Parcel Payroll Security Review

Pashov Audit Group

Conducted by: pashov

March 27th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Parcel Payroll** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Parcel Payroll

Parcel Payroll is a protocol that brings a UX-focused payroll management infrastructure to the blockchain. It integrates with the Gnosis Safe multisig wallet abstraction as well as the Spending Limit module within the Gnosis Safe multisig to set an allowance of spending approvers.

In-scope public state-changing methods:

- `PayrollManager::executePayroll` - validates signed messages, fetches tokens from Gnosis Safe multisig wallets and executes transfers(payments)

More docs

Threat Model

System Actors

- Payroll approver - can sign messages off-chain to approve token spending
- Payroll executor - executes a payroll transaction

Q: What in the protocol has value in the market?

A: The Gnosis Safe allowance of the protocol

Q: What is the worst thing that can happen to the protocol?

1. Malicious user draining the protocol's allowance of a Gnosis Safe multisig wallet
2. Replay attacks on signatures & approvals
3. DoS on `executePayroll` related to token/native asset functionality
4. Tokens left stuck forever in the `PayrollManager` contract

Interesting/unexpected design choices:

It is intended that the protocol receives exactly as much value as it transfers out, otherwise tokens will be stuck in the contract.

The protocol uses Merkle Trees for off-chain storing of approval signatures by Gnosis Safe multisig signers.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **-** **37d25a4f4fb2b72f47ccd4dad2146b07640feb20**

Scope

The following smart contracts were in scope of the audit:

- `payroll/PayrollManager`
- `payroll/Validators`
- `payroll/Modifiers`
- `payroll/Storage`
- `signature/Signature`
- `interfaces/index`

7. Executive Summary

Over the course of the security review, pashov engaged with Parcel Payroll to review Parcel Payroll. In this period of time a total of **16** issues were uncovered.

Protocol Summary

Protocol Name	Parcel Payroll
Date	March 27th, 2023

Findings Count

Severity	**Amount**
Critical	2
Medium	3
Low	3
QA	8
Total Findings	**16**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Contract is missing a payable function which makes it impossible to operate with native assets	Critical	Resolved
[C-02]	Contract fails to handle address(0) as a native asset for it's zero balance left invariant	Critical	Resolved
[M-01]	Using the transfer function of address payable is discouraged	Medium	Resolved
[M-02]	Usage of non-standard ERC20 tokens might lead to stuck funds	Medium	Resolved
[M-03]	Contract inherits from Pausable but does not expose pausing/unpausing functionality	Medium	Resolved
[L-01]	EIP-712 domain separator is not implemented correctly	Low	Resolved
[L-02]	The PayrollManager contract has less than 5% code coverage	Low	Resolved
[L-03]	Missing array arguments length check	Low	Resolved
[QA-01]	Redundant or unused interfaces, enums, variables, modifiers and comments throughout the codebase	QA	Resolved
[QA-02]	Contract and file name not matching	QA	Resolved
[QA-03]	Prefer Solidity Custom Errors over require statements with strings	QA	Resolved
[QA-04]	Inconsistency in file header comments	QA	Resolved
[QA-05]	Incorrect NatSpec	QA	Resolved

[QA-06]	Open TODO in the code	QA	Resolved
[QA-07]	Typos in the code	QA	Resolved
[QA-08]	Inconsistent and unsafe pragma statements used	QA	Resolved

8. Findings

8.1. Critical Findings

[C-01] Contract is missing a `payable` function which makes it impossible to operate with native assets

Severity

Impact: High, as all transactions that use native assets will revert

Likelihood: High, as it is well expected that native assets will be used as a `paymentToken` often

Description

The `PayrollManager` does not have a `receive` function that is marked as `payable` neither any `payable` function at all. This will make it impossible for the contract to work with native assets because all transfers from the Gnosis Safe multisig to him will revert.

Recommendations

Add a `payable` `fallback` or `receive` function in `PayrollManager` to allow for native assets transfers.

[C-02] Contract fails to handle `address(0)` as a native asset for it's zero balance left invariant

Severity

Impact: High, as all transactions that use native assets will revert

Likelihood: High, as it is well expected that native assets will be used as a `paymentToken` often

Description

The `executePayroll` method has the following code at the end of it:

```
// Check if the contract has any tokens left
for (uint256 i = 0; i < paymentTokens.length; i++) {
    IERC20 erc20 = IERC20(paymentTokens[i]);
    if (erc20.balanceOf(address(this)) > 0) {
        // Revert if the contract has any tokens left
        revert("CS018");
    }
}
```

A few lines above the code looks like this:

```
if (tokenAddress[i] == address(0)) {
    // Transfer ether
    payable(to[i]).transfer(amount[i]);
    packPayoutNonce(true, payoutNonce[i]);
}
```

Which shows us that `address(0)` is used to handle native assets transfers. The problem is that the formerly mentioned code does not handle this `address(0)` token correctly, so if the `paymentTokens` array contains it the whole transaction will revert.

Recommendations

Check separately for the native asset balance of the contract in the end of `executePayroll` and ignore `address(0)` when calling `ERC20::balanceOf` for the `paymentTokens` array values.

8.2. Medium Findings

[M-01] Using the `transfer` function of `address payable` is discouraged

Severity

Impact: Medium, as payroll will revert and Merkle Trees & approvals would have to be done again

Likelihood: Medium, as it happens any time the recipient is a smart contract or a multisig wallet that has a receive function taking up more than 2300 gas

Description

The `executePayroll` function uses the `transfer` method of `address payable` to transfer native asset funds to a recipient address. This address is set by the caller but is also encoded in the leaf of a Merkle Tree that is created off-chain. It is possible that this recipient is a smart contract that has a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer`. Examples are some smart contract wallets or multi-sig wallets, so usage of `transfer` is discouraged.

Recommendations

Use a `call` with value instead of `transfer`. The function already has a `nonReentrant` modifier so reentrancy is not a problem here.

[M-02] Usage of non-standard ERC20 tokens might lead to stuck funds

Severity

Impact: High, because tokens will be left stuck in `PayrollManager`

Likelihood: Low, because there aren't many such ERC20 tokens

Description

The `executePayroll` method uses the `transfer` method of `ERC20`, but does not check if the returned `bool` value is `true`. This is problematic, because there are tokens on the blockchain which actually do not revert on failure but instead return `false` (example is `ZRX`). If such a token is used and a transfer fails, the tokens will be stuck in the `PayrollManager` smart contract forever.

Recommendations

Use the `SafeERC20` library from `OpenZeppelin` and change the `transfer` call to a `safeTransfer` call instead.

[M-03] Contract inherits from `Pausable` but does not expose pausing/unpausing functionality

Severity

Impact: Low, as methods do not have `whenNotPaused` modifier

Likelihood: High, as it is certain that contract can't be paused at all

Description

The `Organizer` smart contract inherits from OpenZeppelin's `Pausable` contract, but the `_pause` and `_unpause` methods are not exposed externally to be callable and also no method actually uses the `whenNotPaused` modifier. This shows that `Pausable` was used incorrectly and is possible to give out a false sense of security when actually contract is not pausable at all.

Recommendations

Either remove `Pausable` from the contract or add `whenNotPaused` modifier to the methods that you want to be safer and also expose the `_pause` and `_unpause` methods externally with access control.

8.3. Low Findings

[L-01] EIP-712 domain separator is not implemented correctly

The domain separator in `Signature` is missing the `name`, `version` and `salt` fields defined in EIP-712. The standard states that not all fields are mandatory but adding them would add another layer of security for the usage of off-chain signed messages for the protocol. Refer to the EIP712 doc and add all of the missing domain separator fields.

[L-02] The `PayrollManager` contract has less than 5% code coverage

Some security vulnerabilities are more easily caught in a normal unit testing scenario than in a full line-by-line manual audit. It is strongly recommended to strive for close to 100% code coverage on all of the code that you are about to deploy so no low-hanging fruit bugs are left in the contracts.

[L-03] Missing array arguments length check

In `PayrollManager::executePayroll` there is a check that most arrays have the same length, but a check that `paymentTokens.length == payoutAmounts.length` is missing and should be added.

8.4. QA Findings

[QA-01] Redundant or unused interfaces, enums, variables, modifiers and comments throughout the codebase

The `GnosisSafe` interface in `index` is only used as a parameter type in the `executeAllowanceTransfer` method, but it is not really needed as you can use a type of `address` instead. This way you can remove the `GnosisSafe` interface because its only method `execTransactionFromModule` is not used anywhere in the codebase.

The `Allowance` struct in `AllowanceModule` is not used anywhere and can be removed.

The `getTokenAllowance` method from the `AllowanceModule` interface is not used anywhere and can be removed.

The `Operation` enum in `Storage` is not used anywhere and can be removed.

The `MASTER_OPERATOR` storage variable in `Storage` is only written to but never read from and it is `internal` so it doesn't have a getter. It can be removed.

The `SENTINEL_UINT` constant in `Storage` is not used anywhere and can be removed.

There is no need in `Validators::isApprover` to check that `_addressToCheck != address(0)` . This is because in `Organizer::onboard` there is a `require` statement that enforces an `approver` address to not be `address(0)` . This means `_isApprover` will return `false` anyway, so the check is redundant and can be removed.

There is no need for the `require(_addressToCheck != address(0), "CS003");` check in `isOrgOnboarded` in `Validators`, because `Organizer::onboard` uses only `msg.sender` as the key in the `orgs` mapping, so it can't be `address(0)`. The check can be removed as it is redundant and wastes gas.

There is actually no need to do the `isOrgOnboarded` check in `Validators::isApprover`, because even if it is omitted, the `_isApprover` method will return `false` if an Org was not onboarded. My recommendation here is to remove the `Validators` smart contract and just use

```
require(_addressToCheck != SENTINEL_ADDRESS);
orgs[_safeAddress].approvers[_addressToCheck] != address(0);
```

to check if an address is an approver for a Gnosis Safe wallet.

The `onlyMultisig` modifier in `Modifiers` is redundant as it just enforces a function argument to have the value of `msg.sender`. Remove the modifier and the argument checked and just use `msg.sender` directly instead.

Remove commented out imports in `Organizer` as the code is not used. Old code will be kept in old git commits so if you need them again you can get them from there, but there is no need to keep code commented out like this.

The `EIP712Domain` struct definition in `Signature` is not needed as it is not really used as it is actually hardcoded as a string in the `EIP712_DOMAIN_TYPEHASH` hash calculation. The struct definition can be removed as it is redundant.

The `PayrollTx` struct definition in `Signature` is not needed as it only contains one field. Remove the struct and its usage in `validatePayrollTxHashes` and use the `rootHash` field directly instead . Move the `// hash = encodeTransactionData(recipient, tokenAddress, amount, nonce)` comment to the NatSpec `@param` definition of `rootHash` in `validatePayrollTxHashes`.

Remove the `signer` variable in `validatePayrollTxHashes` and just return its value - no need to extract a variable if it's only going to be used once.

Remove the `encodedHash` variable in `PayrollManager::encodeTransactionData` and just directly return its value since it is only used once.

Remove the `erc20` variables and use their values directly in `PayrollManager::executePayroll` as they are only used once in their scopes.

Remove the `allowance` and `to` variables and use their values directly in `PayrollManager::execTransactionFromGnosis` as they are only used once.

Remove the `signature` parameter from `PayrollManager::execTransactionFromGnosis` as its value is always `bytes("")` - just pass this value directly to the `executeAllowanceTransfer` call instead.

Remove the local array variable `validatedRoots` in `PayrollManager::executePayroll` as all of its values are `true` anyway so it is redundant.

The `flag` parameter of `packPayoutNonce` is always `true`, so it can be removed as well as the code that is executed when it is `false`, because it is never executed (dead code).

[QA-02] Contract and file name not matching

The `Signature.sol` file contains the `SignatureEIP712` smart contract - use the same name for both, for example only `Signature`.

[QA-03] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Remove all `require` statements and use Custom Errors instead.

[QA-04] Inconsistency in file header comments

Most files in the codebase have a comment on the first line of the file that looks like

```
//contracts/Organizer.sol
```

The problem is that some files have it, while others don't, which is inconsistent. Those comments shouldn't really be needed anyway so it's best to remove them altogether from the codebase and if not - add the correct comment to each separate smart contract file.

[QA-05] Incorrect NatSpec

The NatSpec of the `ORG` struct in `Storage` is incorrect as it says there are `claimbles` and `autoClaim` parameters or struct fields but there are no such fields, so they should be removed from the NatSpec doc. Also the order of `approvers` and `approvalsRequired` should be switched.

[QA-06] Open `TODO` in the code

In `Organizer::onboard` there is an open `TODO` which shows code is not production-ready. Implement the comment in the `TODO` or remove it.

[QA-07] Typos in the code

Various typos in the code should be fixed:

`payrill` -> `payroll`

`claimbles` -> `claimables`

`AlowanceModule` -> `AllowanceModule`

[QA-08] Inconsistent and unsafe pragma statements used

It is a best practice to use a stable pragma statement so you can lock the compiler version and get the same bytecode in each compilation deterministically. Also it is recommended to use the same Solidity version throughout the codebase, but this is not the case currently. Change all pragma statements to use the same version and lock the pragma.