# Status Report 2

This status report includes findings identified in the second calendar week of testing of the Chia blockchain implementation. Three consultants (two billable and one shadow) worked on the project during the week of 1/18/21. Focus is continuing to be on (1) consensus code review and (2) Proof of Space (PoS) implementation C++ review.

This status report contains a write-up of five findings, summarized below. Any feedback on these findings from the Chia Team is welcome and will be incorporated in the final document.

- 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.
- P2P Message Response Object Mismatches: 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.
- Chia Node Private Key File Persists on Filesystem after Uninstall: 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.
- 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.
- Private Key and Mnemonic Secret Linger in Memory After Key Deletion: Sensitive wallet information can be obtained in process memory after users explicitly delete their key.

During the last week of testing, two consultants will be staffed on the project.

# 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 11.

| Title | ID | Risk |
|---|---|---|
| Unimplemented End Of Sub Slot Bundle Validation | 007 | High |
| Excess Storage Denial of Service Vectors | 010 | Medium |
| P2P Message Response Object Mismatches | 005 | Low |
| Chia Node Private Key File Persists on Filesystem after Uninstall | 006 | Low |
| Private Key and Mnemonic Secret Linger in Memory After Key Deletion | 009 | Low |

| | |
|---|---|
| **Finding** | **Unimplemented End Of Sub Slot Bundle Validation** |
| **Risk** | **High** Impact: High, Exploitability: High |
| **Identifier** | NCC-CHIA001-007 |
| **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.[1] 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...]

        if eos.challenge_chain.challenge_chain_end_of_slot_vdf.challenge !=
         ➜  last_slot_ch:
            # This slot does not append to our next slot
```

[1] 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.

```
        # 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 |
| 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,[2] transaction mempool,[3] orphan transactions,[4] 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).

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

| | |
|---|---|
| **Finding** | **P2P Message Response Object Mismatches** |
| Risk | **Low** Impact: Low, Exploitability: Low |
| Identifier | NCC-CHIA001-005 |
| Category | Data Validation |
| Component | chia-blockchain |
| Location | https://github.com/chia-network/chia-blockchain/blob/f50a372b509d42bfd63d20de3abf985d1294f22f/src/wallet/wallet_node.py#L413<br>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 |
| **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 |
| **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)<br>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.

# 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 member:

- Gene Hoffman — Chia
  hoffmang@chia.net