Pyth Aptos

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Harrison Green hgarrereyn@osec.io

Aleksandre Khokhiashvili khokho@osec.io

Contents

01 | Executive Summary

Overview

Pyth Foundation engaged OtterSec to perform an assessment of the `Pyth` program. This assessment was conducted between October 1st and October 7th, 2022.

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 over to streamline patches and confirm remediation. We delivered final confirmation of the patches October 10th, 2022.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

We reported a minor denial of service issue with Pyth account initialization (OS-PYA-ADV-00). We note that this was a similar issue to what was found in Wormhole, and are in discussions with the Aptos team to add utility functions to help mitigate this coin registration vulnerability class as a whole.

We also made recommendations around safeguarding against accidents during deployment and governance (OS-PYA-SUG-00, OS-PYA-SUG-01), as well as formal verification of hidden critical invariants (OS-PYA-VER-01).

Overall, the Pyth team was responsive and a pleasure to work with.

02 | Scope

The source code was delivered to us in a git repository at github.com/pyth-network/pyth-crosschain/tree/main/aptos. This audit was performed against commit `51080bc`.

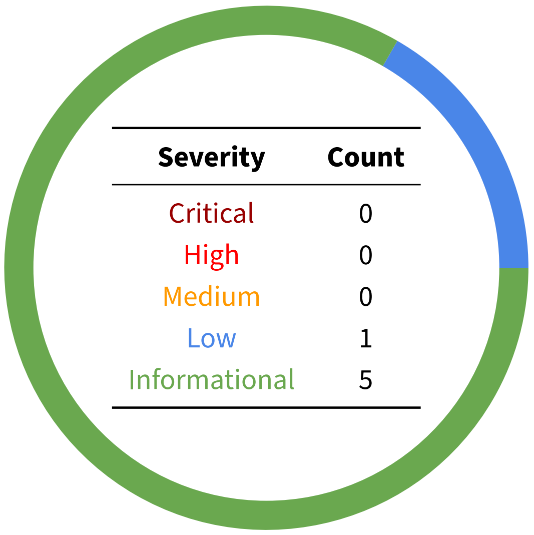
A brief description of the programs is as follows.

Name	Description
pyth-crosschain	Pyth implementation, bridging price information via Wormhole

03 | Findings

Overall, we report 6 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	0
Medium	0
Low	1
Informational	5

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-PYA-ADV-00	Low	Resolved	Pyth deployment can be prevented via resource account DOS.

OS-PYA-ADV-00 [low] [resolved] | Pyth Deployment DOS

Description

Similar to the issue we found in Wormhole, during initialization of the pyth module, it attempts to register an `AptosCoin` account in order to be able to receive fees:

sources/pyth.move RUST

```
module pyth::pyth {
    ...
    fun init_internal(...) {
        ...
        coin::register<AptosCoin>(&wormhole);
    }
    ...
}
```

However, `coin::register` is a one-time operation. If `coin::register` has previously been called on this address, this initialization code will abort and the wormhole program will be unable to initialize.

While it is usually not possible to register coins for users you can not sign for, the Aptos framework provides a special mechanism to register `AptosCoin` for *any* user via `aptos_account::create_account`:

aptos_account.move RUST

```
public entry fun create_account(auth_key: address) {
    let signer = account::create_account(auth_key);
    coin::register<AptosCoin>(&signer);
}
```

Therefore, with this mechanism an attacker could register `AptosCoin` for the wormhole program before deployment in order to prevent it from properly initializing.

Remediation

Check if `AptosCoin` has been registered and conditionally invoke `coin::register` only if it hasn't been registered.

Patch

Resolved in 124589d.

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-PYA-SUG-00	Incorrect `SetGovernanceDataSource` action could leave program in irreparable state.
OS-PYA-SUG-01	Ensure `emitter_chain_ids` and `emitter_addresses` have the same length in `pyth::parse_data_sources`.

OS-PYA-SUG-00 | Prevent Irreparable Governance State

Description

`SetGovernanceDataSource` action is used to change the cross-chain address of the governance message emitter authority.

If an `SetGovernanceDataSource` VAA was executed with an incorrect `DataSource`, it would leave the contract in a state where no new governance messages would be accepted (assuming no one could emit messages as incorrect new data source).

It's improbable that this would happen since it requires an error from a trusted entity. But we recommend safeguarding the implementation and ensuring that the new `DataSource` is valid and operational before switching to it.

Remediation

One possible way to fix the issue would be to introduce a time period where governance messages would be accepted from an old `DataSource`. This would allow for contract repair in case of the error described above.

We recommend introducing a two-step procedure for updating GovernanceDataSource. The first step would stage a new `GovernanceDataSource` but not activate it directly. This new source would be activated once the Aptos contract receives at least one wormhole message from it, implying that the cross-chain address is operational.

OS-PYA-SUG-01 | Assert Length Equality in pyth::parse_data_sources

Description

The `pyth::parse_data_sources` function takes a list of `emitter_chain_ids` and `emitter_addresses`. The function iterates through `emitter_chain_ids` and expects to find a matching address for each entry:

pyth.move MOVE

```move
fun parse_data_sources(
    emitter_chain_ids: vector<u64>,
    emitter_addresses: vector<vector<u8>>): vector<DataSource> {
    let sources = vector::empty();
    let i = 0;
    while (i < vector::length(&emitter_chain_ids)) {
        vector::push_back(&mut sources, data_source::new(
            *vector::borrow(&emitter_chain_ids, i),

     ↪  external_address::from_bytes(*vector::borrow(&emitter_addresses,
     ↪  i))
        ));

        i = i + 1;
    };

    sources
}
```

In the case where `emitter_addresses` is larger than `emitter_chain_ids`, the function will effectively ignore the extra addresses.

Remediation

Add a length equality check to ensure the two arguments have the same length.

Patch

Fixed in #337.

06 | Formal Verification

Here we present a discussion about the formal verification of smart contracts. We include example specifications, recommendations, and general ideas to formalize critical invariants.

ID	Description
OS-PYA-VER-00	Resolving issues with bitwise operations
OS-PYA-VER-01	Document invariants about price timestamps
OS-PYA-VER-02	Specifications around formalizing PriceIdentifier sizing when represented as u8 vectors

OS-PYA-VER-00 | Bitwise Operations

The Move Prover does not work on code that contains bitwise operations such as |, & and ^.

bytecode_translator.rs RUST

```rust
    BitOr | BitAnd | Xor => {
        env.error(&loc, "Unsupported operator");
        emitln!(
            writer,
            "// bit operation not supported: {:?}\nassert false;",
            bytecode
        );
    }
```

In order to get the Move Prover to work, rewrite `parse_magnitude` to use plain subtraction instead of xor operation.

Rewriting this:

sources/i64.move RUST

```rust
    fun parse_magnitude(from: u64, negative: bool): u64 {
        // If positive, then return the input verbatamin
        if (!negative) {
            return from
        };

        // Otherwise convert from two's complement by inverting and
     ↪  adding 1
        let inverted = from ^ 0xFFFFFFFFFFFFFFFF;
        inverted + 1
    }
```

like this:

sources/i64.move RUST

```rust
    fun parse_magnitude(from: u64, negative: bool): u64 {
        // If positive, then return the input verbatamin
        if (!negative) {
            return from
        };
        // Otherwise convert from two's complement by inverting and
     ↪  adding 1
        let inverted = 0xFFFFFFFFFFFFFFFF - from;
        inverted + 1
    }
```

Results in the same functionality(flipping all the bits) implemented in a way which `move-prover` supports.

OS-PYA-VER-01 | Document Timestamp Invariants

We've noticed that there was a hidden invariant with the `PriceFeed` type, where `price.timestamp` was meant to always equal `ema_price.timestamp`. Invariants like these should be documented and made clear. In this case, it's possible to take advantage of `move-prover` and formally verify that this guarantee always holds.

We've modified `pyth::price_feed` module to include the invariant described above:

sources/price_feed.move RUST

```rust
    spec PriceFeed {
        invariant pyth::price::get_timestamp(price) ==
     ↪  pyth::price::get_timestamp(ema_price);
    }

    public fun new(
        price_identifier: PriceIdentifier,
        price: Price,
        ema_price: Price): PriceFeed {
        assert!(pyth::price::get_timestamp(&price) ==
     ↪  pyth::price::get_timestamp(&ema_price), 0);
        PriceFeed {
            price_identifier: price_identifier,
            price: price,
            ema_price: ema_price,
        }
    }
```

In order for the prover to be able to prove this invariant we had to assert it when the type is constructed.

Running `aptos move prove --filter price_feed` tells us that we've codified this invariant successfully.

Additionally we can make sure that every callsite to new passes in `Prices` with identical timestamps. `price_feed::new` is only called once inside `deserialize_price_info`. We've added a following `spec` to verify that passed-in arguments won't result in an assert being triggered:

sources/batch_price_attestation.move RUST

```rust
        spec {
            assert price::get_timestamp(current_price) == ema_timestamp;
        };

        price_info::new(
            attestation_time,
            timestamp::now_seconds(),
            price_feed::new(
                price_identifier,
                current_price,
                pyth::price::new(ema_price, ema_conf, expo,
    ↪   ema_timestamp),
            )
        )
```

`aptos move prove --filter batch_price_attestation` runs successfully and gives us confidence that this code won't run into the above assert.

OS-PYA-VER-02 | PriceIdentifier Sizing

When having types that represent fixed-size addresses, names or identifiers it might be worthwhile to add data invariants that formally verify and ensure that any following code won't accidentally modify the length or initialize the type with the wrong length.

In `pyth` such a type is `PriceIdentifier` and an invariant can be added to this type in the following way:

sources/app/bridge.move RUST

```rust
spec PriceIdentifier {
    invariant vector::length(bytes) == IDENTIFIER_BYTES_LENGTH;
}
```

This invariant can be checked using `aptos move prove --filter price_identifier`.

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