Switchboard Onchain

Security Assessment

August 8th, 2024 — Prepared by OtterSec

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Switchboard engaged OtterSec to assess the `on-demand` program. This assessment was conducted between May 27th and August 8th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 23 findings throughout this audit engagement.

In particular, we identified several critical vulnerabilities, including the discarding of errors from authority and access control checks, allowing unauthorized users to change permissions on any oracle or pull feed account (OS-SVB-ADV-00).

Moreover, an oracle's delegation pool is not added to its corresponding delegation group within the program state, which prevents proper epoch advancement and reward distribution (OS-SVB-ADV-01).

We also made recommendations around modifications to the codebase for improved efficiency (OS-SVB-SUG-04) and suggested the need to ensure adherence to coding best practices (OS-SVB-SUG-05). Additionally, we advised the removal of unutilized and redundant code within the system for increased readability (OS-SVB-SUG-06,OS-SVB-SUG-07), and recommended implementing a two-step process to change the authority of the state, queue, oracle, and pull feed accounts (OS-SVB-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/switchboard-xyz/sbv3. This audit was performed against commit 905f442.

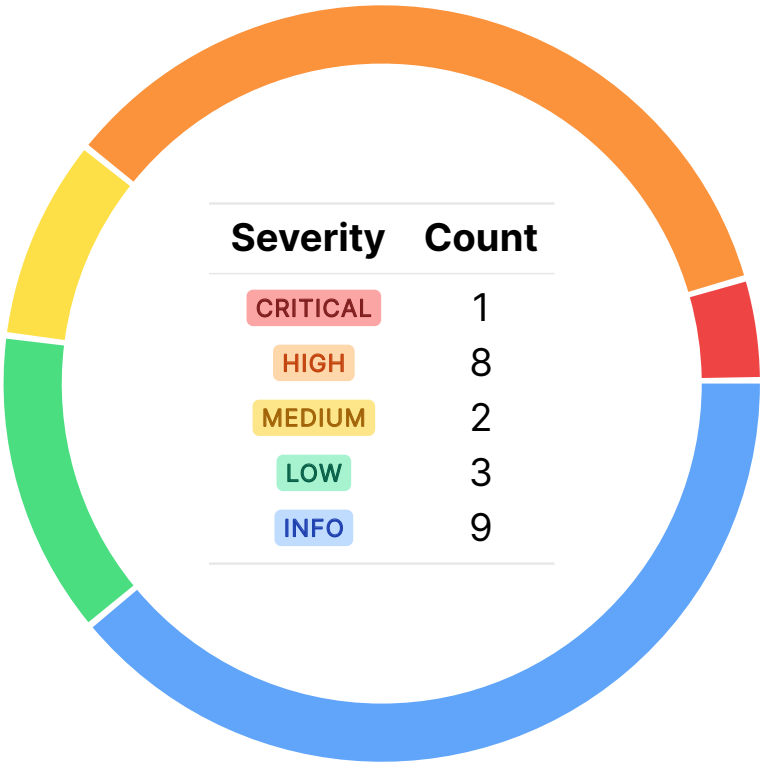
A brief description of the programs is as follows:

Name	Description
on-demand	Built to support high-fidelity financial systems, where users can specify how data is ingested and transformed from on-chain or off-chain sources.

03 — Findings

Overall, we reported 23 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	1
HIGH	8
MEDIUM	2
LOW	3
INFO	9

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-SVB-ADV-00	CRITICAL	RESOLVED	`derive_any_permissioned` discards errors from authority and access control checks, allowing unauthorized users to change permissions on any oracle or pull feed account.
OS-SVB-ADV-01	HIGH	RESOLVED	An oracle's delegation pool is not added to its corresponding delegation group within the program state, preventing proper epoch advancement and reward distribution.
OS-SVB-ADV-02	HIGH	RESOLVED	The `reward` calculation in `OracleHeartbeat` allows oracles to receive maximum rewards even with a zero `reward` score, and oracles receive no `reward` if their `reward_score` exceeds the `slash_score`.
OS-SVB-ADV-03	HIGH	RESOLVED	`OracleHeartbeat` instruction passes the wrong account (`oracle` instead of `oracle_stats`) to `DelegationPoolCpi::advance_epoch` , resulting in incorrect epoch advancement.
OS-SVB-ADV-04	HIGH	RESOLVED	`maybe_execute_stake_rewards` utilizes the wrong reward escrow account in the `OracleHeartbeat` instruction's reward distribution logic.

OS-SVB-ADV-05	HIGH	RESOLVED	Malicious users may disrupt genuine users by repeatedly calling the commit instruction. There is also a risk of manipulating outcomes by calling commit, fetching signatures, and revealing until a desired outcome is achieved.
OS-SVB-ADV-06	HIGH	RESOLVED	The pull feed instructions expect responses from oracles that are part of the same queue as the pull feed being updated. They update the oracles with a new heartbeat timestamp regardless of whether their submissions are valid.
OS-SVB-ADV-07	HIGH	RESOLVED	`PullFeedSubmitResponseV2` and `PullFeedSubmitResponse` instruction does not verify the `offset` field in oracle signatures, allowing users to set it arbitrarily.
OS-SVB-ADV-08	HIGH	RESOLVED	The `OracleSetConfigs` instruction allows the oracle authority to change the `secp256k1_signer` key, undermining the integrity of the verification process, as the updated key may invalidate previous verifications.
OS-SVB-ADV-09	MEDIUM	RESOLVED	In the `RandomnessReveal` instruction, the validity of the secp256k1 key is not checked against the current timestamp. This oversight allows signatures from expired oracles to be utilized.
OS-SVB-ADV-10	MEDIUM	RESOLVED	The `RandomnessCommit` instruction does not ensure that the `oracle` account passed is the same as the `oracle` key stored in `randomness.oracle`.
OS-SVB-ADV-11	LOW	RESOLVED	`OracleInit` utilizes the `address_lookup_table_program` instead of the `stake_program` as the program ID when finding the PDA address for the `delegation_group`.

OS-SVB-ADV-12	LOW	RESOLVED	`parse_remaining_accounts` continues iterating after adding accounts to the `other_accounts` list, throwing an error when this account is incorrectly processed as oracle stats account.
OS-SVB-ADV-13	LOW	RESOLVED	Incorrect error handling in the `QueueInitDelegationGroup` instruction due to error propagation.

Bypass Of Authority/Access Control Checks CRITICAL OS-SVB-ADV-00

Description

The vulnerability in `PermissionSet` instruction arises from how errors are handled within `derive_any_permissioned`. `derive_any_permissioned` calls `derive_permissioned::<OracleAccountData, _>` and `derive_permissioned::<PullFeedAccountData, _>`, but it immediately discards any errors via `ok()`.

>_ *permission/permission_set_action.rs* rust

```rust
fn derive_any_permissioned<'a, F>(account: &'a AccountInfo<'a>, f: F) -> Result<()>
where
    F: Fn(&mut dyn Permissioned) -> Result<()>,
{
    derive_permissioned::<OracleAccountData, _>(account, &f).ok();
    derive_permissioned::<PullFeedAccountData, _>(account, &f).ok();
    Ok(())
}
```

In the closure passed to `derive_any_permissioned`, checks on the granter's authority and access control checks are performed. However, since any errors from `derive_any_permissioned` are discarded, these checks may be bypassed. As a result, any entity may change permissions on any oracle or pull feed account. This will result in unauthorized entities gaining control over oracles and pull feeds, compromising the system's integrity and security.

Remediation

Ensure the errors are not discarded in `derive_any_permissioned`. Instead, they should be properly propagated back to the caller.

Patch

Resolved in 6ed294e.

Failure To Add Delegation Pool To The Delegation Group HIGH OS-SVB-ADV-01

Description

The delegation pool of an oracle is currently not added to the delegation group of its queue within the program state. The delegation group tracks epochs and is crucial for managing epoch-based operations such as reward distribution. If an oracle's delegation pool is not linked to its delegation group, it will prevent the delegation group from advancing epochs as expected. Since epoch advancement is directly tied to reward distribution mechanisms, it also prevents reward distribution in the `OracleHeartbeat` instruction.

Remediation

Ensure to add the delegation pool of an oracle to the delegation group of in its queue.

Patch

Resolved in f3a0733.

Flawed Implementation of Reward Score Calculation HIGH OS-SVB-ADV-02

Description

The vulnerability in the `OracleHeartbeat` instruction stems from the incorrect design of the reward calculation, specifically in the way it handles the relationship between `reward_score` and `slash_score`. The current implementation of the formula for `reward` calculation is such that the final `reward` is proportional to `slash_score` and inversely proportional to `reward_score`. This is counterintuitive zx a higher `slash_score` should result in a lower `reward`, but instead, it increases the `reward`.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn calculate_slash(stats: &OracleStatsAccountData, reward: u64) -> u64 {
    let slash_score = stats.finalized_epoch.slash_score;
    if slash_score == 0 {
        return 0;
    }
    let reward_score = stats.finalized_epoch.reward_score;
    Decimal::from(reward)
        .saturating_mul(reward_score.into())
        .checked_div(slash_score.into())
        .unwrap()
        .to_u64()
        .unwrap_or(0)
}
```

As a result, Oracles that are supposed to be penalized (with a high `slash_score`) will end up receiving higher rewards, which is the opposite of the intended effect. Furthermore, If `reward_score = 0`, the formula simplifies to `reward = reward`. This implies the oracle would receive the maximum `reward` even if it did not perform any attestations, which is a severe flaw. Ideally, an oracle with `reward_score = 0` should receive no `reward`, as it indicates a complete lack of participation. Thus, Malicious oracles may exploit this flaw by not performing any attestations (resulting in `reward_score = 0`) and still receive full rewards, compromising the security and reliability of the entire network.

Remediation

Update the rewards calcualtion to ensure that to ensure that a higher `reward_score` results in a higher `reward`, and a higher

Patch

Resolved in bb597ec.

Improper Account Utilization For Epoch Advancement HIGH OS-SVB-ADV-03

Description

In `OracleHeartbeat`, `actuate` calls `advance_epoch` to synchronize epoch advancements and related state changes across the program accounts involved in staking and delegation. However, the `oracle` account is passed as an argument to `advance_epoch` (`ctx.accounts.oracle.to_account_info()`) instead of the `oracle_stats` account.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn actuate(
    ctx: &Context<Self>,
    params: &OracleHeartbeatParams,
    remaining_accounts: &RemainingAccounts<'info>,
) -> Result<()> {
    [...]
    let advance_epoch_res = DelegationPoolCpi::advance_epoch(
        ctx.accounts.stake_program.to_account_info(),
        ctx.accounts.delegation_pool.to_account_info(),
        ctx.accounts.queue_escrow.to_account_info(),
        ctx.accounts.queue.to_account_info(),
        ctx.accounts.delegation_group.to_account_info(),
        ctx.accounts.oracle.to_account_info(),
    );
    [...]
}
```

`advance_epoch` expects to update and synchronize epoch-related state information. By passing `ctx.accounts.oracle` instead of `ctx.accounts.oracle_stats`, the wrong `current_epoch.id` will be considered in the staking program, resulting in incorrect epoch transitions and misallocation of rewards intended for specific epochs.

Remediation

Ensure to pass the `oracle_stats` account to `advance_epoch`.

Patch

Resolved in 6ed294e.

Assignment Of Incorrect Reward Escrow HIGH OS-SVB-ADV-04

Description

The vulnerability in `maybe_execute_stake_rewards` in `OracleHeartbeat` instruction arises from the incorrect utilization of `remaining_accounts.oracle_switch_reward_escrow` as `oracle_wsol_reward_escrow` for distributing rewards.
`remaining_accounts.oracle_wsol_reward_escrow` is the correct account that should be utilized. Thus, due to the utilization of an incorrect rewards escrow, the oracle's `WSOL` reward escrow will fail to set up properly, and no funds will be transferred, affecting reward distribution.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn maybe_execute_stake_rewards(
    [...]
) -> Result<()> {
    [...]
    if let Some(oracle_wsol_reward_escrow) = &remaining_accounts.oracle_switch_reward_escrow {
        let res = NativeEscrow::spl_transfer(
            &ctx.accounts.token_program,
            &ctx.accounts.queue_escrow.to_account_info(),
            &oracle_wsol_reward_escrow.to_account_info(),
            &ctx.accounts.program_state.to_account_info(),
            &[&[STATE_SEED, &[state.bump]]],
            std::cmp::min(
                reward.saturating_sub(slash),
                oracle_wsol_reward_escrow.amount,
            ),
        );
    [...]
    }
}
```

Remediation

Ensure `oracle_wsol_reward_escrow` is `remaining_accounts.oracle_wsol_reward_escrow` instead of `remaining_accounts.oracle_switch_reward_escrow` in `maybe_execute_stake_rewards`.

Patch

Resolved in c156b82.

Interruptions and Manipulations In RandomnessCommit HIGH OS-SVB-ADV-05

Description

The `RandomnessCommit` instruction may be called by anyone and executed multiple times. This opens up the possibility for a malicious user to disrupt the process for a genuine user. When a genuine user initiates a coin flip and commits randomness, they expect to utilize the oracle's `slot` and `slothash` for their reveal. A malicious user may repeatedly call the `RandomnessCommit` instruction between the genuine user's commit and reveal instructions.

>_ *src/lib.rs* rust

```rust
#[access_control(ctx.accounts.validate(&ctx, &params))]
pub fn randomness_commit<'a>(
    mut ctx: Ctx<'_, 'a, RandomnessCommit<'a>>,
    params: RandomnessCommitParams,
) -> Result<()> {
    RandomnessCommit::actuate(&mut ctx, &params)
}
```

Consequently, the `randomness.oracle`, `randomness.seed_slot`, and `randomness.seed_slothash` values will be updated to the latest values. As a result, the genuine user would need to fetch a new signature from the latest oracle with the latest `slot` and `slothash`, invalidating their process.

Additionally, there is another potential vulnerability where a user may manipulate the system to generate a favorable random value. Before calling the reveal instruction, the user may repeatedly call commit, fetch signature, and reveal instructions until a favorable random value is generated, enabling the user to call the settle flip instruction with the favorable random value.

Remediation

Restrict any calls to the `RandomnessCommit` instruction if it has already been committed and has not yet been revealed. This ensures that once a commit has been made, no further commits can alter the randomness until it has been revealed.

Additionally, the program utilizing the randomness for the coin flip should store the `randomness.seed_slot` during the coin flip and verify it against `randomness.seed_slot` during the settle flip. This ensures that the randomness during the settle flip matches the randomness committed during the coin flip, preventing any manipulation by repeatedly committing and revealing.

Patch

1. Resolved in 9187d5c by restricting the calls to `RandomnessCommit` instruction as suggested.
2. The suggestion for storing the seed slot was acknowledged by the switchboard team, who stated that this should be done on the consumer programs.

Missing Oracle Checks In Pull Feed Instructions HIGH OS-SVB-ADV-06

Description

The `PullFeedSubmitResponseV2`, `PullFeedSubmitResponse`, `PullFeedSubmitResponseMany`, and `PullFeedSubmitResponseManyV2` instructions expect responses from oracles that are part of the same queue as the pull feed being updated. Oracles are associated with specific queues, and a pull feed's integrity depends on responses from oracles within its designated queue. If the contracts do not enforce that oracles must be from the same queue, it opens the door for oracles from other queues to submit responses.

>_ pull_feed/pull_feed_submit_response_action_v2.rs rust

```rust
pub fn actuate(
    ctx: &Context<'_, '_, 'info, 'info, PullFeedSubmitResponseV2<'info>>,
    params: &PullFeedSubmitResponseParamsV2,
    remaining_accounts: &RemainingAccounts<'info>,
    to_execute: &[bool],
) -> Result<()> {
    [...]
    for (idx, submission) in params.submissions.iter().enumerate() {
        let slot = params.slot - submission.offset as u64;
        let oracle_loader = &remaining_accounts.oracles[idx];
        if let Ok(mut oracle) = oracle_loader.load_mut() {
            msg!("Registering heartbeat for oracle {}", oracle_loader.key());
            oracle.last_heartbeat = clock.unix_timestamp;
        }
        [...]
    }
    [...]
}
```

Furthermore, these instructions update the `oracle.last_heartbeat` even when the `to_execute` flag for that oracle is set to false. Hence, by just passing the oracle in `remaining_accounts`, it would be considered as a heartbeat even if the signature for the pull feed is not submitted.

Remediation

Ensure that the oracle submitting the response is a member of the same queue as the pull feed and updates the heartbeat only when `to_execute[idx]` is true.

Patch

Fixed in a34b620.

Failure To Include Offset Value In Signature Verification HIGH OS-SVB-ADV-07

Description

In the `PullFeedSubmitResponseV2` and `PullFeedSubmitResponse` instructions, the `offset` field is not verified in the oracle signature verification. Since the `offset` is controlled by the user and not verified, an attacker may set the `offset` to a value that retrieves a price from a much earlier slot. This allows an attacker to submit a response that appears to be from a different time than the one actually signed by the oracle, undermining the data integrity.

Remediation

Incorporate the `offset` into the message hash used for signature verification or remove it completely.

Patch

Fixed in a34b620.

Ability To Update Signer Key HIGH OS-SVB-ADV-08

Description

In the `OracleSetConfigs` instruction, the oracle authority may change the `secp256k1_signer` of the enclave after verification. The `secp256k1_signer` key is utilized to verify that the signature on the quote (or any related data) is valid and was generated by an authorized party. The `secp256k1_signer` key must match the key used during the signing of quotes to ensure the integrity and authenticity of the oracle's data. The ability to update the `secp256k1_signer` key undermines the quote verification, which includes this key as one of the parameters.

Remediation

Disallow this functionality.

Patch

Resolved in 0109ad4.

Acceptance Of Expired Signatures From Expired Oracles MEDIUM OS-SVB-ADV-09

Description

When the `RandomnessReveal` instruction is invoked, it utilizes a signature generated by an oracle. However, if the validity of the oracle's secp256k1 key is not checked against the current timestamp, the system may accept signatures from expired oracles. This allows an expired oracle, which should no longer be part of the randomness generation process, to influence the outcome, undermining the integrity and security of the randomness generation process.

Remediation

Ensure that the secp256k1 expiration value of the oracle's key is greater than the current timestamp. If the key is expired, the instruction should reject the signature and not proceed with the randomness reveal.

Patch

This issue was acknowledged by the switchboard team

Absence Of Oracle Account Validation MEDIUM OS-SVB-ADV-10

Description

In the current implementation, `RandomnessCommit` instruction takes an `oracle` account as a parameter and also references an `oracle` key stored in the `randomness` account. However, there is no explicit check to ensure that the passed `oracle` account corresponds to the `oracle` key stored in the randomness account. The `oracle` account passed to the instruction may be different from the `oracle` key stored in `randomness.oracle`. This inconsistency may result in a situation where the data in the `randomness` account does not match the oracle data being utilized, potentially resulting in incorrect randomness commitments.

>_ *randomness/randomness_commit_action.rs* rust

```rust
pub fn validate(&self, ctx: &Context<Self>, _params: &RandomnessCommitParams) -> Result<()> {
    let queue = ctx.accounts.queue.load()?;
    if queue.oracle_keys_len == 0 {
        return Err(SwitchboardError::QueueIsEmpty.into());
    }
    let oracle = ctx.accounts.oracle.load()?;
    // Ensure the oracle is successfully verified.
    if oracle.enclave.verification_status == VerificationStatus::VerificationSuccess as u8 {
        // Ensure the oracle key does not expire within the next hour.
        if oracle.enclave.valid_until < Clock::get()?.unix_timestamp + 3600 {
            return Err(SwitchboardError::RandomnessOracleKeyExpired.into());
        }
    } else {
        return Err(SwitchboardError::InvalidQuote.into());
    }
        Ok(())
}
```

Remediation

Add a check to ensure that the `oracle` account passed to the instruction matches the `oracle` key stored in the `randomness` account. If the design is intentionally not to enforce that the `oracle` account matches the `randomness.oracle` key, then storing the `oracle` key in the `randomness` account becomes redundant. Instead, the instruction should explicitly check that the oracle is present in the `queue.oracle_keys`.

Patch

Resolved in bb597ec.

Incorrect PDA Address Calculation LOW OS-SVB-ADV-11

Description

Within `OracleInit` instruction in `actuate`, `Pubkey::find_program_address` is utilized to derive the delegation group program-derived address (PDA). The second parameter to this function should be the program ID of the program that will manage the program-derived address. However, `&ctx.accounts.address_lookup_table_program.key()` is incorrectly utilized as the program ID. The correct program ID should be `&ctx.accounts.stake_program.key()`. Consequently, the delegation group address stored on the address lookup table will be different than the intended one.

Remediation

Ensure that the program-derived address is derived utilizing `&ctx.accounts.stake_program.key()`.

Patch

Resolved in 6ed294e.

Discrepancy In Account Type Handling LOW OS-SVB-ADV-12

Description

In the `PullFeedSubmitResponse` instructions, there is a vulnerability in `parse_remaining_accounts` concerning the improper handling of accounts that are neither `oracles` accounts nor `oracle_stats` accounts. While parsing the remaining accounts in `parse_remaining_accounts`, oracle accounts and oracle stats accounts are added to their respective vectors. If an account is neither an oracle account nor an oracle stats account, it is added to the `other_accounts` hashmap.

>_ pull_feed/pull_feed_submit_response_action.rs rust

```rust
pub fn parse_remaining_accounts(
    ctx: &Context<'_, '_, 'info, 'info, PullFeedSubmitResponse<'info>>,
) -> Result<RemainingAccounts<'info>> {
    let mut oracles: Vec<AccountLoader<'info, OracleAccountData>> = Vec::new();
    let mut oracle_stats: Vec<AccountLoader<'info, OracleStatsAccountData>> = Vec::new();
    let mut other_accounts: HashMap<Pubkey, AccountInfo<'info>> = HashMap::new();
    for accnt in ctx.remaining_accounts.iter() {
        if let Ok(l) = AccountLoader::try_from(accnt) {
            oracles.push(l);
        } else {
            l[...]
            if maybe_stats_loader.is_err() {
                msg!("Unknown account type {}", accnt.key());
                other_accounts.insert(accnt.key(), accnt.clone());
            }
            let stats_loader = maybe_stats_loader?;
            [...]
        }
    }
    [...]
}
```

However, after adding this account to the `other_accounts` list in the `for` loop, the loop execution continues without skipping further processing for this account. Consequently, on the next line (`maybe_stats_loader?`), an error will be thrown since `maybe_stats_loader` is an `Err`, and calling `?` on it will result in an error. This error is unintended because the account should have been handled as an `other_account` and not processed further in the current iteration of the loop.

Remediation

Modify the loop in `PullFeedSubmitResponseV2`, `PullFeedSubmitResponse`, `PullFeedSubmitResponseMany`, and `PullFeedSubmitResponseManyV2` instructions by adding a `continue` statement after inserting the account into `other_accounts` to skip the rest of the loop for non-oracle and non-stats accounts.

Patch

Resolved in c156b82.

Misalignment Of Implementation With Intended Approach LOW OS-SVB-ADV-13

Description

In the `QueueInitDelegationGroup` instruction, the comment for `DelegationGroupCpi::init_cpi` indicates an intention to allow the function to continue execution even if an error occurs during the call (failing open). This implies that even if the initialization of the delegation group fails, the subsequent steps (extending the `LUT`) will still be executed. However, the current implementation does not achieve this because it propagates the error by utilizing the `?` operator.

>_ *queue/queue_init_delegation_group_action.rs* rust

```rust
pub fn actuate(ctx: &Context<Self>, _params: &QueueInitDelegationGroupParams) -> Result<()> {
    [...]
    // Initentionally allow fail open here to extend the lut with
    // the new delegation group and stake accounts
    DelegationGroupCpi::init_cpi(
        ctx.accounts.stake_program.to_account_info(),
        ctx.accounts.payer.to_account_info(),
        ctx.accounts.program_state.to_account_info(),
        ctx.accounts.delegation_group.to_account_info(),
        ctx.accounts.stake_pool.to_account_info(),
        ctx.accounts.system_program.to_account_info(),
        &ctx.accounts.queue.key(),
    )?;
    [...]
}
```

Remediation

Ensure the error from `DelegationGroupCpi::init_cpi` is handled explicitly to achieve the intended fail-open behavior, allowing the function to continue execution even if an error occurs.

Patch

Resolved in 6ed294e.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-SVB-SUG-00	The pattern for the assignment of the new authority may result in the loss of control over specific accounts.
OS-SVB-SUG-01	The message logged when `DelegationPoolCpi::init_cpi` fails suggests that the delegation pool already exists, which is incorrect and misleading.
OS-SVB-SUG-02	Highlighting inconsistencies in the garbage collection process within the `OracleHeartbeat` and `QueueGarbageCollect` instructions.
OS-SVB-SUG-03	`LutExtendCpi::invoke` reverts in case the `LOOKUP_TABLE_MAX_ADDRESSES` limit is reached, resulting in a possible denial-of-service scenario.
OS-SVB-SUG-04	Recommendation for modifying the codebase for improved efficiency and for inclusion of missing validations.
OS-SVB-SUG-05	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-SVB-SUG-06	Multiple cases of dead or irrelevant code are present within the protocol, which may be removed for improved clarity and readability.
OS-SVB-SUG-07	Several unnecessary codes currently exist within the codebase.
OS-SVB-SUG-08	There are numerous fields/accounts that are redundant or not utilized and should be removed:

Unsafe New Admin Assignment OS-SVB-SUG-00

Description

In the current implementation, it is possible to change the authority of the `state`, `queue`, `oracle`, and `pull_feed` accounts in a single step. However, the program does not account for mistyping the new authority address. Not handling this may result in losing control over these accounts if an incorrect address is passed inadvertently. Given the significance of these accounts in the program, splitting this authority update process into two distinct phases is advised.

Remediation

Divide changing the admin into two separate phases:

1. `set_admin_address`, signed by the current authority and utilized for setting the new authority.
2. `update_new_authority`, signed by the new admin address and utilized for updating the authority of the account.

Patch

This issue was acknowledged by the switchboard team

Misleading Error Logging OS-SVB-SUG-01

Description

`OracleUpdateDelegation` instruction logs a message indicating that the delegation pool already exists if `DelegationPoolCpi::init_cpi` returns an error. However, the function only throws an error if the CPI (Cross-Program Invocation) fails for reasons other than the delegation pool already existing. This implies that the logged message is misleading and does not accurately reflect the cause of the error. A similar issue exists in `RewardPoolCpi::init_cpi`.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn actuate(ctx: &Context<Self>, _params: &OracleUpdateDelegationParams) -> Result<()> {
    [...]
    let res = DelegationPoolCpi::init_cpi(
         ctx.accounts.stake_program.to_account_info(),
         ctx.accounts.payer.to_account_info(),
         ctx.accounts.authority.to_account_info(),
         ctx.accounts.oracle_stats.to_account_info(),
         ctx.accounts.delegation_pool.to_account_info(),
         ctx.accounts.stake_pool.to_account_info(),
         ctx.accounts.system_program.to_account_info(),
    );
    if res.is_err() {
        msg!("Delegation pool for this oracle already exists");
    }
    [...]
}
```

Remediation

Implement more specific error handling, making it more granular to provide specific log messages based on the actual cause of the error.

Patch

The issue in `DelegationPoolCpi::init_cpi` was resolved in bb597ec.

Inconsistencies In Garbage Collection Implementation OS-SVB-SUG-02

Description

1. In the current implementation of `OracleHeartbeat` instruction, the insertion of the oracle into the queue is attempted before performing garbage collection, as shown in code snippet of `OracleHeartbeat::actuate` below. As a result, the `oracle_keys` list may be full due to stale or inactive oracles. This results in the rejection of an active oracle trying to insert itself into the queue, even though space may be available after garbage collection.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn actuate(
    ctx: &Context<Self>,
    params: &OracleHeartbeatParams,
    remaining_accounts: &RemainingAccounts<'info>,
) -> Result<()> {
    [...]
    if oracle.is_on_queue == 0 {
        if queue.oracle_keys_len as usize == queue.oracle_keys.len() {
            return Err(error!(SwitchboardError::QueueFull));
        }
        let queue_len = queue.oracle_keys_len as usize;
        queue.oracle_keys[queue_len] = ctx.accounts.oracle.key();
        queue.oracle_keys_len += 1;
        oracle.is_on_queue = 1;
    }
    assert!(queue.oracle_keys_len != 0, "Queue is empty");
    queue.curr_idx += 1;
    queue.curr_idx %= queue.oracle_keys_len;
    [...]
}
```

2. The garbage collection logic in `OracleHeartbeat` and `QueueGarbageCollect` instructions determines the staleness of oracles utilizing different criteria, resulting in inconsistency. In the `OracleHeartbeat` instruction, the staleness of an oracle is determined via the `queue.node_timeout`, however, in the `QueueGarbageCollect` instruction, the staleness of an oracle is determined via a constant `MAX_STALE_SECONDS`.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn actuate(
    ctx: &Context<Self>,
    params: &OracleHeartbeatParams,
    remaining_accounts: &RemainingAccounts<'info>,
) -> Result<()> {
```

```
    [...]
    // Garbage collect
    let gc_idx = queue.gc_idx as usize;
    if ctx.accounts.oracle.key() != ctx.accounts.gc_node.key()
        && queue.try_garbage_collection(gc_idx, &clock, &ctx.accounts.gc_node)?
    {
        emit!(GarbageCollectionEvent {
            oracle: ctx.accounts.gc_node.key(),
            queue: ctx.accounts.queue.key(),
        });
    }
    [...]
}
```

Remediation

1. Perform the garbage collection process before attempting to insert the oracle into the queue.
2. Ensure consistency in the staleness determination in `QueueGarbageCollect` and `OracleHeartbeat` instructions. Allow garbage collection in `QueueGarbageCollect` instruction only if the verification status of the oracle has expired, to be consistent with the garbage collection in the `OracleHeartbeat` instruction.

Patch

Resolved in ba65e19.

Denial Of Service On Exceeding LUT Limit OS-SVB-SUG-03

Description

Lookup address tables (LUTs) are utilized to efficiently manage mappings between addresses (represented as Pubkeys) and associated data or metadata. Each LUT has a defined maximum limit (`LOOKUP_TABLE_MAX_ADDRESSES`) on the number of addresses it can accommodate. This limit is typically set to prevent excessive resource consumption and to ensure efficient lookup operations. As there is no way to change the `lut_slot` fields on the program accounts, reverting the operation when the `LOOKUP_TABLE_MAX_ADDRESSES` limit is reached in `LutExtendCpi::invoke` will effectively result in a denial-of-service scenario.

Remediation

Instead of reverting, the recommended approach is to handle the situation gracefully when the limit is reached.

Code Refactoring OS-SVB-SUG-04

Description

1. Within `Staker`, `next_delegation` tracks the number of used slots in the delegations array. Ideally, all slots from index zero to `next_delegation - 1` should be filled with delegations or empty slots marked by `delegation_pool == Pubkey::default`. `find_delegation` checks for `self.delegations[i].delegation_pool == Pubkey::default()` to identify the end of used slots. However, the code does not explicitly throw an error if it encounters a `Pubkey::default()` value before `next_delegation`.

>_ *staking/src/state/staker.rs* rust

```rust
pub fn find_delegation(&self, delegation_pool: &Pubkey) -> Result<usize> {
    for i in 0..self.next_delegation as usize {
        if self.delegations[i].delegation_pool == Pubkey::default() {
            // Once we enter the blank portion of the list, terminate
            break;
        }
        if self.delegations[i].delegation_pool == *delegation_pool {
            return Ok(i);
        }
    }
    err!(ErrorCode::DelegationNotFound)
}
```

2. `vesting_entry` does not explicitly check if `params.periods` is greater than zero when creating a new `VestingEntry`. If `params.periods` is set to zero, the vesting end time will be equal to the start time.
3. Currently, in the `OracleUpdateDelegation` instruction, it is not checked that the `queue` account represents the queue to which the oracle belongs. If the `queue` account does not match the oracle's queue account, it will result in incorrect entries in the lookup table.
4. `maybe_execute_stake_rewards` does not check whether `state.subsidy_amount` is zero before proceeding with the subsidy transfer logic. Thus, even if there are no subsidies to transfer, the function will still attempt to execute the transfer logic unnecessarily.

>_ *oracle/oracle_heartbeat_action.rs* rust

```rust
pub fn maybe_execute_stake_rewards(
    ctx: &Context<Self>,
    remaining_accounts: &RemainingAccounts<'info>,
    state: &State,
```

```
    stats: &OracleStatsAccountData,
    oracle: &mut OracleAccountData,
) -> Result<()> {
    [...]
    if state.enable_staking == 0 {
        msg!("Staking rewards disabled");
        return Ok(());
    }
    [...]
}
```

Remediation

1. Throw an error when encountering `self.delegations[i].delegation_pool == Pubkey::default()` before `next_delegation`.
2. Check if `params.periods > 0` when creating a new `VestingEntry`.
3. Ensure `queue` account in `OracleUpdateDelegation` instruction is equal to `oracle.queue`.
4. Add a check at the beginning of `maybe_execute_stake_rewards` to return early if `state.subsidy_amount` is zero. This check ensures that the function proceeds with subsidy-elated computations only when there is an actual subsidy amount to transfer.

Patch

1. Issue #3 resolved in 6ed294e.
2. Issue #5 resolved in f3a0733.

Code Maturity OS-SVB-SUG-05

Description

1. In `initialize_stake_pool`, use `InitializeStakePoolParams` within `#[instruction(...)]` instead of `Pubkey`.

>_ *staking/src/instructions/initialize_stake_pool.rs* rust

```rust
#[derive(Accounts)]
#[instruction(authority: Pubkey)]
pub struct InitializeStakePool<'info> {
    /// Payer of rent
    #[account(mut)]
    pub payer: Signer<'info>,
    [...]
}
```

2. Correct the typographical errors in the following areas:
   (a) `slash_score_current` is spelled as `slash_score_current` in `edit_oracle_data` and `write_oracle_data`.
   (b) `initialize_delegation_pool` is spelled as `initialize_delegation_pool` in the file name and in `instructions/mod.rs`.
   (c) `is_removable` is spelled as `is_removable` in `delegate_user_state`.
   (d) `delinquency` is spelled as `delinquency` in `delegation_pool`.
3. When disabling the `oracle_heartbeat` permission via `PermissionSet` instruction, the implementation should include steps to remove the oracle from `queue.oracle_keys`. Keeping disabled oracles in `queue.oracle_keys`, which is meant for active oracles, will unnecessarily utilize space and waste system resources.
4. Since `pull_feed_impl::standard_deviation` calculates the squared difference, the result will always be a non-negative value. Thus, the order of subtraction does not matter and the utilization of `min` and `max` is not necessary since both are `i128`.

Remediation

Implement the above-mentioned suggestions.

Patch

Issue #3 was acknowledged by the switchboard team.

Unutilized Code

OS-SVB-SUG-06

Description

1. `idx` field in `GuardianQuoteVerifyParams` is not utilized anywhere and may be removed.
2. In `OracleHeartbeat` instruction, the `bump`, `owner`, and `oracle` fields on `OracleStatsAccountData` do not need to be updated.
3. The `owner` and `oracle` fields on `OracleStatsAccountData` are the same and not changed anywhere, so one of them may be removed.
4. The `program_authority` and `state` accounts in the `QueueAddMrEnclave` and `QueueRemoveMrEnclave` instructions, and the `state` account in `QueueSetConfigs` seem unnecessary and should be removed.
5. The `QueueLutExtend` instruction is defined but not utilized in the `entrypoint` program.
6. In `StateInit` instruction, the `state.bump = ctx.bumps.state` statement is repeated. Ensure to remove the duplicate statement.
7. `U192` is defined in `math` but not used.
8. The following accounts are passed to instructions unnecessarily and may be removed:
   (a) `owner` and `stake_mint` in `Stake` instruction.
   (b) `owner` in `Grant` instruction.
   (c) `registrar` in the `UpdateVoterWeightRecord` instruction.
   (d) `owner` in the `CreateVoterWeightRecord` instruction.
9. Within `delegation_pool`, `has_reward_vault` may be removed as it is duplicated by `contains_pool`, resulting in redundancy.

Remediation

Ensure to eliminate the above-stated code items.

Patch

The code was removed.

Removal Of Unnecessary Code

OS-SVB-SUG-07

Description

1. In the `GuardianRegister` instruction, the `guardian_queue` account seems unnecessary. The check in `GuardianRegister` instruction may be directly performed between `state.guardian_queue` and `oracle.queue`. Also, in the `OracleInit` instruction and the `PullFeedInit` instruction, the `stake_program` account, `stake_pool` account, and `reward_escrow` account, respectively, may be removed.
2. The `oracle_min_stake`, `allow_authority_override_after`, `require_authority_heartbeat_permission`, `require_authority_verify_permission`, `require_usage_permissions`, `signer_bump`, and `mint` fields on `QueueAccountData` are not utilized.
3. The `flat_reward_cut_percentage`, `enable_slashing`, and `lut_slot` fields on `State` are unutilized. Additionally, the `stake_score` field on `OracleEpochInfo` is not utilized.
4. The `oracle_stats` accounts are unnecessarily checked twice in `parse_remaining_accounts` and validate functions in the `PullFeedSubmitResponse` instruction.
5. In the `OracleUpdateDelegation` instruction, utilize `oracle.load()?.lut_slot` instead of `params.recent_slot`, and remove `recent_slot` from the input parameters.

>_ staking/src/instructions/initialize_stake_pool.rs rust

```rust
    /// CHECK: explicit
    #[account(mut, constraint = lut.key() == derive_lookup_table_address(&lut_signer.key(),
        ↪ params.recent_slot).0)]
    pub lut: AccountInfo<'info>,
```

Remediation

Remove the code instances mentioned in the above list.

Redundant/Unutilized Code OS-SVB-SUG-08

Description

1. The `secp_authority` field on `OracleAccountData`.
2. In the `PullFeedSubmitResponseMany` and `PullFeedSubmitResponseManyV2` instructions, the `FeedInfo` structures are unnecessary since only the `value` field in it is utilized.
3. The `reward_escrow` account in `RandomnessReveal` instruction.
4. The `active_secp256k1_expiration` field in `RandomnessAccountData`.
5. The `signature_instruction_index`, `eth_address_instruction_index`, and `message_instruction_index` fields in `SecpSignatureOffsets`.
6. The `offsets` field in `ParsedSignatureData`.

Remediation

Remove the above-listed code items.

Patch

The code was removed.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.