Least Authority
PRIVACY MATTERS

Centrifuge Chain

Security Audit Report

Centrifuge

Final Report Version: 3 April 2020

Table of Contents

Overview

Background

Centrifuge has requested that Least Authority perform a security audit of the Centrifuge Chain, a Parity Substrate based purpose-specific chain. The Centrifuge Chain is a Proof of Stake chain with block rewards, bridged to Ethereum as its first external network.

Project Dates

- **February 10 - 20**: Code review completed *(Completed)*
- **February 21**: Delivery of Initial Audit Report *(Completed)*
- **March 5**: Delivery of Updated Audit Report *(Completed)*
- **March 30 - April 2:** Verification completed *(Completed)*
- **April 3:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer
- Micro Richter, Cryptography Researcher and Engineer
- Jan Winkelmann, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Centrifuge Chain 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:
- V0.0.3: https://github.com/centrifuge/centrifuge-chain

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

`0556d274d94f29eaaa8f776b84ce5bde065c4983`

For the verification, we examined the Git revision:

`4a3021445a5964bf7263f133656321f38459667f`

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

Supporting Documentation

The following documentation was available to the review team:
- NFT Minting - Bridging (V1).pdf
- Centrifuge Chain V0 - Flimp.pdf
- The Centrifuge Chain Testnets.pdf
- "Centrifuge Chain — the Gateway for Real-World Assets to the Blockchain Multiverse": https://medium.com/centrifuge/centrifuge-chain-the-gateway-for-real-world-assets-to-the-blockchain-multiverse-41dd5597ecf1
- Chain Documentation HackMD.io: https://centrifuge.hackmd.io/6P40PriyRMK4BM0aA8wOUg
- README.md: https://github.com/centrifuge/centrifuge-chain/blob/master/README.md

- Protocol Limitations document: https://develop.developer.centrifuge.io/cent-node/further-reading/protocol-limitations/

Areas of Concern

Our investigation focused on the following areas:

- Worst-case scenarios
  - Ability to change chain history
  - Ability to delete/modify document anchors
  - Unauthorized withdrawal of Radial from bridge contract
    - Double “withdrawal”, etc.
  - Chain halts due to spam/erroneous TXs
  - Ability to submit wrong merkle proofs for NFT minting
  - Ability to bring down/maliciously interact with validators/nodes
- Areas of focus
  - Document anchors
  - Eviction of “old” anchors
  - Fee handling
  - Integration of standard Substrate modules
    - Staking
    - Treasury
    - etc.
  - Asset registration for later NFT minting
  - Use of the asset bridge module (Centrifuge Chain -> Ethereum)
  - Genesis file
  - Standard Substrate modules that are integrated
  - UIs/Front-ends
- Anything else as identified during the initial analysis phase.

Findings

General Comments

We found the code to be very well organized and easy to comprehend. The codebase also included considerable test coverage. The whitepaper, [VPSW17], is technically well written and the authors are transparent about currently unsolved problems within the consensus algorithm. Considering the current state of the industry is experimental, such transparency and analysis is commendable.

However, we found that explanations for most of the chosen Substrate parameters were missing, including the chosen number of MICRO_RADs, the choice for MaximumBlockWeight, and the specific choice of the reward curve, amongst others. As a result, we were unable to effectively reason about the chosen parameters and their interactions, particularly since large parts of the whitepaper specifications have not yet been implemented in the code that was in scope for this audit. We suggest that explanations for the parameters be incorporated into the whitepaper and recommend that another review is conducted once the specifications have been implemented comprehensively in order to check the correctness.

Consensus Design Decisions

We found the Protocol Limitations documentation describing the rationale behind consensus to be well thought out. As this is in the early stages of development, it is understandable that some trade-offs need to be made in initial implementation. As such, we are noting a few potential issues with the consensus design to be considered with future design and development of the protocol. These issues have been discussed during the review with the Centrifuge team and they have stated that their primary focus is to enable onchain privacy of the parties involved in document updates. We caution that this approach may still allow for adversarial participation or blocking of document consensus and should be considered as an area for future improvement.

For example, unfriendly business partners may behave in ways that block state updates of a specific anchor or anchor roots of documents that others do not agree upon (see Issue A and Issue C). Maintaining consistent onchain updates and preventing Byzantine behavior is difficult with decentralized blockchains. Even business partners can sometimes act as counterparties, acting in a malicious or adversarial manner, and the consensus protocol design should be adequately prepared to prevent such types of fraud. At this point, the current design does not sufficiently address these particular concerns and risks. The Centrifuge team has decided that these are acceptable consensus design trade-offs in the short-term and has noted some of this in their Protocol Limitations documentation.

We commend the Centrifuge team for openly addressing the current design limitations and encourage the team to continue iterating on their documented consensus design in order to address the issues with decentralization and adversarial actions.

Specific Issues

We list the issues we found in the code 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: Pre-committing a Document Can Block Counterparty State Updates	Unresolved
Issue B: Fees Must Be Continually Updated to Avoid DoS Issues	Partially Resolved
Issue C: Document Consensus Is Not Enforced	Unresolved
Issue D: Block Proposers May DoS Anchors	Unresolved
Suggestion 1: Use a Lint Program	Resolved
Suggestion 2: Check for Vulnerable Crate Dependencies	Resolved
Suggestion 3: Document A Function That Is Only Used For Testnets	Resolved
Suggestion 4: Expand Documentation Coverage to Include Future Functionality	Partially Resolved
Suggestion 5: Optimized Merkle Proof Checker Depends on the Order of Proofs…	Unresolved

Issue A: Pre-committing a Document Can Block Counterparty State Updates

Location
https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L125

Synopsis
Anyone that has access to an `anchor_id` preimage ID ($d_{next-img}$) can block committing an update of that document by any counterparties involved in that document for up to `pre_commit_expiration_duration_blocks()`. As pre-commits fall off from the eviction process, the attacker could resubmit and continue to block state updates.

Impact
This would allow anyone with the document data for the anchor to `pre_commit` and block updating that document as a result. If this occurs, a document will not be able to complete an update that it would like to `commit`.

Feasibility
An attacker would be required to obtain a document and have knowledge of the $d_{next-img}$ field. This knowledge is intended to be kept to only the set of participants in the document. If the attacker is unable to uncover this preimage, then they will not be able to `pre_commit` to prevent state updates that are of concern to the owners of the document. However, if a counterparty finds a document disastifing to them at any point, they may choose to lock the anchor and continue to relock it after the pre-commit eviction time passes.

Technical Details
If an attacker is able to obtain a document in full, they could propose a `pre_commit` with the information of $d_{next-img}$. This would likely be a counterparty of the document but could be due to an accidental leak of the update's randomness. They could create a `pre_commit` for a specific `anchor_id or signing root,` and if any legitimate participants of the document wish to then update the anchor, they would find themselves blocked by the attackers pre-commits. Pre-commits last 800 (6 second) blocks and do not require state rent but do require a transaction fee.

Mitigation
It may be possible for legitimate owners to fork the document by adding some extra salt to a leaf such that they exclude a participant from a new document update. This is stated as a mitigation in the Protocol Limitations documentation. The documentation also notes that owners of the anchored document should be trusted business partners in the initial implementation to reduce the likelihood of this attack.

Status
The Centrifuge team has stated that they consider this to be a protocol limitation. In the protocol documentation, they note an assumption that counterparties are trusted and centralized business partners and, as a result, these parties are not expected to behave in malicious ways. In the event one of these parties acts maliciously and blocks a document update the document will be abandoned and a new version created.

If counterparties are decentralized, we consider this to be a security issue that will negatively impact participants of document updates and we strongly recommend that investigation into fraud proofs or validity proofs be conducted in order to help prevent malicious actions.

Verification

Unresolved.

Issue B: Fees Must Be Continually Updated to Avoid DoS Issues

Location

https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L121-L123

https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L153-L157

https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L253-L255

Synopsis

Anyone may iterate over random document data and flood a node with transactions to store pre-commit data, anchor data, or other computational or disk heavy transactions. To combat this, Centrifuge implements the transaction weights functionality provided by Substrate. In addition, Centrifuge has developed a specific state rent fee and eviction times that are an additional protection against state bloat from anchor data being committed with little cost.

All fees will be based on a variable amount of tokens that transaction initiators are required to pay to block proposers. This token will have a fluctuating conversion rate to fiat value that must be balanced enough to equal a fiat cost imposed on the proposers that prevents spamming of transactions. Given that the conversion rate could likely be volatile, particularly in the early blocks of the chain, this cost may not be adequately priced at all times and the amount of token required to cover this cost based on fees will be required to fluctuate in some way. In Ethereum, these fees are decided by the miner and how much work the miner is willing to take based on their assessment of the current value of the native token.

Impact

State storage and transactional computation will be forced on nodes. If this is done recursively at little cost as the value of the token used to calculate fees fluctuates, it could cause state storage to bloat and create potential DoS issues by overwhelming full nodes.

Feasibility

There are preventative measures to ensure that data is not stored for indefinite time periods and all transactions are weighted or incur some fee. This is only feasible if the fee is not priced correctly based on the value of the token that the fee is paid in.

Mitigation

As the value of the chain's currency fluctuates, we recommend continuing to investigate the economic burden placed on custom functions to ensure that the cost of these functions prevents unwanted spam.

Status

The Centrifuge team intends to use a centralized form of governance to adjust fees as the value of the native token fluctuates in the early stages of the chain. They are continuing to investigate other methods of pricing the native token such as oracles.

Verification

Partially Resolved.

Issue C: Document Consensus Is Not Enforced

Location

https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L177

Synopsis

The commit function does not check that the document root contains valid signatures of counterparties in the `signature root`.

Impact

Any participant in a document's state update can commit an update without signatures from counterparties. The whitepaper, [VPSW17], mentions that this is intended behavior for this early version.

Remediation

Documentation that explains that a future use will be made of the proof hash supplied to the commit function and that this is intended to be a feature can be found in the Protocol Limitations documentation that was provided to us by the Centrifuge team.

Status

The Centrifuge team considers unenforced document consensus to be a protocol limitation at this time. Any participant of a document may commit an update without the consent of other participants. This would not be a security issue if document updating is a centralized process that does not require consensus, however, if a decentralized update process is desired in the future then checking signatures and acting on the consensus of the document will be required.

Verification

Unresolved.

Issue D: Block Proposers May DoS Anchors

Location

https://github.com/centrifuge/centrifuge-chain/blob/master/runtime/src/anchor/mod.rs#L125

Synopsis

If anchor pre-commits are supplied to nodes via a blockchain transaction, it could be possible that a block proposer could witness a pre-commit transaction and create their own pre-commit for an anchor if they know the `anchor_id` of the next document update. This would block the user wanting to make a state update to a document.

Impact

Those anchored documents that a proposer decides to block will be unavailable until the pre-commit times out and another proposer correctly includes these transactions. Alternatively the document owners would need to recompute a new pre-commit root and have a non-malicious proposer include it.

Feasibility

The value gained from a proposer committing this action is not clear. A block proposer may not be able to identify valuable targets to censor and an incentive to do so would not come from the chains currency, but rather only potentially from the value of harassing the document owners. This would likely only happen in the case that a block proposer wants to damage the network without a direct reward, however there is no cost to a proposer for committing this action.

Mitigation

Begin research into a slashing condition at the consensus layer that will deter miners from committing this random behavior.

Status

The Centrifuge team has stated in their response that because there is no clear reward to performing this attack, that an attack will not take place as a result. Given that incentives to attack a blockchain are not always clear and there may be reasons that attackers would commit to attacks without an immediate or monetary reward, we believe that this should still be considered a security issue. We encourage continued monitoring for this attack, regardless of unknown incentives, and further investigation into slashing conditions to prevent (or minimize) any unknown reward a block proposer may receive for performing this attack.

Verification

Unresolved.

Suggestions

Suggestion 1: Use a Lint Program

Synopsis

While reading the code, it was noticeable that line indentations were not consistent. Running `cargo fmt` also returned a list of edits that would keep the code inline with Rust style guides. Proper formatting will help to ensure that the codebase is legible, does not contain suspicious constructs, and is less polluted, thus easier to maintain.

Mitigation

Use `cargo fmt`, `rustfmt` or a custom configuration with a `rustfmt.toml` to ensure the code is consistent.

Status

A `rustfmt.toml` file has been included and a pull request that includes linting has been added in order to adhere to Rust style guides.

Verification

Resolved.

Suggestion 2: Check for Vulnerable Crate Dependencies

Location

`sp-authority-discovery={git="https://github.com/paritytech/substrate.git", rev="ddb309ae7c70e5e51a60879af18819cf28be4a32"}` in file `cargo.toml`

Synopsis

A vulnerability exists in crate prost version 0.5.0 [RUSTSEC-2020-0002] where parsing a specially crafted message can result in a stack overflow. Since Centrifuge uses the static version "ddb309ae7c70e5e51a60879af18819cf28be4a32" of Substrate's `sp-authority-discovery`, this vulnerability will still exist once Substrate has resolved it.

Mitigation

Upgrade crate prost to a version >= 0.6.1 and periodically check dependencies for newly found vulnerabilities.

Status

A commit that updates dependencies, including Substrate where the upgraded crate prost version 0.6.1 is located, has been added to the code base.

Verification

Resolved.

Suggestion 3: Document A Function That Is Only Used For Testnets

Location

Testnet function `get_authority_keys_from_seed(&str)` in file `chain_spec.rs`

Synopsis

Substrate's staking module introduces account abstractions that help keep funds as secure as possible, however function `get_authority_keys_from_seed(&str)` generates all keys from the same seed, which leads to a situation where the session keys `BabeID`, `ImOnlineId` and `AuthorityDiscoveryId` are identical to the controller account keys. This might be acceptable in a Testnet environment, however, it is not recommended in Mainnet, and according to the Substrate documentation. If one session key is compromised, the attacker is able to both generate slashable behavior and gain access to the controller account.

In addition, if an attacker is able to guess the seed for one of those keys, they may also compromise the stash account.

Mitigation

A testnet implementation might not need any remediation. However, when it comes to mainnet, a Centrifuge specific adaptation of standards like BIP32 or EIP2333 should be used to securely derive all required keys from a single seed.

Document that this function is only intended to be used on testnets or remove it before mainnet release and move testnet functionality elsewhere.

Status

A commit with a comment indicating that this function is for testnet only has been added to the codebase.

Verification

Resolved.

Suggestion 4: Expand Documentation Coverage to Include Future Functionality

Synopsis

Since it is expected that the functionality of the code base will be expanded in the future to include all parts of the specification, [VPSW17], we suggest providing sufficient documentation to keep track of all functionality provided and functionality not yet provided but planned for the future.

Mitigation

Document current functionalities as extensively as needed according to `rustdoc` as to not overlook parts of the desired future implementation of the specification.

Status

Further documentation can be found in the developer site that covers the Centrifuge team's current approach to realizing the concepts and functionality in [VPSW17]. We encourage the team to include documentation on any planned upgrades such as fee governance and oracles.

Verification

Partially Resolved.

Suggestion 5: Optimized Merkle Proof Checker Depends on the Order of Proofs

Location

Function `validate_proofs(H256,&Vec<Proof>, [H256; 3]) -> bool` in file `proofs.rs`

Synopsis

The `validate_proofs()` function optimizes Merkle proof checking. However, this enables a malicious user to just send a pruned proof, p, that does not contain all data required for a proper Merkle proof. Depending on the position of p in `Vec<Proof>`, `validate_proofs()` might give different outcomes and depending on the use of `validate_proofs()` if used outside of the Centrifuge client. In order for this to become an issue, first a proof supplier must compute a valid Merkle proof `p(A)` for some field of a document and prune the proof (e.g. delete (situation specific) branches and the Merkle root from the proof). Then they send the pruned proof `p(A)'` to the validating function.

In the event that the pruned part of proof `p(A)` is contained in another proof `p(B)`, then any user relying on this function will accept `p(A)'`, if in their execution of `validate_proofs()`, `p(B)` appears before `p(A)'` in `Vec<Proofs>`, while no user will accept `p(A)'` if from their perspective, there is no such `p(B)` in `Vec<Proofs>` before `p(A)'`

We do not believe the issue can be used to validate false leaves (e.g. false field entries), the outcome of function `validate_proofs()` is order dependent if pruned proofs appear. Within the Centrifuge Chain, the use of this proof function will only allow one set of proofs to be presented such that incoherent state is not an issue.

Remediation

The following could remediate the vulnerability:

- Implementing an invariant preordering of the proofs
- An additional check for complete Merkle proof data, or

- A conventional non-optimized Merkle proof checker (most convenient)

Document this behavior to prevent any other implementation that uses this batched merkle prover from mistakenly reaching incoherent state in their system.

Status

The Centrifuge team has stated that, due to performance being critical and given that the batched merkle proof verifier can not verify false claims in the Centrifuge Chain, they do not intend to alter the functionality of this proof verifier at this time.

Verification

Unresolved.

Recommendations

We recommend that the *Issues* and *Suggestions* stated above are addressed as soon as possible and followed up with verification by the auditing team.

Regarding the document consensus enforcement and the potential for malicious behavior, both in furtherance of known and unknown incentives, we suggest that the concerns highlighted in this report be given further consideration as the project progresses. Incentive mechanism design is both complex and reactive to use cases as circumstances change. As development and use of the project progresses, we encourage proactive investigation into these security risks and re-evaluation of the approach as various priorities are managed.

Since large parts of the whitepaper specifications have yet to be implemented into the codebase, we also recommend that another audit is conducted in order to comprehensively check for the correctness of the implementation and to identify potential vulnerabilities that might currently exist in those parts of the specification, given that the specification was out of scope for this review.

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.