CANTINA

Parcel

Security Review

Reviewed by

Optimum
Gerard Persoon
Pashov Krum
Christos Papakonstantinou

April 10, 2023

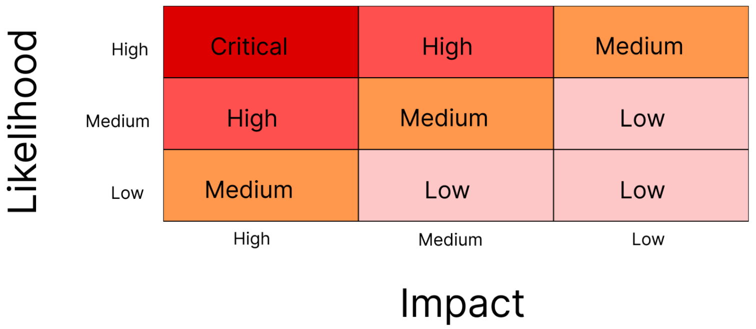
Contents

1 Introduction

1.1 Disclaimer

Cantina provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina code security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.2 Risk assessment



Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	Impact		

1.2.1 Impact

The severity of security issues found during the security review is categorized based on the impact it may have on the assets in the protocol and its users. A high severity issue can result in a loss of a significant portion of assets (greater than 10%) in the protocol or cause significant harm to a majority of users. Medium severity issues may cause losses to only a subset of users or global losses of less than 10%, but are still deemed unacceptable. Low severity issues may lead to losses that are inconvenient but manageable, such as griefing attacks that can be easily repaired or gas inefficiencies.

1.2.2 Likelihood

The likelihood of security issues occurring during the security review is classified based on the level of ease or motivation required to perform the exploit. A high likelihood issue is almost certain to happen, easy to execute, or has a strong incentive. Medium likelihood issues may only be possible under certain conditions or have some level of incentive, but are still relatively probable. Low likelihood issues require rare circumstances to align, or have little-to-no incentive for exploitation.

1.2.3 Action required for severity levels

The severity of issues found during the security review is classified based on the recommended priority for addressing the issue. Critical severity issues must be addressed as soon as possible, particularly if the protocol has already been deployed. High severity issues must be addressed before deployment if not already deployed. Medium severity issues should be addressed at the earliest convenience, while low severity issues could be addressed as time permits.

2 Security Review Summary

Parcel is a platform that enables DAO operators to make both one-off & bulk payouts with a user-friendly interface.

From February 21st to March 3rd with an extension from March 18th to March 21st the Cantina team conducted a review of parcel-payroll on commit hash 3fa78dfb. The team identified a total of **50** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 5
- Medium Risk: 6
- Low Risk: 6
- Gas Optimizations: 13
- Informational: 20

3 Findings

3.1 High Risk

3.1.1 `executePayroll()` can be frontrun with split transactions

Severity: High Risk

Context: PayrollManager.sol#L139-L244, AlowanceModule.sol#L56

Description: The function `executePayroll()` executes multiple transactions in one go, but it can be front-run to execute a subset of the transactions. Note this requires careful crafting of the parameters. This splitting leads to several problems:

- the retrieval from the allowance contract (`execTransactionFromGnosis()`) can be split into a large number of small transactions. This uses up the nonces in the allowance contract. As these nonces are limited to a maximum of 65535 this would finally prevent further payments;
- if a subset is executed (successfully) before the original transaction, then the original transaction will fail and has to be rescheduled (minus the succeeded transaction). This requires gas, effort, and time;
- if the original transaction is delayed multiple times, then the allowance time period (e.g. a month) could be passed and the allowance for that period would be forfeited (lost). This could require coordination efforts to fix.

```
function executePayroll(...) ... {
    ...
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
    ...
}
contract AllowanceModule is ... {
    struct Allowance {
        uint96 amount;
        uint96 spent;
        uint16 resetTimeMin;
        uint32 lastResetMin;
        uint16 nonce; // 16 bits so max 65535
    }
}
```

Recommendation: Use the solution of "Token retrieval not linked to signed transactions".

Parcel: This is solved by a redesign of the `executePayroll()` function.

Cantina Security: Verified.

3.1.2 `offboard()` clears all nonces

Severity: High Risk

Context: Organizer.sol#L93-L106, Storage.sol#L17-L34

Description: The function `offboard()` deletes `orgs[msg.sender]`, which also deletes `packedPayoutNonces[]`. If the same safe would ever `onboard()`, with (more or less) the same `approvers`, then all previous transactions could be re-executed because all the nonces are reset. This could drain the safe.

```
struct ORG {
    uint128 approverCount;
    uint128 approvalsRequired;
    mapping(address => address) approvers;
    uint256[] packedPayoutNonces;
}
mapping(address => ORG) public orgs;

function offboard() external {
    ...
    delete orgs[msg.sender]; // also deletes packedPayoutNonces
    ...
}
```

Recommendation: Apply one of the following solutions:

- use separate contracts for each gnosis safe (see the issue with that name) and deploy a new contract for a new `onboard()`;
- clear the data but set a flag to prevent `onboard()` again;
- don't clear the `packedPayoutNonces[]` and allow `onboard()` again;
- clear the data but keep an onboard nonce which is increased on every onboard and includes that in the signature via `validatePayrollTxHashes()`.

Parcel: Fixed via Proxy Pattern Changes PR 39. The function `offboard()` is no longer present.

Cantina Security: Verified.

3.1.3 More tokens can be retrieved from a safe via frontrunning

Severity: High Risk

Context: PayrollManager.sol#L139-L244

Description: An attacker can retrieve more tokens from the gnosis safe by frontrunning `executePayroll()` and increasing the `payoutAmounts[]` or adding entries with new `paymentTokens[]` to retrieve other tokens. There can also be duplicate transfers of the same token. Normally this is reverted in insufficient signed transactions are available.

However combined with the issue "`safeAddress` not included in signatures", this allows the stealing of ETH/tokens.

```
function executePayroll(...) ... {
    ...
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
    ...
    // send ETH / tokens to recepient
    ...
    // check all ETH / Tokens are sent
}
```

Recommendation: Only retrieve ETH/Tokens that are linked to signed transactions. See the issue "Token retrieval not linked to signed transactions" for a way to do this.

Parcel: This is solved by a redesign of `executePayroll()`.

Cantina Security: Verified.

3.1.4 safeAddress not included in signatures

Severity: High Risk

Context: PayrollManager.sol#L139-L244, PayrollManager.sol#L104-L124, Signature.sol#L39-L52

Description: The validation of signatures doesn't take into account the safeAddress (except when checking the number of approvals). This means signatures from the same approvers who are also involved in other organizations/sub daos/safes, could be reused. As the nonces (e.g. packPayoutNonce()) are stored in a different storage location, the nonces can be reused and thus the payment can be done again.

Combined with the issue "More tokens can be retrieved from a safe via frontrunning", this means ETH/Tokens can be stolen.

```
function executePayroll(...) ... {
    ...
    validateSignatures(safeAddress, roots, signatures);
    ...
    bytes32 leaf = encodeTransactionData(to[i],tokenAddress[i],amount[i],payoutNonce[i]);
    ...
    if (MerkleProof.verify(proof[i][j], roots[j], leaf)) {
        ++approvals;
    }
    if (approvals >= orgs[safeAddress].approvalsRequired && ... ) {
        ...
    }
    ...
}
function validateSignatures(...) ... {
    ...
    address signer = validatePayrollTxHashes(roots[i], signatures[i]);
    ...
}
function validatePayrollTxHashes(...) ... {
    bytes32 digest = ... abi.encode(PAYROLL_TX_TYPEHASH, rootHash) ...
    return digest.recover(signature);
}
```

Recommendation: Include the safeAddress in the signatures, for example in the following way:

```
function validatePayrollTxHashes(...) ... {
-   bytes32 digest = ... abi.encode(PAYROLL_TX_TYPEHASH, rootHash) ...
+   bytes32 digest = ... abi.encode(PAYROLL_TX_TYPEHASH, safeAddress, rootHash) ...
    return digest.recover(signature);
}
```

Note: The recommendation of issue "Use separate contracts for each gnosis safe" also solves this issue, but having a signed safeAddress is always safer.

Parcel: This is solved by using the Proxy Pattern.

Cantina Security: The DomainSeparator contains address(this) so signatures from multiple Gnosis safes can no longer be mixed.

3.1.5 Order of `execTransactionFromGnosis()` and `initialBalances()` is reversed

Severity: High Risk

Context: PayrollManager.sol#L139-L244

Description: The function `executePayroll()` first pulls ETH/tokens and then determines initial balances. Then it transfers ETH/tokens out and checks balances again. The before and after balances can only be equal if no ETH/tokens have been transferred, which is not the intended behaviour.

The after-balance check for tokens is < which seems to be a workaround to get tests working. This way tokens that are previously stored in the contract are used and can be stolen. This is also unwanted, although normally there shouldn't be tokens left in the contract.

```
function executePayroll(...) ... {
    ...
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
    for (uint256 i = 0; i < paymentTokens.length; i++) {
         ...   initialBalances[i] = ...
    }
    ...
    // send ETH / tokens to recepient
    ...
    for (uint256 i = 0; i < paymentTokens.length; i++) {
        ...
        if (...) {
            require(address(this).balance == initialBalances[i], ...);
         } else
         if (IERC20(paymentTokens[i]).balanceOf(address(this)) > initialBalances[i] ) {
                revert("CS018");
         }
    }
}
```

Recommendation: Strict checks are not necessary when using the recommendations for the following issues:

- "Use separate contracts for each gnosis safe";
- "Token retrieval not linked to signed transactions";
- "ETH and tokens can get stuck".

Otherwise, determine the initial balances before `execTransactionFromGnosis()`. Change the tokens check to a strict check.

So change the code to something like the following.

```
function executePayroll(...) ... {
    ...
+   for (uint256 i = 0; i < paymentTokens.length; i++) {
+        ...   initialBalances[i] = ...
+   }
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
-   for (uint256 i = 0; i < paymentTokens.length; i++) {
-        ...   initialBalances[i] = ...
-   }
    ...
    // send ETH / tokens to recepient
    ...
    for (uint256 i = 0; i < paymentTokens.length; i++) {
        ...
        if (...) {
            require(address(this).balance == initialBalances[i], ...);
         } else
-        if (IERC20(paymentTokens[i]).balanceOf(address(this)) >  initialBalances[i] ) {
+        if (IERC20(paymentTokens[i]).balanceOf(address(this)) != initialBalances[i] ) {
                revert("CS018");
         }
    }
}
```

Parcel: This issue is made redundant by the Proxy pattern and a redesign of `executePayroll()`.

Cantina Security: Verified.

3.2 Medium Risk

3.2.1 `deadline` to limit the validity of a transaction

Severity: Medium Risk

Context: PayrollManager.sol#L89-L96

Description: The function `encodeTransactionData()` hashes all the transaction data. Combined with the signed `root` this transaction stays valid as long as it has not been executed. If someone has incorrectly signed a transaction it currently can't be invalidated, see issue "Nonce invalidation functionality is missing". It would be helpful if the transaction has validity and because invalid after some time. This is also a general security precaution. Note: this is also present in the permit function, see ERC20Permit.sol#L49-L68

```
function encodeTransactionData(...) ... {
    return keccak256(abi.encode(to, tokenAddress, amount, payoutNonce));
}
```

Recommendation: Implement a `deadline` to limit the validity of a transaction.

Parcel: Adding an expiry date might create friction between signers while approving older / overdue payments. So we would like to get some feedback from first set of users before making a decision on adding this. A nonce invalidation function has been added.

Cantina Security: Acknowledged.

3.2.2 Nonce invalidation functionality is missing

Severity: Medium Risk

Context: PayrollManager.sol#L24

Description: The `PayrollManager` smart contract deals with nonces and uses them on-chain, but it does not have any functionality for invalidating a nonce. If there was some payroll transaction where all signatures and merkle roots & proofs were computed already, but for any reason (for example people found that the transaction was wrong) it needs to be invalidated, currently this is not possible as there is no nonce invalidation possibility. If this transaction has already gone into the mempool anyone can execute it since the `executePayroll` function lacks access control.

Recommendation: Add a nonce invalidation functionality to the `PayrollManager` smart contract. Here are a few possible ways to do it:

- the nonce is invalidated, by one signer (also has risk of DOS)
- the nonce is invalidated, by n signers (more complicated to keep track off)
- the signature of one signer for one leaf is invalidated (more complicated to keep track off)

Parcel: Any one of the approvers can invalidate the nonce, solved in PR 52.

Cantina Security: Verified.

3.2.3 Several issues with strict after balances check in `executePayroll()`

Severity: Medium Risk

Context: PayrollManager.sol#L139-L244

Description: The pattern to check for exact after balances in `executePayroll()` has some issues. Note: this assumes the fix of issue "Order of `execTransactionFromGnosis()` and `initialBalances()` is reversed" has been applied.

We see the following issues:

- if a token with a transfer fee or a rebalancing token is used, the balances will not match exactly;
- if one of the external contracts that are called (for example an `ETH` recipient or a callback of an `ERC777` token) is malicious, it could transfer some `ETH` or token back to the contract which would revert the call. This could severely hinder the payments;
- the balance check doesn't protect against over- or under-payments because a front runner could call the function `executePayroll()` with updated values for `paymentTokens[]` and `payoutAmounts[]` that satisfy the end check.

```
function executePayroll(...) ... {
    ...
    for (uint256 i = 0; i < paymentTokens.length; i++) {
         ...   initialBalances[i] = ...
   }
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
    ...
    // send ETH / tokens to recepient
    ...
    for (uint256 i = 0; i < paymentTokens.length; i++) {
        ...
        if (...) {
            require(address(this).balance == initialBalances[i], ...);
         } else
         if (IERC20(paymentTokens[i]).balanceOf(address(this)) != initialBalances[i] ) {
                revert("CS018");
         }
    }
}
```

Recommendation: Use the recommendations of the following issues:

- "Use separate contracts for each gnosis safe";
- "Token retrieval not linked to signed transactions";

- "ETH and tokens can get stuck".

Parcel: This has been redesigned using a proxy pattern in PR 53.

Cantina Security: Verified.

3.2.4 ETH and tokens can get stuck

Severity: Medium Risk

Context: PayrollManager.sol#L249

Description: The `Organizer/PayrollManager` contract can receive ETH, which is useful when receiving ETH from the safe. However, ETH might also be sent here accidentally. Also, there are several ways extra ETH/tokens could end up in the `Organizer/PayrollManager` contract:

- due to an error in the code;
- due to ETH/tokens being sent/returned to the contract. For example, when someone receives tokens, recognizes a mistake has been made and returns the tokens;
- due to airdrops. Finally, NFTs could also end up in the contract, although less likely to occur because it won't happen if safetransfer is used.

The code that enables receiving ETH:

```
receive() external payable {}
```

Recommendation: Consider making a function to retrieve ETH, tokens, and possibly NFTs. The owner then has to decide what to do with them. If the solution of "Use separate contracts for each gnosis safe" is used, the functions could return the funds to the safe.

Parcel: Solved by moving to proxy pattern and adding function `sweep()`, in PR 39

Cantina Security: Verified.

3.2.5 Use seperate contracts for each gnosis safe

Severity: Medium Risk

Context: Storage.sol#L34

Description: When several DAOs use the same contract for `Organizer/PayrollManager` it is difficult to separate the token flows. For a recipient who works for different daos it is difficult to see from which dao the tokens originated.

If a bad actor is also using the contract then tokens seem to originate from a bad actor, which might be flagged that way on chain explorers. The combining of token flows resembles a token mixer like tornado cash. If the `USDC/ USDT` of `Organizer/PayrollManager` would be blacklisted then all safes would have an issue.

If there are any vulnerabilities in the contract (see other issues) then funds could be mixed from different safes. This should be avoided if at all feasible. Also, potential users of the Parcel Payments protocol will be more inclined to use the product if funds are guaranteed to be separated.

Additionally having separate contracts for each safe will simplify the code:

- onboarding after offboarding can be implemented in a straightforward way;
- the mapping for safes is no longer necessary, which also saves gas by not having to access it.;
- function `executePayroll()` doesn't have to query the initial balances, it can just return any remaining balance;
- It is far easier to rescue stuck funds. See the issue "ETH and tokens can get stuck".

Also, it will help with the issue "Address restrictions in `AllowanceModule`".

Recommendation: Use a proxy pattern to deploy separate contracts for each gnosis safe.

Note: attention point is that there will be multiple instances of the `Organizer/PayrollManager` contract with multiple instances of `executePayroll()`, with its own `nonReentrant` modifier. So reentrancy between multiple instances would be possible. Because everything else is separated this doesn't induce further risks as far as we can see.

Parcel: Solved in PR 39.

Cantina Security: Verified.

3.2.6 Token retrieval not linked to signed transactions

Severity: Medium Risk

Context: PayrollManager.sol#L139-L244

Description: There is no check done that `tokenAddress[]` is one of the `paymentTokens[]`. So this way you can steal tokens that happen to be left in the `Organizer` contract. Note: normally there should not be any tokens left in the `Organizer` contract. Note: it still requires that sufficient approvers have signed for the non-standard token.

When applied to the "wrong" safe: initially ETH/tokens are transferred from the safe, which is later undone when it turns out no relevant signatures are added. However, in a block explorer, the failed transactions can be seen, which can scare the safe owner.

```
function executePayroll(...) ... {
    ...
    for (uint256 index = 0; index < paymentTokens.length; index++) {
        execTransactionFromGnosis(...);
    }
    ...
    if (tokenAddress[i] == address(0)) {
        ...
        (bool sent, bytes memory data) = to[i].call{  value: amount[i] }("");
    } else {
        ...
        IERC20(tokenAddress[i]).safeTransfer(to[i], amount[i]);
    }
    ...
}
```

Recommendation: Only retrieve ETH/Tokens that are linked to signed transactions.

This can be done in the following way:

- When calling `executePayroll()`: sort the transaction arrays (e.g. `to`, `tokenAddress`, `amount`, `payoutNonce`) on `tokenAddress[]` and `to`.
- Loop through all transactions:
  - check the record is sorted;
  - calculate the leaf and determine which transaction has sufficient approvals and which `payoutNonce[]` hasn't been used yet;
  - store the valid transactions in a memory array.
- Loop over all the valid transactions:
  - sum the `amount[]`s as long as they have the same `tokenAddress[]`;
  - if the sum > 0 then:
    - retrieve the summed amount of ETH/tokens for `tokenAddress[]` from the gnosis safe;
    - loop over all the tokens with the `tokenAddress[]`:
      - optionally combine all the transactions with the same `to`;
      - send the tokens to the `to` recipient;
    - there should be no more tokens left of type `tokenAddress[]` as you have first summed them.

Parcel: Solved in PR 53.

Cantina Security: Verified.

3.3 Low Risk

3.3.1 ETH recipients can block receiving ETH

Severity: Low Risk

Context: PayrollManager.sol#L139-L244

Description: The function `executePayroll()` does multiple ETH transfers and/or token transfers. If one of the ETH recipients reverts (for example it's a contract that can't receive ETH) the entire function `executePayroll()` is reverted and it has the be re-executed without that recipient.

A similar problem can occur when ERC777 tokens are used and the `tokensReceived` hook reverts.

```
function executePayroll(...) ... {
    ...
    if (tokenAddress[i] == address(0)) {
        ...
        (bool sent, bytes memory data) = to[i].call{value: amount[i]}("");
        require(sent, "CS007");
    } else {
        ...
        IERC20(tokenAddress[i]).safeTransfer(to[i], amount[i]);
    }
    ...
}
```

Recommendation: Consider ignoring ETH transfers that don't succeed. With regards to ERC777: these tokens are not used frequently. If necessary, a try - catch construction can be used to ignore failures.

Alternatively, be prepared for this situation in the offline code.

Parcel: Solved in PR 55 and PR 59.

Cantina Security: Verified.

3.3.2 Prefer two-step ownership transfer over a single-step pattern

Severity: Low Risk

Context: Organizer.sol#L126-L128

Description: The `Organizer` smart contract inherits from `Ownable` which has a single-step ownership transfer pattern. Also, since the contract does not want the `renounceOwnership` functionality, it overrides the method and forces it to revert on each call. Single-step ownership is not preferred because if for some reason (fat fingering or mistyping) you send the wrong address as the argument of the `transferOwnership` function, then the `owner` role will be lost forever, leaving all `onlyOwner` methods non-callable anymore.

Recommendation: Use Ownable2Step instead, this way you will also not need to override `renounceOwnership` as this method does not exist in `Ownable2Step`.

Parcel: After refactoring to Proxy Architecture, the Proxy contracts can ONLY be owned by the org safes. The Factory contract implements `Ownable2Step` to secure Parcel's ability to deploy new implementations.

Cantina Security: Verified.

3.3.3 Address restrictions in AllowanceModule

Severity: Low Risk

Context: AlowanceModule.sol#L305-L311

Description: The address of the Organizer/PayrollManager has some restrictions imposed by the AllowanceModule:

- uint48(address(Organizer)) may not be 0;
- uint48(address(Organizer)) may not be the same as another delegate contract.

This is unlikely to occur however someone might deliberately generate a conflicting gnosis safe address, and use that with addDelegate() to prevent adding the Organizer/PayrollManager contract. This would only be relevant if the same Organizer/PayrollManager contract is used on multiple chains and this attack is done before the Organizer/PayrollManager contract is deployed on a new chain.

Note: the solution for the issue "Use separate contracts for each gnosis safe" slightly enhances this issue. This probably slightly increased, however, if it occurs, then a new contract address can be generated easily.

Recommendation: Preferably implement the solution for the issue "Use separate contracts for each gnosis safe".

Parcel: This is solved in PR 39.

Cantina Security: Verified. Note: the salt in function onboard() is important to solve potential conflicts.

3.3.4 Array packedPayoutNonces can cause out of gas errors

Severity: Low Risk

Context: PayrollManager.sol#L24-L49, Organizer.sol#L93-L106, Storage.sol#L17-L22

Description: If (accidentally) a high payoutNonce is used then packPayoutNonce() will increase the orgs[safeAddress].packedPayoutNonces array which can take a lot of gas. It could also run out of gas. Additionally offboard() can also run out of gas while cleaning the array, which is done as part of delete orgs[msg.sender].

```
struct ORG {
    uint128 approverCount;
    uint128 approvalsRequired;
    mapping(address => address) approvers;
    uint256[] packedPayoutNonces;
}

function packPayoutNonce(...) ... {
    ...
    while (orgs[safeAddress].packedPayoutNonces.length <= slot) {
        orgs[safeAddress].packedPayoutNonces.push(0);
    }
}
function offboard() external {
    ...
    delete orgs[msg.sender];
    ...
}
```

Recommendation: Change packedPayoutNonces from an array to a mapping. This also simplifies the code, however, it is not easy to clean the mapping in ‘offboard(). Note: most other implementations of this pattern also use a mapping, for example: Uniswap SignatureTransfer.sol#L19.

```
struct ORG {
    uint128 approverCount;
    uint128 approvalsRequired;
    mapping(address => address) approvers;
-   uint256[] packedPayoutNonces;
+   mapping(uint256 => uint256) packedPayoutNonces;
}
```

Parcel: Solved in PR 57.

Cantina Security: Verified.

3.3.5 Potential out of gas in while loops

Severity: Low Risk

Context: Organizer.sol#L93-L106, ApproverManager.sol#L148-L166

Description: The while loops in offboard() and getApprovers() can run out of gas if a large number of approvers is added. This is unlikely to occur in practice because adding approvers is a privileged operation.

```
function offboard() external {
    ...
    while (currentApprover != SENTINEL_ADDRESS) {
        address nextApprover = orgs[msg.sender].approvers[currentApprover];
        ...
        currentApprover = nextApprover;
    }
}

function getApprovers( ... ) ... {
    ...
    while (currentOp != SENTINEL_ADDRESS) {
        ...
        currentOp = orgs[_safeAddress].approvers[currentOp];
        ...
    }
}
```

Recommendation: Consider removing the linked lists, see the issue "Consider removing linked list".

Parcel: The offboard function has been removed since we migrated to the Proxy architecture. The current setup is allowing us to get a list of approvers directly from the contract whereas if we go the graph way, we will have to consider all the add, remove and swap events on the proxy contract. Hence if there is no security risk involved, we would like to keep this approvers list as a linked list for now.

Cantina Security: Acknowledged.

3.3.6 Domain separator is missing security fields

Severity: Low Risk

Context: Signature.sol#L11

Description: EIP-712 defines multiple fields for the domain separator - name, version, chainId, verifyingContract and salt. While chainId and verifyingContract are present, the rest are not. While the EIP recommends including only the fields that make sense for your application, using the name and version fields seems sensible and can also help protect and secure the protocol by isolating signatures even more narrowly.

Recommendation: Add name and version fields to the domain separator to further isolate the protocol's signed messages from any other possible signed messages. Make sure signatures from different versions are not compatible.

Parcel: Solved in PR 61.

Cantina Security: Verified.

3.4 Gas Optimization

3.4.1 Unnecessary conversions from bytes to string in the Signature contract

Severity: Gas Optimization

Context: Signature.sol#L11, Signature.sol#L16

Description: In the Signature smart contract, the EIP712_DOMAIN_TYPEHASH and the PAYROLL_TX_TYPEHASH are calculated by hashing the encoding of their corresponding struct types using the keccak256 hash function. However, the strings are first converted to bytes before being hashed. Since these strings do not contain any non UTF-8 characters, this conversion is unnecessary and results in unnecessary gas consumption.

Recommendation: It is recommended to remove the conversion from bytes to string, as it is not needed for strings that only contain UTF-8 characters.

Parcel: Solved in PR 52.

Cantina Security: Verified.

3.4.2 Optimize byte operation in packPayoutNonce() and getPayoutNonce()

Severity: Gas Optimization

Context: PayrollManager.sol#L24-L79

Description: Both the functions packPayoutNonce() and getPayoutNonce() have the following code:

```
uint256 slotIndex = payoutNonce / 256;
uint256 bitIndex = payoutNonce % 256;
```

This can be optimized to save some gas.

Recommendation: Consider changing the code to:

```diff
-uint256 slotIndex = payoutNonce / 256;
+uint256 slotIndex = uint248(payoutNonce >> 8);
-uint256 bitIndex  = payoutNonce % 256;
+uint256 bitIndex  = uint8(payoutNonce);
```

For reference see: SignatureTransfer.sol#L142-L145.

Parcel: Solved in PR 57.

Cantina Security: Verified.

3.4.3 Reduce the number of calls to MerkleProof.verify()

Severity: Gas Optimization

Context: PayrollManager.sol#L139-L244

Description: The function executePayroll() loops through the roots to count the approvals. The MerkleProof.verify is a relatively expensive operation. This can be optimized in a few ways.

```
function executePayroll(...) ... {
    ...
    for (uint256 i = 0; i < to.length; i++) {
       ...
        for (uint256 j = 0; j < roots.length; j++) {
            if (MerkleProof.verify(proof[i][j], roots[j], leaf)) {
                ++approvals;
            }
        }
        if (approvals >= ...  && !getPayoutNonce(safeAddress, payoutNonce[i]) {
            ...
        }
        ...
    }
}
```

Recommendation: Consider the following optimizations:

- first, check if the `PayoutNonce` hasn't been used;
- `break` out of the loop once there are enough approvals;
- for each `to`, indicate which `roots` are relevant. This can be done via a bitmap or a list of index numbers for appropriate roots.

Parcel: Solved in PR 57.

Cantina Security: Verified.

3.4.4 Expensive operation can be optimised outside of for-loop

Severity: Gas Optimization

Context: Organizer.sol#L83

Description: In the for-loop of `Organizer::onboard` there is the following code:

```
orgs[safeAddress].approverCount++;
```

This is relatively expensive as it has to happen on each iteration. Since `orgs[safeAddress].approverCount` has a value of 0 before the loop starts you can just set it once, after the for-loop.

Recommendation: Instead of

```
orgs[safeAddress].approverCount++;
```

in the for-loop just do

```
orgs[safeAddress].approverCount = _approvers.length;
```

once after the loop.

Parcel: Solved in PR 39.

Cantina Security: Verified.

3.4.5 Caching variables can result in gas savings

Severity: Gas Optimization

Context: PayrollManager.sol#L188, PayrollManager.sol#L211

Description: In `PayrollManager::executePayroll` you can cache the `to.length` value in a variable, so it is not called 3 times in the start of the method and then multiple times in one of the for loops. You can also extract `orgs[safeAddress].approvalsRequired` to a variable in the same method, otherwise, you will do SLOAD operations in a for loop.

Recommendation: Extract variables to cache the values used.

Parcel: Solved in PR 57. The second part is solved by implementing the Proxy architecture.

Cantina Security: Verified.

3.4.6 Multiple payments to same recipient can be combined

Severity: Gas Optimization

Context: PayrollManager.sol#L139-L244

Description: The function `executePayroll()` can send multiple transactions with the same `tokenAddress` to the same `to` recipient. This costs unnecessary gas. Optimizing this could be relevant if recipients are doing multiple tasks within the same DAO and get separate payments for each task.

Recommendation: When calling `executePayroll()`: sort the transaction arrays (e.g. `to`, `tokenAddress`, `amount`, `payoutNonce`) on `tokenAddress[]` and `to`. Then sum all the payments for the same `tokenAddress` to the same `to` and transfer them in one `safeTransfer()`. See for further details on the algorithm in the issue "Token retrieval not linked to signed transactions".

Note: this does make the code more complicated Note: for the recipient, it is slightly more difficult to seperate and verify the incoming token amounts.

Parcel: Each payout event counts towards an invoice payment and we have future plans to extend the platform and contracts based on these events.

Cantina Security: Acknowledged.

3.4.7 Move code outside `if` and `else` in `executePayroll()`

Severity: Gas Optimization

Context: PayrollManager.sol#L139-L244

Description: In function `executePayroll()`, a call is done to `packPayoutNonce()` both in an `if` and an `else` clause, with the same parameters. This can be put before the `if` to simplify the code and save some gas on the contract deployment.

```
function executePayroll(...) ... {
    ...
    if (tokenAddress[i] == address(0)) {
        packPayoutNonce(safeAddress, payoutNonce[i]);
        ...
    } else {
         packPayoutNonce(safeAddress, payoutNonce[i]);
         ...
    }
    ...
}
```

Recommendation: Consider changing the code to:

```
function executePayroll(...) ... {
    ...
+   packPayoutNonce(safeAddress, payoutNonce[i]);
    if (tokenAddress[i] == address(0)) {
-       packPayoutNonce(safeAddress, payoutNonce[i]);
        ...
    } else {
-       packPayoutNonce(safeAddress, payoutNonce[i]);
        ...
    }
    ...
}
```

Parcel: No longer relevant due to implementation of a try-catch pattern implemented in PR 52.

Cantina Security: Verified.

3.4.8 Redundant event parameters in event `OrgOnboarded`

Severity: Gas Optimization

Context: Organizer.sol#L15-L19

Description: The event `OrgOnboarded` contains both `approvers` and `approvers2`. Both are not needed because the function `onboard()` also emits an `ApproverAdded` for each approver.

```
event OrgOnboarded(
    address indexed orgAddress,
    address[] indexed approvers,
    address[] approvers2
);
```

Recommendation: Consider changing the event to:

```
event OrgOnboarded(
    address indexed orgAddress,
-    address[] indexed approvers,
-    address[] approvers2
);
```

Parcel: Solved in PR 39.

Cantina Security: Verified.

3.4.9 Extracting a storage pointer can result in a gas optimisation

Severity: Gas Optimization

Context: ApproverManager.sol#L25, ApproverManager.sol#L59, ApproverManager.sol#L92, ApproverManager.sol#L132, ApproverManager.sol#L148, PayrollManager.sol#L38-L48, PayrollManager.sol#L68-L77, PayrollManager.sol#L117, PayrollManager.sol#L152, PayrollManager.sol#L211-L224, Organizer.sol#L44-L85, Organizer.sol#L97-L104

Description: In multiple places in the Organizer, ApproverManager and PayrollManager code there are storage reads to the orgs mapping. Extracting a storage pointer to the mapping will result in gas savings.

Recommendation: Preferably implement the recommendation of issue "Use separate contracts for each gnosis safe".

Alternatively extract a storage pointer like `ORG storage organization = orgs[msg.sender];` in all methods of ApproverManager to save gas. This could even be combined with `_onlyOnboarded()`:

```
function _onlyOnboarded(address _safeAddress) internal view returns(ORG storage currentOrg) {
   currentOrg = orgs[_safeAddress];
   require(currentOrg.approverCount != 0,"CS009");
}
```

Parcel: This is no longer relevant due to the implementation of the proxy architecture.

Cantina Security: Verified.

3.4.10 Never updated variable can be `immutable`

Severity: Gas Optimization

Context: Storage.sol#L25

Description: The ALLOWANCE_MODULE variable in Storage is only set in the constructor and never updated, so it can be turned into an immutable as a gas optimization, since immutables are not stored in storage.

Recommendation: Do the following change:

```
- address ALLOWANCE_MODULE;
+ address immutable ALLOWANCE_MODULE;
```

Note: this is especially helpful for proxy-based contracts: immutable can be used there as the values are "hardcoded" in the code.

Parcel: This variable has been made a constant now.

Cantina Security: Verified.

3.4.11 Use Solidity's custom errors to save gas

Severity: Gas Optimization

Context: ApproverManager.sol#L67

Description: The codebase uses `require` statements with strings, where the strings are sometimes some code like `CS003` while other times they are just error messages which are not consistent. Also, different checks use the same error codes, which shouldn't be the case. Using custom errors is much more gas efficient as you can write descriptive and proper names without paying more gas for it.

Recommendation: Replace all `require` statements with Solidity custom errors for better UX and gas savings.

Parcel: Solved in PR 50.

Cantina Security: Verified.

3.4.12 Use `calldata` instead of `memory` for external function array parameters

Severity: Gas Optimization

Context: PayrollManager.sol#L141-L149

Description: Using `memory` for an array function parameter is costly as it copies the whole array from `calldata` to `memory`. This is especially costly in `executePayroll` as there are many array parameters so each of their elements will have to be copied, resulting in big gas waste, while that is not really needed for the application. Note: function `validateSignatures()` can also be updated to `calldata` once `executePayroll` is updated.

Recommendation: Change the `memory` keyword for all array parameters in `executePayroll` to `calldata`, Do the same in `validateSignatures()`. Make sure to not try to change the function arguments in its body (if you need that only then you can copy an array to memory).

Parcel: Those arrays cannot be marked as `calldata` since we are facing `stack too deep error` when we convert these arguments to calldata.

Cantina Security: Acknowledged.

3.4.13 Caching the `domainSeparator` will optimise gas

Severity: Gas Optimization

Context: Signature.sol#L27

Description: The current implementation of `Signature::getDomainSeparator` recomputes the domain separator on each method call. It is a smart optimization to cache the value of the domain separator as it should never change unless the `chain.id` changes which should be a rare occurrence. This will result in gas efficiency when validating payroll transaction hashes.

Recommendation: Cache the domain separator value and only reevaluate it when `chain.id` has changed. You can see an example of this is here

Parcel: Solved in PR 52.

Cantina Security: Verified.

3.5 Informational

3.5.1 Function name removeApprover() doesn't show all functionality

Severity: Informational

Context: ApproverManager.sol#L25-L84

Description: Both functions addApproverWithThreshold() and removeApprover() allow changing the threshold. However removeApprover() doesn't show this in the name of the function.

```
function addApproverWithThreshold(...,  uint128 threshold) ... {
    ...
}
function removeApprover(..., uint128 threshold) ... {
    ...
}
```

Recommendation: Consider changing removeApprover() to removeApproverWithThreshold().

Parcel: Solved in PR 40.

Cantina Security: Verified.

3.5.2 Synchronisation of signers

Severity: Informational

Context: PayrollManager.sol#L104-L124

Description: The function validateSignatures() checks all signers are different, which is important for the security of the protocol. However, this also means that only one root per signer can be used. If signers don't synchronize their signing of roots, then finding the relevant roots might get complicated.

For example, assume the following situation:

- approver 1 has approved the payoutNonce 1,2,3 in root A and payoutNonce 4,5,6 in root B
- approver 2 has approved the payoutNonce 1,2,3,4,5,6 in root C

Then two calls to executePayroll() are required to do all the payouts. Note: the more approvers the more complicated this might get.

```
function validateSignatures(...) ... {
    ...
    for (uint256 i = 0; i < roots.length; i++) {
        address signer = validatePayrollTxHashes(roots[i], signatures[i]);
        // Check if the signer is an approver & is different from the current approver
        require(... , signer > currentApprover, "CS014");
        ...
    }
}
```

Recommendation: If this problem is indeed relevant then the following solution can be used: Have one hash tree per approver that is expanded with every payoutNonce. This way all previously signed payoutNonces are always accessible. This probably requires a different hash approach, like this one: axic/eth2-deposit-contract/blob/master/deposit_contract.sol#L101 .

Parcel: We are generating the trees and roots off-chain when the approvers approve payouts. We are planning to expand trees with every additional approval for an epoch(month) so that when it's time for execution, we will only have one root per approver.

Cantina Security: Acknowledged.

3.5.3 Use address(0) for the zero address

Severity: Informational

Context: PayrollManager.sol#L268

Description: In PayrollManager:: execTransactionFromGnosis one of the arguments of executeAllowanceTransfer looks like

```
0x0000000000000000000000000000000000000000
```

While the intention was obviously to send the zero address as the argument, it is best to use the address(0) approach as it is easier for devs and auditors to be certain it is the zero address.

Recommendation: Do the following change in the arguments list of the executeAllowanceTransfer call

```
- 0x0000000000000000000000000000000000000000,
+ address(0),
```

Parcel: Solved in PR 40.

Cantina Security: Verified.

3.5.4 Function _onlyOnboarded() not used everywhere

Severity: Informational

Context: PayrollManager.sol#L139-L244, ApproverManager.sol#L192-L194

Description: The function executePayroll() has an explicit check for approverCount != 0, while most other functions use _onlyOnboarded(). Using _onlyOnboarded() everywhere makes the code more consistent and easier to maintain.

```
function executePayroll(...) ... {
    // check if safe is onboarded
    require(orgs[safeAddress].approverCount != 0, "CS009");
    ...
}
function _onlyOnboarded(address _safeAddress) internal view {
    require(orgs[_safeAddress].approverCount != 0, "CS009");
}
```

Recommendation: Change executePayroll() to also use _onlyOnboarded().

Parcel: Solved in PR 39. The _onlyOnboarded modifier has been removed since separate contracts are used for each gnosis safe.

Cantina Security: Verified.

3.5.5 Enforce minimum number of approvers

Severity: Informational

Context: ApproverManager.sol#L132-L141

Description: The current minimum number of approvers is 1 and is determined by changeThreshold(). Some safe owners might want to enforce a higher minimum to prevent the accidental removal of approvers.

```
function changeThreshold(uint128 threshold) public whenNotPaused {
    ...
    // There has to be at least one Safe Approver.
    require(threshold >= 1, "CS015");
    ...
}
```

Recommendation: In the onboard() function, add a parameter that defines the minimum number of approvers.

Parcel: It's good to have a feature, but adds one more setting to be managed by DAO operators. I think it's too much complexity for users with very little upside. Probably we can add this in a future version.

Cantina Security: Acknowledged.

3.5.6 Use the `delete` keyword instead of assigning default values

Severity: Informational

Context: ApproverManager.sol#L123

Description: There is no need to assign default values to storage slots that you want to clear, as there is the `delete` keyword for this. You use it in some places but not everywhere - use it everywhere for this use case.

Recommendation:

```
- orgs[msg.sender].approvers[oldApprover] = address(0);
+ delete orgs[msg.sender].approvers[oldApprover];
```

Parcel: Solved in PR 40.

Cantina Security: Verified.

3.5.7 Events parameters order is not consistent

Severity: Informational

Context: ApproverManager.sol#L16

Description: The `RemovedApprover` & `ChangedThreshold` events in `ApproverManager` have the `safeAddress` parameter as the second parameter in the event and also are omitting the `indexed` keyword. This is inconsistent with other events in the codebase.

Recommendation: Move the `safeAddress` parameter to be the first one in the `RemovedApprover` & `ChangedThreshold` events and also add the `indexed` keyword to it.

Parcel: The events are now modified to emit only the added/removed approver address only.

Cantina Security: Verified.

3.5.8 Increase disallowlist for approvers

Severity: Informational

Context: Organizer.sol#L37-L87

Description: The function `onboard()` checks that the `approver` isn't one of the disallowed addresses. This list could be expanded with the following address to prevent mistakes and to maximize seperation of duties:

- `owner`
- `msg.sender` (e.g. the gnosis safe)

```
function onboard(...) ... {
    ...
    require(
        approver != address(0) &&        // approver address cannot be null
        approver != SENTINEL_ADDRESS && // approver address cannot be SENTINEL.
        approver != address(this) &&     // approver address cannot be same as contract.
        ... );
    ...
}
```

Recommendation: Consider expanding the list of disallowed addresses in `onboard()`. This also has to be updated in `addApproverWithThreshold()` and `swapApprover()`.

Parcel: Solved in PR 52.

Cantina Security: Verified.

3.5.9 Use a more recent Solidity version

Severity: Informational

Context: Organizer.sol#L12

Description: The compiler version used `0.8.9` is a bit old (current version is `0.8.19`). This version was released more than a year ago and there have been four applicable bug fixes to this version since then. While it seems that those bugs don't apply to the Parcel project, it is advised to update the compiler to a newer version.

Recommendation: Upgrade the codebase to a more recent compiler version.

Parcel: We have locked the Pragma to `0.18.17` in PR 39.

Cantina Security: Verified.

3.5.10 Emit threshold event on `onboard`

Severity: Informational

Context: Organizer.sol#L57

Description: In `onboard` you set the initial threshold of approvals required for an organisation but there is no event emitted, even though you emit one for each approved added.

Recommendation: Emit the `ChangedThreshold` event on setting the `approvalsRequired` in `onboard`.

Parcel: Solved in PR 39 via `event OrgSetup()`.

Cantina Security: Verified.

3.5.11 Bundle arrays in `executePayroll()`

Severity: Informational

Context: PayrollManager.sol#L139-L244

Description: The function `executePayroll()` has several array parameters. This requires several length checks to sanitize the input. The arrays could be bundled in a struct of arrays. This improves the readability of the rest of the code.

```
function executePayroll(
    address safeAddress,
    address[] memory to,
    address[] memory tokenAddress,
    uint128[] memory amount,
    uint64[] memory payoutNonce,
    bytes32[][][] memory proof,
    bytes32[] memory roots,
    bytes[] memory signatures,
    address[] memory paymentTokens,
    uint96[] memory payoutAmounts ) ... {
}
```

Recommendation: Consider bundling related arrays in an array of structs. This could be the following bundles:

- `to`, `tokenAddress`, `amount`, `payoutNonce`
- `proof`, `roots`, `signatures`
- `paymentTokens`, `payoutAmounts`

Parcel: This change will increase readability, but we intend to proceed with the same pattern for now. We could implement this in a future release.

Cantina Security: Verified.

3.5.12 Simplify check in removeApprover()

Severity: Informational

Context: ApproverManager.sol#L59-L84

Description: The function removeApprover() has a check the make sure the approverCount won't be below threshold. This is slightly difficult to read and can be simplified.

```
function removeApprover(...) ... {
    ...
    require(orgs[msg.sender].approverCount - 1 >= threshold, "CS016");
    ...
    orgs[msg.sender].approverCount--;
    ...
}
```

Recommendation: Consider changing the code to:

```
function removeApprover(...) ... {
    ...
-   require(orgs[msg.sender].approverCount - 1 >= threshold, "CS016");
    ...
    orgs[msg.sender].approverCount--;
+   require(orgs[msg.sender].approverCount >= threshold, "CS016");
    ...
}
```

Parcel: Simplified in PR 50.

Cantina Security: Verified.

3.5.13 Consider removing linked list

Severity: Informational

Context: Organizer.sol#L12

Description: The approvers are stored using a linked list. This has the following disadvantages:

- the functions removeApprover() and swapApprover() require supplying a previous approver;
- if the linked list gets to large out of gas errors could occur, see issue "Potential out of gas in while loops";
- the linked list logic is relatively complicated and difficult to verify.

The advantages are limited:

- it allows getApprovers(), however, because it is intended to be used via the Parcels off-chain payroll product, this can also be done offchain (or for example via TheGraph);
- it allows deleting the mapping in offboard(). However offboard() has limited use, see also issue "offboard() clears all nonces" and "Use separate contracts for each gnosis safe";
- it has been added to help verify the recovered signatures of root hashes and assumes the approvers list is sorted, however:
  - swapApprover() will distort the sort order;
  - offchain sorting of roots[] and signatures[] on signer address is sufficient for executePayroll() and validateSignatures() to function.

Recommendation: Consider removing the linked list to simplify the code.

For example the functions addApproverWithThreshold() and removeApprover() could be simplified to the following:

```
function addApproverWithThreshold(address approver,uint128 threshold) external whenNotPaused {
    _onlyOnboarded(msg.sender);
    require(approver != address(0) && approver != address(this),"CS003");
    require(orgs[msg.sender].approvers[approver] == false, "CS002"); // No duplicate approvers allowed.
    orgs[msg.sender].approvers[approver] = true;
    orgs[msg.sender].approverCount++;
    emit ApproverAdded(msg.sender, approver);
    if (threshold != orgs[msg.sender].approvalsRequired)
        changeThreshold(threshold);
}
function removeApprover(address approver,uint128 threshold) external whenNotPaused {
    _onlyOnboarded(msg.sender);
    require(orgs[msg.sender].approverCount - 1 >= threshold, "CS016");
    require(approver != address(0) ,"CS003");
    require(orgs[msg.sender].approvers[approver] == true, "..."); // Only remove if present
    delete orgs[msg.sender].approvers[approver];
    orgs[msg.sender].approverCount--;
    emit RemovedApprover(approver, msg.sender);
    if (threshold != orgs[msg.sender].approvalsRequired)
        changeThreshold(threshold);
}
```

Parcel: The current setup is allowing us to get a list of approvers directly from the contract whereas if we go the graph way, we will have to consider all the `add`, `remove` and `swap` events on the proxy contract. Hence if there is no security risk involved, we would like to keep this approvers list as a linked list for now.

Cantina Security: Acknowledged.

3.5.14 Naming convention for interfaces is not followed

Severity: Informational

Context: AllowanceModule.sol#L4

Description: The `AllowanceModule` interface does not follow the standard naming convention for Solidity interfaces, which begins with an `I` prefix. This inconsistency can make it harder for developers to understand the purpose and usage of the contract.

Recommendation: It is recommended to prefix the `AllowanceModule` interface with `I`:

```
+interface IAllowanceModule {
-interface AllowanceModule {
    function executeAllowanceTransfer(
        . . .
    ) external;
}
```

Parcel: Solved in PR 40.

Cantina Security: Verified.

3.5.15 Function ordering does not follow the Solidity style guide

Severity: Informational

Context: PayrollManager.sol#L14

Description: The recommended order of functions in Solidity, as outlined in the Solidity style guide, is as follows: constructor(), receive(), fallback(), external, public, internal and private. However, this ordering isn't enforced in the PayrollManager.sol contract.

Recommendation: It is recommended to follow the recommended order of functions in Solidity, as outlined in the Solidity style guide.

Parcel: Solved in PR 51.

Cantina Security: Verified.

3.5.16 offboard() doesn't delete everything

Severity: Informational

Context: Organizer.sol#L93-L106

Description: The function offboard() doesn't remove orgs[msg.sender].approvers[SENTINEL_ADDRESS];. It doesn't pose a security risk though as in onboard() this slot will be overwritten anyway with the current implementation.

```
function offboard() external {
    ...
    address currentApprover = orgs[msg.sender].approvers[SENTINEL_ADDRESS];
    while (currentApprover != SENTINEL_ADDRESS) {
        address nextApprover = orgs[msg.sender].approvers[currentApprover];
        delete orgs[msg.sender].approvers[currentApprover];
        currentApprover = nextApprover;
    }
    ...
}
```

Recommendation: Depending on the solution chosen for issue "offboard() clears all nonces", add the following:

```
delete orgs[msg.sender].approvers[SENTINEL_ADDRESS];
```

Parcel: Offboarding has been removed as we are moving to Proxy architecture.

Cantina Security: Verified.

3.5.17 Inheritance is possibly not needed in the protocol

Severity: Informational

Context: ApproverManager.sol#L8

Description: The inheritance tree of the protocol is a strange one - both ApproverManager and PayrollManager inherit from Storage and Pausable while Organizer inherits from both ApproverManager and PayrollManager.

Also, some contracts are declared abstract while others that should also not be deployed separately are not. As the logic in all contracts is tightly coupled anyway, implementing it in separate contracts seems like not a great fit, so using one big smart contract should be easier to grasp, audit, and reason about in the protocol.

Recommendation: Merge all contracts into one bigger contract since the logic is tightly coupled.

Parcel: Solved in PR 60.

Cantina Security: Verified.

3.5.18 No need to use an assembly block to get the chain ID

Severity: Informational

Context: Signature.sol#L20

Description: Currently the `getChainId` method in `Signature` uses an assembly block to get the current chain id. This is not needed since there is a global variable for this already.

Recommendation: Remove the `getChainId` function and just use `block.chainid` instead.

Parcel: Solved in PR 54.

Cantina Security: Verified.

3.5.19 Redundant code, logic, checks and misnamed variables

Severity: Informational

Context: PayrollManager.sol#L69, ApproverManager.sol#L138, ApproverManager.sol#L11, ApproverManager.sol#L176, ApproverManager.sol#L188, ApproverManager.sol#L10, ApproverManager.sol#L158, Organizer.sol#L48, Organizer.sol#L56, Organizer.sol#L69, Organizer.sol#L22

Description: There are redundant checks, code, and misnamed variables throughout the codebase. Either remove or improve all of them.

Recommendation:

- The `orgs[safeAddress].packedPayoutNonces.length == 0` check in `PayrollManager::getPayoutNonce` is not needed since the other check in the `if` (`orgs[safeAddress].packedPayoutNonces.length <= slotIndex`) contains it anyway, so you can remove it.
- Change `require(threshold >= 1, "CS015");` to `require(threshold != 0, "CS015");` in `ApproverManager:: changeThreshold` for simplicity.
- There is no need for the `_onlyOnboarded(_safeAddress);` check in either `getApproverCount` or `getThreshold` in `ApproverManager` as they are only externally called and will just return 0 if the `_safeAddress` is not onboarded, so checks can be removed.
- The `ApproverRemoved` event in `ApproverManager` is not used and can be removed.
- Rename the `operator` parameter in the `ApproverAdded` event in `ApproverManager` to `approver` for naming consistency.
- Rename the `currentOp` variable in `ApproverManager:: getApprovers` to `currentApprover` for naming consistency.
- There is no need for the `require(_approvers.length > 0, "CS000");` check in `Organizer::onboard` as the other `require` statements contain it already, so it can be removed.
- No need to do `orgs[safeAddress].approverCount = 0;` in `Organizer::onboard` since this value is already enforced to be 0 from the first `require` statement in the method, you can remove the `orgs[safeAddress].approverCount = 0;` code.
- The `currentapprover != approver,` part of the `require` statement in the loop in `Organizer::onboard` is redundant and can be removed since it is checked by the ` No duplicate approvers allowed.` check below.
- Rename the `orgAddress` parameter of the `OrgOffboarded` event in `Organizer` to `safeAddress` for naming consistency.

Parcel: Solved in various commits.

Cantina Security: Verified.

3.5.20 Typos, redundancies, errors and missing data in comments & NatSpec docs

Severity: Informational

Context: Signature.sol#L37, Signature.sol#L25, PayrollManager.sol#L15, ApproverManager.sol#L135, Organizer.sol#L14-L21, ApproverManager.sol#L87, Organizer.sol#L8

Description: Many comments and NatSpec docs have errors or redundancies in them and should be improved.

Recommendation:

- The NatSpec of `validatePayrollTxHashes` is missing `@return` parameter.
- Typo in the NatSpec of `getDomainSeparator` - `seperator` should be `separator`
- Comment on `// Payroll Functions` in `PayrollManager` should be after the `using` statement or can be removed
- Change the `// Validate that threshold is smaller than number of approvers.` comment to `// Validate that threshold is less than or equal to the number of approvers.`
- Remove comments on events in `Organizer` that just duplicate the comment name
- The `/// This can only be done via a Multisig transaction.` part of the `ApproverManager::swapApprover` NatSpec is not 100% correct as it is not enforced that the caller of the `onboard` function is indeed a multisig - it can be an EOA. Update or remove this part of the NatSpec
- Typo in the NatSpec of `Organizer`: `Orgss` should be `Orgs`
- Replace all the occurrences of `ETH/Ether` with `Native Token`, since in the docs it's mentioned that: `The contracts are written in Solidity version 0.8.9 and are compatible to run on all EVM-based chains`

Parcel: Those issues are fixed in several commits.

Cantina Security: Verified.