Chia Consensus and Proof of Space Security Assessment

Chia Network Inc

February 1, 2021

Prepared for

Bill Blanke
Bram Cohen
Gene Hoffman, Jr.
Straya Markovic
Mariano Sorgente

Prepared by

Ava Howell
Aleksandar Kircanski
Ephrayim Kishko

Feedback on this project?
https://my.nccgroup.com/feedback/3d57d875-0446-4bfb-a440-477790182b9d

Executive Summary

Synopsis

In December 2020 and January 2021, Chia Network Inc. engaged NCC Group Crypto Services to conduct a security assessment of the Chia blockchain implementation. Chia's goal is to provide a more equitable coin with a variety of institutional and non-institutional usages. It leverages a novel concept called Proof of Space and Time to offer different incentives when compared to well-known Proof of Work and Proof of Stake systems.

Source code was provided and the Chia team supported NCC Group over a dedicated Keybase channel. A number of conference calls were held, including pre-kickoff, kickoff, two status updates and a readout. NCC Group received valuable feedback during the course of the engagement helping it drive the review in a meaningful direction. Overall, 30 consultant days were spent on the project.

Scope

The scope of the engagement included:

- https://github.com/Chia-Network/chia-blockchain: Python implementation of the Chia consensus protocol, together with other aspects encompassing a blockchain node, such as a wallet, cryptography, etc
- https://github.com/Chia-Network/chiapos: C++ implementation of the Proof of Space mechanism underlying Chia's consensus

Apart from its usage inside `chia-blockchain`, the Chia's VDF implementation was not a part of this review.

Testing Methodology

Two important pieces of documentation were used during the review:

- Chia Proof of Space Construction, v1.1: a spec for the PoS implementation, provided as a PDF
- Chia Consensus: descriptive explanation of the consensus protocol, provided as a google document

The testing strategy was manual code review, while matching the code to the documentation and using a baseline set of issues likely to arise in blockchain as the starting point. The testing methodology also included some dynamic testing using a dedicated Digital Ocean instance mostly to validate assumptions derived from reading the code.

Key findings

- Ambiguous Object Deserialization Scheme Leads to DoS: Ambiguous (de)serialization of blocks, transactions and other data structures is an unwanted property in blockchains. Its consequences in terms of security are somewhat hazy, however, this finding points out that an ambiguity in deserialization leads to a DoS attack.
- Peer Ejection by Web Socket Replacement: An arbitrary web socket connection between any two nodes on the network can be disconnected by an attacker.
- Unimplemented Previous Generator Root Validation: Failure to correctly validate the `previous_generators_root` field of `TransactionsInfo` may lead to broken security assumptions.
- Unimplemented End Of Sub Slot Bundle Validation: An attacker may advertise bogus end of sub slot and have nodes fill their caches with invalid information. This can likely be abused to impede the network's consensus progression.
- Excess Storage Denial of Service Vectors: A misbehaving node may upload excess amounts of data to legitimate nodes on the network, impeding their normal functioning capabilities.

Proof of Space security assessment notes are provided in Section Proofs of Space Implementation Notes on page 4.

Notes and Observations

When it comes to consensus, NCC Group consultants noted two unimplemented consensus controls which are likely of high importance (see finding NCC-CHIA001-011 on page 14 and finding NCC-CHIA001-007 on page 11). Due to TODOs in the code, it is assumed that at least at some point in time, these issues were known to the Chia Team.

In terms of network processing and node interaction, a number of issues was identified, such as Denial of Service via deserialization, peer ejection and `TransactionAck` message spoofing. Overall, it makes sense to assume the blockchain network environment is a highly adversarial and explore misbehaving scenarios during networking code development.

Async-style programming is prone to race conditions. One example is discussed in finding NCC-CHIA001-012 on page 21. Consider spending engineering time devoted to further exploration of how this type of issues can put a node in an inconsistent consensus state. In terms of issues common to Python such as typing problems, several borderline issues were identified, but these did not lead to practically exploitable bugs.

As for private Chia node communication (such as between Full Nodes and private Time Lord nodes), the usage of TLS was reviewed and no issues were found. Regardless, TLS in this context may be replaced by the Noise protocol.[1] In particular, Chia intranets likely do not need crypto agility, X.509 processing capability and other unnecessary functionalities offered by TLS stacks.

When it comes to future reviews, consider reviewing smart contract capabilities together with specific smart contracts and coin features (e.g., colored coins). Since consensus logic is rather complex, it makes sense to dedicate more time on logical consensus review (e.g., block and VDF parameter validation and interaction).

[1] http://www.noiseprotocol.org/noise.html

The Chia Proof-of-Space protocol provides a core mechanism around which consensus can be built. It is described principally in *Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space*[2] and the *Chia Proof of Space Construction*[3] papers.

During the course of the security assessment, the correctness and safety of the Proof of Space implementation used in the Chia full-node implementation[4] was examined. No high severity vulnerabilities were discovered in the plotting, proving, or the crucial verification components. The implementations of the underlying symmetric cryptography used in the Chia Proof of Space construction, ChaCha8 and BLAKE3, were also examined and found to conform with the reference implementations.

Implementation Notes

The following notes are not findings per se, however they are aspects of the `chiapos` implementation that were noted during review.

`chiapos` API Does Not Enforce `k` Parameter Bounds

It was noted that the API exposed by `chiapos`, both through the command-line interface as well as the python bindings, does not enforce the integer bounds on the size parameter k during proof verification. This may lead to incorrect usage of the `chiapos` library. In the case of the principal user of `chiapos`, the Chia python blockchain implementation, it was found that the check around k was performed by the caller:

```
# ...snip...
    def verify_and_get_quality_string(
        self,
        constants: ConsensusConstants,
        original_challenge_hash: bytes32,
        signage_point: bytes32,
    ) -> Optional[bytes32]:
        if (self.pool_public_key is None) and (self.pool_contract_puzzle_hash is None):
            return None
        if (self.pool_public_key is not None) and (self.pool_contract_puzzle_hash is not None):
            return None
        if self.size < constants.MIN_PLOT_SIZE:
            return None
        if self.size > constants.MAX_PLOT_SIZE:
            return None
# ...snip...
```

However, this design breaks with the principle of *misuse-resistance*: that is, it should be difficult for a caller to use a security-oriented API incorrectly. As such, it is recommended to move the validation of k to `chiapos` so that the responsibility of validating k is removed from callers.

Use of Uninitialized Array Memory

The `chiapos` implementation is written in C++, which does not provide any memory safety guarantees. As such, time was taken in attempting to identify any potential memory unsafety which could be exploited by an attacker in order to exfiltrate system information, cede control flow of the Chia application, or perform other undefined behavior. Part of the analysis performed to this end was to perform fuzzing, a type of automated mutation testing, of the parsing of the proof data π. During the run of the fuzzer, no crashes or other unsafety were noted, however the use of uninitialized memory was detected using LLVM/Clang's `MemorySanitizer`:

```
    // Performs one evaluation of the F function on input L of k bits.
    inline Bits CalculateF(const Bits& L) const
```

[2] https://eprint.iacr.org/2017/893.pdf
[3] https://www.chia.net/assets/Chia_Proof_of_Space_Construction_v1.1.pdf
[4] https://github.com/Chia-Network/chiapos

```
    {
// ...snip...
        uint8_t ciphertext_bytes[kF1BlockSizeBits / 8];// NOTE: uninitialized memory
        Bits output_bits;

        // This counter is used to initialize words 12 and 13 of ChaCha8
        // initial state (4x4 matrix of 32-bit words). This is similar to
        // encrypting plaintext at a given offset, but we have no
        // plaintext, so no XORing at the end.
        chacha8_get_keystream(&this->enc_ctx_, counter, 1, ciphertext_bytes);
        Bits ciphertext0(ciphertext_bytes, block_size_bits / 8, block_size_bits);
// ...snip...
```

Note that the `ciphertext_bytes` array is not explicitly initialized (e.g., with `... = { };`), therefore it contains undefined values. In general, operating on such values can be dangerous as they can contain bytes from sensitive system memory, or attacker controlled memory. In the case of `chiapos`, the values are used, however they are simply immediately overwritten as output in `chacha8_get_keystream`:

```
// ...snip...
        U32TO8_LITTLE(c + 0, x0);
        U32TO8_LITTLE(c + 4, x1);
        U32TO8_LITTLE(c + 8, x2);
        U32TO8_LITTLE(c + 12, x3);
        U32TO8_LITTLE(c + 16, x4);
        U32TO8_LITTLE(c + 20, x5);
        U32TO8_LITTLE(c + 24, x6);
        U32TO8_LITTLE(c + 28, x7);
        U32TO8_LITTLE(c + 32, x8);
        U32TO8_LITTLE(c + 36, x9);
        U32TO8_LITTLE(c + 40, x10);
        U32TO8_LITTLE(c + 44, x11);
        U32TO8_LITTLE(c + 48, x12);
        U32TO8_LITTLE(c + 52, x13);
        U32TO8_LITTLE(c + 56, x14);
        U32TO8_LITTLE(c + 60, x15);
// ...snip...
```

As such, there should be no practical security risk arising from this use of uninitialized memory. However, to ensure robustness against potential future code changes and to conform with best practices, NCC Group recommends explicitly initializing `ciphertext_bytes`.

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 25.

Title	Status	ID	Risk
Peer Ejection by Web Socket Replacement	Reported	002	High
Ambiguous Object Deserialization Scheme Leads to DoS	Reported	004	High
Unimplemented End Of Sub Slot Bundle Validation	Reported	007	High
Excess Storage Denial of Service Vectors	Reported	010	Medium
Unimplemented Previous Generator Root Validation	New	011	Medium
Chia Node Private Key File Permissions	Reported	003	Low
P2P Message Response Object Mismatches	Reported	005	Low
Chia Node Private Key File Persists on Filesystem after Uninstall	Reported	006	Low
Private Key and Mnemonic Secret Linger in Memory After Key Deletion	Reported	009	Low
Race Condition via Fake `TransactionAck` Messages In Wallet Nodes	New	012	Low
Data Types not Checked on Payload IDs and Function Names	Reported	001	Informational

Finding	**Peer Ejection by Web Socket Replacement**
Risk	**High** Impact: High, Exploitability: High
Identifier	NCC-CHIA001-002
Status	Reported
Category	Denial of Service
Component	chia-blockchain
Location	https://github.com/chia-network/chia-blockchain/blob/76729e64/src/server/server.py#L183
Impact	An arbitrary web socket connection between any two nodes on the network can be disconnected by an attacker.

Description

Consensus-related communication between the nodes on the Chia network runs over web sockets. The first exchanged message between two newly connected nodes is the handshake message. Apart from network ID, protocol version and other information, the message includes the `node_id` `bytes32` field. This finding evaluates what is the consequence of an attacker's ability to freely choose the `node_id` value.

Consider what happens when a node's listener receives the web socket connection. A `WSChiaConnection` object is created and the handshake is performed. An entry is added to the `all_connections` dict, which holds all of the connections indexed by the `peer_node_id` value. As mentioned, this field is attacker/peer controlled — it comes from the remote peer's handshake message.

When the newly created `WSChiaConnection` object is created, it needs to be placed inside the `all_connections` dict:

```
async def connection_added(self, connection: WSChiaConnection,
 → on_connect=None):
    if connection.peer_node_id in self.all_connections:
        con = self.all_connections[connection.peer_node_id]
        await con.close()
    self.all_connections[connection.peer_node_id] = connection
    if connection.connection_type is not None:
        self.connection_by_type[connection.connection_type][connection.
         → peer_node_id] = connection
        if on_connect is not None:
            await on_connect(connection)
    else:
        self.log.error(
         → f"Invalid connection type for connection {connection}")
```

If the incoming peer claims an existing `peer_node_id`, the original connection is closed. Nothing appears to prevent participants on the network from learning other nodes' IDs and as such an attacker on the network should be able to disconnect arbitrary web socket links between any two peers.

Note: The handshake parameters such as `network_id` are not validated nor ensured to match between connecting client. It is assumed this is the short-term development roadmap.

Recommendation

If a connection between the two peers is broken, peers may end up with stale, non-functional connections in their connection store. This could happen due to a network connection problem or one peer process being killed, resulting in the client disconnecting without the according web socket `CLOSE` message. The only way to deal with this issue is to occasionally try to write to web sockets in the store and consider the connection as broken if the write does not succeed. As such, to deal with stale/broken connections, web socket heartbeat[5] should be used and stale connections should be pruned accordingly.

Currently, the notion of a peer node ID is used in methods such as `send_to_others` and `send_to_all_except`, in order to identify a subset of nodes that messages should be sent to. It should be noted that such use cases could be implemented even if the peer node ID would not originate from the actual sender. Consider swapping out peer-originating node IDs with internal peer IDs, which are just random numbers generated by the node. Alternatively, peer node IDs could be bound by TLS certificates, which is assumed to not be doable as not all connections are meant to be authenticated.

To summarize, the recommendation is to implement a web socket heartbeat in order to keep the connection store fresh and use internally generated (as opposed to client-provided) node IDs for internal peer handling.

[5] Sections 5.5.2 and 5.5.3 in https://tools.ietf.org/html/rfc6455.

Finding **Ambiguous Object Deserialization Scheme Leads to DoS**

Risk **High** Impact: High, Exploitability: High

Identifier NCC-CHIA001-004

Status Reported

Category Other

Component chia-blockchain

Location https://github.com/chia-network/chia-blockchain/blob/76729e64285c85d3bfcaf9a1225d848a86c4d844/src/util/streamable.py#L129

Impact Ambiguous (de)serialization of blocks, transactions and other data structures is an unwanted property in blockchains. Its consequences in terms of security are somewhat hazy, however, this finding points out that an ambiguity in deserialization leads to a DoS attack.

Description In general, the mapping between object attribute content and its serialized representations should be a one-to-one mapping. This rule could be violated in two ways (1) multiple objects serialize to a colliding data blob and (2) multiple data blobs de-serialize to a single object. While violation (1) would be a serious one (as it would mean that a signature validates multiple objects), this finding discusses (2), which shouldn't exist either as it can lead to unforeseen issues.

To encode and decode objects such as transactions and blocks, Chia relies on a custom (de)serialization scheme, defined in `util/streamable.py`. Data to become the content of an object's attribute is parsed using the following function:

```
def parse_one_item(cls: Type[cls.__name__], f_type: Type, f: BinaryIO):
 ➜ # type: ignore
    inner_type: Type
    if is_type_List(f_type):
        inner_type = get_args(f_type)[0]
        full_list: List[inner_type] = []  # type: ignore
        # wjb assert inner_type != get_args(List)[0]  # type: ignore
        list_size: uint32 = uint32(int.from_bytes(f.read(4), "big"))
        for list_index in range(list_size):
            full_list.append(cls.parse_one_item(inner_type, f))
             ➜ # type: ignore
        return full_list
    if is_type_SpecificOptional(f_type):
        inner_type = get_args(f_type)[0]
        is_present: bool = f.read(1) == bytes([1])
        if is_present:
            return cls.parse_one_item(inner_type, f)  # type: ignore
        else:
            return None
```

Suppose the entry the parser is expecting to read is an `Optional[uint32]`. The second `if` condition is triggered and an `f.read(1)` will happen to read out whether the optional value is provided or not. While this is correct, it should be noted that the `f.read(1)` will not throw an exception in the case of EOF. In that case, the `is_present` field will simply be `false` and the `if is_present` condition will not check out.

In other words, having or not having a zero byte to denote that there's no `Optional` field doesn't make a difference. If at the end of the stream, the parser is expecting an `Optional` value, removing or adding a zero suffix does not change the final object (in both cases, the `Optional` field at the end will be a `None`). As such, there can exist multiple data blobs that get de-serialized to the same object. A similar issue exists with some of the other types treated by the `parse_one_item` function, such as `bool`.

Somewhat unexpectedly this property can be converted into a DoS vector. Consider the following class:

```
@dataclass(frozen=True)
@streamable
class SubEpochChallengeSegment(Streamable):
    sub_epoch_n: uint32
    sub_slots: List[SubSlotData]
```

The `SubSlotData` consists only of `Optional` fields:

```
@dataclass(frozen=True)
@streamable
class SubSlotData(Streamable):
    # if infused
    proof_of_space: Optional[ProofOfSpace]
    # [ ... SNIP ...]
    rc_slot_end_info: Optional[VDFInfo]
```

Suppose the parser is deserializing the `SubEpochChallengeSegment` object. In serialized form, the `sub_slots: List[SubSlotsData]` field starts with a 4-byte length, followed by a number of `Optional` fields. The length may be a large number such as $2^{32} - 1$. Due to the property explained above, the byte string can well end there and there's no need for $2^{32} - 1$ entries to follow in the data blob. As such, for the cost of sending a short message, an attacker got the node to perform a large number of steps and also consume a large amount of memory.

Reproduction Steps Run the following program:

```
from dataclasses import dataclass
from typing import List, Optional

from src.util.streamable import Streamable, streamable

from src.types.weight_proof import SubEpochChallengeSegment

x = SubEpochChallengeSegment(3, [])
print(x)
print(bytes(x))

dos = SubEpochChallengeSegment.from_bytes(b'\x00\x00\x00\x03\xff\xff\xff\xff')
```

Recommendation Bail from the `parse_one_item` method if the expected bytes aren't there. In addition, ensure that the data blob consumed by the parser is not followed by any additional bytes. In general, ensure a one-to-one mapping between object content and its serialized form.

Finding Unimplemented End Of Sub Slot Bundle Validation

Risk **High** Impact: High, Exploitability: High

Identifier NCC-CHIA001-007

Status Reported

Category Data Validation

Component chia-blockchain

Location https://github.com/chia-network/chia-blockchain/blob/f50a372b509d42bfd63d20de3abf985d1294f22f/src/full_node/full_node_store.py#L175

Impact An attacker may advertise bogus end of sub slot and have nodes fill their caches with invalid information. This can likely be abused to impede the network's consensus progression.

Description The Chia network's P2P communication includes advertising new signage points using the `new_signage_point_or_end_of_subslot` API endpoint.[6] If the receiving node deems appropriate, it requests the actual signage point based on the advertised data. In some cases, instead of the signage point, the receiving node will request the end of sub slot bundle. This happens if the node does not have the end of sub slot information for the advertised signage point, or if the previous sub slot information is unknown, see `full_node_api.py:354`.

The requested sub slot information is ingested through the `respond_end_of_sub_slot` endpoint and takes an `EndOfSubSlotBundle` as a parameter:

```
class EndOfSubSlotBundle(Streamable):
    challenge_chain: ChallengeChainSubSlot
    infused_challenge_chain: Optional[InfusedChallengeChainSubSlot]
    reward_chain: RewardChainSubSlot
    proofs: SubSlotProofs
```

A consequence of calling `respond_end_of_subslot` is the creation of a new subslot entry, see the `new_finished_sub_slot` function:

```
    def new_finished_sub_slot(
        self,
        eos: EndOfSubSlotBundle,
        sub_blocks: Dict[bytes32, SubBlockRecord],
        peak: Optional[SubBlockRecord],
    ) -> Optional[List[timelord_protocol.NewInfusionPointVDF]]:
        """
        Returns false if not added. Returns a list if added. The list contains al
 ➜  l infusion points that depended
        on this sub slot
        TODO: do full validation here
        """

        # [...SNIP...]
```

[6] Block production in the Chia blockchain happens inside sub-slots. Each sub-slot in the challenge and reward chains is divided into `SIGNAGE_POINTS_PER_SUB_SLOT` smaller VDFs and each signage point records these intermediary VDF outputs. A related notion is the `EndOfSubSlotBundle` which records the VDF state of the three chains at sub slot endpoints.

```
        if eos.challenge_chain.challenge_chain_end_of_slot_vdf.challenge !=
        ➜  last_slot_ch:
            # This slot does not append to our next slot
            # This prevent other peers from appending fake VDFs to our cache
            return None

        # [...SNIP...]

        self.finished_sub_slots.append((eos, [None] *
        ➜  self.constants.NUM_SPS_SUB_SLOT, total_iters))
```

While the `new_finished_sub_slot` method validates whether the three chain's VDF challenges inside the end of sub slot bundle lean on the ongoing context, various other end of sub slot parameters are not validated. This includes VDF proofs, VDF number of iterations and parameters specific to the challenge chain.

The end of sub slot entries inside `finished_sub_slots` participate in several consensus-related code paths. For instance, consider the `full_node_store.py:new_signage_point` method, used to process new signage points. It iterates through the known end of sub slot entries, identifies the one corresponding to the processed signage point and relies on the claimed end of sub slot iteration number. Since this number has not been necessarily validated, the consensus-related decision made by the `new_signage_point` function may be invalid.

Recommendation Address the TODOs in `new_finished_sub_slot` function by fully validating the end of sub slot information. Commented out code validates the VDF proofs inside the end of sub slot data snippet, however, this does not appear to be enough as not all the three chains are validated to lean on the last known end of sub slot entry.

Finding	**Excess Storage Denial of Service Vectors**
Risk	**Medium** Impact: Medium, Exploitability: Medium
Identifier	NCC-CHIA001-010
Status	Reported
Category	Data Validation
Component	chia-blockchain
Location	https://github.com/chia-network/chia-blockchain/blob/f50a372b509d42bfd63d20de3abf985d1294f22f/src/full_node/full_node.py#L956
Impact	A misbehaving node may upload excess amounts of data to legitimate nodes on the network, impeding their normal functioning capabilities.

Description

There have been several memory/storage exhaustion Denial of Service vectors in Bitcoin. Such vectors relied on lack of storage size controls around orphan blocks,[7] transaction mempool,[8] orphan transactions,[9] etc. Memory stores that ingest data without any cost for the attacker are candidates for such storage exhaustion vectors. An additional condition required is a lack of an effective memory store item eviction strategy.

The Chia full node implementation keeps a number of caches during consensus processing:

```
    def __init__(self):
        self.candidate_blocks = {}
        self.seen_unfinished_blocks = set()
        self.disconnected_blocks = {}
        self.unfinished_blocks = {}
        self.finished_sub_slots = []
        self.future_eos_cache = {}
        self.future_sp_cache = {}
        self.future_ip_cache = {}
```

The last three caches do not appear to implement an eviction policy and can be added for free (with the exception of `future_sp_cache` which is not yet fully implemented). For example, processing new infusion point VDFs includes storing them in the `full_node_store.future_ip_cache` map, in the case they don't refer to a known previous block. The new infusion point can store byte strings of arbitrary length (inside VDF proofs) and is not validated before being used. Similar goes for `full_node_store.future_eos_cache` and `future_sp_cache`.

Recommendation

Implement an overall size limit on the mentioned caches, since just limiting the number of entries won't be sufficient. If the size threshold is passed, consider ejecting a random element from the store, or a chosen minimal element strategy (where the definition of "minimal" is chosen accordingly, for instance, the most stale element).

[7] https://github.com/bitcoin/bitcoin/commit/bbde1e99c89392
[8] https://www.reddit.com/r/Bitcoin/comments/3ny3tw/with_a_1gb_mempool_1000_nodes_are_now_down/
[9] https://en.bitcoin.it/wiki/CVE-2012-3789

Finding	**Unimplemented Previous Generator Root Validation**
Risk	**Medium** Impact: High, Exploitability: Undetermined
Identifier	NCC-CHIA001-011
Status	New
Category	Data Validation
Component	chia-blockchain
Location	https://github.com/Chia-Network/chia-blockchain/blob/b82f3ba8a2953de12bddf5c5d6a33e443b51bc8b/src/consensus/block_body_validation.py#L90 (`validate_block_body()`)
Impact	Failure to correctly validate the `previous_generators_root` field of `TransactionsInfo` may lead to broken security assumptions.

Description

A Chia `FullBlock` contains an optional field, `transactions_info`, which contains the reward chain foliage data. The `TransactionsData` struct has the following structure:

```
@dataclass(frozen=True)
@streamable
class TransactionsInfo(Streamable):
    # Information that goes along with each transaction block
    previous_generators_root: bytes32  # This needs to be a tree hash
    generator_root: bytes32  # This needs to be a tree hash
    aggregated_signature: G2Element
    fees: uint64  # This only includes user fees, not block rewards
    cost: uint64
    reward_claims_incorporated: List[Coin]
```

These critical fields, such as the `aggregated_signature` and `generator_root`, are validated during full block body validation. However, `previous_generators_root` is not validated:

```

    # 5. The prev generators root must be valid
    # TODO(straya): implement prev generators

    # 6. The generator root must be the tree-
     ➜ hash of the generator (or zeroes if no generator)
    if block.transactions_generator is not None:
        if block.transactions_generator.get_tree_hash() !=
         ➜ block.transactions_info.generator_root:
            return Err.INVALID_TRANSACTIONS_GENERATOR_ROOT
    else:
        if block.transactions_info.generator_root != bytes([0] * 32):
            return Err.INVALID_TRANSACTIONS_GENERATOR_ROOT
```

Failing to validate that `previous_generators_root` in the correct previous tree hash may lead to broken security assumptions in the foliage chain or other systems that rely on the correctness of `previous_generators_root`. NCC Group noted that `previous_generators_root` is not currently used in consensus logic, which is likely why the validation logic is unimplemented.

Recommendation

As the TODO notes, ensure correct validation of `previous_generators_root` is implemented when `previous_generators_root` is implemented.

Finding	**Chia Node Private Key File Permissions**
Risk	**Low** Impact: Medium, Exploitability: High
Identifier	NCC-CHIA001-003
Status	Reported
Category	Configuration
Component	chia-blockchain
Location	`~/.chia/beta-1.0b19.dev1/config/trusted.key`
Impact	The overall security posture of the Chia node is weakened. A cross-user private key read is possible.

Description Upon initialization, the Chia node generates a private key used for authentication purposes. The file permissions include a flag that allows all users on the Unix system to read the file:

```
~/.chia/beta-1.0b19.dev1/config$ ls -l
total 16
-rw-rw-r-- 1 user user 6921 Jan  1 09:20 config.yaml
-rw-rw-r-- 1 user user 1038 Jan  1 09:20 trusted.crt
-rw-rw-r-- 1 user user 1675 Jan  1 09:20 trusted.key
```

The `trusted.key` file contains raw key material.

Recommendation Set the appropriate `umask` before creating the file (see the `initialize_ssl` function in `init.py`). Consider enforcing the correct file policy by bailing if the `trusted.key` file allows world-reads.

Finding	**P2P Message Response Object Mismatches**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-CHIA001-005
Status	Reported
Category	Data Validation
Component	chia-blockchain
Location	https://github.com/chia-network/chia-blockchain/blob/f50a372b509d42bfd63d20de3abf985d1294f22f/src/wallet/wallet_node.py#L413 https://github.com/chia-network/chia-blockchain/blob/f50a372b509d42bfd63d20de3abf985d1294f22f/src/full_node/full_node.py#L438
Impact	A misbehaving node on the network can respond to P2P messages with messages that deserialize to invalid object types. This will not be detected and cause exceptions or invalid logic execution in the sending client.

Description

The P2P message exchange workflows include a scenario where a node sends a request and waits for the receiving node's reply. This is handled by the `create_request` function. The request and reply messages are tied together by request IDs. The raw response message is in the `result` variable in the code snippet (see `ws_connection.py`):

```
    def __getattr__(self, attr_name: str):
        # TODO KWARGS
        async def invoke(*args, **kwargs):
            attribute = getattr(class_for_type(self.connection_type), attr_name,
            ➜  None)
            if attribute is None:
                raise AttributeError(f"bad attribute {attr_name}")

            msg = Message(attr_name, args[0])

            result = await self.create_request(msg, 60)

            if result is not None:
                ret_attr = getattr(class_for_type(self.local_type),
                ➜  result.function, None)

                req_annotations = ret_attr.__annotations__
                req = None
                for key in req_annotations:
                    if key == "return" or key == "peer":
                        continue
                    else:
                        req = req_annotations[key]
                assert req is not None
                result = req(**result.data)
            return result
```

The raw response is converted to a type that's specified by the `result.function` name from the response. Conceivably, the responder may set `result.function` to an arbitrary API call and get the resulting object to be an arbitrary type allowed by the API list of functions.

As such, in the request-reply workflow, it is necessary for the client code to validate the type of the response object. This is done fairly consistently, however, a few exceptions are noted in this finding.

As specified by `full_node.py`:

```
            for peer in peers_with_peak:
                if peer.closed:
                    to_remove.append(peer)
                    continue
                response = await peer.request_sub_blocks(request)
                if response is None:
                    peers_to_remove.append(peer)
                    continue
                if isinstance(response, RejectSubBlocks):
                    peers_to_remove.append(peer)
                    continue
                elif isinstance(response, RespondSubBlocks):
                    success = await
                     ➜ self.receive_sub_block_batch(response.sub_blocks, peer)
                    if success is False:
                        await peer.close()
                        continue
                    else:
                        batch_added = True
                        break

            for peer in to_remove:
                peers_with_peak.remove(peer)
```

The intent in the code snippet is to remove peers with invalid responses, however, a removal will not happen if an object is neither `None`, `RejectSubBlocks` nor `RespondSubBlocks`.

See also `wallet_node.py`:

```
                weight_request = RequestProofOfWeight(header_block.
                 ➜  sub_block_height, header_block.header_hash)
                weight_proof_response: RespondProofOfWeight = await peer.
                 ➜  request_proof_of_weight(weight_request)
                if weight_proof_response is None:
                    return
                weight_proof = weight_proof_response.wp
                if self.wallet_state_manager is None:
                    return
                valid, fork_point = self.wallet_state_manager.
                 ➜  weight_proof_handler.validate_weight_proof(weight_proof)
```

If the object crafted from the response happens to have `wp` attribute, it will be passed to validation logic, even though the object doesn't necessarily have to be of correct type.

Recommendation Consider extending the `create_request` API to specify allowed return types. This would make the code more robust when it comes to handling unexpected objects received from the responder.

Finding	**Chia Node Private Key File Persists on Filesystem after Uninstall**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-CHIA001-006
Status	Reported
Category	Data Exposure
Component	chia-blockchain
Location	`C:\Users\<USERNAME>\.chia\beta-1.0b21\config\trusted.key`
Impact	Sensitive key material such as a private key is still available on the file system after user uninstall. An incomplete uninstall process may lead to a false sense of security.
Description	Analyzing the uninstall process in Windows for the Chia blockchain application showed that the `trusted.key` file containing a private key is still persisting on the file system after uninstall. Furthermore, this was also evident based on the `C:\Users\<USERNAME>\.chia\beta-1.0b21\` directory available after uninstall.
Recommendation	Application uninstall should not leave any unintended/sensitive files on the file system.

Finding	**Private Key and Mnemonic Secret Linger in Memory After Key Deletion**
Risk	**Low** Impact: High, Exploitability: Low
Identifier	NCC-CHIA001-009
Status	Reported
Category	Data Exposure
Component	chia-blockchain
Location	start_wallet.exe
Impact	Attackers with access to process memory can see sensitive wallet information after users explicitly delete their key.
Description	Regular application usage showed that the secret wallet information was still accessible after the user deleted their key. Secret information such as the private key and the mnemonic were available in memory dumps which can then be re-used to recover the wallet. This became evident based on the `start_wallet.exe` process memory dump

start_wallet.exe (18888) (0x1a3a949d000 - 0x1a3a9625000)


```
000002d0 00 00 00 00 00 00 00 00 2e f5 1b e7 69 01 00 80 ............i...
000002e0 00 00 00 00 00 00 00 00 f0 d9 e9 5b fd 7f 00 00 ...........[....
000002f0 57 02 00 00 00 00 00 00 ff ff ff ff ff ff ff ff W...............
00000300 7b 22 61 63 6b 22 3a 20 74 72 75 65 2c 20 22 63 {"ack": true, "c
00000310 6f 6d 6d 61 6e 64 22 3a 20 22 67 65 74 5f 70 72 ommand": "get_pr
00000320 69 76 61 74 65 5f 6b 65 79 22 2c 20 22 64 61 74 ivate_key", "dat
00000330 61 22 3a 20 7b 22 70 72 69 76 61 74 65 5f 6b 65 a": {"private_ke
00000340 79 22 3a 20 7b 22 66 69 6e 67 65 72 70 72 69 6e y": {"fingerprin
00000350 74 22 3a 20 31 38 30 38 31 32 35 38 2c 20 22 70 t": 18081258, "p
00000360 6b 22 3a 20 22 61 37 37 63 38 35 63 32 65 61 39 k": "a77c85c2ea9
00000370 36 64 66 36 32 34 36 64 38 66 36 65 64 61 33 64 6df6246d8f6eda3d
00000380 34 65 64 65 66 66 30 31 34 30 61 33 31 30 35 61 4edeff0140a3105a
00000390 30 31 32 30 39 62 66 61 61 30 64 30 62 38 64 38 01209bfaa0d0b8d8
000003a0 65 65 61 30 63 63 64 31 65 34 36 37 37 39 31 31 eea0ccd1e4677911
000003b0 61 64 35 37 34 64 63 66 31 64 37 36 63 36 63 30 ad574dcf1d76c6c0
000003c0 30 32 65 61 63 22 2c 20 22 73 65 65 64 22 3a 20 02eac", "seed": 
000003d0 22 6d 65 6c 74 20 74 6f 75 72 69 73 74 20 65 78 "melt tourist ex
000003e0 63 6c 75 64 65 20 61 67 72 65 65 20 62 65 61 75 clude agree beau
000003f0 74 79 20 74 6f 64 64 6c 65 72 20 70 75 72 63 68 ty toddler purch
00000400 61 73 65 20 66 61 73 68 69 6f 6e 20 63 6c 65 76 ase fashion clev
00000410 65 72 20 73 75 62 77 61 79 20 66 69 62 65 72 20 er subway fiber 
00000420 69 73 6f 6c 61 74 65 20 72 6f 74 61 74 65 20 64 isolate rotate d
00000430 6f 6d 61 69 6e 20 76 65 74 65 72 61 6e 20 6b 6e omain veteran kn
00000440 65 65 20 75 72 67 65 20 67 72 61 62 20 6d 65 6c ee urge grab mel
00000450 74 20 76 69 64 65 6f 20 62 6f 61 72 64 20 68 65 t video board he
00000460 69 67 68 74 20 6e 6f 62 6c 65 20 6d 61 6e 61 67 ight noble manag
00000470 65 22 2c 20 22 73 6b 22 3a 20 22 36 39 31 38 32 e", "sk": "69182
00000480 37 66 35 30 61 33 30 63 38 34 36 61 66 64 32 32 7f50a30c846afd22
00000490 64 66 33 32 32 66 30 32 65 62 38 32 64 36 61 62 df322f02eb82d6ab
000004a0 31 38 39 37 33 62 35 31 34 32 37 37 34 39 65 30 18973b51427749e0
000004b0 30 33 36 33 65 33 36 30 66 34 61 22 7d 2c 20 22 0363e360f4a"}, "
000004c0 73 75 63 63 65 73 73 22 3a 20 74 72 75 65 7d 2c success": true},
000004d0 20 22 64 65 73 74 69 6e 61 74 69 6f 6e 22 3a 20  "destination": 
000004e0 22 77 61 6c 6c 65 74 5f 75 69 22 2c 20 22 6f 72 "wallet_ui", "or
000004f0 69 67 69 6e 22 3a 20 22 63 68 69 61 5f 77 61 6c igin": "chia_wal
00000500 6c 65 74 22 2c 20 22 72 65 71 75 65 73 74 5f 69 let", "request_i
00000510 64 22 3a 20 22 37 65 61 34 30 62 39 34 31 30 65 d": "7ea40b9410e
00000520 33 34 33 30 65 30 36 62 66 35 35 63 37 33 34 65 3430e06bf55c734e
00000530 37 37 62 65 35 61 66 61 61 62 61 62 61 66 32 36 77be5afaababaf26
00000540 64 31 36 30 32 35 39 32 38 64 33 65 32 38 33 64 d16025928d3e283d
00000550 61 66 38 32 32 22 7d 00 56 f5 63 e7 a3 02 00 80 af822"}.V.c.....
```


Reproduction Steps 1. Launch the application (Chia.exe)

2. Create a new private key
3. Click to see private key and note contents (ex : Private key & seed)
4. Download process hacker at https://sourceforge.net/projects/processhacker/
5. Search for `start_wallet.exe` in process hacker
6. Right click start_wallet.exe -> Properties -> Memory -> Strings
7. Enter private key/seed for search to display copies in memory

Recommendation Restart the application after key deletion to wipe any potentially sensitive artifacts.

Finding	**Race Condition via Fake `TransactionAck` Messages In Wallet Nodes**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-CHIA001-012
Status	New
Category	Data Validation
Component	chia-blockchain
Location	https://github.com/chia-network/chia-blockchain/blob/b82f3ba8a2953de12bddf5c5d6a33e443b51bc8b/src/wallet/wallet_node_api.py#L64
Impact	During re-orgs, malicious nodes may be able to prevent wallet nodes from re-broadcasting pending transactions. In addition, if a wallet node is connected to just one full node, this full node can prevent the wallet node from broadcasting transactions to future full nodes the wallet connects to.

Description

Wallet nodes need a mechanism to ensure that the transactions they broadcast reach a sufficient number of nodes. This is complicated by the fact that, during re-orgs, some transactions need to be rebroadcasted by wallet nodes, even if they were sent out successfully in the past.

In Chia, this is implemented by counting the `TransactionAck` replies from full nodes. The issue this finding discusses is that there's insufficient authentication on `TransactionAck` messages, which potentially allows full nodes to impede wallet nodes' ability to correctly broadcast transactions.

The `TransactionAck` API endpoint identifies the transaction by a supplied ID and increases the transaction's record `sent` counter. This is done using the `increment_sent` method:

```
    async def increment_sent(self, id: bytes32, name: str,  send_status:
     ➜ MempoolInclusionStatus, err: Optional[Err]) -> bool:
        """
        Updates transaction sent count (Full Node has received spend_bundle and s
 ➜  ent ack).
        """

        current: Optional[TransactionRecord] = await
         ➜ self.get_transaction_record(id)
        if current is None:
            return False

        sent_to = current.sent_to.copy()

        err_str = err.name if err is not None else None
        append_data = (name, uint8(send_status.value), err_str)

        # Don't increment count if it's already sent to othis peer
        if append_data in sent_to:
            return False

        sent_to.append(append_data)

        tx: TransactionRecord = TransactionRecord(
            confirmed_at_sub_height=current.confirmed_at_sub_height,
            confirmed_at_height=current.confirmed_at_height,
```

```
            created_at_time=current.created_at_time,
            to_puzzle_hash=current.to_puzzle_hash,
            amount=current.amount,
            fee_amount=current.fee_amount,
            confirmed=current.confirmed,
            sent=uint32(current.sent + 1),
```

As specified by the first highlighted snippet, the `increment_sent()` function finds the ongoing transaction record and validates if the peer (identified by peer's name) already sent a `TransactionAck` message. In the second highlighted snippet, the `sent` value is increment. The constructed `TransactionRecord` overwrites the previous record, as the primary key of the corresponding database table is the bundle ID. The `sent_to` list accumulates the state as to what nodes increased the `sent` counter in the past.

It should be noted that the `sent_to` list validation in the first highlighted snippet above is not sufficient. The sender of the `TransactionAck` message can vary the `send_status.value` and `err_str` response fields in order to trigger processing of not just one `TransactionAck` message.

Once the `sent` counter is beyond a threshold, the transaction is not re-broadcasted any more. As such, once the transaction ID is known to a malicious participants on the network, nothing appears to prevent them from increasing the wallet node's `sent` counter for that particular transaction, assuming there is a "live" transaction entry in the wallet node's DB. As for the consequences:

- During re-orgs, the `sent` counter gets reset to zero. The transaction ID is known to other nodes before the transactions are re-broadcasted. As such, during re-orgs, wallet nodes may be vulnerable to fake `TranasctionAck` messages from full nodes they connect to. Now, the time window for such `TransactionAck` messages to arrive is small and is between async task suspensions: the counter is reset in the database and the `_resend_queue` task gets placed on the event loop here. It is theoretically possible that in between the transaction counter reset and the `_resend_queue` call, forged `TransactionAck` messages get processed and inflate the counter, resulting in messages never re-broadcasted.
- Re-orgs aside, if the wallet node is connected to just one full node, this full node could orchestrate a number of fake `TransactionAck` messages referring to that transaction, thereby preventing the transaction going out the gate altogether. If, however, the node is connected to multiple nodes, fully preventing the transaction from reaching multiple nodes does not appear doable.
- As for a direct attack, in which a wallet node is simply prevented from broadcasting a transaction does not appear possible because the transaction ID is unknown and Chia wallet nodes broadcast new transactions to all full nodes they are connected to. This finding is rated with low severity due to these considerations.

For reference, links to the source code are provided below. The receiving full nodes are expected to answer to transaction messages with a `TransactionAck` message. For each received `TransactionAck` message, wallet nodes decrease a counter that's kept for that particular transaction. Currently, once four `TransactionAck` messages are received, the transaction message will not be re-broadcasted anymore.

Recommendation If the `sent_to` list is modified to include only the node ID (and not the error message and status), each node will be able to ACK a transaction only once. It should be noted that in theory this does not fully resolve the issue. Technically, a sufficiently large subset of full nodes the wallet connects to could still collude and inflate the counter for any message. Since it's the wallet node that chooses the full nodes it connects to, this appears as a minor issue and could be accepted as a known risk if the `sent_to` list entries are identified only by node IDs.

Finding	**Data Types not Checked on Payload IDs and Function Names**
Risk	**Informational** Impact: Low, Exploitability: None
Identifier	NCC-CHIA001-001
Status	Reported
Category	Data Validation
Component	chia-blockchain
Location	https://github.com/chia-network/chia-blockchain/blob/76729e64/src/server/ws_connection.py#L309
Impact	When Chia nodes decode network messages, the message's Payload ID is fully unconstrained and can be of any type. To avoid any unforeseen issues in future releases, enforcing types on these fields should be considered.

Description

Chia's network messages are CBOR-encoded dictionaries, which must include three key values — the message's function name, the message data and the payload ID — and can contain an arbitrary number of other key/value entries that are ignored by the implementation. The function name decides what function gets called and the message data is expected to be another dictionary, which specifies the API function's arguments. As specified by the `@api_request` decorator (see `util/api_decorators.py`), the supplied dictionaries are converted to objects and type checking is performed on initialization of those objects, see `util/type_checking.py`. The above detailed procedure is critical for avoiding passing arbitrary types into the consensus-critical code paths.

This finding notes that some type-relaxed processing is still present at the layer before consensus. In particular, `Payload` and `Message` classes are not decorated with the `@cbor_message` decorator and as such their fields can get decoded to arbitrary types:

```
@dataclass
class Message:
    # Function to call
    function: str
    # Message data for that function call
    data: Any


@dataclass
class Payload:
    # Message payload
    msg: Message
    # payload id
    id: Optional[bytes8]
```

As for the `Payload.id` field, it can take any type and that type will be passed back to the caller. The payload IDs are used to tie pending requests with result. It appears that the unforeseen type will propagate into the `Dicts` tracking pending requests and results without triggering an exception. When it comes to `Message.function`, it can get decoded to a type different than `str`, but due to the usage of `str.startswith("_")` right after the decoding, it does not appear that a non-`str` type can propagate beyond the decoding.

Recommendation

Enforce type checking on payload ID and message function fields in order to avoid any unforeseen behavior in this and future releases.

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

Appendix B: Project Contacts

The team from NCC Group has the following primary members:

- Javed Samuel — NCC Group
  javed.samuel@nccgroup.com
- Aleksandar Kircanski — NCC Group
  aleksandar.kircanski@nccgroup.com
- Ava Howell — NCC Group
  ava.howell@nccgroup.com
- Ephrayim Kishko — NCC Group
  ephrayim.kishko@nccgroup.com

The team from Chia Network Inc has the following primary members:

- Gene Hoffman — Chia Network Inc.
  hoffmang@chia.net
- Bram Cohen — Chia Network Inc.
  bram@chia.net