
# Security Audit of Charged Particles' Smart Contracts

a report of findings by

Arcadia

May 8th, 2021

## Table of Contents

# Document Info

| | |
|---|---|
| Client | Charged Particles |
| Title | Security Audit of Charged Particles' Smart Contracts |
| Approved By | Rasikh Morani |

## Contact

For more information on this report, contact The Arcadia Media Group Inc.


| Rasikh Morani |
|---|
| (972) 543-3886 |
| rasikh@arcadiamgroup.com |
| https://t.me/thearcadiagroup |

# Executive Summary

A Representative Party of Charged Particles ("CP") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following Charged Particles smart contracts on the Charged Particles repo at Commit #2afb274c16f9c721a915c2fee61c843af3453677.

Arcadia completed this security review using various methods primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

There were 04 issues found, 00 of which were deemed to be ‘critical’, and 02 of which were rated as ‘high’.

After a first review and report, and discussion of findings with the Charged Particles team, they fixed all reported issues, except issue CP4 which is related to gas consumption. However, as the team discussed, code was refined to reduce gas usage in transactions creating Charged Particles. Arcadia then performed a second review of the code at commit #bd1984d1366cd0a9e8682a21aa59da1cbf788e57 specifically regarding only those remediated issues.

| Severity Rating | Number Of Original Occurrences | Number Of Remaining Occurrences |
|---|---|---|
| Critical | 00 | 00 |
| High | 02 | 00 |
| Medium | 01 | 00 |
| Low | 01 | 00 |
| Notice | 00 | 00 |
| Informational | 00 | 00 |

# Findings

## 1. Function addPortions should check releaseTimes in chronological order

- CP-1
- Severity: Medium
- Likelihood: Low
- Impact: Low

- Target: IonTimelock.sol
- Category: Lock Time
- Finding Type: Dynamic

Function `addPortions` should check that releaseTimes should be in chronological order to ensure that function `nextReleaseTime` will always return the actual next release time.

```
function addPortions(uint256[] memory amounts, uint256[] memory releaseTimes)
  external
  virtual
  override
  returns (bool)
{
  require(msg.sender == funder, "ITL:E-103");
  require(amounts.length == releaseTimes.length, "ITL:E-202");

  uint256 totalAmount;
  for (uint i = 0; i < amounts.length; i++) {
    uint256 releaseTime = releaseTimes[i];
    uint256 amount = amounts[i];

    // solhint-disable-next-line not-rely-on-time
    require(releaseTime > block.timestamp, "ITL:E-301");

    portions.push(Portion({
      amount: amount,
      releaseTime: releaseTime,
      claimed: false
```

```
    }));

    totalAmount = totalAmount.add(amount);
  }

  uint256 amountAvailable = token.balanceOf(address(this));
  require(amountAvailable >= totalAmount, "ITL:E-411");

  emit PortionsAdded(amounts, releaseTimes);
  return true;
}
```

**Action Recommended:** Check that all times in releaseTimes parameter are in an ascending order.

```
for (uint i = 0; i < amounts.length; i++) {
    uint256 releaseTime = releaseTimes[i];
    if (i > 0) require(releaseTimes[i] > releaseTimes[i - 1]);
    uint256 amount = amounts[i];
    ...
}
```

## 2. Function addPortions function does not guarantee the locked amount

- CP-2
- Severity: High
- Impact: High
- Target: IonTimelock.sol
- Category: Locked token amount
- Finding Type: Dynamic

Function `addPortions` does not guarantee that the sum of locked amounts is locked in the timelock contract. For example:

- Call `addPortions` to add 10 tokens: This is OK, as this first time the function will check that the contract receives at least 10 tokens.
- Call `addPortions` to add 5 more tokens: The call will be successful (bypassing the require statement require(amountAvailable >= totalAmount, "ITL:E-411"); without any new tokens transferred to the contract for locking.

```
uint256 totalAmount;
   for (uint i = 0; i < amounts.length; i++) {
     uint256 releaseTime = releaseTimes[i];
     uint256 amount = amounts[i];

     // solhint-disable-next-line not-rely-on-time
     require(releaseTime > block.timestamp, "ITL:E-301");

     portions.push(Portion({
       amount: amount,
       releaseTime: releaseTime,
       claimed: false
     }));

     totalAmount = totalAmount.add(amount);
   }

   uint256 amountAvailable = token.balanceOf(address(this));
   require(amountAvailable >= totalAmount, "ITL:E-411");
```

**Action Recommended:** There can be 2 possible solutions:

1. The function should call transferFrom to transfer the token into the contract, and validate that the total received balance is equal to the sum of locked portions. The following is a suggested code snippet:

```
uint256 amountAvailable = token.balanceOf(address(this));
   token.safeTransferFrom(msg.sender, address(this), totalAmount);
   require(token.balanceOf(address(this)).sub(amountAvailable) >= totalAmount,
"ITL:E-411");
```

2. Parameter totalAmount should be the sum of all locked portions in the contract.

## 3. Deployment of IONX

- CP-3
- Severity: High
- Impact: High
- Target: Ion.sol
- Category: Contract owner
- Finding Type: Dynamic

The owner of `IONX` contract should be strictly set/changed to a time-lock or a DAO contract. This is because the owner of IONX can mint new tokens to Universe and Timelock contracts. This is more of a deployment issue that the auditors can only give best advice but cannot intervene in the deployment process.

```
function mintToUniverse(uint256 amount) external onlyOwner returns (bool) {
   require(address(_universe) != address(0x0), "Ion:E-404");
   _mint(address(_universe), amount);
 }

 function mintToTimelock(address ionTimelock, uint256[] memory amounts, uint256[]
memory releaseTimes) external onlyOwner {
   require(address(ionTimelock) != address(0x0), "Ion:E-403");

   uint256 totalAmount;
   for (uint i = 0; i < amounts.length; i++) {
     totalAmount = totalAmount.add(amounts[i]);
   }

   _mint(address(ionTimelock), totalAmount);
   require(IIonTimelock(ionTimelock).addPortions(amounts, releaseTimes), "Ion:E-406");
 }
```

**Action recommended**: Transferring the ownership of Ion to a TimeLock or a DAO contract after deployment.

### 4. ChargedParticle creation transaction gas cost is too high

- CP-4
- Severity: Low
- Impact: Medium
- Target: Proton.sol
- Category: Transaction cost
- Finding Type: Dynamic

The creation of a charged particle through the function `createChargedParticle` of contract `Proton` has a very high gas cost. Specifically, calling the function `createChargedParticle` would cost more than 1M gas. This is very expensive when ETH prices are high as they often are. We recommend optimizing the function to minimize gas consumption.

# Conclusion

Arcadia identified issues that occurred at hash #2afb274c16f9c721a915c2fee61c843af3453677 that were confirmed to be patched as of #bd1984d1366cd0a9e8682a21aa59da1cbf788e57

# Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.