Convergent

Security Assessment

August 15th, 2024 — Prepared by OtterSec

Kevin Chow kchow@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Convergent Finance engaged OtterSec to assess the `v1-contracts` program. This assessment was conducted between June 3rd and July 23rd, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified several critical vulnerabilities, including the possibility of manipulating the CVGT staking state via the ability to set arbitrary CVGT mints (OS-CVG-ADV-00) and spoofing the account containing the staking information in the CVGT staking module, enabling the staking of fake tokens and the withdrawal of legitimate CVGT tokens (OS-CVG-ADV-01). Furthermore, the price feed update functionality may return an outdated or stale price when oracles are broken or untrusted, potentially exposing users to inaccurate pricing data (OS-CVG-ADV-02).

We also recommended modifying the codebase for improved efficiency (OS-CVG-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/Convergent-Finance/v1-contracts. This audit was performed against commit d4e5fa6.

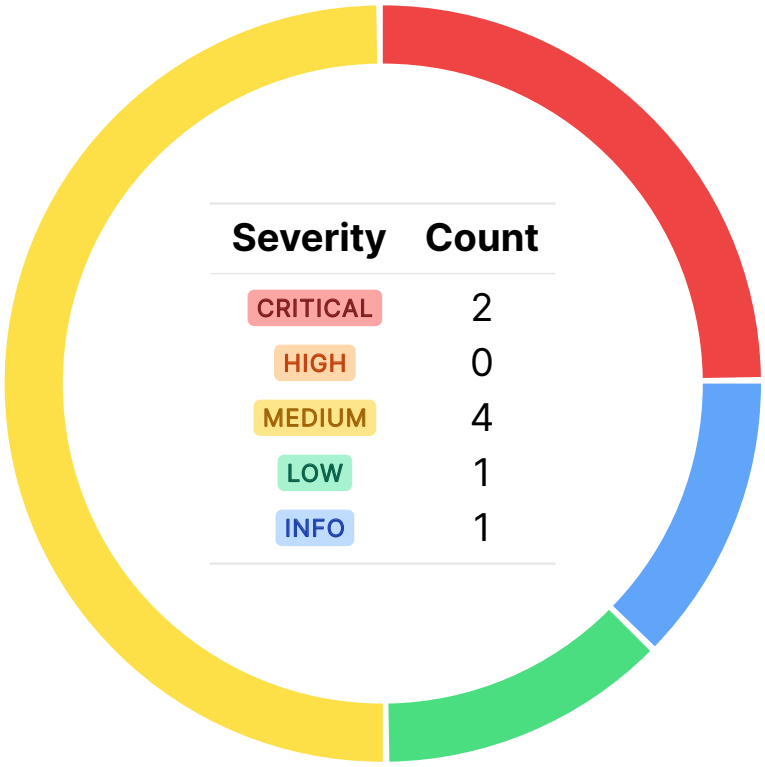
A brief description of the programs is as follows:

Name	Description
v1-contracts	Manages troves, which are debt positions backed by collateral, and also incorporates functionalities for price feeds, stability pools, community issuance, and CVGT staking.

03 — Findings

Overall, we reported 8 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	2
HIGH	0
MEDIUM	4
LOW	1
INFO	1

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-CVG-ADV-00	CRITICAL	RESOLVED	It is possible to manipulate the `CVGT` staking state via the ability to set arbitrary `CVGT` mints and spoofed `cvgt_staking_state` through `initialize_handler` and `config_pool_state_handler`.
OS-CVG-ADV-01	CRITICAL	RESOLVED	`staking_info` account in `cvgt_staking` may be spoofed under a malicious `CVGTStakingPoolState`, allowing attackers to stake fake tokens and potentially withdraw legitimate `CVGT` tokens.
OS-CVG-ADV-02	MEDIUM	RESOLVED	`PriceFeedState::update` may return outdated or stale prices when oracles are broken or untrusted, potentially exposing users to inaccurate pricing data.
OS-CVG-ADV-03	MEDIUM	RESOLVED	`compute_emission_amount` calculates and issues `CVGT` tokens even when the vault is nearly empty, as it only caps the issued amount to the vault's balance without preventing issuance if the vault has insufficient tokens.
OS-CVG-ADV-04	MEDIUM	RESOLVED	Utilizing arbitrary price feed accounts may expose the system to risks from stale or incorrect data.

OS-CVG-ADV-05	MEDIUM	RESOLVED	`move_token_from_redeem` incorrectly burns only the `gas_compensation` amount instead of the full `total_usv_gas_to_burn` amount, which may result in an insufficient burn of `USV` tokens.
OS-CVG-ADV-06	LOW	RESOLVED	To ensure robust pointer management, it's crucial to clear Trove pointers during both insertion and closing operations, in addition to their removal, to prevent potential inconsistencies or errors.

CVGT Staking Pool State Manipulation CRITICAL OS-CVG-ADV-00

Description

There is a potential vulnerability involving the manipulation of the `CVGT` staking state by exploiting how the `CVGT` mint and `CVGTStakingPoolState` accounts are configured. In `init::initialize_handler`, the `CVGT` account is defined as `cvgt: Box<Account<'info, Mint>>`.

>_ *init.rs* rust

```rust
pub struct Initialize<'info> {
    [...]
    #[account()]
    pub cvgt: Box<Account<'info, Mint>>,
    [...]
}
```

This field allows `initialize_handler` to accept any `cvgt` mint account without performing checks to ensure it is legitimate or intended for the staking pool. Similarly, `config_pool_state_handler` within `config_pool_state` allows updating the `cvgt_staking_state` field in the `pool_state` account to point to a new `CVGTStakingPoolState`. Thus, it is possible to set any `CVGT` on a `poolstate` and `stability_pool_state` and also set `CVGTStakingPoolState` to a spoofed `poolState`.

>_ *config_pool_state.rs* rust

```rust
pub fn config_pool_state_handler(ctx: Context<ConfigPoolState>) -> Result<()> {
    let pool_state = &mut ctx.accounts.pool_state;
    let cvgt_staking_state_key = ctx.accounts.cvgt_staking_state.key();
    pool_state.cvgt_staking_state = cvgt_staking_state_key;
    Ok(())
}
```

As a result, this enables an attacker to direct staking operations to a fake staking state, allowing them to manipulate or steal funds. Specifically, in `adjust_trove` and `open_trove`, `increase_f_usv` modifies the `CVGT` staking pool state by increasing the `USV` fee. If an attacker is able to spoof the `CVGTStakingPoolState`, they may increase the `USV` fee in a manner that is not authorized. Furthermore, it may be possible to mutate `CommunityIssuanceConfig` with a spoofed `poolstate` and `stability_pool_state` with the same logic described above.

>_ *adjust_trove.rs* rust

```rust
pub fn increase_f_usv(&self, usv_fee: u64) -> Result<()> {
    let cvgt_staking_account = &self.cvgt_staking_state;
    let mut data = cvgt_staking_account.try_borrow_mut_data()?;
    let mut cvgt_staking = CVGTStakingPoolState::try_deserialize(&mut data.as_ref())
        .expect("Error Deserializing Data");
    cvgt_staking.increase_f_usv(usv_fee);
    cvgt_staking.try_serialize(&mut data.as_mut())?;
    Ok(())
}
```

Remediation

Implement logic to validate that the `CVGT` mint provided during initialization is legitimate and also ensure that the `cvgt_staking_state` being set is indeed valid.

Patch

Resolved in 5275b76.

Spoofing Stake Information CRITICAL OS-CVG-ADV-01

Description

In `cvgt_staking::stake`, the `staking_info` account is initialized based on the user account's public key and is associated with a staking operation. The vulnerability lies in the fact that `staking_info` may be spoofed under a malicious `CVGTStakingPoolState`, resulting in security risks.

>_ *cvgt_staking/stake.rs* rust

```rust
pub struct Stake<'info> {
    [...]
    #[account(
        init_if_needed,
        space = 8 + CVGTStakingInfo::INIT_SPACE,
        payer = user,
        seeds = [
            b"info",
            user.key().as_ref()
        ],
        bump
    )]
    pub staking_info: Box<Account<'info, CVGTStakingInfo>>,
    [...]
```

The attacker may create a `staking_info` account for a fake token, which is not associated with the legitimate `CVGTStakingPoolState`. The `CVGTStakingPoolState` account controls the global state of the staking pool, including mint addresses and other critical data. If the staking information is not correctly verified against the pool state, the attacker may be able to unstake legitimate `CVGT` tokens.

>_ *cvgt_staking/init.rs* rust

```rust
    [...]
    #[account()]
    pub cvgt: Account<'info, Mint>,
    [...]
}
```

Remediation

Ensure that the `staking_info` account is explicitly validated against the `CVGTStakingPoolState`. Also, limit who may initialize or modify `CVGTStakingPoolState` to prevent unauthorized or malicious changes.

Patch

Resolved in 5275b76.

Extended Utilization Of Stale Data MEDIUM OS-CVG-ADV-02

Description

Currently, `PriceFeedState::update` returns the `last_good_price` every time. However, if the `last_good_price` is returned when both oracles are broken or untrusted, there is a risk that this price might be outdated. Thus, if the oracles are down for an extended period and the system continues to utilize the last good price, the system may rely on this price long after the failure. Over time, the market price may diverge significantly from this stale value, resulting in incorrect or sub-optimal decisions based on the outdated price.

>_ *price_feed_state.rs* rust

```rust
    [...]
        // --- CASE 5: Using Pyth, Chainlink is untrusted ---
        Status::UsingPythChainlinkUntrusted => {
            // If Pyth breaks, now both oracles are untrusted
            if is_pyth_broken(pyth_price_message) {
                self.set_status(Status::BothOraclesUntrusted);
                return Ok(self.last_good_price);
            }

            // If Pyth is frozen, return last good price (no status change)
            if is_pyth_frozen(pyth_price_message) {
                return Ok(self.last_good_price);
            }
            [...]
        }
    [...]
```

Remediation

Instead of relying solely on `last_good_price`, implement fallback actions when both oracles are broken. These actions may involve pausing trading and alerting users.

Patch

Acknowledged by the Convergent team.

Vault Depletion Risk MEDIUM OS-CVG-ADV-03

Description

`compute_emission_amount` in `community_issuance_config` manages the token issuance process by adjusting the amount to be issued based on both the passage of time and the available supply. Specifically, the function utilizes `min(amount, token_in_vault)` to ensure that the amount issued does not exceed the tokens available in the vault. While this may seem like a reasonable approach, it may result in unintended consequences if the vault has very few tokens or none at all.

>_ *community_issuance_config.rs* rust

```rust
fn compute_emission_amount(
    config: &CommunityIssuanceConfig,
    current_timestamp: u64,
    token_in_vault: u64,
) -> Option<u64> {
    if !config.enable_emission {
        return Some(0);
    }
    let last_reward_timestamp = config.last_reward_timestamp;
    let mut amount = current_timestamp
        .checked_sub(last_reward_timestamp)?
        .checked_mul(config.emission_rate)?;
    amount = min(amount, token_in_vault);
    Some(amount)
}
```

If `token_in_vault` is very low (or zero), the function will simply issue a very small amount of tokens or none at all. This silent reduction in issuance may not be apparent to users or system administrators, potentially resulting in unintended consequences.

Remediation

Ensure that the function provides clear feedback or error reporting when the vault has insufficient tokens.

Patch

Resolved in 345253c.

Arbitrary Price Feed Utilization MEDIUM OS-CVG-ADV-04

Description

The vulnerability involves utilizing arbitrary or potentially stale price feeds. If a price feed account is not kept up-to-date, it may provide outdated or incorrect prices. This will result in incorrect valuations and decision-making within the system. Allowing arbitrary price feeds implies that the contract may be utilizing feeds that are not validated or endorsed by reputable sources, increasing the risk of relying on inaccurate or stale data.

Remediation

Utilize officially sponsored or verified price feeds.

Patch

Resolved in 4f342a0.

Incorrect Burn Amount MEDIUM OS-CVG-ADV-05

Description

In `move_token_from_redeem` within `redeem_collateral`, tokens are burned from `gas_compensation` if `total_usv_gas_to_burn` is greater than zero. However, the actual amount burned is determined by `pool_state.gas_compensation` rather than `total_usv_gas_to_burn`. The issue with this implementation is that it may not burn the correct amount of tokens as specified by `total_usv_gas_to_burn`. For proper redemption, the exact amount of `total_usv_gas_to_burn` should be burned.

>_ trove-manager/src/instructions/redeem_collateral.rs rust

```rust
fn move_token_from_redeem(
    ctx: Context<RedeemCollateral>,
    redemption_totals: &RedemptionTotals,
) -> Result<()> {
   [...]
    if redemption_totals.total_usv_gas_to_burn > 0 {
        burn(
            ctx.accounts
                .burn_stablecoin_from_gas_compensation_ctx()
                .with_signer(&[&authority_seed[..]]),
            pool_state.gas_compensation,
        )?;
    }
    [...]
}
```

Remediation

Adjust the burn operation to utilize the `total_usv_gas_to_burn` amount directly.

Patch

Resolved in 34b1a5a.

Ensuring Proper Pointer Cleanup LOW OS-CVG-ADV-06

Description

It is recommended to clear `Trove` pointers during insertion and closing functions as a precautionary measure to ensure that the linked list remains consistent and to prevent potential issues that may arise from stale or incorrect pointers. When working with doubly linked lists, each node has pointers to its previous and next nodes. When inserting a new node into the linked list, it is important to ensure that new node initialization is executed correctly and that neighboring pointers are updated properly. Additionally, while clearing pointers, node references should be removed, and node pointers should be set to default.

Remediation

Ensure to clear the `Trove` pointers during insertion and closing.

Patch

Resolved in 6091ea8.

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-CVG-SUG-00	Recommendation for modifying the code base for improved efficiency.

Code Refactoring OS-CVG-SUG-00

Description

Remove view functions, as they are not necessary on Solana.

Remediation

Implement the above-mentioned suggestion.

Patch

Resolved in 5339feb.

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 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.