// HALBORN

Terra - Warp Contracts

CosmWasm Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **December 18th, 2023 - January 19th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Draft Version	01/19/2024
0.2	Draft Review	01/19/2024
1.0	Remediation Plan	01/29/2024
1.1	Remediation Plan Review	02/05/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Warp is a decentralized automation platform for blockchain that goes beyond traditional Cron Jobs. Users can create Jobs, the core feature, by sending WASM messages with specified conditions and rewards. Conditions can range from simple timestamps to complex scenarios, all verifiable on the blockchain. Jobs are signed by the owner during creation and executed by gatekeepers when conditions are met, promoting decentralized automation.

Key concepts include Conditions, which specify the circumstances for job execution, and Templates and Variables for creating reusable, looping jobs. Warp accounts, similar to smart contract wallets, facilitate fund transactions for job-related activities. Keepers, incentivized by rewards, play a vital role in executing smart contract functions, ensuring decentralized and efficient job execution. The platform workflow consists of adding successfully created jobs to a queue until conditions trigger their execution, and an eviction mechanism ensures the continuous activity of jobs by evicting them after a certain amount of time.

Terra engaged Halborn to conduct a security assessment on their smart contracts beginning on December 18th, 2023 and ending on January 19th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team, listed in the Scope section.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided five weeks for the engagement and 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 security risks that were mostly addressed by the Terra team. The main ones were the following:

** (HAL-01) CONTROLLER CONTRACT BALANCE CAN BE DRAINED **

In the create_job function of the warp-controller contract, a critical vulnerability arises due to the absence of validation checks on certain inputs. In a potential attack scenario, a malicious user could set the reward equal to the controller balance, job_amount to 0 and not send any funds. The same user would run that job, receiving the entire balance of the controller contract as a reward.

** (HAL-02) EXECUTION ERROR DURING JOB ACCOUNT INSTANTIATION **

During instantiation of a new job account, account_msgs messages are transformed from Vec to Vec and executed, including the result as a string in the event attributes.

The problem arises in the create_job_account_and_job reply function when it extracts the account_msgs variable from the event attributes and tries to cast it as Vec again, causing a deserialization error.

** (HAL-03) MALICIOUS OWNER COULD CREATE CORRUPTED JOB ACCOUNTS **

In a hypothetical scenario where the controller-owner account is compromised, there is the potential risk of an attack involving the creation of corrupted job accounts with the malicious owner's address as config.creator_addr.

While this corrupted account might initially appear harmless, the malicious owner, having privileges over warp-account-tracker executions, could execute the free_job_account entry point, adding fake accounts to users job account queue. When a user attempts to create a new job, the corrupted account could be allocated, allowing the malicious owner to withdraw any funds deposited into it through a warp_msg execution.

1.3 SCOPE

- Terra:
  - Repository: warp-contracts
  - Commit ID: e000eabd6064b4ab9013fe57366e4d4f114fd47b
  - Smart contracts in scope:
    - warp-controller
    - warp-account
    - warp-account-tracker
    - warp-resolver
    - warp-templates

Out-of-scope: External libraries and financial related attacks.

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of the manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding 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 the coverage of smart contracts. They 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 architecture, purpose, and use of the platform.
- Manual code read and walk through.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Architecture related logical controls.
- Cross contract call controls.
- Scanning of Rust files for vulnerabilities (cargo audit).
- Review of integration tests.
- Deployment to Pisco-1 Testnet for dynamic testing.

2. 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.

2.1 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:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
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:

$$E = \prod m_e$$

2.2 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)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

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

2.3 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.

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
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	6	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CONTROLLER CONTRACT BALANCE CAN BE DRAINED	Critical (10)	SOLVED - 01/12/2024
(HAL-02) EXECUTION ERROR DURING JOB ACCOUNT INSTANTIATION	Medium (5.0)	SOLVED - 01/19/2024
(HAL-03) MALICIOUS OWNER COULD CREATE CORRUPTED JOB ACCOUNTS	Low (2.8)	RISK ACCEPTED
(HAL-04) EVICT JOB FUNCTION DOES NOT RELEASE FUNDING ACCOUNT	Low (2.5)	SOLVED - 01/09/2024
(HAL-05) JOB DURATION IS NOT CHECKED	Low (2.5)	SOLVED - 01/25/2024
(HAL-06) LACK OF VALIDATIONS IN THE CONFIGURATION DATA	Low (2.5)	SOLVED - 01/25/2024
(HAL-07) OWNERSHIP CAN BE TRANSFERRED WITHOUT CONFIRMATION	Low (2.0)	RISK ACCEPTED
(HAL-08) NO OPTION TO MODIFY ACCOUNT TRACKER ADMIN	Low (2.0)	SOLVED - 01/25/2024
(HAL-09) USELESS CODE	Informational (0.0)	SOLVED - 01/25/2024
(HAL-10) REPEATED OPERATIONS	Informational (0.0)	SOLVED - 01/25/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) CONTROLLER CONTRACT BALANCE CAN BE DRAINED - CRITICAL(10)

Description:

The create_job function of the **warp-controller** contract allows any user to create a job containing certain messages and conditions that will be executed in the future by the other user.

Therefore, it is mandatory to include an amount of coins that will be used as a reward for job execution. In addition to that, there are some fees that would be charged to the user at the time of job creation (maintenance, create and burn). The operational_amount input should specify an amount sufficient to cover these expenses. If it is greater, the remainder will be transferred to the job account contract.

If the user provides a funding account address at the time of job creation, all these expenses would be charged to it; if not, the user would have to transfer the funds within the same transaction.

The issue is that there is no verification of the funds sent in the transaction, as well as the parameters directly related to the fixed expenses (reward, operational amount), so the attack scenario is as follows:

- The Controller contract has some funds from the rewards of the pending jobs of all other users.
- The malicious user creates a job and sets the reward amount as the total balance of the Controller contract (minus an estimate of fees). Operational_amount is set to 0 and he does not send any funds to the contract.
- The job is created: the fees are subtracted from the Controller balance and sent to fee_collector.
- The malicious user executes the job and receives the reward, emptying the Controller balance.

This is due to the lack of validation in the create_job function:

- Operational_amount value is not checked. It should be at least equal to or greater than reward + total_fees.
- Info.funds.amount is not checked. It should be at least equal to or greater than operational_amount.

Code Location:

Snippet of code of the create_job function in the **warp-controller** contract:

Listing 1: contracts/warp-controller/src/execute/job.rs (Lines 81,86)

```
71 let total_fees = creation_fee + maintenance_fee + burn_fee;
72
73 if data.funding_account.is_none() && data.recurring {
74     return Err(ContractError::FundingAccountMissingForRecurringJob
↳  {});
75 }
76
77 // ignore operational_amount when funding_account is provided
78 let operational_amount = if data.funding_account.is_some() {
79     Uint128::zero()
80 } else {
81     data.operational_amount
82 };
83
84 // Reward and fee will always be in native denom
85 let native_funds_minus_operational_amount =
↳ deduct_from_native_funds(
86     info.funds.clone(),
87     config.fee_denom.clone(),
88     operational_amount,
89 );
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to include the following validations on `create_job` function:

- `Operational_amount` value should be at least equal to or greater than `reward + total_fees`.
- `Info.funds.amount` value should be at least equal to or greater than `operational_amount`.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID `ae8c020` by adding the corresponding validations on the `operational_amount` and `info.funds.amount` values.

4.2 (HAL-02) EXECUTION ERROR DURING JOB ACCOUNT INSTANTIATION - MEDIUM (5.0)

Description:

One of the inputs of the create_job function is the account_msgs parameter, which allows some messages to be executed at job creation time.

If there is no free account to be assigned to the job, a new one is instantiated. During the instantiation, the account_msgs messages are transformed from Vec<WarpMsg> to Vec<ComosMsg> using the warp_msgs_to_cosmos_msgs function and executed at the end. The executed message is also included as a string in the event attributes.

The issue is that the reply function create_job_account_and_job extracts the account_msgs variable from those event attributes and tries to cast it as Vec<WarpMsg> to run again, so it produces a deserialization error.

It is recommended to leave only one of the executions, the one from the instantiation or the one from the reply function.

Code Location:

Snippet of code of the instantiate function in the **warp-account** contract:

Listing 2: contracts/warp-account/src/contract.rs (Lines 24,27,37)

```
22     CONFIG.save(deps.storage, &config)?;
23
24     let msgs = warp_msgs_to_cosmos_msgs(deps.as_ref(), env, msg.
↳ msgs, &config.owner).unwrap();
25
26     Ok(Response::new()
27         .add_messages(msgs.clone())
28         .add_attribute("action", "instantiate")
29         .add_attribute("job_id", msg.job_id)
30         .add_attribute("contract_addr", instantiated_account_addr)
```

```
31          .add_attribute("owner", msg.owner)
32          .add_attribute(
33              "native_funds",
34              serde_json_wasm::to_string(&msg.native_funds)?,
35          )
36          .add_attribute("cw_funds", serde_json_wasm::to_string(&msg
↳ .cw_funds)?))
37          .add_attribute("account_msgs", serde_json_wasm::to_string
↳ (&msgs)?))
38  }
```

Snippet of code of the create_account_and_job function in the **warp-controller** contract:

Listing 3: contracts/warp-controller/src/reply/account.rs (Lines 84-92,123-129)

```
84  let account_msgs: Option<Vec<WarpMsg>> = serde_json_wasm::from_str
↳ (
85    &account_event
86        .attributes
87        .iter()
88        .cloned()
89        .find(|attr| attr.key == "account_msgs")
90        .ok_or_else(|| StdError::generic_err("cannot find `
↳ account_msgs` attribute"))?
91        .value,
92  )?;
93
94  let mut job = JobQueue::get(deps.storage, job_id)?;
95  job.account = account_addr.clone();
96  JobQueue::sync(deps.storage, env, job.clone())?;
97
98  let mut msgs: Vec<CosmosMsg> = vec![];
99
100 if let Some(cw_funds) = cw_funds.clone() {
101     // Fund account in CW20 / CW721 tokens
102     for cw_fund in cw_funds {
103         msgs.push(match cw_fund {
104             CwFund::Cw20(cw20_fund) => build_transfer_cw20_msg(
105                 deps.api
106                     .addr_validate(&cw20_fund.contract_addr)?
107                     .to_string(),
```

```
                owner.clone(),
                account_addr.clone().to_string(),
                cw20_fund.amount,
            ),
            CwFund::Cw721(cw721_fund) => build_transfer_cw721_msg(
                deps.api
                    .addr_validate(&cw721_fund.contract_addr)?
                    .to_string(),
                account_addr.clone().to_string(),
                cw721_fund.token_id.clone(),
            ),
        })
    }
}

if let Some(account_msgs) = account_msgs {
 // Account execute msgs
 msgs.push(build_account_execute_warp_msgs(
     account_addr.to_string(),
     account_msgs,
 ));
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to leave only one of the executions, the one from the instantiation or the one from the reply function.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID b82d861 by removing the execution from the reply function.

4.3 (HAL-03) MALICIOUS OWNER COULD CREATE CORRUPTED JOB ACCOUNTS - LOW (2.8)

Description:

In the hypothetical case where the controller-owner account was compromised, this could lead to an attack scenario where the attacker could incorporate corrupted job accounts into the system.

The **warp-account** contract can be instantiated by any user, storing a configuration where the config.creator_addr is the info.sender (malicious owner) and the config.owner would be any address specified in the InstantiationMsg.

At first glance, this is not a risk, since this corrupted job account will not be used by the account tracker or the controller contract. However, the malicious owner has privileges over the **warp-account-tracker** executions, so the attack scenario would be as follows:

- The malicious owner instantiates some account contracts by specifying the addresses of the current users as config.owner and his address as config.creator_addr.
- The malicious owner has permission to execute in warp-account-tracker contract, so it executes the free_job_account entry point, adding these fake accounts to each user's free-account-queue.

- When a user tries to create a new job and there are no previous accounts available, the corrupted account would be assigned to the user.
- That job will never be executed because the controller has no permissions on it, but all funds the user deposits into this corrupted account could be withdrawn by the malicious owner by executing a warp_msg of type generic/bank/send/to_address=malicious_addr since the config.creator_addr (malicious owner) has permission to execute.

Although this is an unlikely scenario, it is recommended to remove the controller's owner (config.admin) privilege over execution in the **warp-account-tracker** contract. This way, the account queues could not be manipulated by any external element and only by the Controller contract.

Code Location:

Snippet of code of the execute function in the **warp-account-tracker** contract:

Listing 4: contracts/warp-account-tracker/src/contract.rs (Lines 42-44)

```
35 pub fn execute(
36     deps: DepsMut,
37     _env: Env,
38     info: MessageInfo,
39     msg: ExecuteMsg,
40 ) -> Result<Response, ContractError> {
41     let config = CONFIG.load(deps.storage)?;
42     if info.sender != config.admin && info.sender != config.
↳ warp_addr {
43         return Err(ContractError::Unauthorized {});
44     }
45
46     match msg {
47         ExecuteMsg::TakeJobAccount(data) => {
48             nonpayable(&info).unwrap();
49             execute::account::take_job_account(deps, data)
50         }
51         ExecuteMsg::FreeJobAccount(data) => {
52             nonpayable(&info).unwrap();
53             execute::account::free_job_account(deps, data)
54         }
55         ExecuteMsg::TakeFundingAccount(data) => {
56             nonpayable(&info).unwrap();
57             execute::account::take_funding_account(deps, data)
58         }
59         ExecuteMsg::FreeFundingAccount(data) => {
60             nonpayable(&info).unwrap();
61             execute::account::free_funding_account(deps, data)
62         }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:C/A:M/D:C/Y:N/R:N/S:U (2.8)

Recommendation:

It is recommended to remove the controller's owner (config.admin) privilege over execution in the **warp-account-tracker** contract. This way, the account queues could not be manipulated by any external element and only by the Controller contract.

Remediation Plan:

RISK ACCEPTED: The Terra team accepted the risk of this finding, stating that it does not pose a problem as their mainnet contract will be instantiated with TFL deployment multisig as admin.

4.4 (HAL-04) EVICT JOB FUNCTION DOES NOT RELEASE FUNDING ACCOUNT - LOW (2.5)

Description:

The evict_job function of the **warp-controller** contract allows evicting a job that has been in the JobQueue for too long.

The way it works is similar to the delete_job function: the JobStatus is changed to Evicted and the job account is released.

However, the funding account associated with the job account is not released, and the assets stored in the job are not returned to the owner.

Code Location:

Snippet of code of the evict_job function in the **warp-controller** contract:

Listing 5: contracts/warp-controller/src/execute/job.rs

```
567     // Controller sends eviction reward to evictor
568     msgs.push(build_transfer_native_funds_msg(
569         info.sender.to_string(),
570         vec![Coin::new(eviction_fee.u128(), config.fee_denom.clone
 ↳ ())],
571     ));
572
573     // Controller sends execution reward minus eviction reward
 ↳ back to account
574     msgs.push(build_transfer_native_funds_msg(
575         job.account.to_string(),
576         vec![Coin::new(
577             (job.reward - eviction_fee).u128(),
578             config.fee_denom.clone(),
579         )],
580     ));
581
582     // Free account
```

```
583     msgs.push(build_free_job_account_msg(
584         config.account_tracker_address.to_string(),
585         job.owner.to_string(),
586         account_addr.to_string(),
587         job.id,
588     ));
589 
590     Ok(Response::new()
591         .add_messages(msgs)
592         .add_attribute("action", "evict_job")
593         .add_attribute("job_id", job.id)
594         .add_attribute("job_status", serde_json_wasm::to_string(&
 ↳ job_status)?))
595 }
596 
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to release the funding account once the job has been evicted, as well as returning the assets stored in the job account to the owner.

Remediation Plan:

SOLVED: The Terra team solved this issue in commits c01733e and aeb629b.

4.5 (HAL-05) JOB DURATION IS NOT CHECKED - LOW (2.5)

Description:

The create_job function of **warp-controller** does not validate the duration_days input parameter before storing it in the Job structure, so it could have any value.

In exchange for paying the maximum maintenance fee, duration_days could have an “infinite” value, so this job will never be evicted. Replicating this technique could lead to an overload in the job queue.

Code Location:

Snippet of code of the crete_job function in the **warp-controller** contract:

Listing 6: contracts/warp-controller/src/execute/job.rs (Line 118)

```
95 let state = STATE.load(deps.storage)?;
96
97 let mut job = JobQueue::add(
98     deps.storage,
99     Job {
100         id: state.current_job_id,
101         prev_id: None,
102         owner: job_owner.clone(),
103         // Account uses a placeholder value for now, will update
 ↳ it to job account address if job account exists or after created
104         // Update will happen either in create_job (exists free
 ↳ job account) or reply (after creation), so it's atomic
105         // And we guarantee we do not read this value before it's
 ↳ updated
106         account: info.sender.clone(),
107         last_update_time: Uint64::from(env.block.time.seconds()),
108         name: data.name,
109         status: JobStatus::Pending,
110         terminate_condition: data.terminate_condition,
111         recurring: data.recurring,
112         vars: data.vars,
```

```
113         executions: data.executions,
114         reward: data.reward,
115         description: data.description,
116         labels: data.labels,
117         assets_to_withdraw: data.assets_to_withdraw.unwrap_or(vec!
 ↳ []),
118         duration_days: data.duration_days,
119         created_at_time: Uint64::from(env.block.time.seconds()),
120         // placeholder, will be updated later on
121         funding_account: None,
122     },
123 )?;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to check if the duration_days value of the job is higher than the config.duration_days_max value.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID 86e8e2a by adding the duration_days_limit parameter.

4.6 (HAL-06) LACK OF VALIDATIONS IN THE CONFIGURATION DATA - LOW (2.5)

Description:

In the **warp-controller** contract, there is no validation of queue_size_right > queue_size_left in the instantiate update_config functions. If this condition is not met, a panic will occur during execution of the compute_creation_fee function (underflow Uint) and subsequently a panic in the create_job function.

Also, it is not checked if queue_size_right == queue_size_left and duration_days_max == duration_days_min. This could cause a panic during the execution of the compute_creation_fee and compute_maintenance_fee functions, caused by a division by zero.

Code Location:

Snippet of code of the compute_creation_fee function in the **warp-controller** contract:

Listing 7: contracts/warp-controller/src/execute/fee.rs (Line 11)

```
 4 pub fn compute_creation_fee(queue_size: Uint64, config: &Config)
↳ -> Uint128 {
 5     let x1 = Uint128::from(config.queue_size_left);
 6     let y1 = config.creation_fee_min;
 7     let x2 = Uint128::from(config.queue_size_right);
 8     let y2 = config.creation_fee_max;
 9     let qs = Uint128::from(queue_size);
10
11     let slope = (y2 - y1) / (x2 - x1);
```

Snippet of code of the compute_maintenance_fee function in the **warp-controller** contract:

Listing 8: contracts/warp-controller/src/execute/fee.rs (Line 29)

```
22 pub fn compute_maintenance_fee(duration_days: Uint64, config: &
↳ Config) -> Uint128 {
23     let x1 = Uint128::from(config.duration_days_min);
24     let y1 = config.maintenance_fee_min;
25     let x2 = Uint128::from(config.duration_days_max);
26     let y2 = config.maintenance_fee_max;
27     let dd = Uint128::from(duration_days);
28
29     let slope = (y2 - y1) / (x2 - x1);
30
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add some validations to the instantiate and update_config functions of the **warp-controller** contract:

- queue_size_right must be higher than queue_size_left.
- queue_size_right must not be equal to queue_size_left.
- duration_days_max must not be equal to duration_days_min.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID 86e8e2a by adding the corresponding validations.

4.7 (HAL-07) OWNERSHIP CAN BE TRANSFERRED WITHOUT CONFIRMATION - LOW (2.0)

Description:

The update_config function from **warp-controller** and **warp-templates** contracts allows changing the owner of the contract. This function can only be executed by the current owner, so if any unintentional error occurs in the execution of the transaction (e.g. an incorrect address), the control over the contract and privileged functionalities would be lost.

It is recommended to perform this type of operation using a two-step process: one transaction to set a value of the new_owner and another transaction to confirm this information signed by the new owner himself.

Code Location:

Snippet of code of the update_config function in the **warp-controller** contract:

Listing 9: contracts/warp-controller/src/execute/controller.rs (Line 20)

```
 7 pub fn update_config(
 8     deps: DepsMut,
 9     _env: Env,
10     info: MessageInfo,
11     data: UpdateConfigMsg,
12     mut config: Config,
13 ) -> Result<Response, ContractError> {
14     if info.sender != config.owner {
15         return Err(ContractError::Unauthorized {});
16     }
17
18     config.owner = match data.owner {
19         None => config.owner,
20         Some(data) => deps.api.addr_validate(data.as_str())?,
```

Snippet of code of the update_config function in the **warp-templates** contract:

Listing 10: contracts/warp-templates/src/contract.rs (Line 227)

```
214 pub fn update_config(
215     deps: DepsMut,
216     _env: Env,
217     info: MessageInfo,
218     data: UpdateConfigMsg,
219 ) -> Result<Response, ContractError> {
220     let mut config = CONFIG.load(deps.storage)?;
221     if info.sender != config.owner {
222         return Err(ContractError::Unauthorized {});
223     }
224
225     config.owner = match data.owner {
226         None => config.owner,
227         Some(data) => deps.api.addr_validate(data.as_str())?,
228     };
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to split ownership transfer functionality into set_owner and accept_ownership functions. The latter function allows the transfer to be completed by the recipient.

Remediation Plan:

RISK ACCEPTED: The Terra team accepted the risk of this finding, stating that it does not pose a problem as their mainnet contract will be instantiated with TFL deployment multisig as admin.

4.8 (HAL-08) NO OPTION TO MODIFY ACCOUNT TRACKER ADMIN - LOW (2.0)

Description:

The config.admin parameter of **warp-account-tracker** contract is saved during instantiation and there is no option to modify it. This address is the same as the **warp-controller** owner address.

Since the **warp-controller** contract provides the option to modify the owner address in the update_config function, there should be the option to modify it in the **warp-account-tracker** contract as well, otherwise, in case the owner address was compromised, it would have access to the **warp-account-tracker** contract and there would be no way to avoid it.

Code Location:

Snippet of code of the instantiate function in the **warp-account-tracker** contract:

Listing 11: contracts/warp-account-tracker/src/contract.rs (Line 21)

```
10 pub fn instantiate(
11     deps: DepsMut,
12     env: Env,
13     _info: MessageInfo,
14     msg: InstantiateMsg,
15 ) -> Result<Response, ContractError> {
16     let instantiated_account_addr = env.contract.address;
17
18     CONFIG.save(
19         deps.storage,
20         &Config {
21             admin: deps.api.addr_validate(&msg.admin)?,
22             warp_addr: deps.api.addr_validate(&msg.warp_addr)?,
23         },
24     )?;
25
26     Ok(Response::new()
```

```
27          .add_attribute("action", "instantiate")
28          .add_attribute("contract_addr", instantiated_account_addr.
↳ clone())
29          .add_attribute("account_tracker",
↳ instantiated_account_addr)
30          .add_attribute("admin", msg.admin)
31          .add_attribute("warp_addr", msg.warp_addr))
32 }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to add a mechanism to modify the config.admin address of the **warp-account-tracker**.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID 7f046ac by adding the UpdateConfig function in the warp-account-tracker contract.

4.9 (HAL-09) USELESS CODE - INFORMATIONAL (0.0)

Description:

The reply ID REPLY_ID_CREATE_FUNDING_ACCOUNT_AND_JOB and the corresponding create_funding_account_and_job function from the **warp-controller** contract were inherited from a previous version of the code and are no longer used. They should be removed from the code to save some gas during deployment.

Code Location:

Snippet of code of the reply function in the **warp-controller** contract:

Listing 12: contracts/warp-controller/src/contract.rs (Lines 167-168)

```
160 pub fn reply(deps: DepsMut, env: Env, msg: Reply) -> Result<
 ↳ Response, ContractError> {
161     let config = CONFIG.load(deps.storage)?;
162
163     match msg.id {
164         REPLY_ID_CREATE_JOB_ACCOUNT_AND_JOB => {
165             reply::account::create_account_and_job(deps, env, msg,
 ↳  config)
166         }
167         REPLY_ID_CREATE_FUNDING_ACCOUNT_AND_JOB => {
168             reply::account::create_funding_account_and_job(deps,
 ↳ env, msg, config)
169         }
170         REPLY_ID_CREATE_FUNDING_ACCOUNT => {
171             reply::account::create_funding_account(deps, env, msg,
 ↳  config)
172         }
173         REPLY_ID_INSTANTIATE_SUB_CONTRACTS => {
174             reply::job::instantiate_sub_contracts(deps, env, msg,
 ↳ config)
175         }
176         _ => reply::job::execute_job(deps, env, msg, config),
177     }
```

BVSS:

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

Recommendation:

It is recommended to delete the unused code to save some gas during deployment.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID 1b4a7fb.

4.10 (HAL-10) REPEATED OPERATIONS - INFORMATIONAL (0.0)

Description:

There are some repeated operations in the same code fragment that add nothing to the functionality.

Code Location:

Snippet of code of the create_job function in the **warp-controller** contract:

Listing 13: contracts/warp-controller/src/execute/job.rs (Lines 53,95)

```
53 let state = STATE.load(deps.storage)?;
54
55 let job_owner = info.sender.clone();
56 let account_tracker_address_ref = &config.account_tracker_address.
↳ to_string();
57
58 let _validate_conditions_and_variables: Option<String> = deps.
↳ querier.query_wasm_smart(
59     &config.resolver_address,
60     &resolver::QueryMsg::QueryValidateJobCreation(resolver::
↳ QueryValidateJobCreationMsg {
61         terminate_condition: data.terminate_condition.clone(),
62         vars: data.vars.clone(),
63         executions: data.executions.clone(),
64     }),
65 )?;
66
67 let creation_fee = compute_creation_fee(state.q, &config);
68 let maintenance_fee = compute_maintenance_fee(data.duration_days,
↳ &config);
69 let burn_fee = compute_burn_fee(data.reward, &config);
70
71 let total_fees = creation_fee + maintenance_fee + burn_fee;
72
73 if data.funding_account.is_none() && data.recurring {
```

```
74      return Err(ContractError::FundingAccountMissingForRecurringJob
↳  {});
75  }
76
77  // ignore operational_amount when funding_account is provided
78  let operational_amount = if data.funding_account.is_some() {
79      Uint128::zero()
80  } else {
81      data.operational_amount
82  };
83
84  // Reward and fee will always be in native denom
85  let native_funds_minus_operational_amount =
↳ deduct_from_native_funds(
86      info.funds.clone(),
87      config.fee_denom.clone(),
88      operational_amount,
89  );
90
91  let mut submsgs = vec![];
92  let mut msgs = vec![];
93  let mut attrs = vec![];
94
95  let state = STATE.load(deps.storage)?;
96
97  let mut job = JobQueue::add(
```

Snippet of code of the update_config function in the **warp-controller** contract:

Listing 14: contracts/warp-controller/src/execute/controller.rs (Lines 47-49,67-69)

```
47  if config.burn_fee_rate.u128() > 100 {
48      return Err(ContractError::BurnFeeTooHigh {});
49  }
50
51  if config.creation_fee_max < config.creation_fee_min {
52      return Err(ContractError::CreationMaxFeeUnderMinFee {});
53  }
54
55  if config.maintenance_fee_max < config.maintenance_fee_min {
56      return Err(ContractError::MaintenanceMaxFeeUnderMinFee {});
```

```
57 }
58
59 if config.duration_days_max < config.duration_days_min {
60     return Err(ContractError::DurationMaxDaysUnderMinDays {});
61 }
62
63 if config.cancellation_fee_rate.u64() > 100 {
64     return Err(ContractError::CancellationFeeTooHigh {});
65 }
66
67 if config.burn_fee_rate.u128() > 100 {
68     return Err(ContractError::BurnFeeTooHigh {});
69 }
```

BVSS:

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

Recommendation:

It is recommended to delete the repeated code.

Remediation Plan:

SOLVED: The Terra team solved this issue in commit ID ae4f47c.

AUTOMATED TESTING

5.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. To better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Listing 15: Dependency tree

```
1 No security detections.
```

THANK YOU FOR CHOOSING

// HALBORN