QuillAudits

Audit Report
October, 2021

For

parcel

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Parcel smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	3	6	1
Closed	0	0	3	0

Introduction

During the period of **September 22, 2021, to October 06, 2021** - QuillAudits Team performed a security audit for **Parcel** smart contracts.

The code for the audit was taken from the following official repo of Parcel: https://github.com/ParcelHQ/parcel-smart-contract

Note	Date	Commit hash
Version 1	September	25cc80045aeedd77cf383720f8477c0b87461b11
Version 2	October	97e0d14d7741955da6defd24d66c7517b538eeac

Issues Found

A. Contract – AllowanceModule

High severity issues

No issues were found.

Medium severity issues

A.1 Race Condition

```
Line 444:
function setMaxGasPrice(uint256 newMaxGasPrice) public {
        maxGasPrice[msg.sender] = newMaxGasPrice;
        emit NewMaxGasPrice(msg.sender, newMaxGasPrice);
    }
```

Description
In the contract, the user can call the function setMaxGasPrice to modify his own max gas price. In the scenario where he modifies the max gas price then calls the executeAllowanceTransfer function and the last transaction gets mined first, the user could possibly execute an allowance transfer with a higher or lower max gas price than the one that he set using the setMaxGasPrice function.

Remediation
Add the max gas price as the argument to the executeAllowanceTransfer function and add a require that verifies that the max gas price provided in the arguments is the same as the one that is stored in the smart contract.

Acknowledged
The Parcel team has acknowledged the risk for the reason that the maxGasPrice will not change frequently.

A.2 Loop over a dynamic array

```
Line 311:
for (uint256 i = 0; i < taskIds.length; i++) {
            if (task == taskIds[i]) {
                isActive = true;
                break;
            }
        }
```

```
Line 511:
    function getDelegates(address safe, uint48 start, uint8 pageSize) public
    view returns (address[] memory results, uint48 next) {
            results = new address[](pageSize);
            uint8 i = 0;
            uint48 initialIndex = (start != 0) ? start : delegatesStart[safe];
            Delegate memory current = delegates[safe][initialIndex];
           while(current.delegate != address(0) && i < pageSize) {
                    results[i] = current.delegate;
                    i++;
                    current = delegates[safe][current.next];
            }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Acknowledged

The Parcel team has acknowledged the risk

Low severity issues

A.3 Renounce Ownership

```
Line 55:
contract AllowanceModule is SignatureDecoder, Ownable {
```

Description
Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation
It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Solved
The Parcel team has solved the issue in version 2 by using a multiSig.

A.4 Floating pragma

```
Line 2:
pragma solidity >=0.7.0 <0.8.0;
```

Description
The contract makes use of the floating-point pragma >=0.7.0 <0.8.0 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Parcel team has acknowledged the risk.

A.5 Missing Address Verification

```
Line 120:
constructor(address payable gelato, address gelatoPokeMe){
        GELATO = gelato;
        GELATO_POKE_ME = gelatoPokeMe;
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Solved
The Parcel team has solved the issue in version 2 by adding a require statement to verify the addresses that are coming from the arguments.

A.6 Usage of block.timestamp

```
Line 144:
uint32 currentMin = uint32(block.timestamp / 60);
```

```
Line 160:
uint32 currentMin = uint32(block.timestamp / 60);
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't impact the logic of the smart contract.

Acknowledged
The Parcel team has acknowledged the risk knowing the 900 seconds delay won't impact the business logic.

A.7 Integer overflow

```
Line 224:
allowance.nonce = allowance.nonce + 1;
```

Description
The allowance.nonce variable in an uint16, incrementing this value in solidity version than is < 0.8.0 without any verification can result in an overflow so if the value reached 2^16-1 incrementing will change the nonce value to 0.

Remediation
Use the add function from the SafeMath library. Also returning an error message would help explain why the transaction failed.

Acknowledged
The Parcel team has acknowledged the risk.

Informational Issues

A.8 Use of Inline Assembly

```
Line 320:
function getChainId() public pure returns (uint256) {
        uint256 id;
        // solium-disable-next-line security/no-inline-assembly
        assembly {
            id := chainid()
        }
        return id;
    }
```

```
Line 511:
    assembly {
        mstore(results, i)
    }
```

Description
Inline assembly is a way to access the EVM at a low level. This discards several important safety features in Solidity.

Remediation
When possible, do not use inline assembly because it is a manner to access to the EVM at a low level. An attacker could bypass many important safety features of Solidity.

Acknowledged
The Parcel team has acknowledged the risk.

B. Contract – Resolver

High severity issues

No issues were found.

Medium severity issues

B.1 Looping around unknown number of delegates

```
Line 64:
do {
        uint96 amount;
        (canExec, amount) = _canTransferToDelegate(
            _safe,
            currentNode.delegate,
            _token
        );
        if (canExec) {
            execPayload = _getPayload(
                _safe,
                _token,
                currentNode.delegate,
                amount
            );
            return (canExec, execPayload);
        }
        uint48 nextNode = currentNode.next;
        currentNode = allowanceModule.delegates(_safe, nextNode);
    } while (currentNode.delegate != address(0));
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size data structure that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge data structure, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Acknowledged
The Parcel team has acknowledged the risk.

Low severity issues

B.2 Missing Address Verification

```
Line 120:
constructor(address _allowanceModule) {
        allowanceModule = IAllowanceModule(_allowanceModule);
    }
```

```
Line 155:
function checker(
        address _safe,
        address _token
    ) external view returns (bool canExec, bytes memory execPayload) {
    uint48 entry = allowanceModule.delegatesStart(_safe);
    IAllowanceModule.Delegate memory currentNode = allowanceModule
        .delegates(_safe, entry);
    do {
        uint96 amount;
        (canExec, amount) = _canTransferToDelegate(
            _safe,
            currentNode.delegate,
            _token
        );
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Solved
The Parcel team has solved the issue in version 2 by adding require statements to verify the addresses that are coming from the arguments.

B.3 Floating pragma

```
Line 2:
pragma solidity >=0.7.0 <0.8.0;
```

Description
The contract makes use of the floating-point pragma >=0.7.0 <0.8.0 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Parcel team has acknowledged the risk.

C. Contract – SignatureDecoder

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

C.1 Floating pragma

```
Line 2:
pragma solidity >=0.7.0 <0.8.0;
```

Description
The contract makes use of the floating-point pragma >=0.7.0 <0.8.0 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Parcel team has acknowledged the risk.

D. Contract – Enum

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

D.1 Floating pragma

```
Line 2:
pragma solidity >=0.7.0 <0.8.0;
```

Description
The contract makes use of the floating-point pragma >=0.7.0 <0.8.0 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Parcel team has acknowledged the risk.

Automated Tests

Slither

Constant/View/Pure functions:
FeedRegistryInterface.proposeFeed(address,address,address) : Potentially should be constant/view/pure but is not.
more
Pos: 237:2:

Constant/View/Pure functions:
FeedRegistryInterface.confirmFeed(address,address,address) : Potentially should be constant/view/pure but is not.
more
Pos: 243:2:

No return:
AggregatorInterface.latestAnswer(): Defines a return type but never explicitly returns a value.
Pos: 5:2:

No return:
AggregatorInterface.latestTimestamp(): Defines a return type but never explicitly returns a value.
Pos: 12:2:

No return:
AggregatorInterface.latestRound(): Defines a return type but never explicitly returns a value.
Pos: 19:2:

No return:
AggregatorInterface.getAnswer(uint256): Defines a return type but never explicitly returns a value.
Pos: 26:2:

No return:
AggregatorInterface.getTimestamp(uint256): Defines a return type but never explicitly returns a value.
Pos: 35:2:

No return:
AggregatorV3Interface.decimals(): Defines a return type but never explicitly returns a value.
Pos: 6:2:

No return:
AggregatorV3Interface.description(): Defines a return type but never explicitly returns a value.
Pos: 13:2:

No return:

FeedRegistryInterface.getPhase(address,address,uint16): Defines a return type but never explicitly returns a value.
Pos: 179:2:

No return:

FeedRegistryInterface.getRoundFeed(address,address,uint80): Defines a return type but never explicitly returns a value.
Pos: 192:2:

No return:

FeedRegistryInterface.getPhaseRange(address,address,uint16): Defines a return type but never explicitly returns a value.
Pos: 203:2:

No return:

FeedRegistryInterface.getPreviousRoundId(address,address,uint80): Defines a return type but never explicitly returns a value.
Pos: 215:2:

No return:

FeedRegistryInterface.getNextRoundId(address,address,uint80): Defines a return type but never explicitly returns a value.
Pos: 225:2:

No return:

FeedRegistryInterface.getProposedFeed(address,address): Defines a return type but never explicitly returns a value.
Pos: 251:2:

No return:

FeedRegistryInterface.proposedGetRoundData(address,address,uint80): Defines a return type but never explicitly returns a value.
Pos: 261:2:

No return:

FeedRegistryInterface.proposedLatestRoundData(address,address): Defines a return type but never explicitly returns a value.
Pos: 276:2:

No return:

FeedRegistryInterface.getCurrentPhaseId(address,address): Defines a return type but never explicitly returns a value.
Pos: 291:2:

Mythx

ID	Severity	Name	File
SWC-103	Low	A floating pragma is set.	SignatureDecoder.sol

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Functional test

Function Names	Testing results	Logic result
setAllowance	Passed	Passed
resetAllowance	Passed	Passed
deleteAllowance	Passed	Passed
executeAllowanceTransfer	Passed	Passed
getChainId	Passed	Passed
getTokens	Passed	Passed
getTokenAllowance	Passed	Passed
addDelegate	Passed	Passed
removeDelegate	Passed	Passed
setPaymentToken	Passed	Passed
removePaymentToken	Passed	Passed
getDelegates	Passed	Passed

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.
No instances of Back-Door Entry were found in the contract.
Many issues were discovered during the initial audit; The Parcel Team fixed all of these issues

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Parcel Contracts**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Parcel** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report October, 2021

For

parcel

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com