Gatekeeper Security Review

Pashov Audit Group

Conducted by: 0xdeadbeef, ParthMandale, LordAlive, JoVi

June 28th 2025 - July 1st 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **saguarocrypto/gatekeeper** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Gatekeeper

Gatekeeper is a Solana program that manages slot gating for validators using an efficient bitmap structure to control which slots are permitted for sandwich-facilitating operations. It provides a full CRUD interface—creating, reading, updating, and deleting gating configurations per epoch—secured by multisig authority, enabling precise, scalable slot-level control.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - c009d361238a121e7d4543c2550d75f085a84800

fixes review commit hash - 74998cb761a6d709171b69c5d2028f7c3745fbe5

Scope

The following smart contracts were in scope of the audit:

- `lib`
- `constants`
- `append_data_sandwich_validators_bitmap`
- `clear_data_sandwich_validators_bitmap`
- `close_sandwich_validator`
- `expand_sandwich_validators_bitmap`
- `mod`
- `modify_sandwich_validators`
- `set_sandwich_validators`
- `update_sandwich_validator`
- `validate_sandwich_validators`

7. Executive Summary

Over the course of the security review, 0xdeadbeef, ParthMandale, LordAlive, JoVi engaged with Seguaro to review Gatekeeper. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Gatekeeper
Repository	https://github.com/saguarocrypto/gatekeeper
Date	June 28th 2025 - July 1st 2025
Protocol Type	Sandwich Validator Control

Findings Count

Severity	**Amount**
Critical	2
Medium	1
Low	4
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Critical PDA validation flaw in append_data_sandwich_validators_bitmap	Critical	Resolved
[C-02]	Improper PDA validation in handler enables arbitrary data clearing	Critical	Resolved
[M-01]	Improper PDA validation in expand_sandwich_validators_bitmap handler	Medium	Resolved
[L-01]	Up to 200 slots can be modified instead of 100	Low	Resolved
[L-02]	Conflicting slots in modify_sandwich_validators	Low	Resolved
[L-03]	Rent overfunded during incremental bitmap expansion	Low	Resolved
[L-04]	Append instruction overwrites instead of appending	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Critical PDA validation flaw in `append_data_sandwich_validators_bitmap`

Severity

Impact: High

Likelihood: High

Description

The `append_data_sandwich_validators_bitmap` instruction does **not correctly validate** the `sandwich_validators` PDA account. Specifically, the `AppendDataSandwichValidatorsBitmap` context used in this instruction is not Deriving the PDA account from the seeds.

This flaw allows one authority to **append data to the validator bitmap of a different authority's PDA**, leading to **unauthorized state modifications**.

Unlike a benign validation oversight, this issue poses a **critical security risk**, as it breaks isolation between authorities and can lead to **cross-authority data corruption or manipulation** and will re-write the Data of that PDA account from the 16th byte making it as a critical bug.

Recommendations

Derive the PDA account inside AppendDataSandwichValidatorsBitmap instead of using the passed account to the instruction :

```
pub struct AppendDataSandwichValidatorsBitmap<'info> {
@>   #[account(mut,
        seeds = [SandwichValidators::SEED_PREFIX, multisig_authority.key
          ().as_ref(), &epoch_arg.to_le_bytes()],
        bump
    )]
    pub sandwich_validators: AccountLoader<'info, SandwichValidators>,
    #[account(mut)]
    pub multisig_authority: Signer<'info>,
}
```

[C-02] Improper PDA validation in `handler` enables arbitrary data clearing

Severity

Impact: High

Likelihood: High

Description

The `clear_data_sandwich_validators_bitmap` instruction's handler does **not properly validate** the `sandwich_validators` PDA account. This oversight allows a malicious signer to **supply a different PDA account** (belonging to another signer or authority) and still pass the check.

As a result, the handler may **incorrectly clear bitmap data** belonging to a different PDA by potentially **wiping valid data** corresponding to another authority.

This poses a **critical risk**, as it enables unauthorized modification of account data that should be protected by strict PDA derivation and validation.

Recommendations

Derive the PDA account in the `ClearDataSandwichValidatorsBitmap` struct it self like below :

```
pub struct ClearDataSandwichValidatorsBitmap<'info> {

@>  #[account(
        mut,
        seeds = [SandwichValidators::SEED_PREFIX, multisig_authority.key
          ().as_ref(), &epoch_arg.to_le_bytes()],
        bump
    )]
    pub sandwich_validators: AccountLoader<'info, SandwichValidators>,
    #[account(mut)]
    pub multisig_authority: Signer<'info>,
}
```

8.2. Medium Findings

[M-01] Improper PDA validation in `expand_sandwich_validators_bitmap` handler

Severity

Impact: Low

Likelihood: High

Description

The `expand_sandwich_validators_bitmap` instruction does **not correctly validate** the `sandwich_validators` PDA account. In particular, the `ExpandSandwichValidatorsBitmap` struct fails to **derive the PDA using the correct authority**, which results in executing this instruction with another authority's corresponding PDA account.

While this issue does **not pose an immediate malicious risk**, it reflects a **logical inconsistency** in the program's account validation. This could lead to **unexpected behavior**.

Recommendations

Derive the PDA account inside the `ExpandSandwichValidatorsBitmap` struct like below :

```
pub struct ExpandSandwichValidatorsBitmap<'info> {
@>   #[account(mut,
        close = multisig_authority,
        seeds = [SandwichValidators::SEED_PREFIX, multisig_authority.key
          ().as_ref(), &epoch_to_close.to_le_bytes()],
        bump
    )]
    pub sandwich_validators: AccountInfo<'info>,
    #[account(mut)]
    pub multisig_authority: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

8.3. Low Findings

[L-01] Up to 200 slots can be modified instead of 100

The specs and comment define that a maximum of 100 slots can be modified for each transaction. However, this is enforced per ungate and gate operations so the actual amount can be up to 200.

```
if slots_to_gate.len() > MAX_SLOTS_PER_TRANSACTION {
            return err!(GatekeeperError::TooManySlots);
        }

        if slots_to_ungate.len() > MAX_SLOTS_PER_TRANSACTION {
            return err!(GatekeeperError::TooManySlots);
        }
```

Consider checking if `slots_to_gate.len() + slots_to_ungate.len() > MAX_SLOTS_PER_TRANSACTION`.

[L-02] Conflicting slots in `modify_sandwich_validators`

When calling the modify_sandwich_validators instruction, the same slot can appear in both slots_to_gate and slots_to_ungate, leading to silent mis-configuration. **Recommendation** - Reject transactions where the two lists intersect (e.g., new ConflictSlots error) or automatically treat overlaps as a no-op.

[L-03] Rent overfunded during incremental bitmap expansion

The expand_sandwich_validators_bitmap instruction transfers enough lamports to make the account rent-exempt for the final TARGET_ACCOUNT_SIZE, even though each call may grow the account by only MAX_REALLOC_SIZE (10 KiB). If the expansion halts midway, those surplus lamports remain locked in the account.

Recommendation – Calculate minimum_balance for the next size actually reached in this instruction (current_size + expansion_size) and transfer only the delta. Defer any remaining top-up until the last chunk brings the account to TARGET_ACCOUNT_SIZE.

[L-04] Append instruction overwrites instead of appending

The `append_data_sandwich_validators_bitmap` function is documented as an "append" utility.

```
/// Append data to the sandwich validators bitmap account.
------------
/// Handler for appending data to a sandwich validators bitmap account.
------------
msg!("Appending {} bytes of data to large bitmap", data.len());
------------
msg!("Successfully appended {} bytes to large bitmap", max_write);
```

However - its implementation actually **overwrites** the beginning of the bitmap with the provided data, starting at offset 0. It does not support true append.

```
// Write data to bitmap data section
let max_write = data.len().min(bitmap_data.len());
bitmap_data[..max_write].copy_from_slice(&data[..max_write]);
```

If a user or integrator expects this function to append data, they may inadvertently overwrite the existing gating state, leading to loss of protection or unnecessary gating.

Recommendations

Consider either rewriting the spec or implementing append instead of overwriting.