CHAIN TROOPERS

Energy Web Foundation
Worker Pallet
Security Assessment Report

May 23rd, 2025
Version 1.1

CONFIDENTIAL

Table of Contents

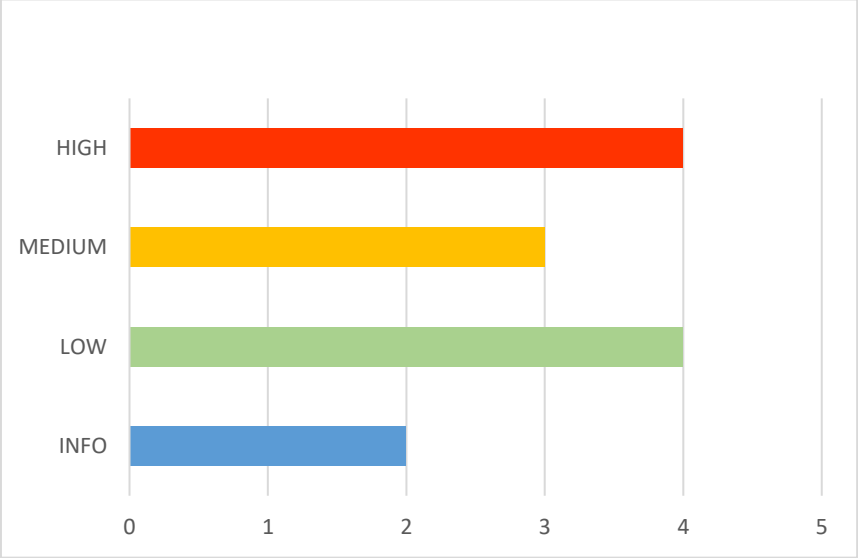
1 Executive Summary

1.1 Introduction

The report contains the results of Energy Web Worker solution pallet security assessment that took place from January 6th, 2025, to January 24th, 2025. The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

In total, the team identified eleven (11) vulnerabilities. There were also two (2) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity. A retesting phase was carried out on May 20th, 2025, and the results are presented in Section 6

1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to three (3) Information Disclosure and one (1) Data Validation issues of HIGH risk. Regarding the Information Disclosure issues, it was identified that the *submit_solution_result* extrinsic was emitting critical information regarding the outcome of the vote in 3 different instances. This would allow workers to vote the correct results without the need to conduct the actual work needed as part of the normal flow of the application. As a result, these workers would benefit from the voting rewards when they shouldn't.

In reference to the High-risk Data Validation issue, it was identified that the *next_voters_batch* function was processing vote rewards using the current set of operators rather than the set of operators that voted in the round leading to the loss of rewards for the operators affected.

The in-scope components were also affected by one (1) Data Validation, one (1) Access Control and one (1) and Business Logic vulnerabilities of MEDIUM risk.

In one case, an operator could vote and unsubscribe without implications or having their vote removed. In another case, when nominated workers are in place, a worker who leaves the subscription pool without voting could brick the voting round. In the final MEDIUM risk issue, we see the partial implementation of the performance bonus feature which while it could be accepted by the system the amount wouldn't be reserved from the registrar's wallet.

There were also four (4) vulnerabilities of LOW risk and two (2) findings of no-risk (INFORMATIONAL). Chaintroopers recommend the immediate mitigation of all HIGH and MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

1.2.1 Retesting Results

Results from resting carried out on May 2025, determined that four (4 out of 4) reported HIGH risk issues were sufficiently addressed.

Furthermore, three (3 out of 3) and four (4 out of 4) MEDIUM and LOW risk issues were sufficiently addressed.

Of the remaining two issues one (1 out of 2) informational issues was accepted as risk to the business and the remaining one is OPEN. More information can be found in Section 6.

1.3 Summary of Findings

The following findings were identified in the examined source code:

Vulnerability Name	Status	Retest Status	Page
lib.rs/submit_solution_result emits the voting round settled result	HIGH	CLOSED	16
lib.rs/submit_solution_result result bruteforcing on Settled voting rounds	HIGH	CLOSED	18
lib.rs/submit_solution_result emits voting results	HIGH	CLOSED	19
data_struct:next_voters_batch for non-nominated worker case is not assessing a snapshot	HIGH	CLOSED	20
lib.rs/calculate_deposit_fee does not reserve bonus_reward from registrar	MEDIUM	CLOSED	23
The length of nominated_workers is not dynamically assessed during a voting round	MEDIUM	CLOSED	24
lib.rs/unsubscribe_from_solution_group is not taking into account current voting periods	MEDIUM	CLOSED	26
lib.rs/remove_allowed_operator can be blocked by malicious operators	LOW	CLOSED	28
voting-rs:get_nomination snapshot not properly validating user stake	LOW	CLOSED	30
lib.rs/remove_allowed_operator relies on determine_user_overall_stake	LOW	CLOSED	31
lib.rs/raise_group_rewards fails silently	LOW	CLOSED	32
Outdated and Vulnerable dependencies in-use at "Cargo.lock"	INFO	INFO	33

voting_threshold_percent allows for minority voting	INFO	Risk Accepted	36

2 Assessment Description

2.1 Target Description

The Energy Web worker solution pallet is a system which allows for the registration of specific solutions and solution groups. These groups in return allow operators to subscribe to them by staking funds and participating in a voting system. To vote in this system, the operator needs to register a worker and the worker to conduct some processing work.

Workers who vote in this system would see some additional rewards if they have conducted the work properly.

2.2 In-Scope Components

The following pallets were in-scope for this assessment:

- *data_structs*
- *lib*
- *rewards*
- *solution*
- *stake_manager*
- *vote_processing*
- *voting*

The components are located at the following URL:

https://github.com/energywebfoundation/ewx-worker-solution-pallet

Component	Commit Identifier
https://github.com/energywebfoundation/ewx-worker-solution-pallet	*6d0ffd0687ebd85e6d629ce4a9c23f720955e85e*

A retesting phase was carried out on May 20th, 2025.

Component	Commit Identifier
https://github.com/energywebfoundation/ewx-worker-solution-pallet	*9d57d8a5d0637e00497d253461fe6d18bc3b73e2*

2.3 Assessment limitations

The following features are planned for future development hence could not be part of the assessment. In case a feature is partially implemented, and a security vulnerability is present it is reported as an actual vulnerability of the system.

- Top performance bonus, this is a feature which would reward a worker with the highest amount of correct vote submissions. This feature is partially implemented.
- Slashing mechanism, with this mechanism voters who abstain from a vote or submit an incorrect result will have part of their stake slashed. This feature is not implemented.
- Commit reveal, the voting system currently doesn't have a commit reveal approach implemented but it is part of future plans.

3 Methodology

3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

3.2 Smart Contracts

The testing methodology used is based on the empirical study "Defining Smart Contract Defects on Ethereum" by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo and T.

Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in “Security Considerations” section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement
- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment
- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

In Substrate Pallets, the list of vulnerabilities that are identified also includes:

- Static or Erroneously Calculated Weights
- Arithmetic Overflows
- Unvalidated Inputs
- Runtime Panic Conditions
- Missing Storage Deposit Charges
- Non-Transactional Dispatch Functions
- Unhandled Errors &Unclear Return Types
- Missing Origin Authorization Checks

4 Scoring System

4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

Rating	CVSS Score
None/Informational	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

5 Identified Findings

5.1 High Severity Findings

5.1.1 lib.rs/submit_solution_result emits the voting round settled result

Description	HIGH

The consensus result is broadcasted before the end of the voting period allowing this way for operators to learn the consensus result and vote for it in the remainder of the voting round.

The unsigned extrinsic *lib.rs/submit_solution_result*, is making changes in the storages *Votes* and *Voters*. In addition to that, when consensus has been reached along with setting the *VotingRoundStatus* to *Settled*, an event will be emitted with the result of the consensus.

```
// Broadcast event

Self::deposit_event(Event::VotingRoundSettled {

  namespace: solution_namespace.to_owned(),

  id: voting_round_id.to_owned(),

  result: result.to_owned(),

 });
```

While this behavior is setting the consensus to a specific value it does not end the voting round, as this will happen once the relevant block has been reached. This allows for more votes to go through.

The behavior of the extrinsic for new votes will now go through the following match statement and will allow for further registration of correct votes.

```
match voting_round_info.status {

    ...
```

```
    VotingRoundStatus::Settled(outcome) => {

        ensure!(outcome == result, Error::<T>::WrongVotingResult);

        // TODO is this something we need to raise offense for?
```

Impact

This behavior allows workers to provide the correct solutions to the voting round without being required to do the actual work. Workers can monitor the network for the event emission which contains the solution that they need to provide and just copy the result. While they can't affect the consensus they will be rewarded as honest voters.

Recommendation

The consensus result should be emitted when the voting period has finished and not before as to not interfere with the voting process.

5.1.2 lib.rs/submit_solution_result result bruteforcing on Settled voting rounds

Description HIGH

When a voting round is Settled but not expired, workers can submit and essentially bruteforce the right solution as calling the libs.rs/submit_solution_result extrinsic with an incorrect result will simply throw an error and not punish the malicious effort.

```
match voting_round_info.status {

    ...

    VotingRoundStatus::Settled(outcome) => {

        ensure!(outcome == result, Error::<T>::WrongVotingResult);

        // TODO is this something we need to raise offense for?
```

Since this extrinsic is unsigned, there is no cost for the malicious worker to brute force the solution.

Impact

Once the voting round has been settled, workers can utilize this behavior to identify the correct consensus result and participate in the voting rewards as honest voters. Since the extrinsic is unsigned there is no cost to the user to call the function and there is no penalty for a worker to abuse this behavior.

Recommendation

Since the current behavior allows for correct votes to go through even if the consensus threshold has been reached, it is advised to allow and log incorrect votes as well.

5.1.3 lib.rs/submit_solution_result emits voting results

Description HIGH

The extrinsic *lib.rs/submit_solution_result* is currently emitting an event of every successful vote going through with the result of the vote. Operators monitoring the network will be able to identify which votes go through and predict the outcome of the vote.

```
lib.rs/submit_solution_result

Self::deposit_event(Event::SolutionResultSubmitted {

     solution_namespace,

     voting_round_id,

     result,

     operator,

     worker,

     reward_period_index: period.index,

});
```

Even in the scenario where a commit reveal approach is implemented it is still plausible even though unlikely for this issue to still be present.

Impact

Operators/Workers can monitor the network, count the votes and vote only when the consensus is clear to maximize their chances of getting the vote correct and participating to the rewards phase.

Recommendation

It is advised to not broadcast the results of each vote prior to the end of the voting round.

5.1.4 data_struct:next_voters_batch for non-nominated worker case is not assessing a snapshot

Description	HIGH

The current implementation of *data_struct.rs/next_voters_batch* for the case where there are non-nominated workers, allows for the extraction of the current list of staked operators and not the list of staked operators at the time of voting round.

This function is triggered as part of the *lib.rs/on_idle* function which is not certain to run at the end of every block. If there isn't sufficient weight remaining at the end of the block execution, the function will skip its operation.

on_idle is preferred to be used for non-critical operations and opportunistic processing.

The *next_voters_batch* is processing batches of voters at a max number of *VOTE_PROCESSING_BATCH_SIZE* (which is currently set to 128). The number of voters/entries in *SolutionGroupStakeRecords* is not limited to this number which allows for the logical assumption that a voting round will need more than one call to the *lib.rs/on_idle* function to fully process its rewards. This further enhances the possibility of this issue occurring as the processing may take multiple blocks/calls to the *on_idle* function.

In this case, if an operator has voted, the voting round has finished and then the operator has unsubscribed from the solution, they may not be included in the processing round as the *lib.rs/on_idle* function will take a later copy of the *SolutionGroupStakeRecords*.

```
/// Returns batch of not yet processed voters.

    pub(crate) fn next_voters_batch<T: Config>(

        &mut self,

        solution_namespace: &SolutionNamespace,

    ) -> Option<Vec<T::AccountId>>
```

```
    where
        T::AccountId: From<AccountId>,
        AccountId: From<T::AccountId>,
    {
        let batch: Option<Vec<T::AccountId>> =
            match VotingRoundToNominatedSnapshot::<T>::get(&self.id) {
....
None => match GroupOfSolution::<T>::get(solution_namespace) {
    Some(group_namespace) => match &self.last_processed {
        Some(last_processed) => {
            let hashed_key = SolutionGroupStakeRecords::<T>::hashed_key_for(
                &group_namespace,
                T::AccountId::from(last_processed.clone()),
            );
            SolutionGroupStakeRecords::<T>::iter_key_prefix_from(
                &group_namespace,
                hashed_key,
            )
            .take(VOTE_PROCESSING_BATCH_SIZE as usize)
            .collect::<Vec<_>>()
            .into()
        },
        None                                                                    =>
SolutionGroupStakeRecords::<T>::iter_key_prefix(&group_namespace)
            .take(VOTE_PROCESSING_BATCH_SIZE as usize)
```

```
            .collect::<Vec<_>>()
            .into(),
    },
    None => None,
},
```

In addition to that, the operators that were part of the voting round and have not left the subscription pool will receive a higher reward than intended, as the *total_weighted_subscription_stake* is evaluated based on the amount of votes that are processed and since the operators who voted and left after the voting round are not processed, the *total_weighted_subscription_stake* will be lower.

In the example of 100 operators voting in the round, with a 100% consensus threshold reached (Absolute majority). 99 of them leave before the *next_voters_batch* has fully processed the results. The 1 remaining operator will receive 100% of the voting rewards instead of 1% which should have been the case.

Impact

Operators who have voted and stayed for the whole voting period may not be able to receive rewards if they leave before the *next_voters_batch* is triggered.

If the above scenario happens, then the operators who voted and stayed for the *next_voters_batch* to kick in will receive a higher reward than intended as there are less "correct" voters to share the rewards with.

Recommendation

It is advised to review the way voter rewards are calculated to include a more accurate list of users.

5.2 Medium Severity Findings

5.2.1 lib.rs/calculate_deposit_fee does not reserve bonus_reward from registrar

Description MEDIUM

A registrar calling the *raise_group_rewards* extrinsic to update the group rewards and include a performance bonus will have the rewards reserved from their balance for subscription and voting rewards but not for the performance bonus.

The extrinsic *libs.rs:raise_group_rewards* is utilizing the function *struct.rs:calculate_deposit_fee* in order to identify the *additional_deposit_fee* and eventually the *additional_depost_fee* is reserved from the registrars wallet.

It was identified that the *calculate_deposit_fee* is not taking into account the *top_performance_bonus* value. This effectively means that the performance bonus is not reserved from the registrar's wallet even though it's possible to process the request and update the *SolutionGroupRewardPeriodConfig* storage and eventually emit an event updating the network that a performance bonus is set.

Impact

This behavior can be misleading to the users of the network and to the registrar using the extrinsic. Users of the network will believe that a performance bonus has been set while it's not. In addition to that, the system will not make sure that the performance bonus amount has been reserved from the registrar ensuring that these funds will be available by the time they are needed. In case the registrar does not have the amount ready to be paid out when needed, this will leave the network itself open to liability.

Recommendation

It is advised to either temporarily prevent the registrar from calling the extrinsic with a populated performance bonus or advance the logic of the extrinsic to consider the performance bonus when calculating the deposit fee.

5.2.2 The length of nominated_workers is not dynamically assessed during a voting round

Description	MEDIUM

Voting round obstruction is currently feasible by a minority of voters. In case a nominated voter deregisters as an operator during the voting round, the round has a good chance of not going through. The *submit_solution_result* extrinsic counts the size of the voting pool according to the time where the voting round was created and does not update it during the round. In the example of 100 nominated workers and a 51% consensus threshold, if one worker leaves the subscription pool it could invalidate the vote. The voting sample will remain at 100 but only 99 votes would be able to go through.

When a new voting round is created, the round may accept votes from all subscribers or from a set of specific workers (nominated workers). This issue affects the nominated workers case. Upon creation of the voting round the system will take a snapshot of the nominated workers at that time.

```
voting.rs/create_voting_round

if solution.nominations_enabled {

   let nominated_voters = get_nomination_snapshot::<T>(&solution.namespace,
group)?;

   VotingRoundToNominatedSnapshot::<T>::insert(voting_round_id.clone(),
nominated_voters);

}
```

The issue arises as part of the *submit_solution_result* extrinsic which accepts the votes for the voting round. In that, the size of the voting pool (*eligible_voters_count*) is calculated according to the snapshot and not according to the current state of that snapshot.

```
lib.rs/submit_solution_result

....

let eligible_voters_count =

match VotingRoundToNominatedSnapshot::<T>::get(&voting_round_id) {

       Some(nominations) => nominations.voters.len() as u32,

       None => SolutionGroupSubscribersCount::<T>::get(&group_namespace),

 };
```

This allows for the case where a nominated worker who was part of the subscription pool at the time of the *creation_voting_round* is no longer part of it to affect the outcome of the vote.

In the example of 100 nominated workers and a 51% consensus threshold, if one worker leaves the subscription pool it could invalidate the vote. The voting sample will remain at 100 but only 99 votes would be able to go through.

Impact

A malicious nominated worker would be able to abuse this issue to invalidate a specific voting round. In addition to that, this is an issue which could be produced normally without any ill intention as operators can register and deregister from groups as they see fit.

Recommendation

In the case where a bricked round is acceptable, the system should terminate the vote immediately.

In all other cases, the nominated worker storage map should be updated dynamically and remove the unsubscribed operators.

5.2.3 lib.rs/unsubscribe_from_solution_group is not taking into account current voting periods

Description	MEDIUM

The current implementation of *libs.rs/unsubscribe_from_solution_group* does not consider if a current voting period is in place. Lack of this control allows for cases where an operator can cast a vote in a voting round via *submit_solution_result* (both in the cases of *NominatedWorkers* and not) and then unsubscribe from the solution group manipulating this way the outcome of the voting round. This issue affects the non-Nominated workers case.

The current format of the *submit_solution_result* assesses the consensus threshold based on the number of available operators at the time of their vote.

A voting round without nominated workers may see the following scenario:

- The requirement for a consensus vote is set at 51% with a pool of 100 workers.
- The pool's length is evaluated dynamically and depending on the duration of the voting round the total number of workers may increase or decrease.
- At this stage 51 votes would reach a consensus, and we consider that no new workers join the pool.
- Two workers cast a vote and then their operators unsubscribe from the solution group.

The result now is that there are two additional votes in the system while at the same time the size of the voting pool is reduced which could cause the voting round to reach a Settled outcome sooner.

Now that the subscriber pool is at 98 operators, the 51% threshold is equivalent to 49.98 votes out of the 98, which rounds up to 50. With this scenario, out of 100 operators with 2 voting and leaving, the consensus is reached by only 50 operators voting.

It is important to state that *unsubscribe_from_solution_group* does have an unsubscription delay (*solution_group.withdrawal_delay*) but this isn't connected to the lifespan of a voting round (or to the additional time where rewards and slashing would require to be calculated).

Impact

Operators can willingly or unwillingly (as part of the normal flow of activities) manipulate the outcome of a voting round by unsubscribing after voting.

Recommendation

This issue can be tackled in a couple of ways.

- An operator shouldn't be able to unsubscribe if a voting round is in effect, e.g. pause unsubscriptions during voting rounds.
- During unsubscription, the operator's vote should be purged

5.3 Low Severity Findings

5.3.1 lib.rs/remove_allowed_operator can be blocked by malicious operators

Description	LOW

The extrinsic *lib.rs/remove_allowed_operator* can be blocked by a malicious operator preventing a registrar from removing that operator from their allowlist.

The extrinsic *remove_allowed_operator* is used to control the number of operators allowed to subscribe to groups. With the *allow_operator* the registrar is setting a list of operators that are allowed and with the *remove_allowed_operator* they can remove them.

With the current format of the extrinsic *remove_allowed_operator* an operator cannot be forced out of a pool which can be the intended behavior. This becomes more of an issue when an operator doesn't want to be removed from that group.

There are two cases here:

- Operator is subscribed - *remove_allowed_operator* won't force them out which is intended behavior.
- Operator is not currently subscribed.

In the latter case the operator can still prevent the registrar from removing them by frontrunning the *remove_allowed_operator* extrinsic with a call to the *stake* extrinsic. Essentially the operator stakes just before the registrar removes them.

Even if the helper function *stake_manager::has_stake* was being used instead the same scenario would persist, as the *has_stake* function queries the *last_key_value* of the *StakeRecord* which would be true and positive.

Impact

A malicious operator who is currently unsubscribed can monitor the transaction pool for a call to *remove_allowed_operator* aimed to remove them from the allow

list. In this case, the operator can send off a call to the *stake* extrinsic in the same block with a higher priority which would block the action of the registrar.

Recommendation

Currently the *remove_allowed_operator* extrinsic cannot be used as a security control.

After discussion with the team, this function works more like a clean-up function and is not essential for the operation of the system. As such this issue is reduced to Low severity.

5.3.2 voting-rs:get_nomination snapshot not properly validating user stake

Description	LOW

If an operator has unsubscribed in the same block as the *start_voting_round* extrinsic is called, then the nomination snapshot will count the worker as part of the voting even though they will never be able to participate.

This issue affects the voting rounds where nominated workers are enabled.

The extrinsic *voting.rs/get_nomination_snapshot* is using the function *stake_manager::determine_user_overall_stake* instead of *stake_manager::has_stake* , allowing this way to include previously deregistered operators who remain in the *WorkerNodeToOperator* storage.

During the *on_initialize* function a pruning of old stakes is taking place reducing the occurrence of this event to 1 block and 2 blocks if this happens during the final block of the reward period (*reward_period.has_finished)*. Due to that the severity of this issue is reduced to Low.

Impact

This behavior would allow for the accidental or intentional manipulation of the voting round. This event has the potential to invalidate the voting round from its creation. In the example where the voting round has a 51% consensus threshold with 100 nominated workers but 50 of them unsubscribe just before the *start_voting_round* extrinsic is called then the voting round will require 51 votes for consensus to be reached out of 50 operators.

Recommendation

It is advised to adjust the *get_nomination_snapshot* function to include only operators with greater than 0 stake.

5.3.3 lib.rs/remove_allowed_operator relies on determine_user_overall_stake

Description	LOW

The extrinsic *lib.rs/remove_allowed_operator* is utilizing *check_account_is_not_subscribed_to_solution_group* as one of its checks which in turn is utilizing the *stake_manager::determine_user_overall_stake* function which will not throw an error in case the operator has previously unsubscribed.

This behavior will essentially block the use of the extrinsic rendering it useless.

Records from *SolutionGroupStakeRecords* are pruned at the beginning of every block during the *on_initialize* function. This reduces the nuisance that this issue would cause to 1 block and in case it coincides with the finalization of the *reward_period* this would be extended by an additional block, hence the severity is reduced to Low.

Impact

This behavior will forbid a registrar from removing an operator from the allowed list for a specific number of blocks. In that timeframe a malicious operator could resubscribe back to a solution group.

Recommendation

It is advised to adjust the *check_account_is_not_subscribed_to_solution_group* function to utilize the *stake_manager:has_stake* function instead.

5.3.4 lib.rs/raise_group_rewards fails silently

Description	LOW

A registrar calling the extrinsic *lib.rs:raise_group_reward* and updating only the *top_performance_bonus* will not be able to successfully register the bonus reward. The extrinsic will exit with an *Ok* with no further changes taking place.

This happens due to the current behavior described in *raise_group_rewards*, where the *additional_deposit_fee* will not take into account the performance bonus. As such the following *if* clause will return without updating the group settings.

```
lib.rs/raise_group_rewards

if additional_deposit_fee.eq(&Self::to_balance(0)) {

   log::debug!("Reward configuration of group {:?} was not changed", namespace);

                return Ok(().into());

}
```

It is important to note here that if the registrar makes this call and updates at least one of the other two variables (staking or voting reward) the extrinsic will proceed and update the storage successfully and emit the relevant event.

Impact

This issue affects the registrar as they won't be able to update the performance bonus easily. A direct request to update the bonus will fail without any information to the user as to what went wrong.

Recommendation

The case where the *raise_group_rewards* extrinsic is called with only the performance bonus should be handled appropriately and not fail without any notification to the user.

5.4 Informational Findings

5.4.1 Outdated and Vulnerable dependencies in-use at "Cargo.lock"

Description	INFO

The team identified multiple dependencies that were outdated and vulnerable. Outdated and vulnerable dependencies are open-source or proprietary components, in the form of libraries or frameworks, that contain known software vulnerabilities or are no longer maintained. Once a vulnerable component is discovered by adversaries, applications using this component can be targeted and exploited.

```
Vulnerable dependencies found at Cargo.lock:

Crate:     curve25519-dalek

Version:   2.1.3

Title:                Timing     variability     in     `curve25519-dalek`'s
`Scalar29::sub`/`Scalar52::sub`

Date:      2024-06-18

ID:        RUSTSEC-2024-0344

URL:      https://rustsec.org/advisories/RUSTSEC-2024-0344

Solution:  Upgrade to >=4.1.3



Crate:     curve25519-dalek

Version:   3.2.0

Title:                Timing     variability     in     `curve25519-dalek`'s
`Scalar29::sub`/`Scalar52::sub`

Date:      2024-06-18

ID:        RUSTSEC-2024-0344

URL:      https://rustsec.org/advisories/RUSTSEC-2024-0344
```

```
Solution:  Upgrade to >=4.1.3


Outdated:

Crate:     parity-wasm

Version:   0.45.0

Warning:   unmaintained

Title:     Crate `parity-wasm` deprecated by the author

Date:      2022-10-01

ID:        RUSTSEC-2022-0061

URL:      https://rustsec.org/advisories/RUSTSEC-2022-0061


Crate:     mach

Version:   0.3.2

Warning:   unmaintained

Title:     mach is unmaintained

Date:      2020-07-14

ID:        RUSTSEC-2020-0168

URL:      https://rustsec.org/advisories/RUSTSEC-2020-0168


Crate:     ansi_term

Version:   0.12.1

Warning:   unmaintained

Title:     ansi_term is Unmaintained

Date:      2021-08-18

ID:        RUSTSEC-2021-0139
```

```
URL:      https://rustsec.org/advisories/RUSTSEC-2021-0139


Crate:     proc-macro-error

Version:   1.0.4

Warning:   unmaintained

Title:     proc-macro-error is unmaintained

Date:      2024-09-01

ID:        RUSTSEC-2024-0370

URL:      https://rustsec.org/advisories/RUSTSEC-2024-0370
```

Impact

When a vulnerable 3rd-party component is discovered by adversaries, all applications that utilize this component can be identified and targeted. Even if the vulnerability might initially look like a small weakness in the application codebase, in some cases it can lead to a full system compromise. Such a breach can have a seriously affect the users' data and potentially lead to lost revenue and reputational damage the organization.

Recommendation

It is recommended to update the related dependencies to the latest version.

5.4.2 *voting_threshold_percent allows for minority voting*

Description INFO

The current implementation allows for the registration of new solutions with a *voting_thresold_percent* from 0-100. A percentage below 33% would allow for a veto-only system.

The only check currently happening for *vote_threshold_percent* is to confirm it's more than 0 and less than 100.

```
solution.rs/new

ensure!(

     vote_threshold_percent <= 100,

     Error::<T>::ArgumentOutOfBounds(FieldCode::VoteThresholdPercent)

);
```

The current consensus threshold cannot be modified. The issue would only be introduced as an intended action or a misconfiguration during the creation of the solution which would be known before the start of any voting round.

After discussion with the team this behavior is intended and as such the issue now is informational.

Impact

The ability to set a low consensus threshold allows for potential manipulation of voting rounds by a minority set of users. According to the current set up, a 1% consensus is feasible. In the example of 100 voters, we have the case where a malicious voter, monitors the transaction pool for the voting round to start and sneaks in any result, capturing this way the round consensus and its potential rewards.

Recommendation

The current behavior allows for the use of the voting system as an approval system (majority) or a veto system (minority protection).

If this is not the intended behavior, then it is advised to conduct further analysis on the lower end of the consensus threshold. Percentages lower than 33% allow for abuse of the voting system.

If updating the consensus thresholds are part of future plans it is strongly advised to separate veto and approval systems as the reward distribution will be able to be abused.

6 Retesting Results

6.1 Retest of High Severity Findings

Results from retesting carried out on May 2025, determined that four (4 out of 4) reported HIGH risk issues were sufficiently addressed (see sections *5.1.1, 5.1.2, 5.1.3, 5.1.4*).

6.1.1 lib.rs/submit_solution_result emits the voting round settled result

The VotingRoundStatus is no longer emitted in the extrinsic submit_solution_result. In addition to that, the round result is no longer calculated within the same extrinsic.

Issue mitigated in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/367.

6.1.2 lib.rs/submit_solution_result result bruteforcing on Settled voting rounds

The reported logic has been removed. The issue is now fixed.

Team comment: Voting rounds are settled after expiration when no other votes are accepted

Issue mitigated in PR, https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/367.

6.1.3 lib.rs/submit_solution_result emits voting results

The SolutionResultSubmitted event has been deprecated, and the extrinsic submit_solution_result uses VoteSubmitted event instead which does not include the result.

The issue is sufficiently mitigated in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/367.

6.1.4 data_struct:next_voters_batch for non-nominated worker case is not assessing a snapshot

Function is not used anymore. Issue is addressed in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/367. The issue is now sufficiently mitigated.

6.2 Retest of Medium Severity Findings

Three (3 out of 3) reported MEDIUM risk issues were sufficiently addressed (see sections 5.2.1, 5.2.2, 5.2.3).

6.2.1 lib.rs/calculate_deposit_fee does not reserve bonus_reward from registrar

The issue is now resolved. solution_group_registration will exit if top_performance bonus is added.

```
ensure!(rewards_config.top_performance_bonus==0, Error::<T>::NotImplemented);
```

raise_group_rewards does not allow the use of top_performance bonus. The fix is introduced in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/393.

6.2.2 The length of nominated_workers is not dynamically assessed during a voting round

The extrinsic submit_solution_result no longer fetches the VotingRoundNominatedSnapshot.

The logic of this operation has been heavily modified. There are no scenarios where a user can unsubscribe within a voting period. All unsubscriptions take place after the conclusion of that period. The issue is now considered resolved. The issue is mitigated in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/409.

6.2.3 lib.rs/unsubscribe_from_solution_group is not taking into account current voting periods

A voting round with non-nominated workers now has a fixed size snapshot. Even if an operator votes and unsubscribes the size of the snapshot will remain the same. This change invalidates the scenario stated in this issue. The fix is in the PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/365. The issue is now fixed.

6.3 Retest of Low Severity Findings

Four (4 out of 4) reported LOW risk issues were sufficiently addressed (see sections 5.3.1, 5.3.2, 5.3.3, 5.3.4).

6.3.1 lib.rs/remove_allowed_operator can be blocked by malicious operators

This issue has been fixed in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/410. The extrinsic libs.rs/remove_allowed_operator uses the function register_unsubscribe which schedules the deregistration of an operator. This operation will work in both cases where the operator's stake is active or not. The cases that were mentioned are currently also covered by tests cases. This issue is now fixed.

6.3.2 voting-rs:get_nomination snapshot not properly validating user stake

The issue is mitigated in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/369. In that, voting.rs/get_nomination_snapshot now validates the operator_stake and if it is equal to 0, the operator will be skipped.

6.3.3 lib.rs/remove_allowed_operator relies on determine_user_overall_stake

Item was mitigated on https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/370/.

During retesting the version `9d57d8a5d0637e00497d253461fe6d18bc3b73e2` was provided. In the latest version, the above segment is altered even further. Function `check_account_is_not_subscribed_to_solution_group` is removed from the code. In the `remove_allowed_operator` extrinsic which was previously calling the aforementioned extrinsic now has an altered logic. The alrtered logic does make use of the has_stake. This issue is considered Fixed.

6.3.4 lib.rs/raise_group_rewards fails silently

When a user is calling the lib.rs:raise_group_reward extrinsic with a reward with includes bonus, the error NotImplemented will be thrown. The issue is resolved.

Item is resolved in PR https://github.com/energywebfoundation/ewx-worker-solution-pallet/pull/392/.

6.4 Retest of Informational Findings

One (1 out of 2) reported issues bearing no risk (INFORMATIONAL) were marked as risk accepted (see sections 5.4.2). The remaining issue is still open 5.4.1.

6.4.1 Outdated and Vulnerable dependencies in-use at "Cargo.lock"

Issue remains Open. The team replied that the issue will be addressed when updating the polkadot-sdk version.

6.4.2 voting_threshold_percent allows for minority voting

The team replied that the WNP aims to give complete flexibility and responsibility to the registrars in order to create consensus systems according to their needs. If a registrar aims for 1% consensus because the use case requires it then the WNP should accept it. Documentation and guidance on each use case is provided from E.W. Dapps to solution registrars.

Document History

Revision	Description	Changes Made By	Date
1.0	*First Version*	*Chaintroopers*	*January 24th, 2025*
1.1	*Retest Report*	*Chaintroopers*	*May 23rd, 2025*