// HALBORN

Valkyrie Protocol

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 20th, 2021 - October 22nd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/20/2021	Luis Quispe Gonzales
0.2	Document Updates	10/18/2021	Luis Quispe Gonzales
0.3	Draft Version	10/22/2021	Luis Quispe Gonzales
0.4	Draft Version Review	11/05/2021	Gabi Urrutia
1.0	Remediation Plan	01/18/2022	Luis Quispe Gonzales
1.1	Remediation Plan Review	01/18/2022	Gabi Urrutia

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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Valkyrie Protocol engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on September 20th, 2021 and ending October 22nd, 2021.

The security engineers involved on the audit are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

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

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

- Remove execution logic from campaign contract.
- Update the logic of some functions in campaign contract to handle correctly edge cases (e.g.: no collateral deposited, no collateral denom specified, no rewards deposited).
- Generates automatically URLs for created campaigns and do not allow their modification.
- Fix calculus for rewards balance in campaigns.
- Remove transfer function in distributor contract to avoid rug-pull related attacks.
- Restrict the modification of execution messages when campaigns are ongoing.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & 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 audit. 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 audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual assessment of use and safety for the critical Rust variables and functions in scope to identify any contracts logic related vulnerability.
- Fuzz testing (Halborn custom fuzzing tool)
- Checking the test coverage (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities (cargo audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.3 SCOPE

Code repository: https://github.com/valkyrieprotocol/contracts

1. CosmWasm Smart Contracts - Campaign flow
    (a) Commit ID: a4b9967c210e4ffd9f96656dcc5cf007611f6097
    (b) Contracts in scope:
        i. campaign
        ii. campaign_manager
2. CosmWasm Smart Contracts - Remaining flows
    (a) Commit ID: 2e1820a0d15bbbf91bf35e71f61418a64e25aad7
    (b) Contracts in scope:
        i. community
        ii. distributor
        iii. governance
        iv. lp_staking

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	6	5	5	5

LIKELIHOOD

IMPACT

(HAL-10)		(HAL-06)	(HAL-03) (HAL-04)	(HAL-01) (HAL-02)
(HAL-14)	(HAL-11) (HAL-12)		(HAL-07)	(HAL-05)
	(HAL-15) (HAL-16) (HAL-17)	(HAL-13)		(HAL-08)
(HAL-19) (HAL-20)				(HAL-09)
(HAL-23)	(HAL-21) (HAL-22)	(HAL-18)		

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) ARBITRARY EXECUTION MESSAGES LEAD TO STEALING ALL FUNDS FROM CAMPAIGNS	Critical	SOLVED - 10/04/2021
(HAL-02) WITHDRAWAL OF ARBITRARY PARTICIPATION REWARDS WITHOUT DEPOSITING COLLATERALS	Critical	PARTIALLY SOLVED
(HAL-03) DEPOSITS GET LOCKED IN CAMPAIGN IF COLLATERAL DENOM IS NOT SPECIFIED	High	SOLVED - 09/30/2021
(HAL-04) CAMPAIGNS CAN HOST MALICIOUS URLS THAT HARM USERS	High	RISK ACCEPTED
(HAL-05) BALANCE DOES NOT UPDATE WHEN USERS CLAIM REWARDS	High	SOLVED - 10/11/2021
(HAL-06) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF VKR TOKENS FROM DISTRIBUTOR	High	SOLVED - 12/29/2021
(HAL-07) USERS CAN PARTICIPATE IN CAMPAIGNS EVEN IF THERE ARE NOT ENOUGH REWARDS	High	SOLVED - 10/13/2021
(HAL-08) NO RESTRICTION TO UPDATE EXECUTION MESSAGES WHEN CAMPAIGNS ARE ONGOING	High	SOLVED - 10/04/2021
(HAL-09) FUNCTION TO UPDATE STAKING CONFIG DOES NOT WORK PROPERLY	Medium	SOLVED - 10/21/2021
(HAL-10) COLLUDED STAKERS CAN TRANSFER VKR TOKENS OUTSIDE GOVERNANCE CONTRACT	Medium	SOLVED - 12/29/2021
(HAL-11) PRIVILEGED ADDRESSES CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 01/18/2022
(HAL-12) NOT ENFORCING SLIPPAGE TOLERANCE COULD LEAD TOKENS LOSS	Medium	SOLVED - 12/30/2021
(HAL-13) DISTRIBUTION TIME FRAME CAN BE REDUCED UNRESTRICTEDLY	Medium	SOLVED - 12/29/2021
(HAL-14) NO MINIMUM THRESHOLD FOR EFFECTIVE DELAY PERIOD	Low	SOLVED - 12/29/2021

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-15) DISTRIBUTION CAN BE UPDATED WITH AN AMOUNT LESSER THAN RELEASED ONE	Low	SOLVED - 10/22/2021
(HAL-16) RATES / PERCENTAGES COULD BE SET TO VALUES GREATER THAN 1	Low	SOLVED - 12/29/2021
(HAL-17) VOTERS CAN FORCE UNEXPECTED QUORUM WITHIN SNAPSHOT PERIOD	Low	RISK ACCEPTED
(HAL-18) ANYONE CAN PARTICIPATE ON BEHALF OF OTHER USERS	Low	SOLVED - 12/29/2021
(HAL-19) DEPOSIT / PARTICIPATION PARAMETERS CAN BE UPDATED IN NOT PENDING CAMPAIGNS	Informational	SOLVED - 12/29/2021
(HAL-20) PASSED POLLS DO NOT AUTOMATICALLY EXPIRE IF NOT EXECUTED	Informational	ACKNOWLEDGED
(HAL-21) MISCALCULATION OF DEPOSIT VALUE WHEN VALIDATING REWARD POOL WEIGHT	Informational	SOLVED - 12/21/2021
(HAL-22) MISCALCULATION OF REMAIN AMOUNT WHEN ADMIN TRANSFERS VKR TOKENS	Informational	SOLVED - 12/29/2021
(HAL-23) FUNCTION WITH UNUSED ARGUMENT	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) ARBITRARY EXECUTION MESSAGES LEAD TO STEALING ALL FUNDS FROM CAMPAIGNS - CRITICAL

Description:

participate and participate_qualify_result functions in **contracts/campaign/src/executions.rs** allow a creator to execute any message on behalf of the **campaign** contract. Consequently, a malicious creator can include an execution message for withdrawing all deposits from a campaign, i.e., stealing deposits from users that have participated in that campaign.

The risk level for this finding increases because there are no restrictions to update execution messages to malicious ones when campaigns are ongoing, see HAL-08 for more details.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Loop of messages to be executed without restrictions on behalf of the **campaign** contract using the participate function:

Listing 1: contracts/campaign/src/executions.rs (Lines 787)

```
787 for execution in campaign_config.executions.iter() {
788         response = response.add_message(message_factories::
                wasm_execute_bin(
789             &execution.contract,
790             execution.msg.clone(),
791         ));
792 }
```

Loop of messages to be executed without restrictions on behalf of the **campaign** contract using the participate_qualify_result function:

Listing 2: contracts/campaign/src/executions.rs (Lines 839)

```
836 if continue_option.can_execute() {
837     let campaign_config = CampaignConfig::load(deps.storage)?;
838
839     for execution in campaign_config.executions.iter() {
840         response = response.add_message(message_factories::
                wasm_execute_bin(
841             &execution.contract,
842             execution.msg.clone(),
843         ));
844     }
845 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to remove all execution logic from the **campaign** contract. If it is not possible, update the logic in participate and participate_qualify_result functions to not allow transfers of tokens outside the **campaign** contract.

Remediation plan:

SOLVED: The issue was fixed in commit 1e1de2243655fa3f083dec248256651e48dbb83b. Valkyrie team removed the execution logic from the **campaign** contract and moved it to the **qualifier** contract, which is written by the campaign creator and should have zero balance.

3.2 (HAL-02) WITHDRAWAL OF ARBITRARY PARTICIPATION REWARDS WITHOUT DEPOSITING COLLATERALS - CRITICAL

Description:

When a campaign is activated and no one has deposited collaterals yet, an attacker can participate an undefined number of times without depositing any collateral. Later on, attacker will be able to withdraw his illegitimate participation rewards.

This issue arises because, when a user participates, the internal require_collateral function in **contracts/campaign/src/states.rs** is triggered and will always return false if collateral amount is zero, i.e.: no one has deposited collaterals yet. As a consequence, validation that user owns enough collateral balance will be bypassed and participation_count field for user will increase by one for each participation, without the need to deposit any collateral.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 3: contracts/campaign/src/executions.rs (Lines 738)

```
738 if campaign_config.require_collateral() {
739     let mut collateral = Collateral::load_or_new(deps.storage, &
        actor)?;
740
741     let collateral_balance = collateral.balance(env.block.height)
        ?;
742
743     if collateral_balance < campaign_config.collateral_amount {
744         return Err(ContractError::Std(StdError::generic_err(
            format!(
745             "Insufficient collateral balance (required: {},
```

```
                current: {})",
746             campaign_config.collateral_amount.to_string(),
747             collateral_balance.to_string(),
748         ))));
749     }
750
751     collateral.lock(campaign_config.collateral_amount, env.block.
        height, campaign_config.collateral_lock_period)?;
752     collateral.save(deps.storage)?;
753 }
```

Listing 4: contracts/campaign/src/states.rs (Lines 52)

```
51 pub fn require_collateral(&self) -> bool {
52     self.collateral_denom.is_some() && !self.collateral_amount.
        is_zero()
53 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Update the logic of require_collateral function to handle correctly the cases where no one has yet deposited collaterals in campaigns.

Remediation plan:

PARTIALLY SOLVED: Commit 5b89ebd6767d031f168ad66dcba7a9fa23b25483 partially fixes the security issue by extending the require_collateral verification even if collateral amount is zero.

On the other hand, by design, this protocol accepts that users participate in campaigns without depositing collaterals, which could lead that malicious users participate without restrictions and in unfair fashion

to earn rewards. The Valkyrie team decided to implement the following measures in commit b3586c8c869b16cdc4a4a1ed8c2d8f46d9531702 to partially reduce the attack surface:

- Make **qualifier** mandatory, so creators can include additional security requirements in their campaigns.
- Limit the participation count per address up to 100 when collateral_denom is **None**.

It is highly recommended that Valkyrie documentation warns creators about the security risks of creating campaigns that do not require collaterals and the security requirements they should consider in their **qualifier** contracts.

3.3 (HAL-03) DEPOSITS GET LOCKED IN CAMPAIGN IF COLLATERAL DENOM IS NOT SPECIFIED - HIGH

Description:

When users call deposit_collateral function in **contracts/campaign/src/executions.rs** for a campaign which collateral_denom has not been specified at instantiation time, the contract will receive the deposits as usual, without throwing any error message to users.

However, for when users call withdraw_collateral function to withdraw their deposits from **campaign** contract, the function will throw an error message and deposits will get locked forever in contract.

Code Location:

Conditional expression in deposit_collateral function does not consider **else** case: reject operation if campaign_config.collateral_denom is **None**:

Listing 5: contracts/campaign/src/executions.rs (Lines 1088)

```
1086  let campaign_config = CampaignConfig::load(deps.storage)?;
1087
1088  if let Some(collateral_denom) = campaign_config.collateral_denom
          {
1089      if *send_denom != collateral_denom {
1090          return Err(ContractError::Std(StdError::generic_err("
                Missing collateral denom")));
1091      }
1092  }
1093
1094  let mut campaign_state = CampaignState::load(deps.storage)?;
```

When users call withdraw_collateral function to withdraw their deposits, function will always throw a **“No collateral”** error message:

Listing 6: contracts/campaign/src/executions.rs (Lines 1153,1154,1155)

```
1141  if let Some(denom) = campaign_config.collateral_denom {
1142      let mut campaign_state = CampaignState::load(deps.storage)?;
1143
1144      campaign_state.collateral_amount = campaign_state.
              collateral_amount.checked_sub(amount)?;
1145      campaign_state.save(deps.storage)?;
1146
1147      response = response.add_message(make_send_msg(
1148          &deps.querier,
1149          denom,
1150          amount,
1151          &info.sender,
1152      )?);
1153  } else {
1154      return Err(ContractError::Std(StdError::generic_err("No
              collateral")));
1155  }
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

Update the logic in deposit_collateral function to reject deposits in campaigns which collateral_denom has not been specified.

Remediation plan:

SOLVED: The issue was fixed in commit e45c721568be661fc15d1dd20473ec54b61d1ca1.

3.4 (HAL-04) CAMPAIGNS CAN HOST MALICIOUS URLS THAT HARM USERS - HIGH

Description:

When a campaign is created with create_campaign function from **contracts/campaign_manager/src/executions.rs** or updated with update_campaign_config function from **contracts/campaign/src/executions.rs**, creator can introduce a malicious external URL that fools users to be redirected to a phishing DApp that can steal their deposits. It is important to note that URLs will appear in official Valkyrie frontend, so legitimate users won't be aware whether those URLs are malicious or not.

Attack scenario:

1. Malicious creator calls create_campaign function to create a campaign with a valid URL.
2. Malicious creator sends the address of the campaign to Valkyrie team, who publishes it in official Valkyrie frontend.
3. Malicious creator calls update_campaign_config function to update URL to a malicious one.
4. Users click a malicious URL that appears in Valkyrie frontend and are redirected to a phishing DApp.
5. Users interact with the phishing DApp and try to participate in the campaign, but their deposits are sent to a malicious address instead of to the actual campaign.

Code Location:

Creator can send an arbitrary config_msg that contains a malicious URL when creating a new campaign:

Listing 7: contracts/campaign_manager/src/executions.rs (Lines 199)

```
195 pub fn create_campaign(
196     deps: DepsMut,
197     env: Env,
198     info: MessageInfo,
199     config_msg: Binary,
200     collateral_denom: Option<Denom>,
201     collateral_amount: Option<Uint128>,
202     collateral_lock_period: Option<u64>,
203     qualifier: Option<String>,
204     qualification_description: Option<String>,
205     executions: Vec<ExecutionMsg>,
206 ) -> ContractResult<Response> {
```

Listing 8: contracts/campaign_manager/src/executions.rs (Lines 227)

```
218 let create_campaign_msg = message_factories::wasm_instantiate(
219     config.code_id,
220     Some(config.governance.clone()),
221     to_binary(&CampaignInstantiateMsg {
222         governance: config.governance.to_string(),
223         fund_manager: config.fund_manager.to_string(),
224         campaign_manager: env.contract.address.to_string(),
225         admin: info.sender.to_string(),
226         creator: info.sender.to_string(),
227         config_msg,
228         collateral_denom,
```

Creator can even update URL to a malicious one for campaign when is in **pending** status:

Listing 9: contracts/campaign/src/executions.rs (Lines 152)

```
143  if let Some(url) = url.as_ref() {
144     validate_url(url)?;
145
```

```
146     if !is_pending(deps.storage)? {
147         return Err(ContractError::Std(StdError::generic_err(
148             "Only modifiable in pending status",
149         )));
150     }
151
152     campaign_config.url = url.clone();
153     response = response.add_attribute("is_updated_url", "true");
154 }
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

It is recommended that **campaign_manager** contract generates automatically URLs for created campaigns (e.g.,: https://app.valkyrieprotocol.com/campaigns/terra1e9...7mpw) and allows users to participate in campaigns directly through Valkyrie frontend, without the need of external sites. Furthermore, creators shouldn't be able to update those URLs.

Remediation plan:

RISK ACCEPTED: The Valkyrie team accepted the risk for this finding. They also claimed that this is a critical part of the creator's process and workflow when creating a campaign, so it must be necessary for the creators to input their link and the protocol must allow for flexibility.

3.5 (HAL-05) BALANCE DOES NOT UPDATE WHEN USERS CLAIM REWARDS - HIGH

Description:

The claim_participation_reward and claim_referral_reward functions in **contracts/campaign/src/executions.rs** allow users to claim their rewards when they participate or promote campaigns.

Every time these functions are called, they do not update total reward balance, which allows other users to participate in campaigns even if there are not enough rewards for them and affects all rewardable ecosystem of Valkyrie protocol.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

After unlocking balance, total participation reward balance does not update when reward is claimed:

Listing 10: contracts/campaign/src/executions.rs (Lines 654)

```
649  let mut campaign_state = CampaignState::load(deps.storage)?;
650
651  let reward_amount = participation.participation_reward_amount;
652
653  participation.participation_reward_amount = Uint128::zero();
654  campaign_state.unlock_balance(&reward_config.
         participation_reward_denom, &reward_amount)?;
655
656  participation.save(deps.storage)?;
657  campaign_state.save(deps.storage)?;
```

After unlocking balance, total referral reward balance does not update when reward is claimed:

Listing 11: contracts/campaign/src/executions.rs (Lines 695)

```
690  let mut campaign_state = CampaignState::load(deps.storage)?;
691
692  let reward_amount = participation.referral_reward_amount;
693
694  participation.referral_reward_amount = Uint128::zero();
695  campaign_state.unlock_balance(
696      &cw20::Denom::Cw20(reward_config.referral_reward_token.clone()
         ),
697      &reward_amount,
698  )?;
699
700  participation.save(deps.storage)?;
701  campaign_state.save(deps.storage)?;
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Fix the logic in claim_participation_reward and claim_referral_reward functions to update total balance when rewards are claimed.

Remediation plan:

SOLVED: The issue was fixed in commit 397af636356390334af89a68f93f8d0340124e61. The Valkyrie team also discovered this security issue while a security audit was in progress and solved it timely.

3.6 (HAL-06) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF VKR TOKENS FROM DISTRIBUTOR - HIGH

Description:

The transfer function in **contracts/distributor/src/executions.rs** allows an admin (Valkyrie team) to transfer an arbitrary amount of VKR tokens from **distributor** contract to a potentially malicious external account.

The maximum amount of VKR tokens to transfer depends on unlocked balance. However, admin can call update_distribution function to unlock all balance by updating start_height (see HAL-13 finding). According to Valkyrie documentation, this contract can concentrate up to 40% of total supply of VKR tokens.

Attack scenario:

1. Malicious (or compromised) admin calls update_distribution function with start_height = <any_value_greater_than_current_block_height> and amount = 0.
2. As a consequence of **Step 1**, the aforementioned function will unlock all balance in contract.
3. Malicious (or compromised) admin calls transfer function to totally withdraw VKR tokens from **distributor** contract.
4. There won't be more VKR tokens to distribute to **lp_staking** and **governance** contracts anymore.

Code Location:

Listing 12: contracts/distributor/src/executions.rs (Lines 305)

```
300  // Execute
301  let mut response = make_response("transfer");
302
303  response = response.add_message(message_factories::wasm_execute(
304      &config.managing_token,
305      &Cw20ExecuteMsg::Transfer {
306          recipient: deps.api.addr_validate(&recipient)?.to_string(),
307          amount,
308      },
309  ));
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

If not used, it is recommended to totally remove `transfer` function to avoid rug-pull related attacks.

Remediation plan:

SOLVED: The issue was fixed in commit 9cb490064cdf6f1d37b2373644d355aaec9f2d8f.

3.7 (HAL-07) USERS CAN PARTICIPATE IN CAMPAIGNS EVEN IF THERE ARE NOT ENOUGH REWARDS - HIGH

Description:

When a campaign is activated and its creator has not deposited participation / referral rewards yet, users are allowed to participate in the campaign, even if there is no balance for rewards distribution. As a consequence, unless campaign creator deposits all rewards accrued unexpectedly in campaign, some users will not be able to claim their rewards.

This issue arises because, when a user participates, the internal validate_balance function in **contracts/campaign/src/states.rs** is triggered and will always return Ok(()) if balance has no elements, i.e.: if creator has not deposited rewards yet.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 13: contracts/campaign/src/states.rs (Lines 157,165)

```
156 pub fn validate_balance(&self) -> StdResult<()> {
157     for (denom, balance) in self.balances.iter() {
158         let locked_balance = self.locked_balance(denom);
159 
160         if *balance < locked_balance {
161             return Err(StdError::generic_err("locked balance can't
                    greater than balance"));
162         }
163     }
164 
165     Ok(())
166 }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Update the logic of `validate_balance` function to handle correctly the cases where creators have not yet deposited rewards in campaigns.

Remediation plan:

SOLVED: The issue was fixed in commit 1026b29b7cefc6a9e3af3800d2c1061718afe14d.

3.8 (HAL-08) NO RESTRICTION TO UPDATE EXECUTION MESSAGES WHEN CAMPAIGNS ARE ONGOING - HIGH

Description:

The update_campaign_config function in **contracts/campaign/src/executions.rs** allows creators to update execution messages when campaigns are ongoing, i.e., last active height is different to **None**. This situation can produce the following consequences:

- A malicious creator can update execution messages when a campaign is ongoing and force contract to transfer the whole collateral deposits to him, i.e., stealing deposits from users that have participated in that campaign. See HAL-01 for more details regarding the exploiting of this vulnerability.
- Users who participate in campaigns may be unaware that the execution messages have changed to malicious or disadvantageous ones and, of course, cannot react timely if the change made is not in the best interest of them.

Code Location:

Listing 14: contracts/campaign/src/executions.rs

```
190  if let Some(executions) = executions.as_mut() {
191      executions.sort_by_key(|e| e.order);
192      campaign_config.executions = executions.iter()
193          .map(|e| Execution::from(deps.api, e))
194          .collect::<StdResult<Vec<Execution>>>()?;
195      response = response.add_attribute("is_updated_executions", "
             true");
196  }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Update the logic of update_campaign_config function to restrict the modification of execution messages when campaigns are ongoing.

Remediation plan:

SOLVED: The issue was fixed in commit 1e1de2243655fa3f083dec248256651e48dbb83b. The Valkyrie team updated the logic of update_campaign_config function to not allow changes in execution messages.

3.9 (HAL-09) FUNCTION TO UPDATE STAKING CONFIG DOES NOT WORK PROPERLY - MEDIUM

Description:

The update_staking_config function in **contracts/governance/src/staking/executions.rs** is not restricted to be called only by admin (Governance contract). Moreover, all changes made are not saved appropriately in contract's storage.

As a consequence of malfunction of aforementioned function, admin will never be able to update distributor field for **governance** contract and could stop receiving rewards adequately.

Code Location:

Listing 15: contracts/governance/src/staking/executions.rs

```
36 pub fn update_staking_config(
37     deps: DepsMut,
38     _env: Env,
39     _info: MessageInfo,
40     distributor: Option<String>,
41 ) -> ContractResult<Response> {
42     // Execute
43     let mut response = make_response("update_staking_config");
44
45     let mut config = StakingConfig::load(deps.storage)?;
46
47     if let Some(distributor) = distributor {
48         config.distributor = Some(deps.api.addr_validate(
               distributor.as_str())?);
49         response = response.add_attribute("is_updated_distributor"
               , "true");
50     }
51
52     Ok(response)
```

Risk Level:

Likelihood - 5
Impact - 2

Recommendation:

Update the logic in update_staking_config function to saves all changes appropriately in contract's storage. Also, restrict access to a function in such a way that can only be called by admin.

Remediation plan:

SOLVED: The issue was fixed in commit 753da9627f9dad0ee089415937178a33ebe4796d.

3.10 (HAL-10) COLLUDED STAKERS CAN TRANSFER VKR TOKENS OUTSIDE GOVERNANCE CONTRACT - MEDIUM

Description:

The run_execution function in **contracts/governance/src/poll/executions.rs** allows stakers (users that have voting power) to execute any message on behalf of **governance** contract.

Because there are no restrictions regarding messages to be executed and **governance** contract can concentrate all staked VKR tokens plus up to 10% of total supply of VKR tokens in rewards (according to Valkyrie documentation), malicious stakers have a strong motivation to collude and approve an execution message to transfer all tokens outside contract.

The risk level for this finding increases because there is no minimum threshold for effective delay period (see HAL-14), so malicious stakers can even execute the transfer message immediately.

Code Location:

Listing 16: contracts/governance/src/poll/executions.rs (Lines 353)

```
348 // Execute
349 let mut response = make_response("run_execution");
350
351 executions.sort_by_key(|e| e.order);
352
353 for execution in executions.iter() {
354     response = response.add_message(message_factories::
        wasm_execute_bin(
355         &deps.api.addr_validate(&execution.contract)?,
356         execution.msg.clone(),
357     ));
358 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Update the logic in run_execution function to not allow transfer of VKR tokens outside **governance** contract. If not possible, at least limit the amount of VKR tokens to be transferred to a reasonable value.

Remediation plan:

SOLVED: The issue was fixed in commit b031a838feaad9ccb0c2b24dff9966bd4f4f85fc.

3.11 (HAL-11) PRIVILEGED ADDRESSES CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

An incorrect use of update_campaign_config and update_config functions in contracts can set owner to an invalid address and inadvertently lose control of the contracts, which cannot be undone in any way. Currently, the owner of the contracts can change **admin / governance address (owner)** using the aforementioned functions in a single transaction and without confirmation from the new address.

The affected smart contracts are the following:

- campaign
- campaign_manager
- distributor
- community

Code Location:

Listing 17: contracts/campaign/src/executions.rs

```
198  if let Some(admin) = admin.as_ref() {
199      campaign_config.admin = deps.api.addr_validate(admin)?;
200      response = response.add_attribute("is_updated_admin", "true");
201  }
```

Listing 18: contracts/campaign_manager/src/executions.rs

```
70  if let Some(governance) = governance.as_ref() {
71      config.governance = deps.api.addr_validate(governance)?;
72      response = response.add_attribute("is_updated_governance", "
            true");
73  }
```

Listing 19: contracts/distributor/src/executions.rs

```
50 if let Some(admins) = admins.as_ref() {
51     config.admins = admins.iter()
52           .map(|v| deps.api.addr_validate(v))
53           .collect::<StdResult<Vec<Addr>>>()?;
54     response = response.add_attribute("is_updated_admins", "true");
55 }
```

Listing 20: contracts/community/src/executions.rs

```
48 if let Some(admins) = admins.as_ref() {
49     config.admins = admins.iter()
50           .map(|v| deps.api.addr_validate(v))
51           .collect::<StdResult<Vec<Addr>>>()?;
52     response = response.add_attribute("is_updated_admins", "true");
53 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

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

If a smart contract accepts more than one admin, it is recommended to aggregate / update them one-by-one, and not massively.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- b76df77a6b72ad9a3a4aee7987d458f4f8f18cb8
- 031e4d54bfbbf2adf399c109c2a2960f7fc262b7

- 0b0dae47029aac265f4d84809a17fd07cdb433e9
- 8b816adaed0ec917b324a8e26921f826b9161b88

3.12 (HAL-12) NOT ENFORCING SLIPPAGE TOLERANCE COULD LEAD TOKENS LOSS - MEDIUM

Description:

The auto_stake function from **contracts/lp_staking/src/executions.rs** does not enforce slippage_tolerance parameter when users provide liquidity to **lp_staking** contract. As a consequence, if a user mistakenly (or fooled by an attacker) provides liquidity with an imbalanced asset pair, he could lose all his leftover tokens.

As an example, if a user provides liquidity of **100 VKR** and **254.039006 UST** for **lp_staking** contract, he receives 150.987532 LP.

On the other hand, if the user provides liquidity of **100 VKR** and **25403.9006 UST**, he also receives 150.987532 LP, the same amount of LP tokens than previous transaction, but spending **100 times more** UST.

Code Location:

Listing 21: contracts/lp_staking/src/executions.rs (Lines 48)

```
43 pub fn auto_stake(
44     deps: DepsMut,
45     env: Env,
46     info: MessageInfo,
47     token_amount: Uint128,
48     slippage_tolerance: Option<Decimal>,
49 ) -> StdResult<Response> {
```

Listing 22: contracts/lp_staking/src/executions.rs (Lines 114)

```
92 // 3. Provide liquidity
93 response = response.add_message(message_factories::
       wasm_execute_with_funds(
94     &config.pair,
```

```
95      vec![Coin {
96          denom: UST.to_string(),
97          amount: uusd_amount.checked_sub(tax_amount)?,
98      }],
99      &PairExecuteMsg::ProvideLiquidity {
100         assets: [
101             Asset {
102                 amount: (uusd_amount.checked_sub(tax_amount))?,
103                 info: AssetInfo::NativeToken {
104                     denom: UST.to_string(),
105                 },
106             },
107             Asset {
108                 amount: token_amount,
109                 info: AssetInfo::Token {
110                     contract_addr: token_addr.to_string(),
111                 },
112             },
113         ],
114         slippage_tolerance,
115         receiver: None,
116     },
117 ));
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Enforce slippage_tolerance parameter in auto_stake function and add a validation routine to ensure that this value is lesser or equal than a predefined **max value**. As a reference, **max slippage tolerance** for Uniswap liquidity pools is 50%.

Remediation plan:

SOLVED: The issue was fixed in commit 5d61e0b50145e9b8e55b72a5eaafc61b7bbbed78.

3.13 (HAL-13) DISTRIBUTION TIME FRAME CAN BE REDUCED UNRESTRICTEDLY - MEDIUM

Description:

The update_distribution function in **contracts/distributor/src/executions.rs** allows admin (Valkyrie team) to change without restrictions start_height and end_height fields. If those changes are made in a wrong or malicious manner, can produce the following consequences:

- Possibility to transfer an arbitrary amount of VKR tokens from **distributor** contract, see HAL-06 for more details.
- Time frame can be reduced to be less than 4 years (minimum distribution time defined in Valkyrie documentation), which impacts negatively VKR token supply and trust from Valkyrie users and community.
- Distribution can be updated with an amount less than released, see HAL-15 for more details.

Code Location:

Listing 23: contracts/distributor/src/executions.rs (Lines 132,137)

```
131  if let Some(start_height) = start_height {
132      distribution.start_height = start_height;
133      response = response.add_attribute("is_updated_start_height", "
             true");
134  }
135
136  if let Some(end_height) = end_height {
137      distribution.end_height = end_height;
138      response = response.add_attribute("is_updated_end_height", "
             true");
139  }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

Update the logic in `update_distribution` function to not allow the change of `start_height` field. Furthermore, add a validation routine in the aforementioned function to ensure that the difference between `end_height` and `start_height` fields is greater or equal than a predefined **minimum threshold**. The value of this threshold should be reflected in Valkyrie documentation, too.

Remediation plan:

SOLVED: The issue was fixed in commit `390a5e069c979b4cd74d3ded8a65f6281a232558`.

3.14 (HAL-14) NO MINIMUM THRESHOLD FOR EFFECTIVE DELAY PERIOD - LOW

Description:

Timelocks are defined in **Governance contracts** to allow protocol users to react timely if a change made is bad faith or is not in the best interest of protocol and its users.

The instantiate and update_poll_config functions from **contracts/governance/src/poll/executions.rs** do not restrict that timelock (execution_delay_period) is greater or equal than a **minimum threshold**. So, malicious changes proposed through voting could even be executed immediately if execution_delay_period is not set appropriately.

Code Location:

Listing 24: contracts/governance/src/poll/executions.rs (Lines 40)

```
36 let poll_config = PollConfig {
37     quorum: msg.quorum,
38     threshold: msg.threshold,
39     voting_period: msg.voting_period,
40     execution_delay_period: msg.execution_delay_period,
41     proposal_deposit: msg.proposal_deposit,
42     snapshot_period: msg.snapshot_period,
43 };
```

Listing 25: contracts/governance/src/poll/executions.rs (Lines 96)

```
95 if let Some(execution_delay_period) = execution_delay_period {
96     poll_config.execution_delay_period = execution_delay_period;
97     response = response.add_attribute("
           is_updated_execution_delay_period", "true");
98 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Add a validation routine inside instantiate and update_poll_config functions to ensure that timelock (execution_delay_period) is greater or equal than a **minimum threshold** that allows Valkyrie users to act timely against any issue that protocol could have when changes are made. The following are some examples of timelocks used on other protocols:

- Uniswap: 48-hours timelock
- Compound: 48-hours timelock
- Aave: 24-hours timelock (Short time lock)

Remediation plan:

SOLVED: The issue was fixed in commit 610784f128f9f7cfbbb30873339904605f213439.

3.15 (HAL-15) DISTRIBUTION CAN BE UPDATED WITH AN AMOUNT LESSER THAN RELEASED ONE - LOW

Description:

The update_distribution function in **contracts/distributor/src/executions.rs** allows admin (Valkyrie team) to change start_height and amount fields in such a way that distribution can be updated with an amount lesser than released one, which directly affects rewards distribution to **lp_staking** and **governance** contracts.

Attack scenario:

1. Malicious (or compromised) admin calls update_distribution function with start_height = <any_value_greater_than_current_block_height> and amount = <any_value_lesser_than_released_amount>.
2. As a consequence of **Step 1**, the aforementioned function will allow the new amount to be lesser than the released one.

Code Location:

Listing 26: contracts/distributor/src/executions.rs (Lines 132,142)

```
131  if let Some(start_height) = start_height {
132      distribution.start_height = start_height;
133      response = response.add_attribute("is_updated_start_height", "
             true");
134  }
135
136  if let Some(end_height) = end_height {
137      distribution.end_height = end_height;
138      response = response.add_attribute("is_updated_end_height", "
             true");
139  }
140
141  if let Some(amount) = amount {
```

```
142     if distribution.released_amount(env.block.height) > amount {
143         return Err(ContractError::Std(StdError::generic_err("
                amount must be less than released_amount")));
144     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Update the logic in update_distribution function to not allow the change of start_height field.

Remediation plan:

SOLVED: The issue was fixed in commit 9cc43540fbdf2d63b385cc7d4480e0e4329316ee. The Valkyrie team updated the logic of update_distribution function to not allow that distribution can be updated with an amount lesser than released one.

3.16 (HAL-16) RATES / PERCENTAGES COULD BE SET TO VALUES GREATER THAN 1 - LOW

Description:

The instantiate, update_config and update_referral_reward_limit_option functions from **contracts/campaign_manager/src/executions.rs** do not restrict that rates / percentages fields are greater than 1.

If they are not correctly set, some operations will always panic and won't allow legitimate users to deposit, withdraw or participate in campaigns; thus generating a denial of service (DoS) in Valkyrie protocol.

The affected fields are the following:

- deposit_fee_rate
- withdraw_fee_rate
- min_referral_reward_deposit_rate
- percent_for_governance_staking

Code Location:

Affected code in instantiate function:

Listing 27: contracts/campaign_manager/src/executions.rs (Lines 25,26)

```
20  Config {
21      governance: deps.api.addr_validate(msg.governance.as_str())?,
22      fund_manager: deps.api.addr_validate(msg.fund_manager.as_str()
            )?,
23      terraswap_router: deps.api.addr_validate(msg.terraswap_router.
            as_str())?,
24      code_id: msg.code_id,
25      deposit_fee_rate: msg.deposit_fee_rate,
26      withdraw_fee_rate: msg.withdraw_fee_rate,
```

Listing 28: contracts/campaign_manager/src/executions.rs (Lines 31)

```
30      referral_reward_token: deps.api.addr_validate(msg.
            referral_reward_token.as_str())?,
31      min_referral_reward_deposit_rate: msg.
            min_referral_reward_deposit_rate,
32  }.save(deps.storage)?;
```

Listing 29: contracts/campaign_manager/src/executions.rs (Lines 38)

```
34  ReferralRewardLimitOption {
35      overflow_amount_recipient: msg.referral_reward_limit_option.
            overflow_amount_recipient
36          .map(|r| deps.api.addr_validate(r.as_str()).unwrap()),
37      base_count: msg.referral_reward_limit_option.base_count,
38      percent_for_governance_staking: msg.
            referral_reward_limit_option.percent_for_governance_staking
            ,
39  }.save(deps.storage)?;
```

Affected code in update_config function:

Listing 30: contracts/campaign_manager/src/executions.rs (Lines 91,96)

```
90  if let Some(deposit_fee_rate) = deposit_fee_rate.as_ref() {
91      config.deposit_fee_rate = *deposit_fee_rate;
92      response = response.add_attribute("is_updated_deposit_fee_rate
            ", "true");
93  }
94
95  if let Some(withdraw_fee_rate) = withdraw_fee_rate.as_ref() {
96      config.withdraw_fee_rate = *withdraw_fee_rate;
```

Listing 31: contracts/campaign_manager/src/executions.rs (Lines 121)

```
120 if let Some(min_referral_reward_deposit_rate) =
        min_referral_reward_deposit_rate.as_ref() {
121     config.min_referral_reward_deposit_rate = *
            min_referral_reward_deposit_rate;
122     response = response.add_attribute("
            is_updated_min_referral_reward_deposit_rate", "true");
123 }
```

Affected code in update_referral_reward_limit_option function:

Listing 32: contracts/campaign_manager/src/executions.rs (Lines 160)

```
159  if let Some(percent_for_governance_staking) =
         percent_for_governance_staking.as_ref() {
160      limit_option.percent_for_governance_staking = *
             percent_for_governance_staking;
161      response = response.add_attribute("
             is_updated_percent_for_governance_staking", "true");
162  }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Add a validation routine inside instantiate, update_config and update_referral_reward_limit_option functions to ensure that **aforementioned fields** are lesser or equal than 1.

Remediation plan:

SOLVED: The issue was fixed in commit 7810214ad4d5e353350e4e5df396d5496693fb67.

3.17 (HAL-17) VOTERS CAN FORCE UNEXPECTED QUORUM WITHIN SNAPSHOT PERIOD - LOW

Description:

The snapshot_poll function in **contracts/governance/src/poll/executions.rs** is not restricted and allow voters to force unexpected quorum within snapshot_period in some edge scenarios. To better illustrate the issue, a comparison between a **regular scenario** and a **forcing quorum scenario** will be included, taking the following parameters:

- poll_config.quorum = 0.1
- Total votes = 180 VKR
- Staked amount = 2000 VKR
- Snapshot period is active

Regular scenario:

1. User calls stake_governance_token function to stake 20 VKR and then calls cast_vote function to vote with his 20 VKR in the poll.
2. Total votes now are 200 VKR. Internally, in Gov contract, snapshot_staked_amount function is triggered and because snapshot period is active, now snapped_staked_amount is 2020 VKR.
3. Nobody else votes and, when someone calls end_poll function, quorum is not reached **(200 VKR / 2020 VKR < 0.1)** and poll is rejected.

Forcing quorum scenario:

1. Before voting, user calls snapshot_poll function. Internally, in Gov contract, snapshot_staked_amount function is triggered and because snapshot period is active, now snapped_staked_amount is 2000 VKR.

2. User calls stake_governance_token function to stake 20 VKR and then calls cast_vote function to vote with his 20 VKR in the poll.
3. Total votes now are 200 VKR. Because snapped_staked_amount is already set in **Step 1**, it remains in 2000 VKR.
4. Nobody else votes and, when someone calls end_poll function, quorum is reached **(200 VKR / 2000 VKR >= 0.1)** in this scenario.

Code Location:

The snapshot_poll function can be called by anyone and triggers internally snapshot_staked_amount function:

Listing 33: contracts/governance/src/poll/executions.rs (Lines 412)

```
395 pub fn snapshot_poll(
396     deps: DepsMut,
397     env: Env,
398     _info: MessageInfo,
399     poll_id: u64,
400 ) -> ContractResult<Response> {
401     // Validate
402     let mut poll = Poll::load(deps.storage, &poll_id)?;
403
404     if poll.status != PollStatus::InProgress {
405         return Err(ContractError::Std(StdError::generic_err("Poll
                is not in progress")));
406     }
407
408     // Execute
409     let mut response = make_response("snapshot_poll");
410
411     let contract_available_balance = load_available_balance(deps.
            as_ref(), env.block.height)?;
412     let staked_amount = poll.snapshot_staked_amount(
413         deps.storage,
414         env.block.height,
415         contract_available_balance,
416     )?;
417     poll.save(deps.storage)?;
```

The snapshot_staked_amount function set snapped_staked_amount value only if snapshot has not occurred before:

Listing 34: contracts/governance/src/poll/states.rs (Lines 228)

```
224  if self.snapped_staked_amount.is_some() {
225          return Err(StdError::generic_err("Snapshot has already
                 occurred"));
226  }
227
228  self.snapped_staked_amount = Some(contract_available_balance);
229
230  Ok(contract_available_balance)
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

If not used, it is recommended to remove snapshot_poll function to avoid issues describe above. Another alternative could be to restrict its access, applying the principle of least privilege.

Remediation plan:

RISK ACCEPTED: The Valkyrie team accepted the risk for this finding.

3.18 (HAL-18) ANYONE CAN PARTICIPATE ON BEHALF OF OTHER USERS - LOW

Description:

The participate function from **contracts/campaign/src/executions.rs** allows a user to participate in a campaign if he has previously deposited collateral tokens in that campaign. However, anyone can participate on behalf of user specified in actor argument because the function does not validate that the sender is the user that participates.

Although the rewards for participation are sent to actor, it is important to apply the principle of least privilege in these cases to not affect users with unexpected operations.

Code Location:

Listing 35: contracts/campaign/src/executions.rs (Lines 720,724)

```
716 pub fn participate(
717     deps: DepsMut,
718     env: Env,
719     info: MessageInfo,
720     actor: String,
721     referrer: Option<Referrer>,
722 ) -> ContractResult<Response> {
723     // Validate
724     let actor = deps.api.addr_validate(&actor)?;
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Update the logic in `participate` function to not include `actor` as argument. Instead, the participation should be done on behalf of the sender.

Remediation plan:

SOLVED: The issue was fixed in commit `57e2d2f3a82ade965d774c3c31d06049dc2859bc`.

3.19 (HAL-19) DEPOSIT / PARTICIPATION PARAMETERS CAN BE UPDATED IN NOT PENDING CAMPAIGNS - INFORMATIONAL

Description:

The update_campaign_config and set_no_qualification functions in **contracts/campaign/src/executions.rs** allow the update of some deposit / participation parameters in campaigns that are in **not pending** status, i.e., last active height is different to **None**. This situation could create disadvantages for users that have already participated.

The affected parameters are the following:

- collateral_amount
- collateral_lock_period
- qualifier

Code Location:

Listing 36: contracts/campaign/src/executions.rs

```
169  if let Some(collateral_amount) = collateral_amount {
170      campaign_config.collateral_amount = collateral_amount;
171      response = response.add_attribute("
             is_updated_collateral_amount", "true");
172  }
```

Listing 37: contracts/campaign/src/executions.rs

```
174  if let Some(collateral_lock_period) = collateral_lock_period {
175      campaign_config.collateral_lock_period =
             collateral_lock_period;
176      response = response.add_attribute("
             is_updated_collateral_lock_period", "true");
```

Listing 38: contracts/campaign/src/executions.rs

```
179  if let Some(qualifier) = qualifier.as_ref() {
180      campaign_config.qualifier = Some(deps.api.addr_validate(
             qualifier)?);
181      response = response.add_attribute("is_updated_qualifier", "
             true");
182  }
```

Listing 39: contracts/campaign/src/executions.rs

```
260 campaign_config.qualifier = None;
261 response = response.add_attribute("is_updated_qualifier", "true");
262
263 campaign_config.save(deps.storage)?;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Update the logic of update_campaign_config and set_no_qualification functions to restrict the modification of parameters detailed above when campaigns are in **not pending** status.

Remediation plan:

SOLVED: The issue was fixed in commit ba7b525dac227e768da2bee30e3115244dc95778.

3.20 (HAL-20) PASSED POLLS DO NOT AUTOMATICALLY EXPIRE IF NOT EXECUTED - INFORMATIONAL

Description:

The governance contract does not define a **maximum threshold** to execute passed polls before make them expire, as defined in Valkyrie documentation. Moreover, the execute_poll function in **contracts/governance/src/poll/executions.rs** only validates that passed polls can not be executed before execution_delay_period, but do not include any logic to restrict the execution of expired polls.

Code Location:

Listing 40: contracts/governance/src/poll/executions.rs

```
297  if poll.status != PollStatus::Passed {
298          return Err(ContractError::Std(StdError::generic_err("Poll
                 is not in passed status")));
299  }
300
301  if poll.end_height + poll_config.execution_delay_period > env.
         block.height {
302      return Err(ContractError::Std(StdError::generic_err("Execution
             delay period has not expired")));
303  }
304
305  let mut executions = poll.executions;
306  if executions.is_empty() {
307      return Err(ContractError::Std(StdError::generic_err("The poll
             does not have executions")));
308  },
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Define a **maximum threshold** to execute passed polls before make them expire and include a routine in execute_poll function to validate if a passed poll has not expired before allow its execution.

Remediation plan:

ACKNOWLEDGED: The Valkyrie team acknowledged this finding.

3.21 (HAL-21) MISCALCULATION OF DEPOSIT VALUE WHEN VALIDATING REWARD POOL WEIGHT - INFORMATIONAL

Description:

The validate_reward_pool_weight function from **contracts/campaign/src/executions.rs** miscalculates deposit value because **referral_reward_amount** is used instead of **referral_reward_value** in the calculus of return value:

$$participation\_reward\_value + referral\_reward\_amount$$

At the end of deposit function, the miscalculated value is added as an attribute and emitted as part of a "wasm" event, which could affect the adequate traceability of the operation.

Code Location:

Affected code in deposit function:

Listing 41: contracts/campaign/src/executions.rs (Lines 320)

```
320 let (key_denom, referral_reward_pool_ratio, deposit_value) =
        validate_reward_pool_weight(
321     &deps.querier,
322     deps.api,
323     &campaign_config,
324     &reward_config,
325     participation_reward_amount,
326     referral_reward_amount,
327 )?;
```

Listing 42: contracts/campaign/src/executions.rs (Lines 334)

```
331 response = response.add_attribute("participation_reward_amount",
        participation_reward_amount.to_string());
```

```
response = response.add_attribute("key_denom", Denom::from_cw20(key_denom).to_string());
response = response.add_attribute("referral_reward_pool_ratio", referral_reward_pool_ratio.to_string());
response = response.add_attribute("deposit_value", deposit_value);
```

Affected code in validate_reward_pool_weight function:

Listing 43: contracts/campaign/src/executions.rs (Lines 450,470)

```
let referral_reward_value = swap_simulate(
    &querier,
    &global_campaign_config.terraswap_router,
    cw20::Denom::Cw20(reward_config.referral_reward_token.clone()),
    key_denom.clone(),
    referral_reward_amount,
)?;

let referral_reward_pool_rate = Decimal::from_ratio(
    referral_reward_value,
    participation_reward_value + referral_reward_value,
);

if referral_reward_pool_rate < global_campaign_config.min_referral_reward_deposit_rate {
    return Err(StdError::generic_err(format!(
        "Referral reward rate must be greater than {}",
        global_campaign_config.min_referral_reward_deposit_rate.to_string(),
        )));
}

Ok((key_denom, referral_reward_pool_rate, participation_reward_value + referral_reward_amount))
```

Risk Level:

Likelihood - 2

Impact - 1

Recommendation:

Use **referral_reward_value** instead of **referral_reward_amount** when deposit values is calculated in validate_reward_pool_weight function.

Remediation plan:

SOLVED: The issue was fixed in commit a1e310e5b92c58b9626b5e1e934d00ff88771dd9.

3.22 (HAL-22) MISCALCULATION OF REMAIN AMOUNT WHEN ADMIN TRANSFERS VKR TOKENS - INFORMATIONAL

Description:

The transfer function from **contracts/community/src/executions.rs** calculates the remain_amount when a sender transfer an amount of VKR tokens. If the sender is admin (Governance contract), the remain_amount is miscalculated because **amount** is not subtracted from **balance.free_balance**.

At the end of the function, the miscalculated value is added as an attribute and emitted as part of a "wasm" event, which could affect the adequate traceability of the operation.

Code Location:

Listing 44: contracts/community/src/executions.rs (Lines 165,172)

```
165 let remain_amount = if config.is_admin(&info.sender) {
166     let balance = state.load_balance(&deps.querier, &env, &config
        .managing_token)?;
167
168     if balance.free_balance < amount {
169         return Err(ContractError::Std(StdError::generic_err("
            Insufficient balance")));
170     }
171
172     balance.free_balance
173 } else {
```

Listing 45: contracts/community/src/executions.rs (Lines 204)

```
202 response = response.add_attribute("recipient", recipient);
203 response = response.add_attribute("amount", amount);
204 response = response.add_attribute("remain_amount", remain_amount);
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Update the calculus of remain_amount to subtract **amount** from **balance.free_balance** in transfer function when sender is admin.

Remediation plan:

SOLVED: The issue was fixed in commit d648cabc4629bef8e742140fd4f479df5341a3ef.

3.23 (HAL-23) FUNCTION WITH UNUSED ARGUMENT - INFORMATIONAL

Description:

The _height argument in load_available_balance function from **contracts/governance/src/common/states.rs** is not used as part of the internal logic of this function. So, its presence is not necessary for the correct working of the aforementioned function and only incurs in additional gas fees consumption.

Code Location:

Listing 46: contracts/governance/src/common/states.rs (Lines 31)

```
31 pub fn load_available_balance(deps: Deps, _height: u64) ->
       StdResult<Uint128> {
32     let contract_config = ContractConfig::load(deps.storage)?;
33     let contract_balance = query_cw20_balance(
34         &deps.querier,
35         &contract_config.governance_token,
36         &contract_config.address,
37     )?;
38     let poll_state = PollState::load(deps.storage)?;
39     let available_balance = contract_balance.checked_sub(
           poll_state.total_deposit)?;
40
41     Ok(available_balance)
42 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If not used, it is recommended to remove `_height` argument in `load_available_balance` function in order to reduce gas fees consumption.

Remediation plan:

ACKNOWLEDGED: The Valkyrie team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN