Pyth Data Association Pythnet

Security Assessment

July 31, 2023

Prepared for:
Pyth Data Association

Prepared by: **Samuel Moelius, Troy Sargent, and Vara Prasad Bandaru**

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.
228 Park Ave S #80688
New York, NY 10003
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

Executive Summary

Engagement Overview

Pyth Data Association engaged Trail of Bits to review the security of various components of Pythnet, an application-specific blockchain operated by Pyth's data providers.[1] Pythnet acts as a conduit over which prices are fed to various other blockchains.

A team of three consultants conducted the review from May 8 to June 2, 2023, for a total of nine engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The system is extremely complex, and various aspects of it do not appear to have kept up with this complexity:

- While there is documentation for the system as a whole, individual components lack documentation.
- Many critical components lack unit tests. In many places where components interact, integration tests are still in development.
- Several components rely on outdated dependencies (TOB-PYTH-13).

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Pyth Data Association take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Develop metrics to determine each component's maintenance cost, and track those metrics over time.** Appendix E provides recommendations to aid in developing such metrics. If a component is deemed to have a maintenance cost disproportionate to its benefit, consider removing the component.
- **Write documentation for each component.** Having documentation for each component will aid in its implementation and in the implementation of the components that must interact with it.

[1] https://docs.pyth.network/design-overview/pythnet

- **Add tests.** Tests are one of the most effective ways of exposing errors, so it is important that each component has a comprehensive set of unit tests and that integration tests exist wherever components interact.
- **Add logging throughout the codebase.** Logging can help raise alerts about abnormal situations, including malicious actions. For dealing with such situations, appendix D provides incident response plan recommendations.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	3
Low	5
Informational	9
Undetermined	2

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	3
Auditing and Logging	1
Configuration	1
Data Validation	7
Denial of Service	1
Patching	3
Testing	1
Undefined Behavior	4

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

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

The following engineers were associated with this project:

Samuel Moelius, Consultant
samuel.moelius@trailofbits.com

Troy Sargent, Consultant
troy.sargent@trailofbits.com

Vara Prasad Bandaru, Consultant
vara.bandaru@trailofbits.com

Project Timeline

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

Date	Event
May 5, 2023	Pre-project kickoff call
May 12, 2023	Status update meeting #1
May 19, 2023	Status update meeting #2
May 26, 2023	Status update meeting #3
June 5, 2023	Delivery of report draft
June 5, 2023	Report readout meeting
June 9, 2023	Delivery of updated report draft
June 14, 2023	Delivery of updated report draft
July 31, 2023	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of various components of the Pythnet blockchain. Specifically, we sought to answer the following non-exhaustive list of questions:

- Did the modifications to the Solana validator introduce vulnerabilities (e.g., denial-of-service vectors)?
- Are the Pythnet multisig wallets configured such that restricted actions can be performed and programs can be upgraded only with a quorum?
- Is the Merkle tree library robust against verification of forged leaves?
- Does the oracle program properly authenticate data publishers?
- Do calculations performed with wrapping arithmetic result in incorrect data?
- Are each account's program ID, ownership, and/or discriminator checked prior to interaction with the account?

Project Targets

The engagement involved a review and testing of the targets listed below.

Pythnet validator, accumulator

Repository	https://github.com/pyth-network/pythnet
Version	`83a5f1bdc93d63973c755cb0a9b4f02279f4cdec` (May 5, 2023) `28647f6a3ea2f1cd6aea71e12645956736b7d9c2` (May 8, 2023)
Type	Rust
Platform	POSIX

Message buffer, remote executor (program and CLI), Merkle tree library

Repository	https://github.com/pyth-network/pyth-crosschain
Version	`1902dbaa3080281b4e5b5748df630c0b13cc22ec` (May 5, 2023) `78d3c5c4caa6aae9452868542c0b649f36fcc205` (May 9, 2023)
Type	Rust/Anchor
Platform	Pythnet (Solana fork)

xc_admin front end

Repository	https://github.com/pyth-network/pyth-crosschain
Version	`78d3c5c4caa6aae9452868542c0b649f36fcc205` (May 9, 2023)
Type	Typescript/Next.js
Platform	Web application

Oracle (Rust portion)

Repository	https://github.com/pyth-network/pyth-client
Version	`410f0bd094f0728715f555e0cc84b33888f8337f` (May 17, 2023)
Type	Rust
Platform	Pythnet (Solana fork)

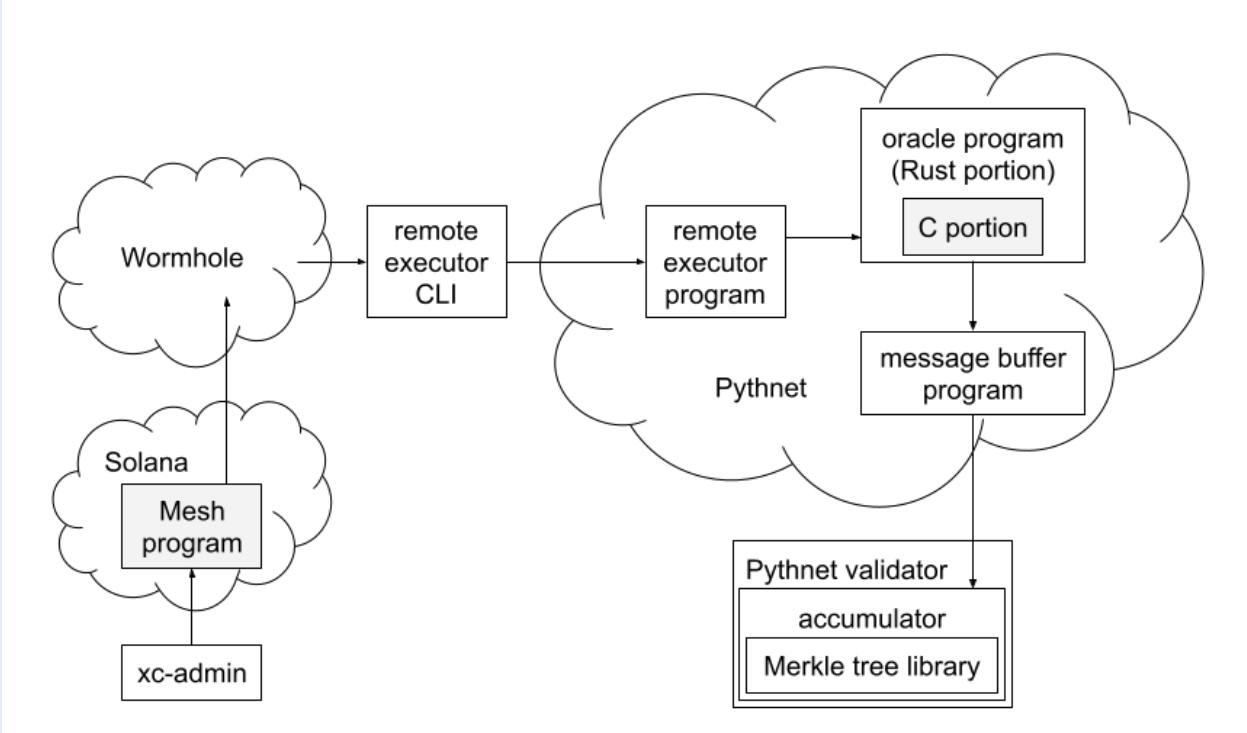

Figure 1: Diagram of the reviewed components and how they interact: the components in gray (the Mesh program and the C portions of the oracle program), while relevant to the audit, were not reviewed.

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches include the following:

- **Documentation review.** We carefully reviewed the How Pyth Works documentation and the Pyth operations key management document.
- **Manual review.** We reviewed the differences between the Solana and Pythnet validators, as well as the source code for the accumulator, the message buffer, the Merkle tree library, the Rust portions of the oracle program, the remote executor program, and the remote executor command-line interface (CLI). We also briefly reviewed parts of the `xc-admin` front end related to proposal handling.
- **Static analysis.** We ran Clippy with the pedantic lints enabled on all the bodies of Rust source code mentioned in the previous bullet.
- **Dependency review.** We used `cargo-audit` to look for dependencies with known vulnerabilities. We also used `cargo-upgrade` to check for outdated dependencies.
- **Test coverage review.** Where tests were available, we verified that they passed. We also computed the oracle program's test coverage using `cargo-llvm-cov`. Wherever there appeared to be gaps in the coverage report, we manually checked whether an applicable test existed in the source code. Finally, we ran Necessist to look for bugs in the oracle program's tests.
- **Fuzzing.** We adapted the oracle program's `quickcheck` tests into fuzz targets and fuzzed them. We also fuzzed the oracle program's `process_instruction` function.
- **Deployment review.** We reviewed the deployment of the Solana multisig wallets and the programs on Pythnet. For the former, we used Solana Explorer and Solscan. For the latter, we connected to `pythnet.rpcpool.com` using the `solana` command-line tool.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- There is insufficient documentation describing the correct behavior of each of the components. This hindered the review in two ways:
    - It was often difficult to determine what the correct behavior of the component under review should be. For example, the oracle instructions are

largely undocumented, aside from code comments. Thus, we often had to ask about or infer the correct behavior of oracle instructions.

    - Often, the only way to determine whether an interacted-with component was being used correctly was to look at the source code for that component. Ideally, documentation would be available for this purpose because it can lead to more expedient answers.

- We did not review the Mesh source code because it was out of scope, even though the Mesh program is critical to Pythnet's functionality. We did briefly review each instruction's signature to determine its purpose, but no more.
- We reviewed only the oracle program's Rust source code, not its C source code, which was out of scope. In some cases, this meant we had an incomplete understanding of its functionality or the cause of a failure (TOB-PYTH-21).
- We had very little time to review the `xc-admin` front end (approximately two engineer-days). However, this was the lowest-priority target.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

- Clippy: A collection of lints to catch common mistakes and improve Rust code
- `cargo-audit`: A Cargo subcommand to audit `Cargo.lock` files for crates with security vulnerabilities reported to the RustSec Advisory Database
- `cargo-llvm-cov`: A Cargo subcommand for obtaining and analyzing LLVM source–based code coverage
- `cargo-upgrade`: A Cargo subcommand to upgrade dependencies in a `Cargo.toml` file to their latest versions
- Necessist: A mutation testing tool that can identify bugs in tests

Fuzzing

We developed fuzzers for the targets listed in the table below.

The `test_price_feed_message_roundtrip`, `test_twap_message_roundtrip`, and `test_twap` targets were adaptations of existing `quickcheck` tests. Fuzzing the test_twap target resulted in an arithmetic overflow (TOB-PYTH-18).

We also fuzzed the oracle program's `process_instruction` function, allowing the fuzzer to mutate both instruction data and account contents. This produced panics in the `c_upd_aggregate` function, which is called by the `upd_price` function before the price account is validated.

Fuzz Target	Finding
`test_price_feed_message_roundtrip`	None
`test_twap_message_roundtrip`	None

`test_twap`	TOB-PYTH-18
`process_instruction` (in oracle program)	TOB-PYTH-21

Deployment Review

We were asked to review the Pyth deployment by checking the client-provided properties in the table below. All the properties but one (TOB-PYTH-17) appear to hold as stated.

ID	Property	Finding
1	Each of the following tasks can happen only with a 4/8 proposal in the upgrade multisig wallet `FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj`: a. Upgrading the oracle program on Pythnet `FsJ3A3u2vn5cTVofAjvy6y5kwABJAqYWpe4975bi2epH` b. Upgrading the remote executor on Pythnet `exe6S3AxPVNmy46L4Nj6HrnnAVQUhwyYzMSNcnRn3qq` c. Upgrading the Mesh program on Solana `SMPLVC8MxZ5Bf5EfF7PaMiTCxoBAcmkbM2vkrvMK8ho` d. Configuring (i.e., changing the threshold and members) of the upgrade multisig `FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj` e. Configuring (i.e., changing the threshold and members) of the price feed multisig `92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8`	None
2	The oracle program can be configured only with a 3/6 proposal in the price feed multisig wallet `92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8`	TOB-PYTH-17

Table 1: Properties verified during the deployment review

Below, we describe our process for checking the above properties.

1. **Each of the following tasks can happen only with a 4/8 proposal in the upgrade multisig wallet `FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj`.**

   We verified that multisig `FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj` has a threshold of 4 and eight registered keys (see table 2 below).

We determined that the Mesh program's upgrade authority is `6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT`. We verified that this address is a PDA generated from the following seeds:

- `squad`
- `FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj`
- 1 (authority index)
- `authority`
- 254 (bump)
- `SMPLVC8MxZ5Bf5EfF7PaMiTCxoBAcmkbM2vkrvMK8ho` (program ID)

This PDA is owned by the multisig program. Thus, upgrading the multisig would require a 4/8 proposal in the upgrade multisig.

a. **Upgrading the oracle program on Pythnet `FsJ3A3u2vn5cTVofAjvy6y5kwABJAqYWpe4975bi2epH`**

   By connecting to https://pythnet.rpcpool.com, we determined that the program's upgrade authority is `DgpbK8SiypiUHBkBTAunMnwRWF3McGGR4iKxTrTfTXq4`. We verified that this address is a PDA generated from the following seeds:

   - `EXECUTOR_KEY`
   - `6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT`
   - 254 (bump)
   - `exe6S3AxPVNmy46L4Nj6HrnnAVQUhwyYzMSNcnRn3qq` (program ID)

   The `exe6S3AxPVNmy46L4Nj6HrnnAVQUhwyYzMSNcnRn3qq` program ID is the address of the remote executor on Pythnet.

b. **Upgrading the remote executor on Pythnet `exe6S3AxPVNmy46L4Nj6HrnnAVQUhwyYzMSNcnRn3qq`**

   By connecting to https://pythnet.rpcpool.com, we determined that the program's upgrade authority is `DgpbK8SiypiUHBkBTAunMnwRWF3McGGR4iKxTrTfTXq4` (the same as that of the oracle program).

c. **Upgrading the Mesh program on Solana SMPLVC8MxZ5Bf5EfF7PaMiTCxoBAcmkbM2vkrvMK8ho**

The steps required to verify this property are described under item 1 above.

d. **Configuring (i.e., changing the threshold and members) of the upgrade multisig FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj**

We determined that multisig FVQyHcooAtThJ83XFrNnv74BcinbRH3bRmfFamAHBfuj has external authority 6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT. We scanned the instructions in the Mesh source code for those that could change the threshold or add or remove members. All such instructions used an `MsAuth` or `MsAuthRealloc` context, which requires the external authority as a signer.

As stated in item 1, 6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT is a PDA that belongs to the Mesh program and is associated with the upgrade multisig. Thus, configuring the upgrade multisig would require a 4/8 proposal in the upgrade multisig.

e. **Configuring (i.e., changing the threshold and members) of the price feed multisig 92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8**

We determined that multisig 92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8 has external authority 6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT. Additional details are the same as for item d.

2. **The oracle program can be configured only with a 3/6 proposal in the price feed multisig 92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8**

We verified that multisig 92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8 has a threshold of 3 and six registered keys (see table 2 below).

However, strictly speaking, this property does not hold. The price feed multisig uses the same program as the upgrade multisig (SMPLVC8MxZ5Bf5EfF7PaMiTCxoBAcmkbM2vkrvMK8ho). This program's upgrade authority is 6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT, which is a PDA associated with the upgrade multisig. Thus, a 4/8 proposal in the upgrade multisig could change the Mesh program's code, effectively overriding the 3/6 requirement in the price feed multisig (TOB-PYTH-17).

Key	FVQyHco...	92hQkq8...
E5KR7yfb9UyVB6ZhmhQki1rM1eBcxHvyGKFZakAC5uc	✔	✔
opsLibxVY7Vz5eYMmSfX8cLFCFVYTtH6fr6MiifMpA7	✔	
3S2N8k3tKUMtp9m1fp4GSc5k6AtZfkuujpqAfZV6K846	✔	
7fbLBsSd5oZGtYgw8eJVvAqrMbQUTEdDTtgNnpqqTw7y	✔	
ADp5q8hLHH9FUeNMSzvMJWKnZ9cWdFQL9YkJspB9dHk4	✔	
Bt46Ktdz6PWg8YqgEUC6dwr4ottkcAuLFS2vRRR1KKjA	✔	✔
C2x5zd2CTDAQWWX3ngJ4qrFKudVXo5DNLsjXwr8tkQ9d	✔	✔
H6uQwqZBebe8RRwjzyuJkkngTkgjPvYwnXXdpK3qoTnp	✔	✔
ACzP6RC98vcBk9oTeAwcH1o5HJvtBzU59b5nqdwc7Cxy		✔
Eh32UTCozwrmyhjEazGMuuWG1e2bXG2kNYmhzHRNp5Sa		✔

Table 2: Keys registered to the two multisig wallets

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 contains several instances where overflows may occur but are not explicitly handled (TOB-PYTH-18 and TOB-PYTH-19). Adopting checked arithmetic or justifying uses of unchecked arithmetic would specify the desired behavior of arithmetic operations more clearly.	**Moderate**
Auditing	The remote executor, oracle, and message buffer programs do not generate logs for critical operations (TOB-PYTH-11). In some cases, errors are silently ignored. The system would benefit from monitoring in the event a data provider experiences an outage or bug, as these may require a response. Such failure cases should be identified, and a plan to respond to them should be created (see appendix D). Pyth Data Association notes that it has an off-chain monitoring system for the oracle program specifically and that the system uses account data directly, rather than events.	**Weak**
Authentication / Access Controls	There are multiple unused roles in the oracle program that are instead designated to a single account. It would be ideal to follow the principle of least privilege and distribute roles. Transferring roles could be made more robust by requiring the new role holder to provide a signature (TOB-PYTH-5). System documentation should describe the roles and list which actors are to fill them.	**Moderate**

Complexity Management	The risks associated with maintaining a fork of Solana (TOB-PYTH-1) may exceed the functionality it provides (see appendix E). The functionality of the smart contracts could potentially be simplified by requiring a quorum of data providers to aggregate price data off-chain and post a signed message similar to a multisig message. Components are not documented, which makes them difficult to understand and test. Component interfaces are similarly undocumented and, in many cases, untested.	**Further Investigation Required**
Cryptography and Key Management	We did not consider cryptography or key management because the Mesh multisig source code was out of scope.	**Not Considered**
Decentralization	The oracle program relies on trusted data providers, and updates to the smart contracts are performed by a single entity (the multisig wallet).	**Weak**
Documentation	While inline documentation is consistently available, the system lacks explanations of the architecture, actors, and important assumptions. It would be beneficial to create diagrams that show which actions are available to actors and what important preconditions/postconditions exist.	**Weak**
Front-Running Resistance	Rating the front-running resistance would be premature without reviewing the system in its entirety. For example, one concern is that operation ordering could affect the oracle program, whose C code was out of scope.	**Further Investigation Required**
Low-Level Manipulation	Low-level manipulation of data structures appears prominently in the Pythnet validator, the Merkle tree library, and the oracle program. These parts of the system should employ fuzzing to help identify problems. The lack of fuzzing may have contributed to some of the identified findings (TOB-PYTH-8, TOB-PYTH-9, and TOB-PYTH-21).	**Moderate**

Testing and Verification	The Pythnet validator lacks unit tests (TOB-PYTH-2). It does have limited integration tests; however, even those do not exercise all critical functionality. Additionally, some of the tests did not build without modifications to the code or toolchains. While `quickcheck` tests are used in some places, it would be ideal to use coverage-guided fuzzing for greater effectiveness. On a positive note, the oracle program has a relatively thorough test suite. However, even the oracle program would benefit from fuzzing, as well as testing the code guarded by the `pythnet` feature. Pyth Data Association notes that testing was being developed concurrently with the audit.	**Weak**

Summary of Findings

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

ID	Title	Type	Severity
1	Maintaining a forked codebase carries risk	Patching	Informational
2	Insufficient test coverage	Testing	Informational
3	Wormhole dependency is not pinned	Patching	Informational
4	Use of environment variables risks misconfiguration	Configuration	Informational
5	Insecure transfer of admin role	Access Controls	Medium
6	Insufficient validations on admin key in initialize instruction	Access Controls	Low
7	Buffer accounts cannot be deleted after updating allowed programs	Undefined Behavior	Low
8	Validator could panic while reading buffer account containing 255 messages	Data Validation	High
9	Resizing a buffer account could lead the validator to panic	Data Validation	High
10	Program may not write all messages even when account has free space	Undefined Behavior	Low
11	Insufficient event generation	Auditing and Logging	Low
12	Remote executor lacks overflow protection	Data Validation	Informational

13	Outdated dependencies	Patching	**Informational**
14	Account size calculations are undocumented	Undefined Behavior	**Informational**
15	Remote executor CLI error handling could lead to loss of rent	Undefined Behavior	**Low**
16	Inefficient prove function could lead to DoS	Denial of Service	**Informational**
17	Price feed multisig can be upgraded without 3/6 proposal	Access Controls	**Undetermined**
18	Arithmetic overflow in PriceCumulative::update	Data Validation	**Medium**
19	Potential overflow in get_status_for_update	Data Validation	**Informational**
20	Price and publisher accounts are not properly closed	Data Validation	**Medium**
21	Price account is not validated before calling c_upd_aggregate	Data Validation	**Undetermined**

Detailed Findings

1. Maintaining a forked codebase carries risk	
Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-PYTH-1
Target: Pythnet validator	

Description
The Pythnet validator is a fork of the Solana validator. Thus, Pythnet could be susceptible to bugs that are silently patched on Solana.

Blockchain maintainers frequently patch bugs without revealing their security implications so as not to tip off attackers. This has occurred on Ethereum and Optimism, for example. If something similar occurred on Solana, Pythnet could become vulnerable.

Exploit Scenario
Solana silently patches a critical bug. Eve notices the bug from the patch, realizes that the bug has security implications that affect Pythnet, and exploits the bug on Pythnet.

Recommendations
Short term, take the following steps:

- Minimize the number of files that are changed in Pythnet relative to the Solana codebase. Prepare an accurate list of all such changes. Taking these steps will make it easier to rebase the changes onto updated versions of the Solana codebase.
- Keep abreast of changes to the Solana codebase, and incorporate those changes into the Pythnet codebase as often as possible. Doing so will help ensure Pythnet receives fixes to bugs that are silently patched.

Long term, investigate whether maintaining a codebase that shares no lineage with Solana would meet Pythnet's needs. Such a codebase would likely expose a significantly smaller attack surface.

References
- Geth v1.9.17 Post Mortem
- Disclosure: Fixing a critical bug in Optimism's Geth fork

2. Insufficient test coverage

Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-PYTH-2
Target: Pythnet validator, `pyth-agent/integration-tests`, `pyth-client/program/rust`, `pyth-crosschain/governance/xc_admin`	

Description

The Pythnet validator is a fork of the Solana validator. However, there are no tests of the changed code. Other parts of the code, including the oracle code enabled by the `pythnet` feature and all `xc_admin` packages except for `xc_admin_common`, are similarly untested.

Commit `83a5f1b` contained one relevant test that did not build (figure 2.1), but the test was subsequently removed.

```
#[test]
fn test_pyth_pda_addresses() {
    use solana_sdk::pyth::wormhole::WORMHOLE_PID;
    // TODO: make this a const
    let (accumulator_message_pda, _) =
        Pubkey::find_program_address(&[b"AccumulatorMessage"], &WORMHOLE_PID);

    //TODO: make this a const
    //seeds = ["emitter"], seeds::program = cpiProgramId
    let (emitter_pda_key, _) =
        Pubkey::find_program_address(&[b"emitter"], &sysvar::accumulator::id());
    //TODO: make this a const
    //seeds = ["Sequence", wormholeEmitter], seeds::program = wormholeProgram
    let (sequence_pda_key, _) = Pubkey::find_program_address(
        &[b"Sequence", &emitter_pda_key.to_bytes()],
        &WORMHOLE_PID,
    );

    println!(
        r"
    accumulator_message_pda: {accumulator_message_pda:?}
    emitter_pda_key: {emitter_pda_key:?}
    sequence_pda_key: {sequence_pda_key:?}
    "
    );
}
```

Figure 2.1: `pythnet/runtime/src/bank.rs#L18419–L18444`

The integration tests in the `pyth-agent` repository appear capable of exercising the changed code. However, the most complicated operation they perform is to create a message buffer (figure 2.2). They do not, for example, attempt to read from the accumulator accounts.

```
   ix = create_buffer({
        "allowed_program_auth": oracle_auth_pda,
        "base_account_key": address,
        "target_size": 1024
    }, {
        "admin": parsed_funding_keypair.public_key,
    },
        remaining_accounts = [
            AccountMeta(pubkey=message_buffer_pda, is_signer=False,
is_writable=True)
        ]
    )
    tx.add(ix)

await provider.send(tx, [parsed_funding_keypair])
```

Figure 2.2: pyth-agent/integration-tests/tests/test_integration.py#L385–L398

In addition, there appears to be a bug in the integration tests. Specifically, the line in figure 2.3 uses `os.path.exists`, which returns `false` for a symbolic link that points to a nonexistent file. Very likely, the line is meant to use `os.path.lexists`, which returns `true` for any existing symbolic link, regardless of what it points to.

```
if os.path.exists("keystore"):
    os.remove("keystore")
os.symlink(agent_keystore_path, "keystore")
```

Figure 2.3: pyth-agent/integration-tests/tests/test_integration.py#L342–L344

Exploit Scenario

A bug is found in the Pythnet validator code. The bug could have been discovered by more thorough unit or integration tests.

Recommendations

Short term, develop unit tests for each of the following:

- The Pythnet validator code that was changed relative to Solana
- The oracle code enabled by the `pythnet` feature
- All currently untested packages in `xc_admin`

Strive to test both happy (i.e., successful) and sad (i.e., failing) paths. Use a code coverage tool such as `cargo-llvm-cov` to verify the tests' thoroughness. Try to achieve similar

functional coverage through the integration tests. Taking these steps will help ensure the correctness of the changed code.

Long term, regularly review the tests' coverage to ensure that they are relevant and test all important conditions.

3. Wormhole dependency is not pinned

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-PYTH-3
Target: `Cargo.toml, geyser/Cargo.toml`	

Description

The `serde_wormhole` dependency is not pinned to a specific tag or revision (figures 3.1 and 3.2). A change pushed to the Wormhole repository could adversely affect the Pythnet validator.

The Wormhole repository appears to be rapidly changing. As of this writing, it has nearly 3,000 commits, the last of which was less than one hour ago. The fact that it receives so many changes increases the likelihood that one could affect the Pythnet validator. Moreover, it increases the likelihood that a maliciously introduced change could go unnoticed.

```
[patch.crates-io]
serde_wormhole = { git = "https://github.com/wormhole-foundation/wormhole" }
```

Figure 3.1: pythnet/Cargo.toml#L97–L98

```
[patch.crates-io]
serde_wormhole = { git = "https://github.com/wormhole-foundation/wormhole" }
```

Figure 3.2: pythnet/geyser/Cargo.toml#L23–L24

The need for the patch appears to be a nonstandard configuration in the Wormhole repository itself. Specifically, it patches the `serde_wormhole` dependency (figure 3.3), which then causes dependents (like the Pythnet validator) to have to patch as well. Arguably, packages within the Wormhole repository should refer to `serde_wormhole` by relative path, rather than relying on a patch (figure 3.4).

```
[patch.crates-io]
serde_wormhole = { path = "serde_wormhole" }
```

Figure 3.3: wormhole/sdk/rust/Cargo.toml#L11–L12

```
[dependencies]
anyhow = "1"
bstr = { version = "1", features = ["serde"] }
schemars = { version = "0.8.8", optional = true }
```

```
serde = { version = "1", default-features = false, features = ["alloc", "derive"] }
serde_wormhole = "0.1.0"
```

Figure 3.4: wormhole/sdk/rust/core/Cargo.toml#L13–L18

Exploit Scenario

Eve, an insider at Wormhole with merge privileges, realizes that changes made to `serde_wormhole` affect Pythnet validators. Eve intentionally introduces a flaw that goes unnoticed. Following the next Pythnet deployment, Eve exploits the flaw.

Recommendations

Short term, pin the `serde_wormhole` dependency to a specific tag or revision. Doing so will help ensure changes to `serde_wormhole` are noticed so they can be tested before they are deployed.

Long term, pursue a change to the Wormhole repository that eliminates the patch in figure 3.3. Doing so will allow Pythnet to eliminate its patches to the `serde_wormhole` dependency. This will result in a clearer, more maintainable, and more standardized codebase. (See also TOB-PYTH-14.)

4. Use of environment variables risks misconfiguration

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-PYTH-4
Target: `pythnet/runtime/src/bank.rs`	

Description

The Pythnet validator allows certain program IDs (e.g., the message buffer's program ID) to be overridden by environment variables (figure 4.1). Since this feature is intended to be used only on the testnet, it requires additional safeguards.

```
    let message_buffer_pid = Self::env_pubkey_or(
        "MESSAGE_BUFFER_PID",
        Pubkey::new_from_array(MESSAGE_BUFFER_PID),
    )?;
...
fn env_pubkey_or(
    var: &str,
    default: Pubkey,
) -> std::result::Result<Pubkey, AccumulatorUpdateError> {
    Ok(std::env::var(var)
        .as_deref()
        .map(Pubkey::from_str)
        .ok()
        .transpose()?
        .unwrap_or(default))
}
```

Figure 4.1: `pythnet/runtime/src/bank.rs#L2529–L2720`

Exploit Scenario

Alice, a Pythnet validator operator, verifies that her validator works correctly on the Pyth testnet. Alice tries to connect to the Pyth mainnet, but forgets to unset the relevant environment variables. Alice observes failures and wastes time and energy trying to determine the cause.

Recommendations

Short term, use the `ClusterType` field to determine the program IDs. For example, when the cluster type is `MainnetBeta`, use the program IDs that are currently hard coded in the Pythnet validator. Similarly hard code program IDs for the other cluster types (e.g., `Testnet`). Doing so will reduce the likelihood of a misconfigured validator being used on the Pyth network.

Long term, carefully consider which validator parameters should be configurable. Avoid allowing configuration of parameters that could affect consensus or could result in loss of funds. Having fewer parameters will reduce the likelihood that users misconfigure them.

5. Insecure transfer of admin role

Severity: **Medium**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-PYTH-5
Target: `pythnet/message_buffer/programs/message_buffer/src/lib.rs`	

Description

The message buffer program allows the existing whitelist admin to transfer the admin role to another pubkey (figure 5.1). Currently, the new admin never has to accept and sign this update instruction, which allows the admin role to be granted to a pubkey that is not controlled by the entity thought to control the address.

```
/// Sets the new admin for the whitelist
pub fn update_whitelist_admin(ctx: Context<UpdateWhitelist>, new_admin: Pubkey) ->
Result<()> {
    let whitelist = &mut ctx.accounts.whitelist;
    whitelist.validate_new_admin(new_admin)?;
    whitelist.admin = new_admin;
    Ok(())
}
```

Figure 5.1: pyth-crosschain/pythnet/message_buffer/programs/message_buffer/src/lib.rs#L43–L49

Exploit Scenario

The message buffer program's whitelist admin, Bob, intends to transfer ownership to Alice and makes a typo. Because Alice was not required to sign the instruction, the update succeeds and control over the message buffer's whitelist is lost.

Recommendations

Short term, require the new admin to be a signer on the `update_whitelist_admin` instruction to ensure they control the address the role is transferred to.

Long term, make sure that irreversible actions are not performed without safeguards to prevent loss of control over important roles.

6. Insufficient validations on admin key in initialize instruction

Severity: **Low**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-PYTH-6
Target: `pythnet/message_buffer/programs/message_buffer/src/lib.rs`	

Description

The message buffer program has an `initialize` instruction that sets the initial whitelist and an admin account that can manage the whitelist (figure 6.1). The admin account is required to not be the default public key (figure 6.2). However, the admin's signature is not required, which would allow pubkeys that are not controlled by the desired entity to be set as the admin upon initialization.

```
#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(mut)]
    pub payer:          Signer<'info>,
    #[account(
        init,
        payer = payer,
        seeds = [b"message".as_ref(), b"whitelist".as_ref()],
        bump,
        space = 8 + Whitelist::INIT_SPACE
    )]
    pub whitelist:      Account<'info, Whitelist>,
    pub system_program: Program<'info, System>,
}
```

Figure 6.1:
pythnet/message_buffer/programs/message_buffer/src/lib.rs#L152–L165

```
/// Initializes the whitelist and sets it's admin to the provided pubkey
/// Once initialized, the authority must sign all further changes to the whitelist.
pub fn initialize(ctx: Context<Initialize>, admin: Pubkey) -> Result<()> {
    require_keys_neq!(admin, Pubkey::default());
    let whitelist = &mut ctx.accounts.whitelist;
    whitelist.bump = *ctx.bumps.get("whitelist").unwrap();
    whitelist.admin = admin;
    Ok(())
}
```

Figure 6.2:
pythnet/message_buffer/programs/message_buffer/src/lib.rs#L19–L27

Exploit Scenario
Bob, a Pyth developer, deploys the message buffer program and intends to set Alice as the admin but makes a typo. Control over the whitelist is lost because neither Alice nor Bob control the pubkey registered as the program admin.

Recommendations
Short term, validate that the admin is a signer of the `initialize` instruction.

Long term, perform additional validations as outlined in TOB-PYTH-5.

7. Buffer accounts cannot be deleted after updating allowed programs

Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-PYTH-7
Target: `message_buffer/src/instructions/delete_buffer.rs`	

Description

The `delete_buffer` instruction checks that the auth PDA account associated with the buffer account is in the `whitelist.allowed_programs` state variable. If the admin updates the `allowed_programs` to a new list of PDA accounts, the delete operation for buffer accounts associated with the PDA accounts in the old list will fail.

Every message buffer account is associated with an `allowed_program_auth` account. All the operations involving a buffer account validate that the associated `allowed_program_auth` account is whitelisted. This includes the `delete_buffer` instruction (figure 7.1).

```
pub fn delete_buffer<'info>(
    ctx: Context<'_, '_, '_, 'info, DeleteBuffer<'info>>,
    allowed_program_auth: Pubkey,
    base_account_key: Pubkey,
    bump: u8,
) -> Result<()> {
    [...]
    ctx.accounts
        .whitelist
        .is_allowed_program_auth(&allowed_program_auth)?;
```

Figure 7.1: pyth-crosschain/pythnet/message_buffer/programs/message_buffer/src/instructions/delete_buffer.rs#L21–L23

The admin can update `whitelist.allowed_programs` using the `set_allowed_programs` instruction (figure 7.2).

```rust
pub fn set_allowed_programs(
    ctx: Context<UpdateWhitelist>,
    allowed_programs: Vec<Pubkey>,
) -> Result<()> {
    let whitelist = &mut ctx.accounts.whitelist;
    whitelist.validate_programs(&allowed_programs)?;
    whitelist.allowed_programs = allowed_programs;
    Ok(())
}
```

Figure 7.2: pyth-crosschain/pythnet/message_buffer/programs/message_buffer/src/lib.rs#L33–L41

Because the delete operation requires the auth account to be in `whitelist.allowed_programs`, the `delete_buffer` instruction will fail for all buffer accounts whose auth accounts are not part of the new list after the update.

The admin would have to update the `allowed_programs` state variable to restore the old list, perform the delete operations, and update `allowed_programs` again to the new list.

Exploit Scenario

The message buffer program stores the messages from the oracle program. The `whitelist.allowed_programs` contains the oracle program's auth PDA account. The oracle program is deployed at a new address and creates a new PDA account for the message buffer program.

Alice, the admin of the message buffer program, updates `allowed_programs` to replace the old PDA account with the new one. Alice tries to delete the old buffer accounts, but the operation fails. The validator continues including the data from the old buffer accounts in the Merkle tree, which reduces its efficiency.

Recommendations

Short term, remove the check validating that the auth PDA account associated with the buffer account is present in `whitelist.allowed_programs`.

Long term, implement comprehensive unit tests to test the operations after multiple state transitions.

8. Validator could panic while reading buffer account containing 255 messages

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-8
Target: `pythnet/runtime/src/bank.rs`	

Description

The validator iterates over the `end_offsets` field without checking its length. As a result, the validator might read the message data as one of the `end_offsets` and use it as a pointer to read data from the buffer account. The validator panics when the pointer is greater than the account size.

The `end_offsets` are used as pointers into the message part of the buffer account. `end_offsets` is a fixed array of size 255 (figure 8.1) and contains nonzero offsets in the beginning of the array, each corresponding to a message. The rest of the elements are set to a default value of zero.

```
pub struct MessageBuffer {
    [...]
    /// endpoints of every message.
    /// ex: [10, 14]
    /// => msg1 = account_info.data[(header_len + 0)..(header_len + 10)]
    /// => msg2 = account_info.data[(header_len + 10)..(header_len + 14)]
    pub end_offsets: [u16; 255], // 510

                          /* messages */
```

Figure 8.1: message_buffer/src/state/message_buffer.rs#L36–L55

The validator iterates over `end_offsets` using a `Cursor` and stops reading only when an element is zero (figure 8.2).

```
let data = account.data();
[...]
while let Some(end) = cursor.read_u16::<LittleEndian>().ok() {
    if end == 0 {
        break;
    }
```

```
    let end_offset = header_len + end;
    let accumulator_input_data = &data[header_begin as usize..end_offset as usize];
    inputs.push(accumulator_input_data);
    header_begin = end_offset;
}
```

Figure 8.2: pythnet/runtime/src/bank.rs#L2554–L2573

When the number of messages written to the buffer account is exactly 255, all elements of `end_offsets` will be nonzero. As a result, the `Cursor` will also read from the `messages` field that is stored after `end_offsets` in memory. The offset calculated from the message data could point to an invalid location and the validator would panic while accessing the data with that invalid offset.

Exploit Scenario

An allowed program writes 255 messages to a buffer account. The size of the buffer account is 10,000 bytes. The first two bytes of the first message are equal to 11,000 when read as `u16`. The validator attempts to read the buffer account using the offset 11,000 while constructing the Merkle tree and panics with an out-of-range error.

Recommendations

Short term, update the loop implementation to iterate over the length of the `end_offsets` and break when an offset is zero.

Long term, improve unit and integration tests to cover the edge cases involving the value boundaries. Avoid manually decoding values and use appropriate libraries to decode the values to the target types.

9. Resizing a buffer account could lead the validator to panic

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-9
Target: `message_buffer/src/instructions/resize_buffer.rs`, `pythnet/runtime/src/bank.rs`	

Description

The `resize_buffer` instruction does not consider the `messages` field while calculating the minimum target size required for a buffer account. As a result, the account could be resized to a smaller size, leading the validator to panic while reading the messages.

The `resize_buffer` instruction reallocates the account to the given `target_size` argument. The instruction checks only that the `target_size` covers the buffer header, not that it also covers the messages written to it (figure 9.1).

```
pub fn resize_buffer<'info>(
    [...]
    target_size: u32,
) -> Result<()> {
   [...]
    require_gte!(
        target_size,
        MessageBuffer::HEADER_LEN as u32,
        MessageBufferError::MessageBufferTooSmall
    );

    if target_size_delta >= 0 {
        [...]
        message_buffer_account_info
            .realloc(target_size, false)
            .map_err(|_| MessageBufferError::ReallocFailed)?;
    } else {
        message_buffer_account_info.realloc(target_size, false)?;
    }
```

Figure 9.1: message_buffer/src/instructions/resize_buffer.rs#L67–L88

The validator reads the messages using the `end_offsets` field, which contains offsets into the buffer account data (figure 9.2).

```
let data = account.data();
[...]
```

```
while let Some(end) = cursor.read_u16::<LittleEndian>().ok() {
    if end == 0 {
        break;
    }

    let end_offset = header_len + end;
    let accumulator_input_data = &data[header_begin as usize..end_offset as usize];
    inputs.push(accumulator_input_data);
    header_begin = end_offset;
}
```

Figure 9.2: pythnet/runtime/src/bank.rs#L2554–L2573

When the buffer account is reallocated to a `target_size` that is less than the message location calculated from an offset, the validator panics with an out-of-range error while reading the buffer account.

While we consider this issue to be of high severity, it requires the admin to perform certain actions, so we also consider it to be of high difficulty.

Exploit Scenario

A buffer account has a size of 9522 bytes (header + messages): 522 + 9000. An allowed program writes a message, and one of the `end_offsets` is 1000. Eve, an attacker, steals the admin key and resizes the account to 600 bytes (less than `header_len` + offset).

The validator attempts to read the message from the account using `end_offsets` but panics with an out-of-range error when reading the data up to the range 1522 (`header_len` + offset).

Recommendations

Short term, check that `target_size` is greater than or equal to `HEADER_LEN` + maximum `end_offset` in the `resize_buffer` function.

Long term, develop thorough unit and integration tests for the Pythnet validator. Doing so could help expose bugs like this one.

10. Program may not write all messages even when account has free space

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-PYTH-10
Target: `message_buffer/src/state/message_buffer.rs`	

Description

The `put_all` instruction checks that the account has sufficient space before writing each message. Because the account size is converted using lossy conversion before comparison, the compared size may be less than the actual size, leading the program to not write all messages.

The account size is converted from `usize` to `u16` for comparison (figure 10.1). If the destination length is greater than `u16::MAX`, the length will be truncated. As a result, the amount of data that could be written to the buffer account will be limited by the truncated size, which will be much less than the actual size.

```
pub fn put_all_in_buffer(
    &mut self,
    destination: &mut [u8],
    values: &Vec<Vec<u8>>,
) -> (usize, u16) {

    for (i, v) in values.iter().enumerate() {
        [...]
        let end = offset.checked_add(len.unwrap());
        [...]
        if end > destination.len() as u16 {
            return (i, start);
        }
```

Figure 10.1: message_buffer/src/state/message_buffer.rs#L111–L113

Exploit Scenario

A buffer account is reallocated to size `HEADER_LEN + u16::MAX + 1`. The destination length will be `u16::MAX + 1`. After the program converts the destination length to `u16`, the length will be zero. An allowed program attempts to write messages, but the `put_all` instruction returns without writing any messages.

Recommendations

Short term, convert the `end` variable from `u16` to `usize` for comparison. Additionally, consider restricting the maximum size of the buffer account to `u16::MAX` in the `realloc_buffer` instruction.

Long term, consider enabling Clippy lints such as `cast_possible_truncation` at the `deny` level. Doing so will help prevent similar bugs.

11. Insufficient event generation

Severity: **Low**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-PYTH-11
Target: `message_buffer`, `remote_executor`	

Description

Events generated during program execution aid in monitoring, baselining behavior, and detecting suspicious activity. Without events, users and blockchain monitoring systems cannot easily detect behavior that falls outside the baseline conditions. This may prevent malfunctioning programs or malicious attacks from being discovered.

Multiple critical operations do not emit events. As a result, it will be difficult to review the correct behavior of the programs once they have been deployed. The following operations should trigger events:

- Updating the admin and allowed programs
- Creating, resizing, and deleting buffer accounts
- Writing messages to the buffer account
- Posting Verified Action Approvals (VAAs) from the remote executor
- Performing any of the following operations from within the oracle program:
    - Adding, updating, or deleting a product, publisher, or price
    - Changing the oracle program's settings
    - Updating the oracle program's authorities

Additionally, the oracle program's `upd_price_no_fail_on_error` function (figure 11.1) performs an operation and explicitly discards any error. By doing so, the caller cannot tell whether an error actually occurred, or whether an error was the expected one. At a minimum, the function should log errors.

```
pub fn upd_price_no_fail_on_error(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
```

```
) -> ProgramResult {
    match upd_price(program_id, accounts, instruction_data) {
        Err(_) => Ok(()),
        Ok(value) => Ok(value),
    }
}
```

Figure 11.1: pyth-client/program/rust/src/processor/upd_price.rs#L67–L76

The oracle program performs essentially no logging. There is a `log` module that can be enabled by a feature (figure 11.2). However, enabling the feature results in build failures.

```
#[cfg(feature = "debug")]
mod log;
```

Figure 11.2: pyth-client/program/rust/src/lib.rs#L16–L17

Exploit Scenario

Alice runs a Pythnet validator and calls the oracle program's `upd_price_no_fail_on_error` function, expecting a particular error. The actual error is different from the one she expects, but Alice is unaware.

Recommendations

Short term, add events for all operations that may contribute to a higher level of monitoring and alerting. Consider using the Anchor `emit` construct to log structured information.

Long term, consider using a blockchain monitoring system to track any suspicious behavior in the programs. A monitoring mechanism for critical events would quickly detect any compromised system components. Additionally, develop an incident response plan if Pythnet does not already have one. Doing so will help ensure that issues are dealt with promptly and without confusion.

12. Remote executor lacks overflow protection

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-12
Target: `pyth-crosschain/governance/remote_executor/programs/remote-executor/Cargo.toml`	

Description

Release builds of the remote executor lack overflow protection because of a workspace misconfiguration. Future changes to the remote executor could lead to arithmetic errors.

Building the remote executor causes Cargo to produce the warning in figure 12.1. The declaration that Cargo is reacting to appears in figure 12.2. Specifically, when a workspace includes multiple packages, Cargo considers `profile` declarations that appear in the workspace's root `Cargo.toml` file only. Hence, aside from emitting the warning, Cargo ignores the `profile` declaration in figure 12.2.

```
warning: profiles for the non root package will be ignored, specify profiles at the
workspace root:
package:   .../governance/remote_executor/programs/remote-executor/Cargo.toml
workspace: .../governance/remote_executor/Cargo.toml
```

Figure 12.1: Warning produced when the remote executor is built

```
[profile.release]
overflow-checks = true
```

Figure 12.2: pyth-crosschain/governance/remote_executor/programs/remote-executor/Cargo.toml#L19–L20

This finding is set to informational because use of arithmetic does not feature prominently in the remote executor.

Exploit Scenario

Alice, a Pyth developer, makes a change to the remote executor that introduces an arithmetic error. The bug goes unnoticed because the remote executor lacks overflow protections, and as a result, funds are lost.

Recommendations

Short term, move the declaration in figure 12.2 to the `Cargo.toml` file in the workspace root. Doing so will enable overflow checks for release builds of the remote executor.

Long term, consider enabling Clippy lints such as `arithmetic_side_effects` at the deny level. Doing so will help identify such problems.

13. Outdated dependencies

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-PYTH-13
Target: `pyth-crosschain/governance/remote_executor/cli/Cargo.toml`, `pyth-crosschain/pythnet/pythnet_sdk/Cargo.toml`, `pyth-client/program/rust/Cargo.toml`	

Description

Updated versions of many of the dependencies for the remote executor CLI, the Pythnet software development kit (SDK), and the oracle program are available. Because silent bug fixes are common, all dependencies should be periodically reviewed and updated wherever possible.

Some of these outdated dependencies have updated versions that are considered incompatible by Cargo; because of this, simply running `cargo update` will not update them in the project's `Cargo.lock` file. Dependencies for which incompatible upgrades are available appear in tables 13.1, 13.2, and 13.3.

Dependency	Version currently in use	Latest version available
`anchor-client`	0.25.0	0.27.0
`base64`	0.13.0	0.21.0
`clap`	3.2.22	4.2.7
`shellexpand`	2.1.2	3.1.0

Table 13.1: `remote_executor` dependencies for which incompatible upgrades are available

Dependency	Version currently in use	Latest version available
`borsh`	0.9.1	0.10.3
`rand`	0.7.0	0.8.5

Table 13.2: `pythnet_sdk` dependencies for which incompatible upgrades are available

Dependency	Version currently in use	Latest version available
`bindgen`	0.60.1	0.65.1
`hex`	0.3.1	0.4.3

Table 13.3: `pyth-oracle` dependencies for which incompatible upgrades are available

Exploit Scenario

Eve learns of a vulnerability in an outdated version of a remote executor CLI dependency. Knowing that the CLI tool still relies on this outdated version, Eve exploits the vulnerability.

Recommendations

Short term, update the dependencies to their latest versions wherever possible. Once the dependencies are updated, verify that all unit tests pass. Document any reasons for not updating a dependency. Using outdated dependencies could mean critical bug fixes are missed.

Long term, regularly run `cargo upgrade --incompatible`. This will help ensure that the project stays up to date with its dependencies.

14. Account size calculations are undocumented

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-PYTH-14
Target: `pyth-crosschain/governance/remote_executor/cli/src/main.rs`, `pyth-client/program/rust/src/accounts/price.rs`	

Description

The remote executor CLI tool manually calculates the sizes of two Wormhole-related accounts. However, the steps involved in the calculation are undocumented, which hinders maintainability.

The CLI tool creates two accounts for interacting with Wormhole: a signature set account and a VAA account. The tool calculates these accounts' sizes (yellow in figure 14.1) so that it can fund them sufficiently to avoid paying rent (red in figure 14.1). As the figure shows, the steps in the calculation are undocumented, so if a developer needs to fix or modify the calculation, they will have little information to work with.

```
let signature_set_size = 4 + 19 + 32 + 4;
let posted_vaa_size = 3 + 1 + 1 + 4 + 32 + 4 + 4 + 8 + 2 + 32 + 4 +
vaa.payload.len();
let posted_vaa_key = PostedVAA::key(&wormhole, vaa.digest().unwrap().hash);

process_transaction(
    &rpc_client,
    vec![
        transfer(
            &payer.pubkey(),
            &signature_set_keypair.pubkey(),
            rpc_client.get_minimum_balance_for_rent_exemption(signature_set_size)?,
        ),
        transfer(
            &payer.pubkey(),
            &posted_vaa_key,
            rpc_client.get_minimum_balance_for_rent_exemption(posted_vaa_size)?,
        ),
    ],
    &vec![&payer],
)?;
```

Figure 14.1: pyth-crosschain/governance/remote_executor/cli/src/main.rs#L95–L114

Similar problems exist in the oracle program (figures 14.2 and 14.3).

```
pub const MESSAGE_SIZE: usize = 1 + 32 + 8 + 8 + 4 + 8 + 8 + 8 + 8;
```

Figure 14.2: pyth-client/program/rust/src/accounts/price.rs#L246

```
pub const MESSAGE_SIZE: usize = 1 + 32 + 16 + 16 + 8 + 4 + 8 + 8 + 8;
```

Figure 14.3: pyth-client/program/rust/src/accounts/price.rs#L330

The Mesh source code (figure 14.4) provides a good example of a well-documented size calculation.

```
pub const SIZE_WITHOUT_MEMBERS: usize = 8 + // Anchor disriminator
2 +         // threshold value
2 +         // authority index
4 +         // transaction index
4 +         // processed internal transaction index
1 +         // PDA bump
32 +        // creator
1 +         // allow external execute
4 +         // for vec length
32;       // external authority
```

Figure 14.4: mesh/programs/mesh/src/state/mesh.rs#L20–L29

Exploit Scenario

Alice, a Pyth developer, is tasked with modifying the remote executor's account size calculations. Alice misinterprets the purpose of one of the steps and inadvertently introduces a bug.

Recommendations

Short term, document the purpose of each of the individual addends (like in figure 14.4) in the remote executor's account size calculations. Doing so will produce more maintainable code and reduce the likelihood that bugs are introduced in the future.

Long term, pursue a change to the Wormhole API to perform these calculations on behalf of clients. Arguably, Wormhole should be responsible for performing these calculations because Wormhole consumes these accounts. (See also TOB-PYTH-3.)

15. Remote executor CLI error handling could lead to loss of rent

Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-PYTH-15
Target: `pyth-crosschain/governance/remote_executor/cli/src/main.rs`	

Description

Early in its execution, the remote executor CLI transfers funds to two created accounts. If a subsequent operation fails, a user will need to manually recover those funds, which may require the assistance of Wormhole.

The relevant code appears in figure 15.1. The CLI tool creates two accounts, a signature set account and a Verified Action Approval (VAA) account, and transfers funds to them (yellow in figure 15.1). The account addresses are not communicated to the user. There are several opportunities for the tool to fail thereafter, in either instruction creation or execution (red in figure 15.1). If a failure occurs, the user will need to manually determine the account addresses (e.g., by reviewing transaction logs) so that they can recover the funds transferred to those accounts.

```
process_transaction(
    &rpc_client,
    vec![
        transfer(
            &payer.pubkey(),
            &signature_set_keypair.pubkey(),
            rpc_client.get_minimum_balance_for_rent_exemption(signature_set_size)?,
        ),
        transfer(
            &payer.pubkey(),
            &posted_vaa_key,
            rpc_client.get_minimum_balance_for_rent_exemption(posted_vaa_size)?,
        ),
    ],
    &vec![&payer],
)?;

// RENT HACK ENDS HERE

// First verify VAA
let verify_txs = verify_signatures_txs(...)?;

for tx in verify_txs {
    process_transaction(&rpc_client, tx, &vec![&payer, &signature_set_keypair])?;
```

```
}
...
process_transaction(
    &rpc_client,
    vec![post_vaa(...)?],
    &vec![&payer],
)?;

// Now execute
process_transaction(
    &rpc_client,
    vec![get_execute_instruction(...)?],
    &vec![&payer],
)?;
```

Figure 15.1:
pyth-crosschain/governance/remote_executor/cli/src/main.rs#L99–L165

Moreover, based on our current understanding of the `verify_signatures_txs` function, one of the instructions it generates transfers ownership of the signature set account to Wormhole. Hence, if the tool fails following the execution of `verify_signatures_txs`, the user may need assistance from Wormhole to recover their funds.

Exploit Scenario

Alice routinely uses the remote executor CLI to post VAAs. Occasionally, the tool fails (e.g., because of network problems). In such cases, Alice simply tries again until the tool succeeds. After a long period of such use, Alice wonders why there is a deficit in her wallet's balance.

Recommendations

Short term, at a minimum, provide sufficient logging information so that if a failure occurs, the information needed to recover lost funds is readily available to the user. Doing so will reduce the likelihood that such funds are lost indefinitely (e.g., because the user is unaware of the loss).

Long term, investigate the following possibilities:

- Verify that transactions are successful when simulated before performing them with real funds.
- Provide a way for the CLI tool to resume a failed operation (i.e., pick up where it left off).

Taking either of these steps will help prevent conditions that warrant fund recovery.

16. Inefficient prove function could lead to DoS

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-PYTH-16
Target: `pyth-crosschain/pythnet/pythnet_sdk/src/accumulators/merkle.rs`	

Description
The runtime of the Merkle tree `prove` function's current implementation is O(n), where n is the number of leaves in the tree. One would expect the runtime to be O(log n). If a client's software could be caused to call the function repeatedly, it could result in a denial-of-service (DoS) condition.

The relevant code appears in figure 16.1. Given an `item`, the `prove` function constructs a Merkle path for the `item` by first hashing it and then finding the position of the hash among the leaves. The `prove` function accomplishes the latter by performing a linear search (yellow in figure 16.1). Additionally, `prove` unnecessarily compares the `item`'s hash to that of all the internal nodes before considering the leaves.

```
fn prove(&'a self, item: &[u8]) -> Option<Self::Proof> {
    let item = hash_leaf::<H>(item);
    let index = self.nodes.iter().position(|i| i == &item)?;
    Some(self.find_path(index))
}
```

Figure 16.1: pyth-crosschain/pythnet/pythnet_sdk/src/accumulators/merkle.rs#L107–L111

The `pyth_sdk` Merkle tree implementation places no requirements on the order of the leaves. Hence, sorting the leaves by their hash would have no impact on other parts of the Merkle tree implementation. Moreover, ordering the leaves in this way would allow the `prove` function to use a binary search instead of a linear one.

Exploit Scenario
Alice writes software that relies on the `pyth_sdk` Merkle tree implementation. Alice's software can be made to call `prove` repeatedly. Eve uses this fact to perform DoS attacks against users of Alice's software.

Recommendations
Short term, take the following steps:

- Order the Merkle tree leaves in `MerkleAccumulator::nodes` by their hash.

- When searching for a leaf with a particular hash, do not compare the hash to that of the internal nodes.

Taking these steps will allow the leaves to be searched using binary search, which will result in a more efficient implementation.

Long term, incorporate benchmarking into Pythnet's testing process. This problem could be found by running the `prove` function with inputs of varying sizes.

17. Price feed multisig can be upgraded without 3/6 proposal

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Access Controls	Finding ID: TOB-PYTH-17
Target: price feed multisig (`92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8`)	

Description
A stated requirement of the Pyth deployment is that "the oracle program can only be configured with a 3/6 proposal in the price feed multisig `92hQkq8kBgCUcF9yWN8URZB9RTmA4mZpDGtbiAWA74Z8`." However, this requirement does not hold because the price feed multisig can be upgraded by parties other than price feed multisig keyholders.

The price feed multisig uses the Mesh program (`SMPLVC8MxZ5Bf5EfF7PaMiTCxoBAcmkbM2vkrvMK8ho`). This program's upgrade authority is `6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT`, which is a PDA associated with the upgrade multisig. There is overlap among the upgrade and price feed multisig keys (see table 2 in Deployment Review). However, there are four keys registered to the upgrade multisig that are not registered to the price feed multisig:

- `opsLibxVY7Vz5eYMmSfX8cLFCFVYTtH6fr6MiifMpA7`
- `3S2N8k3tKUMtp9m1fp4GSc5k6AtZfkuujpqAfZV6K846`
- `7fbLBsSd5oZGtYgw8eJVvAqrMbQUTEdDTtgNnpqqTw7y`
- `ADp5q8hLHH9FUeNMSzvMJWKnZ9cWdFQL9YkJspB9dHk4`

Moreover, the upgrade multisig's threshold is 4. Thus, the holders of the above keys could change the Mesh program's code without the consent of any price feed multisig keyholder.

Exploit Scenario
The holders of the above keys conspire to manipulate Pyth prices by performing a malicious upgrade of the Mesh program. The price feed multisig keyholders learn of the conspiracy but realize they have no means of stopping it.

Recommendations
Short term, take one of the following steps:

- If the risk associated with upgrading the Mesh program is acceptable, prominently document this fact. Doing so will help alert users to this possibility.

- Deploy a fresh copy of the Mesh program with an upgrade authority that is not `6oXTdojyfDS8m5VtTaYB9xRCxpKGSvKJFndLUPV3V3wT`, and use that fresh copy for the price feed multisig. Doing so will prevent the upgrade multisig keyholders from overriding the will of the price feed multisig keyholders.

Long term, develop more detailed deployment documentation. For example, such documentation should include an expanded version of the diagram that appears near the top of the Pyth operations key management document. Ideally, the expanded diagram would mention all parties that interact with the Pyth network, including the two multisigs' keyholders. Taking these steps could help expose situations where privileges afforded to one group could be used to adversely affect another.

18. Arithmetic overflow in PriceCumulative::update

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-18
Target: `pyth-client/program/rust/src/accounts/price.rs`	

Description

The `PriceCumulative::update` method uses unchecked arithmetic that can overflow, resulting in incorrect price calculations.

The relevant code appears in figure 18.1. The method adds the result of three calculations to three fields. The additions to those fields are unchecked.

```
pub fn update(&mut self, price: i64, conf: u64, slot_gap: u64) {
    self.price += i128::from(price) * i128::from(slot_gap);
    self.conf += u128::from(conf) * u128::from(slot_gap);
    // This is expected to saturate at 0 most of the time (while the feed is up).
    self.num_down_slots += slot_gap.saturating_sub(PC_MAX_SEND_LATENCY.into());
}
```

Figure 18.1: pyth-client/program/rust/src/accounts/price.rs#L164–L169

Figure 18.2 shows part of a test involving `PriceCumulative::update`. The problem can be demonstrated by duplicating the highlighted line.

```
let mut price_cumulative_overflow = PriceCumulative {
    price:          0,
    conf:           0,
    num_down_slots: 0,
    unused:         0,
};
price_cumulative_overflow.update(i64::MIN, u64::MAX, u64::MAX);
```

Figure 18.2: pyth-client/program/rust/src/tests/test_twap.rs#L149–L155

The `PriceCumulative::update` method is called from the `PriceAccountV2::update_price_cumulative` method in figure 18.3.

```
impl PriceAccountV2 {
    /// This function gets triggered when there's a succesful aggregation and
updates the cumulative sums
    pub fn update_price_cumulative(&mut self) -> Result<(), OracleError> {
        if self.agg_.status_ == PC_STATUS_TRADING {
```

```
            self.price_cumulative.update(
                self.agg_.price_,
                self.agg_.conf_,
                self.agg_.pub_slot_.saturating_sub(self.prev_slot_),
            ); // pub_slot should always be >= prev_slot, but we protect ourselves
against underflow just in case
            Ok(())
        } else {
            Err(OracleError::NeedsSuccesfulAggregation)
        }
    }
}
```

Figure 18.3: pyth-client/program/rust/src/accounts/price.rs#L124–L138

Exploit Scenario

A volatile asset experiences widely varying price and confidence reports. Eventually, one of the calls to `PriceCumulative::update` results in arithmetic overflow. Incorrect prices are reported to various exchanges, and users suffer financial loss.

Recommendations

Short term, use checked arithmetic in the `PriceCumulative::update` method and return an error when any operation overflows. Doing so will help ensure correct price calculations.

Long term, consider incorporating fuzzing into the CI process, as this bug was found through fuzzing. Fuzzing regularly could help expose similar bugs.

19. Potential overflow in get_status_for_update

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-19
Target: `program/rust/src/accounts.rs`	

Description

The negation of the `threshold_conf` variable can overflow if its value is `i64::MIN` (figure 19.1); this error will not be caught in release builds of the program because the arithmetic overflow will wrap and not panic. Furthermore, this behavior would violate the invariant that `threshold_conf` is a positive value because `i64::MIN` will be returned.

```
if threshold_conf < 0 {
    threshold_conf = -threshold_conf;
}
```

Figure 19.1: program/rust/src/utils.rs#L208–L210

Currently, the value of `threshold_conf` cannot reach `i64::MIN` in practice because it is divided by `MAX_CI_DIVISOR` (figure 19.2) and thus `threshold_conf` will be greater than `i64::MIN`. However, this potential overflow should be handled by using the `i64::checked_abs` function.

```
let mut threshold_conf = price / MAX_CI_DIVISOR;
```

Figure 19.2: program/rust/src/utils.rs#L206

Exploit Scenario

Bob, a Pyth developer, modifies the code such that `threshold_conf` can be `i64::MIN`, and the overflow wraps, causing unexpected behavior.

Recommendations

Short term, use `checked_abs` to get the absolute value of `threshold_conf`.

Long term, review uses of arithmetic that do not use checked arithmetic, and perform fuzzing with overflow protections to identify overflows.

20. Price and publisher accounts are not properly closed

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PYTH-20
Target: `program/rust/src/accounts.rs`	

Description

The instructions `del_price` and `del_publisher` are intended to close price and publisher programs, but they only transfer the programs' lamports and do not clear the data they hold (figure 20.1). This may result in instructions reading data that was marked as deleted but not cleared from state, or it may allow previously deleted accounts to be reinitialized.

```
let lamports = price_account.lamports();
**price_account.lamports.borrow_mut() = 0;
**funding_account.lamports.borrow_mut() += lamports;
```

Figure 20.1: program/rust/src/processor/del_price.rs#L84-L86

Exploit Scenario

Alice discovers a bug in her code that causes incorrect price data to be published to Pythnet. To prevent the bad data from being used, Alice decides to delete the price data account. Alice sends multiple instructions in a transaction, one of which is `del_price`. The price account is deleted but still used as though it were not by a subsequent instruction in the same transaction.

Recommendations

Short term, overwrite accounts with a unique discriminator upon deletion.

Long term, consider using Anchor, which handles this issue, with the `#[account(close = <target_account>)]` macro.

21. Price account is not validated before calling c_upd_aggregate

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-PYTH-21
Target: `pyth-client/program/rust/src/processor/upd_price.rs`	

Description

Fuzzing the `upd_price` instruction produces panics in the C function `c_upd_aggregate`. Since C is an unsafe language, care must be taken to validate the arguments of functions written in C.

The relevant code appears in figure 21.1. The `c_upd_aggregate` function is called before `price_account` has been validated. Calling `c_upd_aggregate` with a malformed price account can result in a panic.

```
pub fn upd_price(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    ...
    if clock.slot > latest_aggregate_price.pub_slot_ {
        unsafe {
            aggregate_updated = c_upd_aggregate(
                price_account.try_borrow_mut_data()?.as_mut_ptr(),
                clock.slot,
                clock.unix_timestamp,
            );
        }
    }
```

Figure 21.1: pyth-client/program/rust/src/processor/upd_price.rs#L98–L171

Adding a line like the following, just prior to the call to `c_upd_aggregate`, eliminates all panics that we observed:

```
let _price_data = load_checked::<PriceAccountV2>(price_account,
    cmd_args.header.version)?;
```

Since the C code was out of scope, we did not further investigate the cause of the panics.

Exploit Scenario

Alice, a Pyth developer, introduces a bug into `c_upd_aggregate` that allows arbitrary code execution. Eve, an attacker, uses the fact that `upd_price` does not validate the price account before calling `c_upd_aggregate` to exploit the bug.

Recommendations

Short term, to eliminate panics caused by malformed price accounts, ensure that the price account is a valid `PriceAccountV2` account before calling the `c_upd_aggregate` function.

Long term, consider incorporating fuzzing into the CI process, as this bug was found through fuzzing. Fuzzing regularly could help expose similar bugs.

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 table describes 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
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
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
Front-Running Resistance	The system's resistance to front-running attacks
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

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. Non-Security–Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Avoid copying literals.** Using a constant variable improves readability and helps prevent developers from introducing bugs while copying and updating. We identified the following literals:
  - The seeds of the whitelist account, which are copied in the following structs:
    - `WhitelistVerifier`
    - `UpdateWhitelist`
    - `Initialize`
    - `CreateBuffer`
    - `ResizeBuffer`
    - `DeleteBuffer`

```
seeds = [b"message".as_ref(), b"whitelist".as_ref()],
```

Figure C.1: `WhitelistVerifier.whitelist` in `state/whitelist.rs`

  - The length of the `MessageBuffer.end_offsets`

```
pub end_offsets: [u16; 255], // 510
```

Figure C.2: `MessageBuffer` struct in `state/message_buffer.rs`

```
end_offsets: [0u16; u8::MAX as usize]
```

Figure C.3: `MessageBuffer::new` method in `state/message_buffer.rs`

```
self.end_offsets = [0u16; u8::MAX as usize];
```

Figure C.4: The `MessageBuffer.refresh_header` method in `state/message_buffer.rs`

- The seed of the message buffer account

```
&[
    [...]
    b"message".as_ref(),
```

Figure C.5: The `accumulator_input_seeds` macro in `macros.rs`

Update the macro implementation to use the `instructions::MESSAGE` constant, which is defined to store the seed of the buffer account.

- **Correct the following comments:**

```
/// Note: manually checking because using anchor's `AccountLoader.load()`
/// will panic since the `AccountInfo.data_len()` will not match the
/// size of the `MessageBuffer` since the `MessageBuffer` struct does not
/// include the messages.
pub fn check_discriminator(message_buffer_account_info: &AccountInfo) ->
Result<()> {
```

Figure C.6: Comment on `MessageBuffer.check_discriminator` method in `state/message_buffer.rs`

The `AccountLoader.load()` method will not panic because messages are missing from the `MessageBuffer` struct.

```
pub struct ResizeBuffer<'info> {
    [..]
    // Also pays for account creation
    #[account(mut)]
    pub admin: Signer<'info>,
```

Figure C.7: Comment on `ResizeBuffer.admin` in `instructions/resize_buffer.rs`

The admin does not pay for account creation but pays the additional rent required when increasing the account size.

```
// allow for delta == 0 in case Rent requirements have changed
// and additional lamports need to be transferred.
// the realloc step will be a no-op in this case.
if target_size_delta >= 0 {
```

Figure C.8: Comment in `resize_buffer` function in `instructions/resize_buffer.rs`

The `realloc` step will not be a no-op because the `target_size_delta` variable will be zero when the target size is less than the old size. In this case, `realloc` will reduce the account size.

```
/// Sets the programs that are allowed to invoke this program through CPI
///
/// * `allowed_programs` - Entire list of programs that are allowed to
///                        invoke this program through CPI
pub fn set_allowed_programs(
    ctx: Context<UpdateWhitelist>,
    allowed_programs: Vec<Pubkey>,
) -> Result<()> {
```

Figure C.9: Comment on `set_allowed_programs` instruction in `lib.rs`

`allowed_programs` are not programs but PDA accounts of the programs that are allowed to invoke the message buffer program. There are other instances in the codebase where the comments about `allowed_programs` are incorrect.

- **Follow the principle of least privilege for cross-program invocations:**

```
system_program::transfer(
    CpiContext::new_with_signer(
        system_program.to_account_info(),
        Transfer {
            from: payer.to_account_info(),
            to:   new_account_info.to_account_info(),
        },
        seeds,
    ),
    target_rent - new_account_info.lamports(),
)?;
```

Figure C.10: A snippet of the `CreateBuffer::create_account` method in `instructions/create_buffer.rs`

The `new_account_info` account signs the above Cross Program Invocation (CPI) call but is not required to. Following the principle of least privilege helps reduce the impact of an external bug.

- **Use `AccountLoader` construct to access and validate message buffer accounts.** Using Anchor constructs simplifies the code, improves readability, and helps validate the input accounts. The following instructions implement the additional logic to validate message buffer accounts:
    - `CreateBuffer`
    - `ResizeBuffer`
    - `PutAll`

- **Remove the unneeded `instruction` attribute from the `Accounts` structs.** Removing the unnecessary attribute improves readability. The attribute appears on the following structs:
  - `ResizeBuffer`
  - `PutAll`

```
#[derive(Accounts)]
#[instruction(
    allowed_program_auth: Pubkey, base_account_key: Pubkey,
    buffer_bump: u8, target_size: u32
)]
pub struct ResizeBuffer<'info> {
```

Figure C.11: `ResizeBuffer` struct in `instructions/resize_buffer.rs`

- **Remove the following unreachable code:**

```
if target_size_delta >= 0 {
    [...]
} else {
    // Not transferring excess lamports back to admin.
    // Account will retain more lamports than necessary.
    message_buffer_account_info.realloc(target_size, false)?;
}
```

Figure C.12: Unreachable code in `instructions/resize_buffer.rs`

`target_size_delta` is of type `usize` and will always be greater than or equal to zero. The `else` block is unreachable.

- **Remove unused accounts from the `Accounts` structs:**

```
#[derive(Accounts)]
pub struct UpdateWhitelist<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,
```

Figure C.13: `UpdateWhitelist` struct in `src/lib.rs`

None of the instructions that use the `UpdateWhitelist` struct use the `payer` account.

- **Return an error if the buffer account is already initialized in the `create_buffer` instruction.** The instruction logs a message only if the account is already initialized. Returning an error would protect against any unwanted changes from reinitialization.

```
if is_uninitialized_account(buffer_account) {
   [...]
} else {
    // FIXME: change this to be emit!(Event)
    msg!("Buffer account already initialized");
}
```

Figure C.14: The `create_buffer` function in `instructions/create_buffer.rs`

- **Read `bump` value from the message buffer account.** The `delete_buffer` and `resize_buffer` instructions take `bump` as an instruction argument. Reading `bump`s from the account data reduces the need for additional validation and reduces the attack surface.

```
pub fn delete_buffer<'info>(
    [...]
    bump: u8,
) -> Result<()> {
```

Figure C.15: The `delete_buffer` function in `instructions/delete_buffer.rs`

- **Check that the `base_account_key` argument is not equal to `Pubkey::default()` in the `create_buffer` instruction.** Doing so will help ensure that `base_account_key` is not cleared accidentally.

```
pub fn create_buffer<'info>(
    [...]
    base_account_key: Pubkey,
) -> Result<()> {
```

Figure C.16: The `create_buffer` function in `instructions/create_buffer.rs`

- **Limit the maximum size of a message buffer account to `u16::MAX` in the `resize_buffer` instruction.** Buffer accounts contain `end_offsets`, which are pointers into the account data. These are represented by `u16`. As a result, the addressable size of the buffer account is limited by `u16::MAX + HEADER_LEN`.

- **Avoid reimplementing logic.** The code to compute and validate the buffer account PDA is reimplemented in the `resize_buffer` and `delete_buffer` instructions. The `MessageBuffer.validate` function performs the same operations and should be used instead of repeating the code. Repeating code increases the code complexity and hinders readability.

- **Use `MessageBuffer::HEADER_LEN` as `header_end_index` instead of computing it again.**

```
let header_end_index = mem::size_of::<MessageBuffer>() + 8;
```

Figure C.17: A snippet of the `put_all` function in `instructions/put_all.rs`

- **Return an explicit error when the number of messages to write is greater than the `end_offsets` length.** `end_offsets` stores the offset for each message and can store only 255 offsets. When the number of messages is greater than 255, the execution fails implicitly. However, an explicit error code will help in debugging and make reviewing the codebase easier.

```
pub fn put_all_in_buffer(
    [...]
    values: &Vec<Vec<u8>>,
) -> (usize, u16) {

    for (i, v) in values.iter().enumerate() {
        [...]
        self.end_offsets[i] = end;
```

Figure C.18: A snippet of the `MessageBuffer.put_all_in_buffer` function in `state/message_buffer.rs`

- **Add owner checks for message buffer accounts in `put_all`, `resize_buffer`, and `delete_buffer` instructions.** Validating the account ownership reduces the attack surface.

- **Simplify the calculation of internal Merkle tree nodes.** The `MerkleAccumulator`'s associated function, `new`, computes a Merkle tree for a given number of items. The code that computes the hash of the internal Merkle tree nodes is shown in figure C.19. This code iterates over an inclusive range and then offsets the loop counter variable by one, which could be simplified, as shown in figure C.20.

```
// Filling the node hashes from bottom to top
for k in (1..=depth).rev() {
    let level = k - 1;
    let level_num_nodes = 1 << level;
[...]
```

Figure C.19: `pythnet/pythnet_sdk/src/accumulators/merkle.rs#L155–L158`

```
// Filling the node hashes from bottom to top
for k in (0..depth).rev() {
    let level = k;
    let level_num_nodes = 1 << level;
[...]
```

Figure C.20: Suggested refactoring

- **Remove the developer's name from the `Anchor.toml` file since the path will not exist on most machines and doing so will help protect the developer's privacy:**

```
wallet = "/Users/<redacted>/.config/solana/id.json"
```

Figure C.21: pyth-crosschain/governance/remote_executor/Anchor.toml#L11

- **Eliminate the coupling between the `test_adversarial.rs` and `executor_simulator.rs` files.** The `test_adversarial.rs` file performs several test attacks (figure C.22). However, to understand what those attacks involve, one must look in `executor_simulator.rs` (figure C.23). Ideally, the information needed to understand the attacks would be contained within a single file.

```
let vaa_account_wrong_data = bench.add_vaa_account(
    &emitter,
    &vec![transfer(
        &executor_key,
        &receiver,
        Rent::default().minimum_balance(0),
    )],
    VaaAttack::WrongData,
);
...
// VAA with random bytes
assert_eq!(
    sim.execute_posted_vaa(
        &vaa_account_valid,
        &vec![],
        ExecutorAttack::WrongVaaAddress(vaa_account_wrong_data)
    )
    .await
    .unwrap_err()
    .unwrap(),
    ErrorCode::AccountDidNotDeserialize.into_transation_error()
);
...
```

Figure C.22: pyth-crosschain/governance/remote_executor/programs/remote-executor/src/tests/test_adversarial.rs#L49–L150

```
/// Add VAA account with emitter and instructions for consumption by the
remote_executor
pub fn add_vaa_account(
    &mut self,
    emitter: &Pubkey,
    instructions: &Vec<Instruction>,
    validity: VaaAttack,
```

```
) -> Pubkey {
    ...
    let data: Vec<u8> = match validity {
        VaaAttack::WrongData => (0..vaa_bytes.len()).map(|_|
rand::random::<u8>()).collect(),
        _ => vaa_bytes,
    };
...
}
...
// ExecutorAttack overrides
match executor_attack {
    ExecutorAttack::WrongVaaAddress(key) => account_metas[1].pubkey = key,
    ExecutorAttack::WrongEmptyClaimAddress => {
        account_metas[2].pubkey = Pubkey::find_program_address(
            &[
                CLAIM_RECORD_SEED.as_bytes(),
                &Pubkey::new_unique().to_bytes(),
            ],
            &self.program_id,
        )
        .0
    }
    ExecutorAttack::WrongClaimAddress(key) => account_metas[2].pubkey = key,
    ExecutorAttack::WrongSystemProgram => account_metas[3].pubkey =
Pubkey::new_unique(),
    _ => {}
};
```

Figure C.23: pyth-crosschain/governance/remote_executor/programs/remote-executor/src/tests/executor_simulator.rs#L147–L318

- **Address the warnings that are produced when Clippy is run on `pythnet_sdk`.** More generally, regularly run Clippy on `pythnet_sdk`, and address any warnings that are produced.

```
warning: this call to `as_ref` does nothing
   --> src/accumulators/merkle.rs:149:57
    |
149 |                 tree[(1 << depth) + i] =
hash_leaf::<H>(items[i].as_ref());
    |
^^^^^^^^^^^^^^^^ help: try this: `items[i]`
    |
    = help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#useless_asref
    = note: `#[warn(clippy::useless_asref)]` on by default
```

```
warning: using `clone` on type `<H as Hasher>::Hash` which implements the
`Copy` trait
   --> src/accumulators/merkle.rs:174:23
    |
174 |             path.push(self.nodes[index ^ 1].clone());
    |                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try removing
the `clone` call: `self.nodes[index ^ 1]`
    |
    = help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#clone_on_copy
    = note: `#[warn(clippy::clone_on_copy)]` on by default
```

Figure C.24: Some of the warnings produced when Clippy is run on `pythnet_sdk`

- **Do not write to `*buf` in `<MessageData as BorshDeserialize>::deserialize` when an error occurs (figure C.25).** The current approach makes it impossible for the caller to try deserializing `buf` using another method.

```
*buf = &buf[3..];
Ok(PostedMessageUnreliableData {
    message: <MessageData as BorshDeserialize>::deserialize(buf)?,
})
```

Figure C.25: pyth-crosschain/pythnet/pythnet_sdk/src/wormhole.rs#L65–L68

- **Rename `Accumulator::from_set` to `from_iter` or something similar (figure C.26).** The method's argument type is not a set.

```
fn from_set(items: impl Iterator<Item = &'a [u8]>) -> Option<Self>;
```

Figure C.26: pyth-crosschain/pythnet/pythnet_sdk/src/accumulators.rs#L30

- **Consider making `MerkleAccumulator::new`'s argument type `impl IntoIterator<...>` instead of a slice (figure C.27).** Making this change will save callers from having to `collect` the argument first.

```
pub fn new(items: &[&[u8]]) -> Option<Self> {
```

Figure C.27: pyth-crosschain/pythnet/pythnet_sdk/accumulators/merkle.rs#L138

- **Update the following ambiguous comment to explain clearly why trees with a size of 1 are not tested.**

```
// Note that this is testing proofs for trees size 2 and greater, as a size 1
tree the root is
// its own proof and will always pass. This just checks the most obvious case
```

```
that an empty or
// default proof should obviously not work, see the proptest for a more
thorough check.
```

Figure C.28: pyth-crosschain/pythnet/pythnet_sdk/accumulators/merkle.rs#L303–L305

The comment in figure C.29 is more precise and helpful.

```
// Reject 1-sized trees as they will always pass due to root being the only
elements
// own proof (I.E proof is [])
```

Figure C.29: pyth-crosschain/pythnet/pythnet_sdk/accumulators/merkle.rs#L445–L446

- **Format the oracle program source code with `rustfmt`.** Currently, the source code appears to be unformatted, which decreases its readability.
- **Address the following Clippy warning, which is produced by running Clippy with `-- -W clippy::pedantic` on the oracle program:**

```
warning: this argument (8 byte) is passed by reference, but would be more
efficient if passed by value (limit: 8 byte)
  --> program/rust/src/utils.rs:88:14
   |
88 |     cmd_hdr: &CommandHeader,
   |              ^^^^^^^^^^^^^^ help: consider passing by value instead:
`CommandHeader`
   |
   = help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#trivially_copy_pass_
by_ref
   = note: `-W clippy::trivially-copy-pass-by-ref` implied by `-W
clippy::pedantic`
```

Figure C.30: One of the Clippy warnings produced when Clippy is run with `-- -W clippy::pedantic` on the oracle program

- **In the two `as_bytes` implementations in the `prices.rs` file, change all uses of the `clone_from_slice` method to the `copy_from_slice` method to ensure `u8`'s `Copy` trait implementation is used:**

```
pub fn as_bytes(&self) -> [u8; Self::MESSAGE_SIZE] {
    let mut bytes = [0u8; Self::MESSAGE_SIZE];

    let mut i: usize = 0;

    bytes[i..i + 1].clone_from_slice(&[Self::DISCRIMINATOR]);
    i += 1;
```

Figure C.31: pyth-client/program/rust/src/accounts/price.rs#L276–L282

- **Since the system instructions `create_account` and `transfer` do not require the system program, remove it from the `account_infos` array:**

```
let create_instruction = create_account(from.key, to.key, lamports,
try_convert(space)?, owner);
invoke_signed(
    &create_instruction,
    &[from.clone(), to.clone(), system_program.clone()],
    &[seeds],
)?;
```

Figure C.32: pyth-client/program/rust/src/accounts.rs#L169–L174

```
let transfer_instruction = transfer(from.key, to.key, amount);
invoke(
    &transfer_instruction,
    &[from.clone(), to.clone(), system_program.clone()],
)?;
```

Figure C.33: pyth-client/program/rust/src/utils.rs#L287–L291

- **Use an assertion to check the invariant that `price_data.num_` is less than or equal to `crate::c_oracle_header::PC_COMP_SIZE` in `del_publisher`.** Doing so will help protect the code against future changes.

- **Correct the following inaccurate comments.** Currently, the `data_curation_authority` role has no special privileges, and the `security_authority` role's only privilege is to resize accounts.

```
/// An authority that can  :
/// - Add mapping accounts
/// - Add price accounts
/// - Add product accounts
/// - Delete price accounts
/// - Delete product accounts
/// - Update product accounts
pub data_curation_authority: Pubkey,
```

```
/// An authority that can  :
/// - Add publishers
/// - Delete publishers
/// - Set minimum number of publishers
pub security_authority:      Pubkey,
```

Figure C.34: pyth-client/program/rust/src/accounts/permission.rs#L27–L39

- **Add a statement of the form `price_data.update_price_cumulative().unwrap();` after the assignments following the `// Slot decreases` comment in `test_twap.rs`.** Currently, the slot decreases functionality is not tested correctly.

```
// Slot decreases
price_data.agg_ = PriceInfo {
    price_:           300,
    conf_:            6,
    status_:          PC_STATUS_TRADING,
    corp_act_status_: 0,
    pub_slot_:        1,
};
price_data.prev_slot_ = 5;
// Add here: price_data.update_price_cumulative().unwrap();

assert_eq!(price_data.price_cumulative.price, 1 - 2 * 10);
assert_eq!(price_data.price_cumulative.conf, 2 + 2 * 5);
assert_eq!(price_data.price_cumulative.num_down_slots, 3);
```

Figure C.35: pyth-client/program/rust/src/tests/test_twap.rs#L199–L211

- **Simplify the code in figure C.36.** The code can be rewritten as in figure C.37.

```
match (*key, command) {
    (pubkey, _) if pubkey == self.master_authority => true,
    (pubkey, OracleCommand::ResizePriceAccount) if pubkey ==
self.security_authority => {
        true
    } // Allow for an admin key to resize the price account
    _ => false,
}
```

Figure C.36: pyth-client/program/rust/src/accounts/permission.rs#L45–L51

```
match command {
    _ if *key == self.master_authority => true,
    OracleCommand::ResizePriceAccount if *key == self.security_authority => {
        true
```

```
    } // Allow for an admin key to resize the price account
    _ => false,
}
```

Figure C.37: Proposed simplification

- **Make the `rerun-if-changed` directive in the oracle program's build script more precise.** Currently, the oracle program is rebuilt on nearly every Cargo command.

```
println!("cargo:rerun-if-changed=../")
```

Figure C.38: `pyth-client/program/rust/build.rs#L64`

- **Make the code more consistent by using `pyth_assert` in the locations in figures C.39 through C.41.** Doing so will make the code more readable.

```
if command_header.version != PC_VERSION {
    return Err(OracleError::InvalidInstructionVersion);
}
```

Figure C.39: `pyth-client/program/rust/src/instruction.rs#L114–L116`

```
if price_data.num_ >= PC_COMP_SIZE {
    return Err(ProgramError::InvalidArgument);
}
```

Figure C.40: `pyth-client/program/rust/src/processor/add_publisher.rs#L67–L69`

```
if cmd_args.publisher == price_data.comp_[i].pub_ {
    return Err(ProgramError::InvalidArgument);
}
```

Figure C.41: `pyth-client/program/rust/src/processor/add_publisher.rs#L72–L74`

D. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
    - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which Pyth Data Association will compensate users affected by an issue (if any).**
    - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
    - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific components of the system.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers, etc.) and how it will onboard them.**
    - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop "muscle memory." Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

E. Architectural Complexity Considerations

This appendix provides recommendations to aid in determining whether the current architecture is too complex and should be simplified. More specifically, it provides suggestions for judging whether a component incurs an undue maintenance burden, as well as references for real-world situations involving systems determined to be overly complex.

Discussion

As mentioned under Observations and Impact, various aspects of the system do not appear to have kept up with its complexity. These aspects include documentation, testing, dependency management, and logging. Simplifying a complex architecture is one way of reducing its maintenance burden.

Ways in which the Pythnet architecture might be simplified include the following:

- Base an individual node's software on something simpler than a Solana validator.
- Deploy fewer programs on Pythnet (e.g., consolidate the remote executor program, oracle program, and/or message buffer program).
- Have the remote executor CLI talk directly to Solana rather than receive messages through Wormhole.
- Deploy a multisig to Pythnet and have it interact directly with the contracts on Pythnet, rather than transferring messages through Wormhole and the remote executor CLI.

To be clear, we are not advocating for any one of these simplifications. We are simply pointing out that there are ways the architecture might be simplified.

Moreover, we cannot offer recommendations on how to simplify the architecture at this time. Such recommendations would require data not considered during this audit (see, for example, the Metrics section below).

Finally, it is easy to point to a component and argue that it provides a benefit. Thus, simplifying the architecture may involve tough choices. Consider the following deliberately exaggerated example.

Suppose that before submitting a VAA to the remote executor program, the remote executor CLI first sends the VAA to a satellite in orbit. The purpose is to help ensure that VAAs can be recovered following earthly events such as earthquakes, fires, floods, etc. The benefits are real in the sense that sending VAAs to a satellite will help ensure their

longevity. However, it is not clear that this benefit would justify the cost of maintaining a satellite.

Metrics

Building software requires cost benefit analysis as new features and overengineering may detract from addressing long-standing technical debt or investing in proactive measures such as testing. Consider the following questions when deciding whether a component incurs a disproportionate maintenance burden.

- How frequently has the component been updated? Is the code relatively stable, or does it require frequent updates?
- Do external factors (network bandwidth, lack of memory, lack of disk space, etc.) often cause the component to fail?
- Do external changes tend to break the component (e.g., does it rely on features that are unpublished, unstable, or otherwise not guaranteed)?
- How often are GitHub issues opened about the component?
- Do pull requests involving the component tend to require long review cycles?
- Do GitHub checks for the component's pull requests tend to pass on the first attempt? Or do developers struggle to get the checks to pass?

References

Here are some references to aid in determining whether a system is overly complex.

- Creately; "Sequence Diagram Tutorial – Complete Guide with Examples," by Amanda Athuraliya, last updated December 12, 2022.

  This post explains how to create sequence diagrams (i.e., diagrams showing "how different parts of a system work in a 'sequence' to get something done") and provides numerous examples of such diagrams. The post makes the following point:

  > *Make sure that the diagram fits on a single page and leaves space for explanatory notes too.*

  Thus, if a system's use case cannot be represented by a sequence diagram that fits on a single page, it may indicate that the system is overly complex.

- Robert Nord and Ipek Ozkaya, "10 Years of Research in Technical Debt and an Agenda for the Future," *SEI Blog* (blog), *Software Engineering Institute*, August 22, 2023.

This post puts forth an interesting definition of technical debt: an invisible feature that provides negative value. If a system has many such features, it may indicate that the system is overly complex.

- David Heinemeier Hansson, "Even Amazon can't make sense of serverless or microservices, *HEY World*, May 4, 2023.

  Devansh, "Amazon Prime Video reduced costs by 90% by ditching Microservices," *Dev Genius* (blog), May 9, 2023.

  Both posts describe the same event: Amazon Prime Video's move away from microservices. A system that exhibits similarities to Amazon Prime Video may be overly complex.