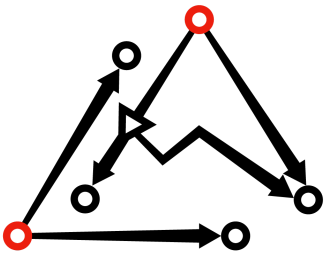

Least Authority

PRIVACY MATTERS

Compound + Pando

Security Audit Report

Fox One

Final Audit Report: 10 September 2021

Table of Contents

Overview

Background

Fox One has requested that Least Authority perform a security audit of Pando and Compound. Pando is a decentralized financial network built with the Mixin Trusted Group (MTG) technology, a multi-signature custodian consensus solution, and an alternative to smart contacts on the Mixin Network. Pando's underlying financial algorithm is inspired by MakerDao and Synthetix. Compound is an implementation of the compound protocol based on Mixin MTG technology.

Project Dates

- **June 2 - July 28**: Code review *(Completed)*
- **July 30**: Delivery of Initial Audit Report *(Completed)*
- **September 9:** Verification completed *(Completed)*
- **September 10:** Delivery of Final Audit Report *(Completed)*

Review Team

- Alicia Blackett, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer
- David Braun, Security Researcher and Engineer
- Jehad Baeth, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of Pando and Compound followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Pando: https://github.com/https://github.com/fox-one/pandofox-one/pando
- Compound: https://github.com/fox-one/compound

Specifically, we examined the Git revisions for our initial review:

Pando: `04289223f8af2bbbca185e526c6e6b8a6a50280b`
Compound: `ab236d9ba64eb7773883a895d8c95a734c8dd60d`

For the verification, we examined the Git revision:

Pando: `8b718e53debe3030819250f8df9d0a360515c6c7`
Compound: `c25066a86b4237f872d036a63b21b48c9e7927e0`

For the review, these repositories were cloned for use during the audit and for reference in this report:

Pando: https://github.com/LeastAuthority/fox-one-pando
Compound: https://github.com/LeastAuthority/fox-one-compound

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Pando
  - `README.md`: https://github.com/fox-one/pando/blob/main/README.md
  - `Design.md`: https://github.com/fox-one/pando/blob/main/design.md
  - Pando User Interface Guide.pdf (*provided to Least Authority via email on 13 April 2021*)
- Compound
  - `README.md`: https://github.com/fox-one/compound/blob/master/README.md
  - `Deploy.md`: https://github.com/fox-one/compound/blob/audit/docs/deploy.md
  - `Design.md`: https://github.com/fox-one/compound/blob/audit/docs/design.md
  - Compound Documentation: https://github.com/fox-one/compound/tree/master/docs
- Mixin Network Developer Documentation: https://developers.mixin.one/document/mainnet/overview
- Guidance to Create a Mixin dAPP and get its keystore.pdf (*provided to Least Authority via Telegram on 11 June 2021*)

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the network;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the network's intended use or disrupt its execution;
- Vulnerabilities in the network code;
- Protection against malicious attacks and other ways to exploit the network;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Fox One's Compound and Pando are protocols intended to bring Decentralized Finance (DeFi) capabilities to the Mixin Network. The Mixin Network is a punitive Proof of Stake (PoS), cross-chain network consisting of a maximum of 50 mainnet nodes. Transactions on the Mixin Network include a memo field that can be loaded with data.

The Mixin Trusted Group (MTG) is a para-chain of the Mixin network, consisting of a set of nodes that interact with the Mixin Network via a multi-signature implementation. Each node in the MTG posts transactions to the Mixin Network, and scans the encrypted contents of the memo field of each transaction. By succeeding in decrypting the data in the memo field, the node identifies a transaction from another node in the MTG.

The Compound and Pando protocols are to be deployed to the MTG Network, by which user transactions that are verified by the MTG are written to the Mixin Network.

System Design

Our team conducted a broad and comprehensive review of the Compound and Pando protocol code bases. To aid in this effort, we solicited the help of a Mandarin Chinese translator, to facilitate discussion with the Fox One team and to assist in document translation from Mandarin Chinese to English. In our efforts of reviewing the system design, we found that there are several areas of concern in the design of these protocols and the dependencies on which they rely.

In our review of the code base and the project documentation for Compound and Pando, our team concluded that security has not been sufficiently prioritized in the system design, as evident in the absence of a coherent security plan, a lack of adherence to accepted security practices, and excess in concentration of authority. These factors increase the risk of exposing the users of Compound and Pando to loss of assets. As a result, we recommend that the development process adopted by the Fox One team is guided by a comprehensive security roadmap (Issue B).

Cryptography

Our team conducted a thorough investigation of the implementation of cryptographic primitives in the Compound and Pando protocols. We found that AES encryption is performed in unauthenticated CBC mode, which is vulnerable to padding oracle attacks by utilizing arbitrary values while checking a message's correctness. We recommend using authenticated AES-GCM mode for encryptions (Issue E).

Additionally, our team analyzed the key generation, encryption, and decryption used for the memo field. To send a raw memo, the user encrypts the raw memo using AES in CBC mode (Issue E). The encrypted memo is then decrypted by the MTG node to perform the operation. Since a shared key between the user and the MTG node must be generated, an Elliptic Curve Diffie Hellman Key Exchange (ECDH) is performed. The user generates a one-time Ed25519 key pair and uses the (shared) Ed25519 public key of the MTG nodes to generate a shared key by scalar multiplication of these values (the passed private key value in this operation is undergoing clamping). After encrypting the raw memo with the resulting shared key, the encrypted memo along with the generated Ed25519 user public key is sent to the MTG nodes. An MTG node uses its Ed25519 private key and the received Ed25519 public key of the user to perform the equivalent operation to generate the shared key to decrypt the encrypted memo.

While using ECDH to agree on an encryption algorithm is a known best practice, our team noted that the generated key pairs are on the Edwards form of Curve25519. Since only scalar multiplication is used within this step and even though scalar multiplication on Ed25519 might be less efficient, this doesn't pose a problem in this setting. Since, also, only one coordinate of Ed25519 points is passed and with only one coordinate, operations are performed, the same argumentation as for the Montgomery form of Curve25519 with only the x-coordinate holds, where points don't need to be validated. Our team did not identify any security vulnerabilities resulting from this implementation.

Shared Key Security

In an MTG network, the participating nodes all utilize a configuration file that includes, in plain text, the key pairs required by nodes to participate in the network. Our team found that generating keys externally and distributing them manually is insecure. In addition, our team did not identify any provisions in the code base or the documentation for managing an unexpected event that requires a new key to be distributed. We recommend that the shared key be managed in adherence with best practices, in order to reduce the risks of key compromise (Issue A). We also recommend that the security roadmap include plans to protect users in case of key compromise (Issue A).

Excess Centralization

The design of Compound and Pando relies on seven MTG nodes. A four-node quorum is required to validate transactions. The quorum is implemented via an M-of-N multi-signature scheme. Our team was unable to verify that the node operators are independent and approximate that there is a high risk of collusion between the MTG members. Collusion risk aside, an attacker would only need to compromise four machines to compromise the entire system. By comparison, the Ethereum distributed ledger comprises more than 5,000 nodes.

Although the Compound and Pando protocols are inspired by similar decentralized systems, which as a result of being decentralized, provide users and the system itself a minimum level of assurance of the security and stability of the network, the implementation as reviewed by our team is characterized by excessive centralization that could put users of the system at risk (Issue G).

Code Quality

In our review of the Compound and Pando code bases, we found the code is not written in adherence with best practices. For example, our team found instances of unused code, abbreviated variable names, and excessively long functions. These coding practices inhibit the understanding of the intended functionality of the code and increase the likelihood of implementation errors going unnoticed. We recommend improving the code quality of the Compound and Pando implementations in order to minimize the potential for errors or misunderstanding by both implementers and security reviewers (Suggestion 4).

Tests

The test coverage for Compound and Pando was found to be insufficient, with 7% test coverage for Compound and 5% coverage for Pando. Sufficient test coverage (i.e. ideally >80%) tests for success and failure cases aid in identifying errors and bugs and protect against potential edge cases, which may lead to security critical vulnerabilities or exploits. A robust test suite includes a minimum of unit tests and integration tests. In addition, end-to-end testing is recommended so that it can be determined if the implementation behaves as intended. Finally, a script should be provided with clear instructions so that the tests can be run via a single command, and a testnet should be implemented. We recommend that all of the aforementioned best practices be implemented as it relates to test coverage (Suggestion 1).

Documentation

The project documentation provided by the Fox One team is accurate, but insufficient in providing a clear and comprehensive understanding of the architecture of the system, in addition to missing a description for each of the components and how their interaction is represented in the code. A large code base like Pando (10k lines of code) and Compound (25k lines of code) reflects a complex system with intricate interactions, which make the system more difficult to reason about, thus necessitating thorough and comprehensive documentation.

While the Fox One team was very helpful and responsive in answering our team's questions, which alleviated some of the challenges created by the lack of project documentation, the provided documentation is not suitable for facilitating a review by security researchers and is only sufficient as a reminder reference for the development team already familiar with the system. For example, the Compound Governance documentation for the `inject-ctoken` command is "inject the ctoken to the multi-sign wallet", which does not provide any useful information to reviewers or new maintainers of the code base.

Furthermore, specification documentation for Compound and Pando has not been created, clearly describing each of the systems, their intended functionality, and how the functionality is realized in the

components that make up each code base. As a result, we recommend creating a specification for both Compound and Pando (Issue C).

Finally, the deployment documentation is unclear and the instructions are overly complex. Simple, straightforward, working instructions for executing the code and running the tests by executing a minimum number of steps is considered to be a security best practice. As a result, we recommend that the Fox One team improve the project documentation to facilitate a better understanding of the system by users, developers, and security researchers (Suggestion 2).

Code Comments

The Pando and Compound code bases have insufficient code comments. The documentation contained within the code should be comprehensive and document every function and entry point, explaining clearly the intended functionality of each of the components. This minimizes the opportunity for missing potential errors and inconsistencies in the intended behavior of a system, which may result in security vulnerabilities (Suggestion 3).

Scope and Dependencies

The Pando and Compound protocols are critically dependent on the Mixin Network and the MTG, both of which were out of scope for this security review. We recommend that a security review of critical components of those systems be performed as part of the Compound and Pando security roadmap (Issue B).

Additionally, our team identified critical security functions such as PIN and token generation performed by the `mixin-sdk`, which was out of scope for this security review. We recommend that a security review of the `mixin-sdk` also be included in the security roadmap (Issue B).

The Compound and Pando protocol rely on an oracle to fetch market pricing data for assets transacted in both protocols. Oracles are security-critical components with many known security vulnerabilities. As a result, we recommend that a review of the oracle component be included in the security roadmap (Issue B).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: No Provision to Handle Compromise of Shared MTG Key [Compound & Pando]	Unresolved
Issue B: Security Roadmap Nonexistent [Compound & Pando]	Unresolved
Issue C: Protocol Specifications Nonexistent [Compound & Pando]	Unresolved
Issue D: Secrets Are Shared and Persist in Plain Text [Compound & Pando]	Unresolved
Issue E: Use of Unauthenticated Encryption Mode [Compound & Pando]	Unresolved
Issue F: Input Not Checked When Adding or Removing PKCS #7 Padding	Unresolved

Issue A: No Provision to Handle Compromise of Shared MTG Key [Compound & Pando]

Location

Compound: `audit/deploy/config.node.yaml.tpl#L31`

Pando: `cmd/pando-worker/README.md` (and examples of configuration files at `https://gist.github.com/yiplee/0d41b232c93dbb8048d0fd30951d967b`)

Synopsis

In Compound and Pando, for each node, key pairs are generated outside of the protocol. For example, in Compound one key pair is unique to each node and used for signing transactions. Another key pair is shared among the MTG nodes, and is used for the process of encrypting and decrypting data stored in the memo field. Key pairs along with other secrets are stored unencrypted in the `config.yaml` file of each MTG node. We found no processes in place to deal with the compromise, revocation, or unplanned change of keys.

As part of the key revocation process, nodes using the public key of a key pair will need to be informed that the public key may no longer be used. New keys must be generated in instances of a member node leaving the group, the keys becoming compromised, a newly discovered security vulnerability having an impact on the security of the keys, or the lifetime of the key has been reached.

In the event of a key compromise, the compromised key must be revoked and the node operators must manually receive a new keypair. User notification and key replacement becomes more complex as the system scales. The key revocation and replacement process must include a security assessment of data encrypted by the compromised key.

Impact

In the case that an unexpected key replacement is required, and is not executed in a timely and well coordinated manner, nodes using revoked keys could be severed from the MTG network, and the MTG could potentially be destabilized, putting users at risk.

Preconditions

An attack would require access to the machine running the node, and access to the `config.yaml` file.

Feasibility

All cases of unplanned key changes, as previously mentioned, will result in this issue.

Remediation

We recommend using specialized key management tooling which automates the creation, storage, and replacement of keys in a secure manner.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to “system constraints”. As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue B: Security Roadmap Nonexistent [Compound & Pando]

Synopsis

A clear and transparent security roadmap conveys concisely how the risks and concerns that can impact financial protocols are being addressed. The Compound and Pando protocols have the potential to accumulate and control significant amounts of funds. Thus, there are many security risks to different aspects of these systems. We identify three different categories of risks that surround such systems:

- Technical risks: Faulty implementation of the system, usage of dependencies, inter-node communication, attacks targeting the Compound or Pando infrastructure causing service interruption, hijacking of trusted MTG nodes, and tampering with oracles servers.
- Economic risks: Exploits of the economic model of Compound and Pando and how the protocols react to blackswan events.
- Third-party security failure risks: Failures of third-party services utilized by Compound and Pando, such as the Mixin Network or the oracles used to fetch market information.

Systems like Compound and Pando would benefit from having a publicly accessible and comprehensive security roadmap that can increase trust by the community and broaden user adoption.

Impact

The impact of the risks can include, but is not limited to, users’ loss of funds, system interruption, and the protocol losing adoption.

Mitigation

Systems similar to Compound and Pando, such as MakerDao, have publicly accessible security roadmaps that demonstrate what has been done and what is planned for improving the securing of these critical systems. Compound and Pando can benefit from adopting a similar approach to the security of their protocol.

To increase the overall security of the protocols, we recommend creating a publicly accessible security roadmap and including the following components:

- Develop and document a security roadmap for Compound and Pando;
- Publish Compound and Pando code bases as open source;
- Include multiple audits by different security teams in the security roadmap;
    - Mixin Network and MTG
    - `mixin-sdk`

    - Oracle Component
    - Pando Rings Web
    - Pando Leaf Web
- Plan audits at appropriate milestones in the development roadmap of both protocols; and
- Participate in bug bounty programs.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to “system constraints”. As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue C: Protocol Specifications Nonexistent [Compound & Pando]

Location

Compound: `design.md`

Pando: `design.md`

Synopsis

Compound

The implementation of Compound for Mixin is inspired by the Compound Protocol by Compound Finance. The Compound Protocol is a smart contract for Ethereum for borrowing and supplying assets. Since the underlying machinery of Mixin Network differs strongly from the Ethereum Virtual Machine and Ecosystem, our team recommends thorough documentation and analysis of the Compound Protocol in the setting of Mixin Network.

Pando

Pando’s design is inspired by MakerDao and Synthetix, very complex systems with carefully balanced incentives and governance mechanisms to ensure that the systems remain stable and free from exploitation. To inspire user trust and provide assurance in the system, it is critical to provide a clear protocol specification of how the system is intended to behave, and to verify through 3rd party security teams that the code is working in accordance with the specification. For comparison, see MakerDAO's protocol specification.

Impact

Compound & Pando

Our team found that the Compound and Pando design document provided for this security review to be insufficient. A usable protocol specification must specify all of the functionality in detail, provide rationale for why each function is necessary, and explain how it all fits together. To facilitate verification of code compliance, the specification must be comprehensive, detailed, and specific. A specification is complete enough if it can enable a software developer to create an implementation of the system without any other source of guidance.

Bugs and vulnerabilities in large and complex systems such as Pando and Compound for the Mixin Network are difficult to find, and the probability of implementation errors going unnoticed is high. Ideally, bugs should be identified at the protocol layer before code is written. A well-written specification facilitates reasoning about the protocol design independent of its implementation. The specification is a

prototype written in prose so that its design can be discussed, argued about, and refined at a high level of abstraction. Like an architectural blueprint, it enables design problems to be addressed at a high level before they become expensive at the implementation level.

Remediation

Compound

We recommend writing a technical specification for the Compound Protocol for the Mixin Network according to industry standard best practices. See A practical guide to writing technical specs, the Compound Protocol Whitepaper, and the Compound Protocol specifications by Compound Finance for guidance and inspiration.

Pando

We recommend writing a technical specification for the Pando protocol according to industry standard best practices. See A practical guide to writing technical specs and the MakerDAO specifications for guidance and inspiration.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to "system constraints". As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue D: Secrets Are Shared and Persist in Plain Text [Compound & Pando]

Location

Compound:

`audit/deploy/config.node.yaml.tpl#L31`

`audit/deploy/config.node.yaml.tpl#L26`

Pando:

`cmd/pando-worker/README.md` (and examples of configuration files at `https://gist.github.com/yiplee/0d41b232c93dbb8048d0fd30951d967b`)

Synopsis

To deploy an MTG node, the `config.yaml` configuration file is used, which contains the two sets of public and private keys and the PIN number for the wallet that is associated with the node, all in plain text.

Impact

An attacker gaining access to the private keys or the node wallet PIN number can gain unauthorized access to assets or information and spoof any of the nodes in the MTG.

Preconditions

After deployment, if the configuration files remain on the system unencrypted or with liberal access rights, the secret information could become compromised.

Mitigation

We recommend restricting the access rights on the configuration file or consider using a keystore and removing secret information from the configuration file.

Remediation

We recommend developing a secrets management process to include cryptographic keys, passwords, and PIN numbers to deploy secrets securely. This includes having a secrets management infrastructure in the build, deploy, and production phases.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to “system constraints”. As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue E: Use of Unauthenticated Encryption Mode [Compound & Pando]

Location

Compound:

pkg/aes/aes.go#L35

pkg/aes/aes.go#L56

Pando:

pkg/aes/aes.go#L35

pkg/aes/aes.go#L56

Synopsis

The Compound and Pando protocols utilize the AES-CBC mode, while the AES-GCM authenticated-encryption mode is known to be more secure.

Impact

CBC mode is vulnerable to padding oracle attacks by utilizing arbitrary values leaked by the oracle while checking a message’s correctness. While Galois/Counter Modes are impervious to such attacks, they handle encryption and validate data integrity, thus also providing protection against data tampering.

Technical Details

Both Compound and Pando utilize AES-CBC mode. Example:

```
func Encrypt(data []byte, key, iv []byte) ([]byte, error) {
      ckey, err := aes.NewCipher(key)
      if nil != err {
            return nil, err
      }
```

```
encrypter := cipher.NewCBCEncrypter(ckey, iv)
```

AES-CBC can be vulnerable against Bit-Flipping attacks as explained in Bit Flipping Attack on CBC Mode.

Remediation

We recommend utilizing AES with a Galois/Counter Mode AES-GCM based authenticated encryption algorithm.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to "system constraints". As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue F: Input Not Checked When Adding or Removing PKCS #7 Padding [Compound & Pando]

Location

Compound:

`pkg/aes/aes.go#L10-L15`

`pkg/aes/aes.go#L17-L26`

Pando:

`pkg/aes/aes.go#L10-L15`

`pkg/aes/aes.go#L17-L26`

Synopsis

Neither the adding PKCS #7 padding nor the removing PKCS #7 padding functions check the input in both Compound and Pando repositories. It is possible to input data of incompatible size in regard to block size into the removing PKCS #7 padding function, which can result in problems decrypting content in AES in CBC mode.

Impact

In its current usage, the adding PKCS #7 padding function is only used in test cases. Although it is a helper function, it should still be implemented correctly in case functionalities of the system are expanded in the future.

The removing PKCS #7 padding function is used in `pkg/mtg/encrypt.go#L17` for decryption of business data and the memo field. Since this comes through the network, input should be checked to stay functional. While in `pkg/mtg/encrypt.go#L17` the compatibility to block size is checked, this should be done at the stage of removing padding, in addition to checks for emptiness.

Preconditions

To exploit this issue, incompatibly padded data can be sent through the network, which would be tested for decryption. Since no input checks when removing the padding follow, decryption errors may occur.

Technical Details

For adding a padding, no checks on block size are implemented. It is possible that the input block size is negative or too big.

For removing a padding, no checks are implemented. It is possible to input data of zero length, or that data is not passed compatible with block size. This could also result in problems in AES encryption and decryption in CBC mode in later stages or in future references to this function.

Remediation

We recommend checking the input before adding and removing padding.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to "system constraints". As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Issue G: Excess Centralization [Compound & Pando]

Synopsis

The design of Compound and Pando relies on seven MTG nodes. A four-node quorum is required to validate transactions.

Impact

The impact can include, but is not limited to, a loss of users funds, price and market manipulation, and service interruption depending on the number of nodes out of the total number of nodes involved going rogue.

Preconditions

A minimum of M MTG nodes out of N MTG nodes getting hacked and taken over or a minimum of M MTG nodes out of N MTG nodes colluding.

Technical Details

The quorum is implemented via an M-of-N multi-signature scheme. Our team was unable to verify that the node operators are independent and approximate that there is a high risk of collusion between the MTG members. Collusion risk aside, an attacker would only need to compromise four machines to compromise the entire system. By comparison, the Ethereum distributed ledger comprises more than 5,000 nodes.

Mitigation

We recommend the Fox One team have an open access policy that allows more nodes to join both Pando and Compound MTG networks. This would provide more safety as more nodes join the network, thus increasing the required number of nodes to collude to cause a negative impact.

Status

The Fox One team has responded that they do not intend to resolve this issue at this time due to "system constraints". As a result, the issue remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestions

Suggestion 1: Implement a Test Suite [Compound & Pando]

Location

Compound: `fox-one-compound`

Pando: `fox-one-pando`

Synopsis

A robust test suite improves code quality by providing a mechanism through which new developers gain an understanding of how the code works through use cases and provides a means to interact with them. In addition, tests build confidence that the code works as intended for known use cases.

A comprehensive test suite is particularly security-critical for any system that will be trusted with large amounts of value, such as Compound and Pando. There are currently only a small number of unit tests in both systems (e.g. Compound has only 7% and Pando has 5% test coverage).

We recommend identifying test cases and separating tests according to correct case, failure case, and edge case scenarios. These three categories should cover intended functionality for correct case tests, unintended functionality for failure case tests, and functionality of the implementation in edge cases tests.

Lastly, we recommend adding a testnet to handle test situations. Without a testnet in place, it is not possible for the development team to properly test interactions within the network.

Mitigation

We recommend the adoption industry standard best practices for testing. We suggest beginning with the Testing Pyramid by writing unit tests to cover at least 80% of the codebase, followed by the implementation of integration tests to test interactions between the major components. We recommend separating the test suite into correct case, failure case, and edge case tests, in addition to verifying that all cases are sufficiently covered.

Additionally, we recommend writing property based tests using the testing/quick standard library. Property based tests are useful for testing finance related smart contracts, in which edge cases may be difficult to anticipate in unit tests but can have significant and security critical consequences. Lastly, we recommend implementing a testnet for testing situations and transfers.

Status

The Fox One team has responded that they intend to implement this suggestion at a future time. However, the suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 2: Improve Project Documentation [Compound & Pando]

Location

Compound: `fox-one-compound`

Pando: `fox-one-pando`

Synopsis

The general documentation provided by the Fox One team for this security review was insufficient. Robust and comprehensive general documentation allows a security team to assess the in-scope components and understand the expected behavior of the system being audited. In addition, clear and concise user documentation provides users with a guide to utilize the application according to security best practices.

Sufficient project documentation should include clear and simple instructions to deploy a test environment and to run automatic integration tests against it. The instructions should enable a developer without any knowledge of the project to onboard quickly and to observe the behavior of running code. If necessary, a script should be provided that installs all necessary dependencies automatically and includes sample configuration files for running with reasonable defaults. Open source projects have the potential to be more secure than closed source projects because they have the advantage of more people looking at the code, but this works in practice only if the project provides a smooth ramp up for new developers who are not yet deeply invested in the details of the codebase. In addition and equally important, this facilitates an easier understanding and provides important context to security reviewers of the systems.

For this reason, it is becoming common practice to include Development Containers with VS Code projects, facilitating the ease at which new development contributors begin work. To protect these quickstart scripts from data degradation, successful execution can be added as a necessary passing condition to continuous integration tests, in order to ensure that every new release provides a means for new developers to test the code with a single instruction.

Mitigation

We recommend improving and expanding the project's general documentation by creating a high-level description of the system, each of the components, and interactions between those components. This can include developer documentation and architectural diagrams.

Status

The Fox One team has responded that they intend to implement this suggestion at a future time. However, the suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 3: Increase Code Comments [Compound & Pando]

Location

Compound: `fox-one-compound`

Pando: `fox-one-pando`

Synopsis

The code comments within the Compound and Pando codebase are minimal and insufficient. The documentation contained within the code should be comprehensive and document every function and entry point, which significantly aids security researchers in identifying implementation errors and potential security vulnerabilities.

Mitigation

We recommend that the code comments be improved and expanded to explain the intended functionality of each of the components, entry points, and functions.

Status

The Fox One team has responded that they intend to implement this suggestion at a future time. However, the suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 4: Improve Code Quality [Compound & Pando]

Location

Compound (example; non-exhaustive):

- Shortened variable names: `audit/worker/snapshot/market_status.go`
- Unused code: `pkg/mtg/encrypt.go#L43-L58`
- Excessively long functions: `worker/snapshot/liquidation.go#L15-L277`
- Utilizing Golang's condition-less switch statement to write clean and readable code: `worker/snapshot/market_status.go#L12`

Pando (example; non-exhaustive):

- Excessively long functions: `maker/vat/frob.go#L12`
- Code that can be written in a more clean form: `maker/proposal/make.go#L35`

Synopsis

In conducting a line-by-line review of the codebase, our team identified several instances of code smells or practices that impact the quality, readability, and maintainability of the Compound and Pando's code bases. For example, we identified shortened variable names that make it hard to understand the information represented by the variable. In addition, we found several instances of unused code. The Fox One team has responded that they intend to start a code cleaning process.

We also found a considerable amount of long functions within the code base. It is considered a best practice to limit function bodies for readability and to encourage better code design (i.e. small functions composed together).

Mitigation

We recommend adhering to best practices in setting variable names. We suggest removing unused and unnecessary code from Compound and Pando code bases. Finally, we suggest decomposing large functions into smaller, more easily tested, and understood component functions.

Status

The Fox One team has responded that they intend to implement this suggestion at a future time. However, the suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 5: Add Audit Logs [Compound & Pando]

Synopsis

Preventative measures such as access control and network segmentation are a common defense against attacks. These measures can be supplemented by enabling logging on the most sensitive components of the architecture and enabling alerts to Fox One in case of suspicious behavior patterns. Logging improves the likelihood of an attack being identified and addressed or prevented.

Mitigation

MTG nodes are manually deployed and the server on which this takes place should have logging enabled. If a malicious user compromises a node, the actions of the user will be recorded in the log files. API's are common targets of DDoS and authentication attacks. Enabling logging on API's (or the associated API gateway or load balancer) would allow Fox One to create alerts for multiple failed authentication attempts, or abnormal API calls. Fox One uses sql databases to record pricing information. Logging of the database could enable Fox One to detect if the prices are being manipulated inside the database by a malicious user.

Status

The Fox One team has responded that they intend to implement this suggestion at a future time. However, the suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.