Ore V2 Audit Report

Signature: b92c99931dc36c7c795c7232ce1578ab

Ore V2 Audit Report

Executive Summary

From June 25, 2024, to July 4, 2024, the Ore project engaged Fuzzland to conduct a thorough security audit of their Ore V2 project. This audit focused on Ore Miner programs. The primary objective was to identify and mitigate potential security vulnerabilities, risks, and coding issues to enhance the project's robustness and reliability. Fuzzland conducted this assessment over 14 person-days, involving 2 engineers who reviewed the code over a span of 7 days. Employing a multifaceted approach that included static analysis, fuzz testing, and manual code review, Fuzzland team identified 3 issues across different severity levels and categories.

Project Summary

Ore is a fair-launch, proof-of-work digital currency that is accessible for mining by all participants. The protocol rewards miners with Ore tokens upon successful completion of Equix proof-of-work calculations. Additionally, users have the option to enhance their earned rewards through a staking mechanism, wherein they stake Ore tokens to receive augmented returns.

Scope

Project Name	Ore V2
Repository Link	https://github.com/regolith-labs/ore
Commit	3b2bf98be990f14f667dfa1de2a70c8d89a9fe88 cbfc7afc2bb5c4551fff9132d31b5892f3faa746
Fix Commit	ec441fef54fc9e4df15ab65ab0634b049daf8505
Language	Rust - Solana

Methodology

Our audit methodology comprised a comprehensive and systematic approach to uncover potential security vulnerabilities, risks, and coding issues. The key components of our methodology included:

- **Static Analysis**: We perform static analysis using our proprietary internal tools for Solana programs and CodeQL to identify potential vulnerabilities and coding issues.
- **Fuzz Testing**: We execute fuzz testing by utilizing our proprietary internal fuzzers to uncover potential bugs and logic flaws.
- **Manual Code Review**: Our engineers manually review code to identify potential vulnerabilities not captured by previous methods.

Engagement Summary

The engagement involved a team of skilled consultants / engineers who were responsible for various phases of the audit process, including onboarding, initial audits, additional audits, and quality assurance. Below is a summary of the engagements with specific dates and details.

Dates	Consultants / Engineers Engaged	Details
6/24/2024	Chaofan Shou, Eda Zhang	Onboarding
6/24/2024 - 7/4/2024	Chaofan Shou, Tony Ke	Audit #1
7/5/2024	Chaofan Shou, Tony Ke	Audit #2

Vulnerability Severity

We divide severity into four distinct levels: high, medium, low, and info. This classification helps prioritize the issues identified during the audit based on their potential impact and urgency.

- **High Severity Issues** represent critical vulnerabilities or flaws that pose a significant risk to the system's security, functionality, or performance. These issues can lead to severe consequences such as fund loss, or major service disruptions if not addressed immediately. High severity issues typically require urgent attention and prompt remediation to mitigate potential damage and ensure the system's integrity and reliability.
- **Medium Severity Issues** are significant but not critical vulnerabilities or flaws that can impact the system's security, functionality, or performance. These issues might not pose an immediate threat but have the potential to cause considerable harm if left unaddressed over time. Addressing medium severity issues is important to maintain the overall health and efficiency of the system, though they do not require the same level of urgency as high severity issues.
- **Low Severity Issues** are minor vulnerabilities or flaws that have a limited impact on the system's security, functionality, or performance. These issues generally do not pose a significant risk and can be addressed in the regular maintenance cycle. While low severity issues are not critical, resolving them can help improve the system's overall quality and user experience by preventing the accumulation of minor problems over time.
- **Informational Severity Issues** represent informational findings that do not directly impact the system's security, functionality, or performance. These findings are typically observations or recommendations for potential improvements or optimizations. Addressing info severity issues can enhance the system's robustness and efficiency but is not necessary for the system's immediate operation or security. These issues can be considered for future development or enhancement plans.

Below is a summary of the vulnerabilities with their current status, highlighting the number of issues identified in each severity category and their resolution progress.

	Number	Resolved
High Severity Issues	1	1
Medium Severity Issues	0	0
Low Severity Issues	1	1
Informational Severity Issues	1	1

Disclaimer

The audit does not ensure that it has identified every security issue in the project, and it should not be seen as a confirmation that there are no more vulnerabilities. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value projects to commission several independent audits, a public bug bounty program, as well as continuous onchain security auditing and monitoring. Additionally, this report should not be interpreted as personal financial advice or recommendations.

Findings

[High] Flashloan Leading to Manipulation of Staking Mechanism

In the following code, Ore Mining program would provide rewards to miners, and if a miner has staked, the reward would be boosted based on the formula: $\frac{min(staked, max\_staked)}{max\_staked}$.

mine.rs regolith-labs/ore

```
        let staking_reward = proof
            .balance
            .min(config.max_stake)
            .saturating_mul(reward)
            .saturating_div(config.max_stake);
        reward = reward.saturating_add(staking_reward);
```

master · Rust

The `max_stake` can be updated by anyone if they have an account that holds more `balance` than the current `max_stake`.

crown.rs regolith-labs/ore

```
    config.max_stake = proof_new.balance;
```

master · Rust

The `balance` can be increased by staking as in the following code:

stake.rs regolith-labs/ore

```
    proof.balance = proof.balance.saturating_add(amount);
```

master · Rust

The issue is that the attacker can use flashloan to increase their `balance` and subsequently `max_stake` in a single transaction at little cost. Then, as `max_stake` is inflated, the reward boost for every staked miner becomes smaller.

Detailed attack steps (in a single transaction):

1. The attacker gains a flashloan of X amount of Ore tokens. (s.t. X >> `max_stake`)
2. The attacker stakes X tokens and increases their `balance` to X.
3. The attacker crowns themselves to set the `max_stake` to X.
4. The attacker claims X amount of tokens back and repays the flashloan with those tokens.
5. Now, the `max_stake` is high enough that no one can get their reward significantly boosted.

Recommended Fix

Check `last_stake_at` at crown.rs regolith-labs/ore.

Status

Fixed

[Low] Potential Backrun Leading to Manipulation of Staking Mechanism

An attacker can potentially backrun the initialization by staking very few tokens and consistently getting a 2x mining reward boost.

initialize.rs regolith-labs/ore

```
    config.max_stake = 0;
```

master · Rust

Detailed attack steps:

1. The attacker gains Ore v1 tokens and converts them to Ore v2 tokens immediately after initialization.
2. In the same transaction, the attacker creates and opens a significant amount of Proof accounts. Each account is then staked with X Ore tokens. (X ~= 1)
3. Crown one of the Proof accounts, making `max_stake` becomes X.
4. Wait for 1 minute. Then, the attacker bulk submits mine transactions in bundles for every Proof account. Hopefully, no one stakes more than X and crowns themselves.
5. Now, with a very limited cost, the attacker can mine a significant number of tokens and have the reward 2x boosted.

Recommended Fix

Initialize `max_stake` with a large enough number instead of 0.

Status

Fixed

[Info] Transaction Fail Silently in Crown

Even if the transaction does not successfully crown a proof account, it is still successful. This may potentially cause misunderstandings.

crown.rs regolith-labs/ore

```rust
    if config.top_staker.ne(&Pubkey::new_from_array([0; 32])) {
        // Load current top staker
        load_any_proof(proof_info, false)?;
        if proof_info.key.ne(&config.top_staker) {
            return Ok(());
        }

        // Compare balances
        let proof_data = proof_info.data.borrow();
        let proof = Proof::try_from_bytes(&proof_data)?;
        if proof_new.balance.lt(&proof.balance) {
            return Ok(());
        }
    }
```

master · Rust

Recommended Fix

Throw an error or explicitly document this behavior.

Status

Acknowledged