Pyth Network TON Oracle Integration

Security Assessment (Summary Report)

November 26, 2024

Prepared for:
Pyth Data Association
Pyth Data Association

Prepared by: **Guillermo Larregay and Tarun Bansal**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 16, 2024	Pre-project kickoff call
October 28, 2024	Status update meeting #1
November 4, 2024	Delivery of report draft
November 4, 2024	Report readout meeting
November 26, 2024	Delivery of final summary report

Project Targets

The engagement involved a review and testing of the following target.

Pyth Crosschain

Repository	https://github.com/pyth-network/pyth-crosschain
Version	`commit 29e5c9b`
Type	FunC
Platform	TVM

Executive Summary

Engagement Overview

Pyth Network engaged Trail of Bits to review the security of the Pyth TON Oracle integration, which allows the use of Pyth Oracles in the TON network. The price updates are sent by a set of allowed addresses using Wormhole, and the governance account can modify the system configuration.

A team of two consultants conducted the review from October 21 to November 1, 2024, for a total of three engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase using manual processes.

Observations and Impact

During the audit, we reviewed the provided codebase to understand all of the execution flows and edge cases. We also reviewed all messages and the provided format template. We used the existing Pyth and Wormhole documentation as the basis for understanding the expected behavior of the code.

The audit scope was the `target_chains/ton/contracts/` directory of the monorepo. We reviewed the FunC files located in the `contracts/` subdirectory and the test files in the `tests/` subdirectory.

We identified eight total issues, including two high-severity, four medium-severity, and two informational issues. Most are related to the validation of message data and could negatively affect the TON Oracles in several ways, ranging from reverting transactions to potential denial of service.

Recommendations

The findings in this report should be addressed as part of a direct remediation or any refactoring that may occur when addressing other recommendations. Additionally, the test suite should be improved to reflect the codebase's current state and to cover as many test scenarios and edge cases as possible.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The code does not perform any critical arithmetic algorithms. We did not identify any rounding errors or precision losses that could affect the protocol.	**Strong**
Auditing	No events (external messages) are emitted for critical parameter updates, such as changing the data sources set or the Wormhole guardian set. At the time of the audit, we were unaware of any off-chain monitoring system that could trigger notifications if unexpected activity is detected. Although we are unaware of the existence of an incident response plan, the documentation mentions a bug bounty program for reporting vulnerabilities.	**Moderate**
Authentication / Access Controls	All messages from the Pyth network are passed using Wormhole to the TON network. The message emitter address and network are provided in the Wormhole headers, and the order of messages from the same sender is guaranteed using a sequence number. This allows the original sender to be validated. Additionally, for Pyth-specific actions such as governance-initiated parameter changes or updating the price feeds, a set of valid senders is kept and used to provide role-based access control for critical functions in the protocol. Roles do not overlap, and the least-privilege principle is followed. However, we found some issues with these validations that could allow rogue actors to send messages (e.g., TOB-PYTHTON-3, TOB-PYTHTON-6).	**Moderate**

Complexity Management	The codebase is easy to read and understand. In general, functions are short and have a clear scope, and there is no significant code repetition. However, as mentioned in the Code Quality appendix, some improvements could be made. Functions and parameters should be better documented.	**Moderate**
Decentralization	The governance role can upgrade the contract without letting users opt out of the upgrade. Although no direct funds are at risk in the contract, third parties depending on Oracles could be affected. Users can get the price feed information, but only privileged publishers can submit new prices. Governance maintains this set, and permission can be granted or revoked at any time. Published prices are not directly submitted to the Oracle. They must pass through an aggregation stage, which can filter out incorrect price updates before reaching end users.	**Moderate**
Documentation	The documentation is clear and covers how Pyth works at a high level. Users can understand how the system works by reading the existing documentation. However, there is no specific documentation for the TON contracts, as most of the information comes from the few in-code comments, test cases, and additional documentation files provided by the team. Documents specific to developers should be included to ease the onboarding process and provide better and clearer information to collaborators.	**Moderate**
Low-Level Manipulation	We identified a few instances of low-level code consisting of functions defined to wrap assembly instructions. No specific tests are performed on the functions, although they are short and easy to understand.	**Satisfactory**
Testing and Verification	Some parts of the test suite reference old versions of the data structures used in the code, and the test suite does not consider all edge cases.	**Moderate**

	There are no test cases for certain invalid messages, which could have detected some of the issues mentioned in this report. Test coverage is measured for the TypeScript test definitions, but not for the FunC code files, which could result in coverage gaps. Some functions (for example, the Merkle tree validation or the utils functions) do not have specific unit tests.	
Transaction Ordering	We did not perform a review that focused on transaction ordering issues, although we noted that all state-changing messages are atomic and privileged. Since the TON network is asynchronous, specific tests should be performed for message delays in propagation through shards or other situations, if required.	**Further Investigation Required**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	The old guardians cannot update the price feed during the guardian set transition period	Data Validation	**Medium**
2	The update_guardian_set function allows setting an empty guardian set	Data Validation	**High**
3	A single guardian signature can be used to pass the quorum	Data Validation	**High**
4	The upgrade_code_hash variable is not reset after upgrade execution	Data Validation	**Informational**
5	Data sources length is not validated	Data Validation	**Medium**
6	One invalid signature can make verification fail even if quorum can be achieved	Data Validation	**Medium**
7	Price feed publish_time is not validated	Data Validation	**Medium**
8	Risk of hitting the cell limit in the contract	Data Validation	**Informational**

Detailed Findings

1. The old guardians cannot update the price feed during the guardian set transition period

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-PYTHTON-1
Target: `target_chains/ton/contracts/contracts/Wormhole.fc`	

Description

The Wormhole guardian set update can take up to 24 hours to propagate to all of the target chains; therefore, the protocol needs to keep the old guardian set active for 24 hours from the execution of the guardian set upgrade message. However, the strict equality check for the guardian set index does not allow the old guardian set to perform any actions on the protocol once the guardian set has been updated.

```
throw_unless(ERROR_INVALID_GUARDIAN_SET,
    (current_guardian_set_index == vm_guardian_set_index) &
    ((expiration_time == 0) | (expiration_time > now()))
);
```

Figure 1.1: The incorrect comparison operator in `Wormhole.fc#L135-L138`

Exploit Scenario

During the propagation period of the new guardian set, a new price upgrade message arrives, signed by the old guardian set. The Pyth contract fails to verify the message, throwing an `ERROR_INVALID_GUARDIAN_SET` error; as a result, the price update information is lost.

Recommendations

Short term, change the operator to >= from == when comparing the `current_guardian_set_index` and `vm_guardian_set_index`.

Long term, ensure that the test suite covers all possible protocol flows. Add new test cases for each functionality.

2. The update_guardian_set function allows setting an empty guardian set

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-2
Target: `target_chains/ton/contracts/contracts/Wormhole.fc`	

Description

The `update_guardian_set` function does not check that the `guardian_length` is non-zero while parsing the upgrade payload, allowing the current active guardians to set a new empty guardian set.

Once the guardian set becomes empty, it is not possible to make any updates to the data because the `parse_and_verify_wormhole_vm` function throws the error `ERROR_INVALID_GUARDIAN_SET_KEYS_LENGTH` for every action requiring guardian set signature verification.

```
(int, int, int, cell, int) parse_encoded_upgrade(int current_guardian_set_index,
slice payload) impure {
    int module = payload~load_uint(256);
    throw_unless(ERROR_INVALID_MODULE, module == UPGRADE_MODULE);

    int action = payload~load_uint(8);
    throw_unless(ERROR_INVALID_GOVERNANCE_ACTION, action == 2);

    int chain = payload~load_uint(16);
    int new_guardian_set_index = payload~load_uint(32);
    throw_unless(ERROR_NEW_GUARDIAN_SET_INDEX_IS_INVALID, new_guardian_set_index ==
(current_guardian_set_index + 1));

    int guardian_length = payload~load_uint(8);
    cell new_guardian_set_keys = new_dict();
    int key_count = 0;
    while (key_count < guardian_length) {
        builder key = begin_cell();
        int key_bits_loaded = 0;
        while (key_bits_loaded < 160) {
            int bits_to_load = min(payload.slice_bits(), 160 - key_bits_loaded);
            key = key.store_slice(payload~load_bits(bits_to_load));
            key_bits_loaded += bits_to_load;
            if (key_bits_loaded < 160) {
                throw_unless(ERROR_INVALID_GUARDIAN_SET_UPGRADE_LENGTH, ~
payload.slice_refs_empty?());
                payload = payload~load_ref().begin_parse();
            }
        }
        slice key_slice = key.end_cell().begin_parse();
        new_guardian_set_keys~udict_set(8, key_count, key_slice);
        key_count += 1;
    }
    throw_unless(ERROR_GUARDIAN_SET_KEYS_LENGTH_NOT_EQUAL, key_count ==
guardian_length);
    throw_unless(ERROR_INVALID_GUARDIAN_SET_UPGRADE_LENGTH, payload.slice_empty?());

    return (action, chain, module, new_guardian_set_keys, new_guardian_set_index);
}
```

Figure 2.1: `guardian_length` is used without checking it to be greater than zero in *`Wormhole.fc#L189-L223`*

Exploit Scenario

A guardian set upgrade message arrives with an incorrect payload of zero `guardian_length` and zero keys. The Pyth contracts parse the upgrade correctly, setting the guardian set to a null dictionary, and the contracts' Wormhole functionality breaks, not allowing any further action that requires guardian verification.

Recommendations

Short term, check that the `guardian_length` field is non-zero before upgrading the current guardian set, or that the `key_count` is non-zero after upgrading the set.

Long term, ensure that the test suite covers all possible protocol flows. Add new test cases for each functionality.

3. A single guardian signature can be used to pass the quorum

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-3
Target: `target_chains/ton/contracts/contracts/Wormhole.fc`	

Description

The `verify_signatures` function of `Wormhole.fc` parses all of the signatures and their associated guardian indexes from the Wormhole-provided message body. It then recovers the signer Ethereum address from the signature and checks if it matches the provided index's stored guardian address. If the signer matches the guardian key, then the number of valid signatures is increased by one:

```
() verify_signatures(int hash, cell signatures, int signers_length, cell
guardian_set_keys, int guardian_set_size) impure {
    slice cs = signatures.begin_parse();
    int i = 0;
    int valid_signatures = 0;

    while (i < signers_length) {
        int bits_to_load = 528;
        builder sig_builder = begin_cell();

        while (bits_to_load > 0) {
            int available_bits = cs.slice_bits();
            int bits = min(bits_to_load, available_bits);
            sig_builder = sig_builder.store_slice(cs~load_bits(bits));
            bits_to_load -= bits;

            if (bits_to_load > 0) {
                cs = cs~load_ref().begin_parse();
            }
        }

        slice sig_slice = sig_builder.end_cell().begin_parse();
        int guardian_index = sig_slice~load_uint(8);
        int r = sig_slice~load_uint(256);
        int s = sig_slice~load_uint(256);
        int v = sig_slice~load_uint(8);
        (_, int x1, int x2, int valid) = check_sig(hash, v >= 27 ? v - 27 : v, r,
s);
        throw_unless(ERROR_INVALID_SIGNATURES, valid);
        int parsed_address = pubkey_to_eth_address(x1, x2);
        (slice guardian_key, int found?) = guardian_set_keys.udict_get?(8,
guardian_index);
        int guardian_address = guardian_key~load_uint(160);
        throw_unless(ERROR_INVALID_GUARDIAN_ADDRESS, parsed_address ==
guardian_address);
        valid_signatures += 1;
        i += 1;
    }

    ;; Check quorum (2/3 + 1)
    ;; We're using a fixed point number transformation with 1 decimal to deal with
rounding.
    throw_unless(ERROR_NO_QUORUM, valid_signatures >= (((guardian_set_size * 10) /
3) * 2) / 10 + 1);
}
```

Figure 3.1: The `verify_signatures` function in `Wormhole.fc#L86-L124`

At the end of the execution, the function checks that the total number of valid signatures is more than 2/3 + 1 set of the stored guardian set.

However, it never checks that the same guardian index and address are repeated in the user-provided message. A rogue guardian can repeat the same signature and index multiple times to pass the quorum and update the guardian set.

Exploit Scenario

The current guardian set consists of 13 guardians. A rogue guardian discovers that by repeating its own signature 13 times, it can bypass the Pyth signature verification. From this point on, it can create any signed arbitrary message and send it directly to Pyth without Wormhole consensus.

Recommendations

Short term, check that all of the signatures in the Wormhole messages are from a different signer. Ensure that the quorum is achieved with different signatures from the guardian set.

Long term, consider all edge cases in the signature process, and prepare countermeasures for invalid or malicious payloads. Add new test cases to the test suite.

4. The upgrade_code_hash variable is not reset after upgrade execution

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-PYTHTON-4
Target: `target_chains/ton/contracts/contracts/Pyth.fc`	

Description

The `execute_upgrade_contract` function does not reset the value of the `upgrade_code_hash` global variable after upgrading the contract code:

```
() execute_upgrade_contract(cell new_code) impure {
    load_data();
    int hash_code = cell_hash(new_code);
    throw_unless(ERROR_INVALID_CODE_HASH, upgrade_code_hash == hash_code);

    ;; Set the new code
    set_code(new_code);

    ;; Set the code continuation to the new code
    set_c3(new_code.begin_parse().bless());

    ;; Throw an exception to end the current execution
    ;; The contract will be restarted with the new code
    throw(0);
}
```

Figure 4.1: The `execute_upgrade_contract` function in `Pyth.fc#L257-L271`

This allows anyone to call the function with the same code and succeed.

Recommendations

Short term, reset the `upgrade_code_hash` global variable after successfully upgrading the contract.

Long term, verify that all global variables keep a consistent value after the contract execution.

5. Data sources length is not validated

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-5
Target: `target_chains/ton/contracts/contracts/Pyth.fc`	

Description

In `execute_data_sources`, the new set of data sources comes from the `payload` slice. However, there is no check that all of the data sources in the payload were processed and added to the new set.

```
() execute_set_data_sources(slice payload) impure {
    int num_sources = payload~load_uint(8);
    cell new_data_sources = new_dict();

    repeat(num_sources) {
        (cell data_source, slice new_payload) = read_and_store_large_data(payload,
272); ;; 272 = 256 + 16
        payload = new_payload;
        slice data_source_slice = data_source.begin_parse();
        int emitter_chain_id = data_source_slice~load_uint(16);
        int emitter_address = data_source_slice~load_uint(256);
        cell data_source = begin_cell()
            .store_uint(emitter_chain_id, 16)
            .store_uint(emitter_address, 256)
        .end_cell();
        int data_source_key = cell_hash(data_source);
        new_data_sources~udict_set(256, data_source_key,
begin_cell().store_int(true, 1).end_cell().begin_parse());
    }

    is_valid_data_source = new_data_sources;
}
```

Figure 5.1: The `execute_set_data_sources` function in `Pyth.fc#L308-L327`

This can allow the new data sources dictionary to have fewer sources than expected if the payload is incorrect. In the worst case, if `num_sources` is zero, the system is left in a state with no valid data sources, preventing price updates from occurring until governance fixes the situation.

Exploit Scenario

Governance sends a malformed message to update the data sources set, where the `num_sources` field is set to zero, and five data sources are meant to be added. After `execute_set_data_sources` is executed, the global `is_valid_data_source` dictionary is empty, and no more price updates can be performed.

Recommendations

Short term, ensure that the whole message is processed by checking that the payload slice is empty before setting the new data sources dictionary.

Long term, ensure that the test suite correctly covers all possible protocol flows. Add new test cases for each functionality.

6. One invalid signature can make verification fail even if quorum can be achieved

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-6
Target: `target_chains/ton/contracts/contracts/Wormhole.fc`	

Description

The `verify_signatures` function throws when a signature is invalid or if the recovered signer address does not match the guardian key for that index:

```
() verify_signatures(int hash, cell signatures, int signers_length, cell
guardian_set_keys, int guardian_set_size) impure {
    slice cs = signatures.begin_parse();
    int i = 0;
    int valid_signatures = 0;

    while (i < signers_length) {
        int bits_to_load = 528;
        builder sig_builder = begin_cell();

        while (bits_to_load > 0) {
            int available_bits = cs.slice_bits();
            int bits = min(bits_to_load, available_bits);
            sig_builder = sig_builder.store_slice(cs~load_bits(bits));
            bits_to_load -= bits;

            if (bits_to_load > 0) {
                cs = cs~load_ref().begin_parse();
            }
        }

        slice sig_slice = sig_builder.end_cell().begin_parse();
        int guardian_index = sig_slice~load_uint(8);
        int r = sig_slice~load_uint(256);
        int s = sig_slice~load_uint(256);
        int v = sig_slice~load_uint(8);
        (_, int x1, int x2, int valid) = check_sig(hash, v >= 27 ? v - 27 : v, r,
s);
        throw_unless(ERROR_INVALID_SIGNATURES, valid);
        int parsed_address = pubkey_to_eth_address(x1, x2);
        (slice guardian_key, int found?) = guardian_set_keys.udict_get?(8,
guardian_index);
        int guardian_address = guardian_key~load_uint(160);
        throw_unless(ERROR_INVALID_GUARDIAN_ADDRESS, parsed_address ==
guardian_address);
        valid_signatures += 1;
        i += 1;
    }

    ;; Check quorum (2/3 + 1)
    ;; We're using a fixed point number transformation with 1 decimal to deal with
rounding.
    throw_unless(ERROR_NO_QUORUM, valid_signatures >= (((guardian_set_size * 10) /
3) * 2) / 10 + 1);
}
```

Figure 6.1: The `verify_signatures` function in `Wormhole.fc#L86-L124`

This behavior prevents validating messages where one signature is considered invalid, even if the remaining signatures are correct and enough to achieve quorum.

Exploit Scenario

A message is received from Wormhole with 18 signatures, and the current guardian set is composed of 20 signers. When the message is analyzed, signature number 14 is invalid. The message is not validated and therefore its payload is not executed, even if the remaining 17 signatures are correct.

Recommendations

Short term, reconsider the calculation of quorum to account for invalid signatures.

Long term, add the relevant test cases to the test suite. Consider cases where the different actors are malicious. Document the expected behavior when invalid signatures are present in the received Wormhole message.

7. Price feed publish_time is not validated	
Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-7
Target: `target_chains/ton/contracts/contracts/Pyth.fc`	

Description

The `upgrade_price_feeds` function takes the `publish_time` variable from the message payload, and checks whether the update is more recent than the one stored. However, it never validates that the incoming `publish_time` is not in the future. Storing a timestamp ahead of the current time will prevent further price updates for that particular feed up to that specific point in time.

```
if (publish_time > latest_publish_time) {
    cell price_feed = begin_cell()
        .store_ref(store_price(price, conf, expo, publish_time))
        .store_ref(store_price(ema_price, ema_conf, expo, publish_time))
    .end_cell();

    latest_price_feeds~udict_set(256, price_id,
begin_cell().store_ref(price_feed).end_cell().begin_parse());
}
```

Figure 7.1: Usage of the `publish_time` before updating the price feed in `Pyth.fc#L212-L219`

Exploit Scenario

A price feed update message comes with a publish timestamp set 24 hours in the future. The price update is stored, and all further updates for the next 24 hours will be ignored.

Recommendations

Short term, validate the `publish_time` variable value using the current block timestamp as a reference.

Long term, document the boundaries for each message field value, and consider messages with out-of-bounds cases in the test suite.

8. Risk of hitting the cell limit in the contract	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTHTON-8
Target: `target_chains/ton/contracts/contracts/common/storage.fc`	

Description
According to FunC documentation, the maximum limit of cells in a single smart contract is 65,536. As a result, the maximum number of entries in a dictionary with no repeated cells is 32,768.

In Pyth, several dictionaries are stored:

- `latest_price_feeds`, whose maximum amount of entries is limited by the number of supported price IDs.
- `data_sources`, which is fixed at the time of deployment.
- `is_valid_data_source`, which is defined by the `execute_set_data_sources` function every time it is called.
- `guardian_sets`, which increases every time a new guardian set is announced by Wormhole.
- `consumed_governance_actions`, which increases every time a new governance instruction is executed.

If any of the ever-increasing dictionaries reaches a high enough number of entries, the limit could be reached. Additionally, updating extensive dictionaries costs a significant amount of gas, so it is also possible that, over time, modifications to these dictionaries become excessively expensive.

Recommendations
Short term, test and determine the realistic on-chain storage needs for Pyth contracts. When possible, do not store unnecessary information in dictionaries.

Long term, consider using contract storage sharding for keeping large amounts of data on-chain.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.

Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains findings that are not directly related to security issues but could improve clarity, robustness, or auditability.

- **Avoid using literal constants (magic numbers) for field values.** Replace them with named constants. Some occurrences include:
    - `version == 1` in Wormhole.fc
    - `action == 2` in Wormhole.fc
    - `message_type == 0` in Pyth.fc
- **Remove unnecessary checks for governance actions.** The `execute_governance_payload` does not handle the `SET_VALID_PERIOD` and `REQUEST_GOVERNANCE_DATA_SOURCE_TRANSFER` action cases, which can be removed.
- **Consider adding the `impure` modifier to functions.** The `get_guardian_set_internal`, `read_and_verify_header`, and `parse_governance_instruction` functions can throw, but they are not marked as impure. This is not currently an issue in the code, as all usages of these functions use the returned values, but can be an issue in the future if a new call is made and the return values are ignored. In this scenario, the compiler will remove the function call.
- **Avoid redefining global variables.** In FunC, if a global variable is redefined locally, it is not considered a new variable. Instead, the global variable is used, as there is no concept of local variable shadowing.
    - The first argument of the `parse_encoded_upgrade` function matches the `current_guardian_set_index` global variable. The global value will be affected if the code is modified and a write operation is performed on the variable.
    - The `governance_chain_id` global variable is overwritten with itself.
- **The value of the `governance_contract` global can be used directly.** The call to `get_governance_contract` can have unexpected side effects because of `load_data`.
- **Ensure that the storage is read only once in each transaction or that all global variable changes are persisted immediately.** In the current state of the contracts, some execution flows call `load_data` several times. If, at some point, one of the

global variables is changed, and `load_data` is called before the new value is saved to storage, the new value will be lost.

- **Remove unused variables and document the reason why they are not used.** Some message fields and variables are ignored, such as the `prev_publish_time` field in the `update_price_feeds` function. Returning and ignoring values is a bad practice, as it suggests that the code fails to use or perform additional checks on those values.
- **Remove the `dump_guardian_sets` function**. Production code should not have debug functions.
- **Remove unused global variables from storage.** The `data_sources` and `num_data_sources` global/storage variables are not used in the codebase.
- **Document the different fees, how they are calculated, and who will pay them.** The contract has three fees: compute fee, update fee, and the TON network storage fee. In the current state of the contract, these are not documented. The first two are assumed to be the gas required to perform the update action. In any case, no functions allow the withdrawal of the contract balance, and it is not clear who is responsible for ensuring that the contract balance is enough to cover storage costs at all times.

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not a comprehensive analysis of the system.

From November 13 to November 14, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Pyth Network team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the eight issues described in this report, Pyth Network has resolved five issues and has not resolved the remaining three issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	The old guardians cannot update the price feed during the guardian set transition period	Resolved
2	The update_guardian_set function allows setting an empty guardian set	Resolved
3	A single guardian signature can be used to pass the quorum	Resolved
4	The upgrade_code_hash variable is not reset after upgrade execution	Resolved
5	Data sources length is not validated	Resolved
6	One invalid signature can make verification fail even if quorum can be achieved	Unresolved
7	Price feed publish_time is not validated	Unresolved
8	Risk of hitting the cell limit in the contract	Unresolved

Detailed Fix Review Results

TOB-PYTHTON-1: The old guardians cannot update the price feed during the guardian set transition period

Resolved in PR #2059. The new operator in the guardian set index validation allows old guardians to send messages during the guardian set transition period.

TOB-PYTHTON-2: The update_guardian_set function allows setting an empty guardian set

Resolved in PR #2073. The `parse_encoded_upgrade` function now checks that the new guardian set is not empty.

TOB-PYTHTON-3: A single guardian signature can be used to pass the quorum

Resolved in PR #2073. The `verify_signatures` function now ensures that the signer guardian index does not have duplicate entries, thus preventing the use of repeated signatures to pass the quorum.

TOB-PYTHTON-4: The upgrade_code_hash variable is not reset after upgrade execution

Resolved in PR #2073. The `execute_upgrade_contract` now sets the `upgrade_code_hash` state variable to 0.

TOB-PYTHTON-5: Data sources length is not validated

Resolved in PR #2092. After processing all of the data sources, the `execute_upgrade_contract` function now checks that the `payload` slice is empty.

TOB-PYTHTON-6: One invalid signature can make verification fail even if quorum can be achieved

Unresolved. The issue has not been resolved.

The client provided the following context for this finding's fix status:

> *This behavior is consistent with the current Ethereum contract. Consumers are always expected to provide valid signatures.*

TOB-PYTHTON-7: Price feed publish_time is not validated

Unresolved. The issue has not been resolved.

The client provided the following context for this finding's fix status:

> *We've identified that timestamps on Pythnet are not guaranteed to be synchronized with timestamps on other chains. There are no strict guarantees on time drifts, and Pythnet's view of time might be ahead of other chains.*
>
> *Applying this logic to TON, we conclude:*

1. *There shouldn't be a requirement that the publish time isn't in the future. If TON is behind Pythnet, such a check would prevent publishing of any prices.*
2. *The exploit scenario specified requires a malicious Wormhole message. In such a case, more serious issues could arise.*
3. *As long as the majority of validators in Pythnet are honest (due to the timestamp calculation algorithm), the current timestamp would never drift into the future.*

TOB-PYTHTON-8: Risk of hitting the cell limit in the contract
Unresolved. The issue has not been resolved.

The client provided the following context for this finding's fix status:

> *While we acknowledge the finding of the cell limit, out of all the dictionaries, only latest_price_feeds will be affected by this issue since the rest will not change that much, furthermore, our analysis indicates that reaching the cell limit of 32,768 entries is not an immediate concern given current usage patterns and projected growth rates. Although contract storage sharding is a valid long-term solution, implementing it at this stage would introduce unnecessary complexity and potential risks that outweigh the theoretical benefits. We have decided to maintain the current implementation while continuing to monitor dictionary sizes as part of our regular system health checks.*

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.