific error message.

Impact

- Unused variables and functions introduce unnecessary complexity and increase maintenance overhead.
- Redundant functions can cause confusion and potential inconsistencies.
- Misleading function and parameter names reduce code readability and increase the likelihood of incorrect usage.
- Commented-out code may indicate incomplete implementations or deprecated logic that should be removed for clarity.
- Unclear error handling can make debugging difficult when a function unexpectedly reverts without an informative message.

Recommendation

- Remove unused functions, variables, and redundant code.
- Rename misleading functions and parameters for better clarity.
- Remove commented-out and deprecated code to maintain a clean codebase.
- Ensure error messages are explicitly stated when reverting execution.

Developer Response The developers implemented the suggested fixes.

4.1.45 V-GRP-VUL-045: Insufficient sanity checks over rounds' periods

Severity	Warning	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	validate_round_detail()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/255`, e948d74		

When creating a round using the `create_round` function in the `round` contract, the creator must provide parameters for round creation via `CreateRoundParams`. These parameters are validated by the `validate_round_details()` function, which ensures that the start of the application and voting periods is earlier than their respective end times.

However, the validation process lacks two checks:

1. It does not prevent setting the start of these periods in the past.
2. It does not enforce a minimum duration between the start and end times of these periods.

Impact Allowing the start of the application or voting periods to be set in the past can result in rounds that are immediately invalid or unusable, disrupting the intended process. Additionally, without enforcing a minimum duration for these periods, rounds could be created with extremely short application or voting windows, limiting participation and potentially leading to an unfair process. These issues could cause operational inconsistencies, where rounds are either ineffective or fail to provide sufficient time for users to engage.

Recommendation To mitigate these issues, the following validations should be added:

1. Ensure that the start of the application period is greater than the current timestamp of the ledger. If no application period is set, the voting period start time must be greater than the current timestamp.
2. Enforce a minimum delay between the current timestamp and the start of the application/voting period.
3. Define a minimum duration for both the application and voting periods to ensure fair participation and prevent rounds with impractically short timeframes.

Developer Response The developers implemented the suggested fix.

4.1.46 V-GRP-VUL-046: Function is_registered behaves in a strange way

Severity	Warning	**Commit**	902900a
Type	Maintainability	**Status**	Fixed
File(s)	stellar/contract/lists/src/internal.rs		
Location(s)	is_registered()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/265, f2827f9		

The function `is_registered` in the `lists` contract is used to check whether a given account has been added to a given list with a given status. However, there is a strange behavior when the `list_id` and `required_status` fields are both set to `None`. In such a case, the function will return `true` if the provided account has any existing registration on any list.

```
1  fn is_registered(env: &Env,
2     list_id: Option<u128>,
3     registrant_id: Address,
4     required_status: Option<RegistrationStatus>,
5  ) -> bool {
6     extend_instance(env);
7     let registration_ids = get_user_registration_ids_of(env, &registrant_id);
8     if required_status.is_none() && list_id.is_none() {
9        return !registration_ids.is_empty();
10    }
11    // ... skipped ...
12 }
```

Snippet 4.24: Code snippet from the `is_registered()` function.

Impact Since anyone can create a list and register a given `registrant_id` with it, this function can be used to trick it into returning `true` if needed. This behavior could introduce security vulnerabilities in other contexts.

Recommendation It is recommended to remove this behavior or return `false` in this case.

Developer Response The developers implemented the suggested fix.

4.1.47 V-GRP-VUL-047: Inconsistency in assigning update_ms field

Severity	Warning	**Commit**	902900a
Type	Maintainability	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs` , `stellar/contract/lists/src/internal.rs`		
Location(s)	apply_to_round() , apply_to_round_batch() , create_list()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/267`, 0e441fa		

The `lists` and `round` smart contracts both operate on data structures that have a field named `updated_ms`. While both are used for the same purpose of providing information about the last update time for some element, the logic behind how this field is initialized is inconsistent.

In the case of the `lists` contract, the field is initialized to the current time. However, in the case of the `round` contract, the function `apply_to_round()` and `apply_to_round_batch()` initialize the field to `None`.

Impact It is a matter of personal preference how one chooses to initialize a variable, but using different methods can lead to inconsistencies between modules, which can cause confusion and potential problems with maintenance and usability.

Recommendation It is recommended to ensure consistency in the initialization logic among contracts.

Developer Response The developers fixed the inconsistency by initializing the `update_ms` to the current time.

4.1.48 V-GRP-VUL-048: Function set_expected_amount doesn't check the time of invocation

Severity	Warning	**Commit**	902900a
Type	Logic Error	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_expected_amount()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/273`, 875fc87		

The function `set_expected_amount()` is used to update the `expected_amount` parameter of a round. This parameter has an informational purpose for voters and is not intended to change during the voting phase. However, the current implementation does not check the timing of the invocation and allows changes to this value until the round has finished.

```
1 if round.round_complete_ms.is_some() {
2   panic_with_error!(env, RoundError::RoundAlreadyCompleted);
3 }
```

Snippet 4.25: Code snippet from the `set_expected_amount()` function.

Impact If the value is changed during the voting process, the behavior of voters may be influenced, and the voting will not be fair.

Recommendation It is recommended to restrict the time during which this function can be invoked.

Developer Response The developers restricted the time to be before the voting period starts.

4.1.49 V-GRP-VUL-049: Payouts can be set before voting ends

Severity	Warning	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_payouts()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/263`, db74dc1		

The `set_payouts()` function in the `RoundContract` is responsible for setting the payouts for a round. This function allows the contract owner or an authorized admin to specify the payout details for approved projects. However, the current implementation lacks an important validation step: it does not check whether the voting phase for the round has been completed before allowing payouts to be set.

Impact The absence of a check to ensure that the voting phase is over before setting payouts can lead to premature allocation of funds, potentially based on incomplete or inaccurate voting results.

Recommended Mitigation Steps To mitigate this issue, implement a validation step within the `set_payouts()` function to ensure that the voting phase has concluded before allowing any payouts to be set.

Developer Response The developers implemented the suggested fix.

4.1.50 V-GRP-VUL-050: Insufficient checks when setting new application period

Severity	Warning	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	`stellar/contract/round/src/internal.rs`		
Location(s)	set_application_config()		
Confirmed Fix At	`https://github.com/PotLock/grantpicks/pull/264`, 14f6f9f		

The `set_applications_config()` function is responsible for configuring the application period for a round. It takes parameters to set whether applications are allowed, and optionally, the start and end times for the application period.

The function currently lacks adequate validation checks to ensure the integrity of the application period configuration. The function should ensure that:

- The application start time is not set in the past.
- The duration between the start and end times is meaningful, i.e., it meets a minimum duration requirement.
- The application end time is set before the voting start time to prevent overlap between application and voting periods.

Additionally, when `allow_applications` is set to `false`, the function should automatically set both `start_ms` and `end_ms` to `None`, regardless of their current values, to clearly indicate that no application period is active.

Impact Without these validations, the contract may allow for invalid application periods, such as those starting in the past or overlapping with voting periods

Recommended Mitigation Steps Implement checks to ensure that the application start time is not in the past, the duration between start and end times is meaningful, and the application end time is before the voting start time. Additionally, when `allow_applications` is set to `false`, ensure that both `start_ms` and `end_ms` are set to `None`.

Developer Response The developers implemented the suggested fix.

4.1.51 V-GRP-VUL-051: Lists are not checked for existence before updating

Severity	Warning	**Commit**	902900a
Type	Data Validation	**Status**	Fixed
File(s)	stellar/contract/round/src/internal.rs		
Location(s)	change_voting_wl_list_id(), change_application_wl_list_id()		
Confirmed Fix At	https://github.com/PotLock/grantpicks/pull/271, 434a0c3		

The functions `change_application_wl_list_id()` and `change_voting_wl_list_id()` are used by admins to change the previously assigned list identifiers. However, neither of these functions ensures that the newly assigned list ID points to an existing list.

```
fn change_voting_wl_list_id(env: &Env, list_id: u128){
  let config = read_config(env);
  config.owner.require_auth();
  let mut updated_config = config.clone();
  updated_config.voting_wl_list_id = Some(list_id);
  write_config(env, &updated_config);
}
```

Snippet 4.26: Snippet from `change_voting_wl_list_id()`

Impact Since these lists are used in the protocol for eligibility checks, passing an incorrect list identifier will lead to denial of service.

Recommendation It is recommended to implement the list existence check before storing the new identifier.

When planning a fix for this issue, is advisable to consider some of the minor issues highlighted in Typos, redundant code and other minor issues which recommend:

- Getting rid of the `application_wl_list` as this is never used in the contract.
- Changing the name of the `voting_wl_list_id` to `kyc_list_id` to better represent its purpose in the protocol.

Developer Response The developers implemented the suggested fix.