Clockwork Threads

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Maher Azzouzi maher@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Clockwork engaged OtterSec to perform an assessment of the `clockwork` program. This assessment was conducted between October 23rd and December 6th, 2022. For more information on our auditing methodology, see Appendix B.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches December 8th, 2022.

Key Findings

Over the course of this audit engagement, we produced 4 findings total.

In particular, we found issues with account trigger validation (OS-CLW-ADV-00). We also made recommendations around checking for rent exemption (OS-CLW-SUG-00) and specifying authority in seeds (OS-CLW-SUG-01).

Overall, we commend the Clockwork team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/clockwork-xyz/clockwork. This audit was performed against commit 45ab0ad. As a note, even though there were three programs in this repository, our review was only scoped to the thread program.

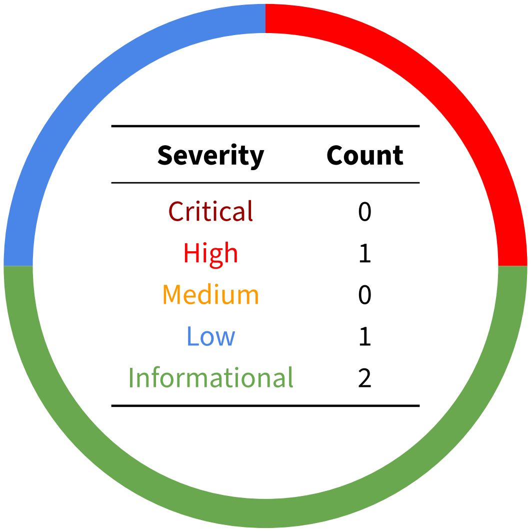
A brief description of the programs is as follows.

Name	Description
thread	Automation engine for the Solana blockchain. Threads enable users to define custom triggers upon which a defined series of instructions will be executed by the distributed Workernet. Examples of triggers include account data changes, time intervals, and on demand.

03 | Findings

Overall, we report 4 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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	1
Medium	0
Low	1
Informational	2

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-CLW-ADV-00	High	Resolved	Account trigger validation doesn't properly validate data changes
OS-CLW-ADV-01	Low	Resolved	An edge case in crank reimbursement calculations could cause unnecessary aborts.

OS-CLW-ADV-00 [high] [resolved] | Account Trigger Validation

Description

Account trigger validation for threads doesn't work as intended.

While the code calculates the hash of the account data, it does not actually compare this calculated value against the prior data hash.

programs/thread/src/objects/thread.rs RUST

```rust
match exec_context.trigger_context {
    TriggerContext::Account {
    data_hash: prior_data_hash,
} => {
    prior_data_hash.hash(&mut hasher);
    hasher.finish()
}
```

As a result, threads that rely on account triggers could be repeatedly invoked, even if the underlying account data did not change.

Remediation

Verify the account's data hash is different than the previous data hash.

programs/thread/src/objects/thread.rs RUST

```rust
require!(
    data_hash.ne(&prior_data_hash),
    ClockworkError::TriggerNotActive
)
```

Patch

Resolved in 831a4c2.

OS-CLW-ADV-01 [low] [resolved] | Crank Reimbursement Edge Case

Description

After the target program invocation, in certain edge cases, the signatory might receive additional lamports.

As a result, the following reimbursement calculation might abort due to use of `checked_sub`.

```rust
let signatory_lamports_post = signatory.lamports();
let signatory_reimbursement = signatory_lamports_pre
    .checked_sub(signatory_lamports_post)
    .unwrap();
```

Remediation

Only perform the `checked_sub` if the signatory's lamport balance decreased after invocation.

Patch

Resolved in 06eaa13.

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 antipatterns and could lead to security issues in the future.

ID	Description
OS-CLW-SUG-00	`thread_withdraw` should check for rent exemption as well.
OS-CLW-SUG-01	Refactor authority seed specifications to be more declarative.

OS-CLW-SUG-00 | Withdrawal Rent Exemption Check

Description

In thread_withdraw, it is recommended to check if the rent exemption is still valid or not, as this function withdraws lamports directly from the thread account.

```rust
// Withdraw balance from thread to the pay_to account
**thread.to_account_info().try_borrow_mut_lamports()? = thread
    .to_account_info()
    .lamports()
    .checked_sub(amount)
    .unwrap();
```

Remediation

Check if the withdrawal would put the account balance below the rent exemption threshold.

Patch

Resolved in 742a970.

OS-CLW-SUG-01 | Refactor Authority Specification

Description

As a general design principle, it might be helpful to specify authority instead of `queue.authority` in the seeds. This way, you strictly relate the queue and the authority, even without the `has_one` check.

```rust
[account(
        mut,
        seeds = [
            SEED_QUEUE,
            queue.authority.as_ref(),
            queue.id.as_bytes(),
        ],
        bump,
        has_one = authority,
        close = close_to
)]
```

Remediation

In the seeds, replace queue with authority.

```rust
[account(
        mut,
        seeds = [
            SEED_THREAD,
            authority.as_ref(),
            queue.id.as_bytes(),
        ],
        bump,
        has_one = authority,
        close = close_to
)]
```

A | Vulnerability Rating Scale

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

Critical

Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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 other 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.