// Private CosmWasm Smart Contract Security Assessment 03.13.2024 - 05.17.2024

Amplifier Contracts

Axelar

HALBORN

Amplifier Contracts - Axelar

Prepared by: HALBORN

Last Updated 05/19/2024

Date of Engagement by: March 13th, 2024 - May 17th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	0	1	4	6

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
   - 7.1 Update worker set fails if workers do not register a public key
   - 7.2 Failed poll causes failure to change working set
   - 7.3 The voting verifier does not check for message duplicates
   - 7.4 Lack of access control in messages verification
   - 7.5 Bad messages from nexus will revert the routing process
   - 7.6 Rewards participants are not asserted to be unique
   - 7.7 Missing signature nonce in registering multisig public key
   - 7.8 Anyone can register a new worker set
   - 7.9 Rewards messages could be batched
   - 7.10 Voting verifier storage grows infinitely
   - 7.11 Prover does not batch workers pubkey query

8. Automated Testing

1. Introduction

Axelar engaged Halborn to conduct a security assessment on their smart contracts beginning on March 13th, 2024 and ending on May 17th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team. Commit hashes and further details can be found in the Scope section of this report.

Axelar Amplifier is a decentralized cross chain messaging protocol, based on smart contracts deployed on the Axelar blockchain.

2. Assessment Summary

The team at Halborn assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple risks, which were addressed and acknowledged by the Axelar team. The main ones were the following:

- Ensure worker set update cannot fail
- Filter duplicate messages at all levels of the protocol
- Enforce access control on the message verification
- Handle gracefully bad messages from the nexus gateway

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning of Rust files for common vulnerabilities (`cargo audit`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: axelar-amplifier

(b) Assessed Commit ID: c179252

(c) Items in scope:

- ampd
- contracts/aggregate-verifier
- contracts/connection-router
- contracts/gateway
- contracts/multisig
- contracts/multisig-prover
- contracts/nexus-gateway
- contracts/rewards
- contracts/service-registry
- contracts/voting-verifier

Out-of-Scope: External libraries and financial attacks., New features/implementations after/with the remediation commit IDs., Changes that occur outside of the scope of PRs.

REMEDIATION COMMIT ID:

- a68eb5ba68eb5b

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	4	6

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UPDATE WORKER SET FAILS IF WORKERS DO NOT REGISTER A PUBLIC KEY	Medium	RISK ACCEPTED
FAILED POLL CAUSES FAILURE TO CHANGE WORKING SET	Low	SOLVED - 03/27/2024
THE VOTING VERIFIER DOES NOT CHECK FOR MESSAGE DUPLICATES	Low	RISK ACCEPTED
LACK OF ACCESS CONTROL IN MESSAGES VERIFICATION	Low	RISK ACCEPTED
BAD MESSAGES FROM NEXUS WILL REVERT THE ROUTING PROCESS	Low	RISK ACCEPTED
REWARDS PARTICIPANTS ARE NOT ASSERTED TO BE UNIQUE	Informational	ACKNOWLEDGED
MISSING SIGNATURE NONCE IN REGISTERING MULTISIG PUBLIC KEY	Informational	ACKNOWLEDGED
ANYONE CAN REGISTER A NEW WORKER SET	Informational	ACKNOWLEDGED
REWARDS MESSAGES COULD BE BATCHED	Informational	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
VOTING VERIFIER STORAGE GROWS INFINITELY	Informational	ACKNOWLEDGED
PROVER DOES NOT BATCH WORKERS PUBKEY QUERY	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 UPDATE WORKER SET FAILS IF WORKERS DO NOT REGISTER A PUBLIC KEY

// MEDIUM

Description

The `update_worker_set` function calls `multisig::get_public_key` for each active worker in the registry. If one of the workers did not submit a public key to the multisig contract, the `multisig::get_public_key` for that worker will fail with an error which causes the function to revert.

- `contracts/multisig-prover/src/execute.rs`:

```
126 let mut pub_keys = vec![];
127 for worker in &workers {
128     let pub_key_query = multisig::msg::QueryMsg::GetPublicKey {
129         worker_address: worker.address.to_string(),
130         key_type: config.key_type,
131     };
132     let pub_key: PublicKey =
133 deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
134         contract_addr: config.multisig.to_string(),
135         msg: to_binary(&pub_key_query)?,
136     }))?;
137     pub_keys.push(pub_key);
    }
```

- `contracts/multisig/src/state.rs`:

```
92 pub fn load_pub_key(store: &dyn Storage, signer: Addr, key_type: KeyType) ->
93 StdResult<HexBinary> {
94     pub_keys().load(store, (signer, key_type))
   }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:P/S:C (4.7)

Recommendation

It is recommended to ensure that worker set can still be updated by preventing the execution failure. More generally, queries should not revert.

Remediation Plan

RISK ACCEPTED: The **Axelar team** accepted the risk of this issue, also mentioned that:

> It can be mitigated by governance (i.e. only authorize workers that have registered a key)

7.2 FAILED POLL CAUSES FAILURE TO CHANGE WORKING SET

// LOW

Description

In the **multisig-prover** contract, the `confirm_worker_set` function does not clear the `NEXT_WORKER_SET` state variable when a poll failed. When trying to set a new worker set, the `save_next_worker_set` function will compare if the current `NEXT_WORKER_SET` is different from the desired one.
If so, the `save_next_worker_set` function will return an error and cause failure of that feature.

- `contracts/multisig-prover/src/execute.rs`:

```
178 fn save_next_worker_set(
179     storage: &mut dyn Storage,
180     new_worker_set: &WorkerSet,
181 ) -> Result<(), ContractError> {
182     if different_set_in_progress(storage, new_worker_set) {
183         return Err(ContractError::WorkerSetConfirmationInProgress);
184     }
185 
186     NEXT_WORKER_SET.save(storage, new_worker_set)?;
187     Ok(())
188 }
```

- `contracts/multisig-prover/src/execute.rs`:

```
296 fn different_set_in_progress(storage: &dyn Storage, new_worker_set:
297 &WorkerSet) -> bool {
298     if let Ok(Some(next_worker_set)) = NEXT_WORKER_SET.may_load(storage) {
299         return next_worker_set.signers != new_worker_set.signers
300             || next_worker_set.threshold != new_worker_set.threshold;
301     }
302 
303     false
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:P/S:U (3.8)

Recommendation

It is recommended to be able to change a worker set when a poll failed.

Remediation Plan

SOLVED: The issue was solved by enabling governance to enforce a new worker set if needed. The **Axelar team** also mentioned that:

> Once a worker set is scheduled for rotation we mustn't allow any other worker set, because those sets get registered with gateway contracts on other chains. Essentially, the confirmation is just waiting to get confirmation from the external chain that the new worker set has been communicated.

Remediation Hash

https://github.com/axelarnetwork/axelar-amplifier/commit/a68eb5b3c28d9f6c0bd665ba012cbec13970f3a8

7.3 THE VOTING VERIFIER DOES NOT CHECK FOR MESSAGE DUPLICATES

// LOW

Description

In the **voting-verifier** contract, an attacker could feed the messages `[M1, M1, M2]` and create a new poll. The `POLL_MSG` data will be: `{hash1: idx_in_poll 2, hash2: idx_in_poll 3}`, when the length of the poll would be 3.

- `contracts/voting-verifier/src/execute.rs`:

```
69 pub fn verify_messages(
70     deps: DepsMut,
71     env: Env,
72     messages: Vec<Message>,
73 ) -> Result<Response, ContractError> {
```

This could lead to denial of service of the verifying logic depending on the implementation of verifiers (although ampd will send the votes for `[M1, M1, M2]` which is does not trigger any failure) ; this could also be costly in gas if an attacker sends a huge list, potentially DOS if attackers verifies `[M1, M1, M1, M1]` and then the gateway verifies `[M1]`, the gateway verification will do nothing has `M1` is already `IN_PROGRESS`. The poll should continue normally, except the fact that each worker will need to send 4 votes instead of 1 because of the poll size.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to filter messages duplicates in the `verify_messages` function.

Remediation Plan

RISK ACCEPTED: The **Axelar team** accepted the risk of this issue, also mentioned that:

> DOS is less of a concern, someone can always just send non-existent messages to make verifiers waste time, so we need to solve this in a different way. However, the gateway already filters duplicates and the verifier should be consistent.

7.4 LACK OF ACCESS CONTROL IN MESSAGES VERIFICATION

// LOW

Description

According to documentation (cf. schema in `doc/src/contracts/voting_verifier.md`), the `verify_messages` of the voting verifier is supposed to be called by the aggregate verifier, but no verification ensures that, and anyone can call that function.
The normal process should be: `relayer -> gateway -> aggregator verifier -> voting verifier`, but can be bypassed directly by `attacker -> voting verifier`
This could lead to DOS if it collides with the normal flow of events (although it is not an issue currently: if attackers verifies `[M1, M2]` and then the gateway verifies `[M1, M2]`, the gateway verification will do nothing has `M1` and `M2` are already IN_PROGRESS. The poll should continue normally.).

- `contracts/voting-verifier/src/execute.rs`:

```
69 pub fn verify_messages(
70     deps: DepsMut,
71     env: Env,
72     messages: Vec<Message>,
73 ) -> Result<Response, ContractError> {
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to ensure that the voting verifier cannot be called directly, and the verifiers cannot be bypassed upfront.

Remediation Plan

RISK ACCEPTED: The **Axelar team** accepted the risk of this issue, also mentioned that:

> The endpoint on the gateway is essentially just there for convenience, so relayers only have to interact with a single contract, but should not be relevant for the safety of the protocol. So it's fine to interact with the voting-verifier and the gateway should react appropriately.

7.5 BAD MESSAGES FROM NEXUS WILL REVERT THE ROUTING PROCESS

// LOW

Description

If the Nexus gateway emits a message that has an unknown chain name for the router, any attempt to route a set of messages with that bad message within it will result in reverting the transaction. That can be problematic if the relayers keep trying to route this message along other valid messages, since the transaction will always fail.

- `contracts/connection-router/src/contract/execute.rs`:

```
169 None if sender != self.config.nexus_gateway => {
170     self.config.nexus_gateway.clone()
171 }
172 _ => return Err(report!(Error::ChainNotFound)),
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to filter Nexus messages if the router does not support them.

Remediation Plan

RISK ACCEPTED: The **Axelar team** accepted the risk of this issue, also mentioned that:

> We don't want to fail messages silently, so filtering them out wouldn't work. The philosophy we're following is that a successful call should have the expected effect (messages will get verified, routed, etc) and in case of error any command can be retried. So maybe it's more of an issue of how we can provide relayers with the information of which chains are available. In practice, we expect to operate the verification and routing calls in-house and provide external users with a more ergonomic sdk, so all in all this issue has low impact.

7.6 REWARDS PARTICIPANTS ARE NOT ASSERTED TO BE UNIQUE

// INFORMATIONAL

Description

The **rewards** contract is not explicitly protected against receiving the same `RecordParticipation(pool_id, event_id, worker_address)` more than once. Although the voting verifier is safe against calling the `EndPoll` more than once, it is safer to explicitly disallow recording the same participation multiple times. Recording participation multiple times on a given event would boost the participation percentage of all workers that followed the consensus on that event, and lower the one from non consensus workers on that event. It could therefore affect which worker can receive rewards.

- `contracts/rewards/src/contract/execute.rs`:

```
72  pub fn record_participation(
73      &mut self,
74      event_id: nonempty::String,
75      worker: Addr,
76      pool_id: PoolId,
77      block_height: u64,
78  ) -> Result<(), ContractError> {
79      let cur_epoch = self.current_epoch(block_height)?;
80
81      let event = self.load_or_store_event(event_id, pool_id.clone(),
82  cur_epoch.epoch_num)?;
83
84      self.store
85          .load_epoch_tally(pool_id.clone(), event.epoch_num)?
86          .unwrap_or(EpochTally::new(
87              pool_id,
88              cur_epoch,
89              self.store.load_params().params,
90          ))
91          .record_participation(worker)
92          .then(|mut tally| {
93              if matches!(event, StorageState::New(_)) {
94                  tally.event_count += 1
95              }
96              self.store.save_epoch_tally(&tally)
97          })
    }
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:C (1.9)

Recommendation

It is recommended to explicitly disallow recording the same participation multiple times.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> That is a good point, but it will require a drastically different data structure. Currently this check is impossible to make. Most likely we won't change it. Each caller contract can only impact it's own rewards pool, so if they make a mistake in recording the participation it's their own fault.

7.7 MISSING SIGNATURE NONCE IN REGISTERING MULTISIG PUBLIC KEY

// INFORMATIONAL

Description

In the **multisig** contract's `register_pub_key` function, the objective is to prevent anyone from registering a public key that belongs to someone else. It would be safer to add a nonce to the payload to sign and some additional data because the users could have previously signed another person's address in for any usage in the past.
As a reference, `ERC20Permit` uses both a `permitHash` string and a nonce per user.
If anyone gets a signature of his own address by the signer, that person could add `(attacker_address, victim_public_key)` to the signers and prevent that victim from registering, since the public key is a unique index.

- `contracts/multisig/src/contract/execute.rs`:

```
133 pub fn register_pub_key(
134     deps: DepsMut,
135     info: MessageInfo,
136     public_key: PublicKey,
137     signed_sender_address: HexBinary,
138 ) -> Result<Response, ContractError> {
139     let signed_sender_address: Signature =
140         (public_key.key_type(), signed_sender_address).try_into()?;
141
142     let address_hash = Keccak256::digest(info.sender.as_bytes());
```

BVSS

AO:S/AC:L/AX:H/C:N/I:M/A:N/D:N/Y:N/R:N/S:C (0.4)

Recommendation

It is recommended to add a nonce to the public key registration.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> This will most likely not be changed. The attacker first needs to get the owner of the key to sign the attacker's address, and even then they can only block them from using that public key for multisig. The verifier can easily create a different key and use that.

7.8 ANYONE CAN REGISTER A NEW WORKER SET

// INFORMATIONAL

Description

In the **multisig** contract, there is no access control on the register of a new worker set, instead of verifying that the message comes from prover. Because of that, anyone can add a worker set to the registry. However, this has no actual exploit because the decision to use a certain worker set is taken through a vote from the current workers. The registry will simply be filled with unused data from callers who are required to pay gas fees for their calls.

- `contracts/multisig/src/contract.rs`:

```
73 ExecuteMsg::RegisterWorkerSet { worker_set } => {
74     execute::register_worker_set(deps, worker_set)
75 }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is recommended to restrict the access control to the multisig prover.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> It's actually correct as is. RegisterWorkerSet just makes the multisig contract aware of a particular worker set, but the decision which worker set to use is handled by the multisig prover contract when calling StartSigningSession , which indeed requires authorization. So if someone registers a bogus worker set, it's just never going to get used.

7.9 REWARDS MESSAGES COULD BE BATCHED

// INFORMATIONAL

Description

The voting verifier sends one Wasm message per consensus participant. This could be more efficient by batching participants in one wasm message.

- `contracts/voting-verifier/src/execute.rs`:

```
197 let rewards_msgs = poll_result
198 .consensus_participants
199 .iter()
200 .map(|address| WasmMsg::Execute {
201     contract_addr: config.rewards_contract.to_string(),
202     msg: to_binary(&rewards::msg::ExecuteMsg::RecordParticipation {
203         chain_name: config.source_chain.clone(),
204         event_id: poll_id
205             .to_string()
206             .try_into()
207             .expect("couldn't convert poll id to nonempty string"),
208         worker_address: address.to_string(),
209     })
210     .expect("failed to serialize message for rewards contract"),
211     funds: vec![],
212 });
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is recommended to batch all rewards information into a single Wasm message.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> We have discussed this internally but since we can easily adjust this even after the protocol is live we won't act on it before the MVP and will measure the gas consumption afterwards to check its impact.

7.10 VOTING VERIFIER STORAGE GROWS INFINITELY

// INFORMATIONAL

Description

The `POLL_MESSAGES` hash map grows infinitely, increasing the storage size of the contract. Deleting the messages after usage can reduce the storage size.

- `contracts/voting-verifier/src/execute.rs`:

```
127 for (idx, message) in msgs_to_verify.iter().enumerate() {
128     POLL_MESSAGES.save(
129         deps.storage,
130         &message.hash(),
131         &state::PollContent::<Message>::new(message.clone(), id, idx),
132     )?;
133 }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is recommended to delete the messages after their usage.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> For most of its life, the verifier pulled double duty by also preventing double spending (you can't verify again if it's already verified). This is no longer the case, so we could potentially implement a retention policy for poll outcomes.

7.11 PROVER DOES NOT BATCH WORKERS PUBKEY QUERY

// INFORMATIONAL

Description

The update worker set function sends a query for each active worker. This could be replaced by adding a list query to save complexity.

- `contracts/multisig-prover/src/execute.rs`:

```
127 for worker in &workers {
128     let pub_key_query = multisig::msg::QueryMsg::GetPublicKey {
129         worker_address: worker.address.to_string(),
130         key_type: config.key_type,
131     };
132     let pub_key: PublicKey =
133 deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
134         contract_addr: config.multisig.to_string(),
135         msg: to_binary(&pub_key_query)?,
136     }))?;
137     pub_keys.push(pub_key);
    }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is recommended to send a unique query that would return all the needed necessary information in a unique call.

Remediation Plan

ACKNOWLEDGED: The **Axelar team** acknowledged this finding, also mentioning that:

> We generally opted for simplicity over performance when we didn't have a good reason otherwise. We can revisit this in the future if we find it's too costly.

8. AUTOMATED TESTING

The following vulnerabilities were identified with `cargo audit`. Even though no applicable exploit was found, it is best practice to ensure no third party vulnerability remains in the project.

```
Crate:     cosmwasm-std
Version:   1.4.0
Title:     Arithmetic overflows in cosmwasm-std
Date:      2024-04-24
ID:        RUSTSEC-2024-0338
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0338
Solution:  Upgrade to >=1.4.4, <1.5.0 OR >=1.5.4, <2.0.0 OR >=2.0.2

Crate:     h2
Version:   0.3.21
Title:     Degradation of service in h2 servers with CONTINUATION Flood
Date:      2024-04-03
ID:        RUSTSEC-2024-0332
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0332
Solution:  Upgrade to ^0.3.26 OR >=0.4.4

Crate:     h2
Version:   0.3.21
Title:     Resource exhaustion vulnerability in h2 may lead to Denial of Service (DoS)
Date:      2024-01-17
ID:        RUSTSEC-2024-0003
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0003
Solution:  Upgrade to ^0.3.24 OR >=0.4.2

Crate:     mio
Version:   0.8.8
Title:     Tokens for named pipes may be delivered after deregistration
Date:      2024-03-04
ID:        RUSTSEC-2024-0019
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0019
Solution:  Upgrade to >=0.8.11

Crate:     quinn-proto
Version:   0.10.4
Title:     Denial of service in Quinn servers
Date:      2023-09-21
ID:        RUSTSEC-2023-0063
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0063
Solution:  Upgrade to ^0.9.5 OR >=0.10.5
```

```
Crate:     rsa
Version:   0.8.2
Title:     Marvin Attack: potential key recovery through timing sidechannels
Date:      2023-11-22
ID:        RUSTSEC-2023-0071
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0071
Solution:  No fixed upgrade is available!

Crate:     rustls
Version:   0.19.1
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on
network input
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Solution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0

Crate:     rustls
Version:   0.21.7
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on
network input
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Solution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0

Crate:     serde-json-wasm
Version:   0.5.1
Title:     Stack overflow during recursive JSON parsing
Date:      2024-01-24
ID:        RUSTSEC-2024-0012
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0012
Solution:  Upgrade to >=1.0.1 OR >=0.5.2, <1.0.0

Crate:     shlex
Version:   1.2.0
Title:     Multiple issues involving quote API
Date:      2024-01-21
ID:        RUSTSEC-2024-0006
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0006
Solution:  Upgrade to >=1.3.0

Crate:     tungstenite
```

```
Version:   0.20.0
Title:     Tungstenite allows remote attackers to cause a denial of service
Date:      2023-09-25
ID:        RUSTSEC-2023-0065
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0065
Solution:  Upgrade to >=0.20.1

Crate:     webpki
Version:   0.21.4
Title:     webpki: CPU denial of service in certificate path building
Date:      2023-08-22
ID:        RUSTSEC-2023-0052
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0052
Solution:  Upgrade to >=0.22.2

Crate:     whoami
Version:   1.4.1
Title:     Stack buffer overflow with whoami on several Unix platforms
Date:      2024-02-28
ID:        RUSTSEC-2024-0020
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0020
Solution:  Upgrade to >=1.5.0
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.