Y-Foundry DAO
Audit Report

Prepared for Y-Foundry DAO, 08th February 2023

Table of Contents

Introduction

SCV was engaged by Y-Foundry DAO to assist in identifying security threats and vulnerabilities that have the potential to affect their security posture. Additionally, SCV will assist the team in understanding the risks and identifying potential mitigations.

Scope

SCV performed the security assessment on the following codebase:

- yfd-governance:
    - https://github.com/Y-Foundry-Dao/yfd-governance
    - `8cac3bc6effcdef30984d78abbc36318d85ee0f5`
- yfd-claim:
    - https://github.com/Y-Foundry-Dao/yfd-claim
    - `a2d19513730f01137b824e8abf02c9d241f814ba`
- yfd-token:
    - https://github.com/Y-Foundry-Dao/yfd-token
    - `54678990bb1e1876cf467c093fae780eb9786fd0`
- yfd-integration (tests and deployment infrastructure)
    - `https://github.com/SCV-Security/yfd-integration`
    - `C301579f10f24606856fe67c13e0ce075c07202a`

Remediations were applied in several PRs up the the following reference:

- yfd-governance:
    - https://github.com/Y-Foundry-Dao/yfd-governance
    - *PR#32*
- yfd-claim:
    - https://github.com/Y-Foundry-Dao/yfd-claim
    - *PR#3*
- yfd-token:
    - https://github.com/Y-Foundry-Dao/yfd-token

- *PR#3*

- yfd-integration (tests and deployment infrastructure)
  - *https://github.com/SCV-Security/yfd-integration*
  - *PR#11*

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to Y-Foundry Dao. Testing includes, but is not limited to, the following:

- Understanding the application and its code base purpose;
- Deploying SCV in-house tooling to automate dependency analysis and static code review;
- Analyze each line of the code base and inspect application security perimeter;
- Review underlying infrastructure technologies and supply chain security posture;

Code Criteria and Test Coverage

This section below represents how *SUFFICIENT* or *NOT SUFFICIENT* each code criteria was during the assessment:

Criteria	Status	Notes
Provided Documentation	**SUFFICIENT**	The project did provide technical documentation.

Code Coverage Test	**SUFFICIENT**	• yfd-governance
		  ○ 39.01% coverage, 903/2315 lines covered
		• yfd-claim
		  ○ 63.78% coverage, 81/127 lines covered
		• yfd-token
		  ○ 93.85% coverage 366/390 lines covered
		
		The yfd-integration repository provided further integration testing making the code coverage sufficient.

Code Readability	**SUFFICIENT**	N/A
Code Complexity	**SUFFICIENT**	N/A

Vulnerabilities Summary

#	Summary Title	Risk Impact	Status
1	`lock_proposer_yfd` incorrectly determines proposer locked balance	**Critical**	**Resolved**
2	Any user can call `finalize_vote` and block forge from finalizing proposal	**Critical**	**Resolved**
3	Attacker can block all claims	**Critical**	**Resolved**
4	Unbounded entries in the codebase	**Critical**	**Partially Resolved**
5	`calc_fyfd` will panic when block height is higher than deposit height and lock duration	**Critical**	**Resolved**
6	`locked_fyfd` and `locked_yfd` incorrectly calculates unlocked stake ledger	**Critical**	**Resolved**
7	`ParameterChangeType::Activate` will not activate parameters	**Severe**	**Resolved**
8	`FUNDS_DISBURSED` is never saved to true	**Severe**	**Resolved**
9	Governance cannot modify `LockingBlockMultiplier` value	**Medium**	**Resolved**
10	`disbursement` lacking validations	**Medium**	**Resolved**
11	Permissionless `MintNft` message allows an attacker to specify custom metadata	**Medium**	**Resolved**
12	Native tokens cannot be whitelisted	**Medium**	**Resolved**
13	`LOCKED_STAKE_LEDGER` not updated during penalty	**Medium**	**Resolved**
14	Anyone can start proposals in the proposal contract	**Medium**	**Resolved**
15	Governance parameter's minimum and maximum values are not validated	**Low**	**Acknowledged**
16	`VaultRelease ProposalType` is not properly validated	**Low**	**Acknowledged**

	in `create_proposal`		
17	Proposal justification link is always empty	**Low**	**Resolved**
18	Users can still vote if an emergency vote is completed	**Low**	**Resolved**
19	`query_funds` will fail for fully funded proposals when the proposal state is `Affirm`	**Low**	**Resolved**
20	`query_votes` does not include `affirm_with_penalty` votes	**Low**	**Resolved**
21	Empty `address_whitelist` will block proposal creation	**Low**	**Resolved**
22	`AddressWhitelistChange::Change` does not remove old roles	**Low**	**Acknowledged**
23	Liquidated Vault contract specifies incorrect contract name	**Informational**	**Resolved**
24	Token contract has default name	**Informational**	**Resolved**
25	`do_transfer` should return a descriptive error message rather than an overflow error	**Informational**	**Resolved**
26	Proposal contract uses outdated code id values	**Informational**	**Acknowledged**
27	Max name limit can be bypassed with `AddressWhitelistChange::Change`	**Informational**	**Resolved**

Detailed Vulnerabilities

1 - `lock_proposer_yfd` incorrectly determines proposer locked balance

Risk Impact: Critical - **Status**: Resolved

Description

The `lock_proposer_yfd` function calculates the proposer address's amount of locked `YFD`. To do so, both the `YFD` and `FYFD` balances are determined. In `contracts/forge/src/contract.rs:593`, the `locked_fyfd` balance is incorrectly set to the amount of locked `YFD` rather than `FYFD`. Both the `locked_fyfd` and `locked_yfd` values are determined by the sum of the proposer's lock balances in `LOCKED_STAKE_LEDGER`. The error exists on line `598` where the `lock.locked_yfd` value is used rather than `lock.locked_fyfd`.

Recommendations

We recommend updating the value on line `contracts/forge/src/contract.rs:598` to `lock.locked_fyfd` rather than `lock.locked_yfd`.

2 – Any user can call `finalize_vote` and block forge from finalizing proposal

---

Risk Impact: Critical - **Status**: Resolved

Description

The `finalize_vote` function in `contracts/cw20-vote/src/contract.rs:118` does not restrict the caller, so any address may call this function. This issue is present in vote contracts that have been instantiated by the forge. This is problematic because the forge's `FinalizeProposal` functionality attempts to finalize the vote, but if the vote has been finalized already it will return an error and effectively block the forge's ability to finalize the proposal.

Recommendations

We recommend only allowing the caller of the `FinalizeVote` message to be the `INSTANCE_OWNER`.

3 – Attacker can block all claims

Risk Impact: Critical - **Status**: Resolved

Description

The `receive_cw20` function in `claim:src/contract.rs:60-75` does not check to ensure that only expected CW20s are being sent. This will allow an attacker to create a malicious CW20 that can block claims. For claims, the `transfer_msg` function is used to handle the transfer, so the funds get sent to the NFT owner.

This is problematic because anyone can send malicious CW20 tokens to the contract (see line `72`), which will cause the specific proposal id to have reimbursement from the attacker's CW20. The attacker could craft a CW20 where the `Cw20ExecuteMsg::Transfer` message returns an error. When the transfer is invoked from the claim contract to this CW20, it will cause the transaction to revert. As a result, when the user wants to claim shares with their NFT, an error will occur in line `167`. This blocks all valid reimbursements, including CW20 and native assets.

In addition, if a small amount of CW20 tokens are sent, there is a potential scenario that some stakers will receive 0 funds due to rounding issues. As default CW20 token transfers disallow 0 amount transfers, the claim functionality will be blocked for the user.

Recommendations

We recommend creating a whitelist for the CW20s that are allowed for the claims contract. Additionally, consider only allowing privilege addresses to reimburse to prevent unintended consequences.

4 − Unbounded entries in the codebase

Risk Impact: Critical - **Status**: Partially Resolved

Description

Some instances inside the codebase attempt to iterate over all the vectors or keys stored in the storage. An out-of-gas error will occur if too many keys need to be processed. This may impact the specific functionality of the protocol. This also creates the opportunity for attackers to take advantage of these unbounded iterations to negatively impact the functionality of the protocol.
Some of the unbounded iterations we identified:

1. The vote contract relies on all staker's staked balance as seen in `contracts/forge/src/contract.rs:531`. The initial balances are fetched from `query_all_stakes` which is based on `USER_STAKE_LEDGER`. The `USER_STAKE_LEDGER` is set once a new user stakes YFD.

   An attacker can speed up this by creating many accounts and staking YFD tokens. Eventually, this would be an issue if many people stake YFD tokens. As a result, the contract instantiation would fail because `initial_balances` need to be iterated when creating accounts to prevent dupe (see `contracts/cw20-vote/src/contract.rs:40, 71`)

2. In `contracts/forge/src/contract.rs:863`, the code attempts to perform an unbounded loop in `USER_STAKE_LEDGER`. If a user has staked YFD tokens many times, the `unlock_proposer_yfd` function will fail due to being out of gas, preventing a user from unlocking their YFD tokens.

3. In `contracts/forge/src/contract.rs:1112`, the `execute_claim` function will fail if the user staked too many assets. This is because `USER_STAKE_LEDGER` will become too large to process, eventually causing an out-of-gas failure

4. In `contracts/forge/src/contract.rs:1331`, `validate_active_proposal_limits` will fail if the user has created many proposals in the past. This is because `LOCKED_STAKE_LEDGER` will grow too large, eventually causing out-of-gas failure when trying to process it.

5. In `contracts/forge/src/contract.rs:1447`, `query_all_accounts` attempts to loop through all users that staked YFD tokens in the contract. Consequently, the function will fail if too many users staked YFD tokens.

6. In `contracts/forge/src/vault_proposal.rs:35-42`, the `validate_vault_proposal` function performs an unbounded loop for all created proposals. All tickers from `VAULT_PROPOSALS` are fetched as an unbounded loop to ensure the proposed ticker is never stored in any vaults. This will become an issue if more vault proposals are created, eventually causing an out-of-gas failure.

7. In `src/contract.rs:185` of the `yfd-claim` repository, all keys stored inside `DISBURSEMENTS` would get iterated infinitely without a limit. If there are too many keys to process, the function will run out of gas.

   An attacker can keep creating many disbursements with a small number of funds to cause the function to fail out of gas. For example, the attacker can call `receive_native` or `Cw20HookMsg::Disbursement` many times to cause the `DISBURSEMENTS` state filled with many keys, causing a potential out-of-gas failure. This will cause users to be unable to execute the claim function and query all claims via `query_pending_claims` query.

Additionally, the below iterations are also processed without limits. However, it does not pose a direct risk as the values are supplied by privileged roles or governance.

- `contracts/cw721-fee-distribution/src/execute.rs:55-70`
- `contracts/cw721-fee-distribution/src/query.rs:17-32`
- `contracts/forge/src/contract.rs:1230-1241`

This is especially important for functionality where there are multiple unbounded iterations invoked during the context of an entire transaction as gas is cumulatively consumed. While some of the items mentioned above in isolation may be a lower concern, the gas consumption is compounded for transactions that invoke many iterations.

Recommendations

Consider applying a pagination mechanism to all unbounded iterations above or modifying the implementation to iterate by `O(1)`.

Revision Notes

The client disagreed with the recommendation of pagination for remediation as proposed by SCV. Pagination would solve the issue of the unbounded lock/deposit limit reaching a transaction's absolute gas ceiling as defined by the chain.

The client also mentioned that tests and simulations of staking and proposal submissions were run on LocalTerra, showing that based on their calculations, a single instance of a forge contract can provide sufficient capacity within the terra gas limit to allow for at least 15,000 active concurrent `FYFD` lock positions active at any one time.

Going forward, governance solutions will evolve, and they look to treat the current design of governance as a proof of concept that fully on-chain governance is possible. Additionally, the DAO may look to modify the governance token lock and claim contract designs to accommodate a larger number of participants or by placing upper limits on participation per instance of governance contract.

The following remediations were implemented:

1. `FYFD` locking records are purged from the contract when YFD is reclaimed by the depositor.
2. Increase the `YFD` minimum deposit to greater than 10,000 `YFD` at launch with a consideration to raise the minimum `YFD` locking requirement to greater than or equal to 125,000 `YFD`.

5 – `calc_fyfd` will panic when block height is higher than deposit height and lock duration

Risk Impact: Critical - **Status**: Resolved

Description

In `contracts/forge/src/fyfd_math.rs:17`, the `calc_fyfd` function does not verify that the current height is not too large. Consequently, the `staked_blocks` will become a very large number which eventually causes `decayed_fyfd` to be larger than `max_fyfd`, causing an underflow during subtraction.

This affects `lock_proposer_yfd`, `query_token_info`, and `query_balance_detail` functions. Note that all three functions attempt to process all `USER_STAKE_LEDGER` owned by a user, which means if any of the staked deposit height and lock duration is lower than the block height, all 3 functions will fail permanently.

We consider this a critical issue because this will happen in the future time when the block height is high enough. The result of this will be that the `calc_fyfd` will fail because `overflow-checks` are enabled in the release profile.

Here's a test case demonstrating the issue:

```
#[test]
fn overflow_poc() {
    let stake = Stake {
        depositor: Addr::unchecked("depositor"),
        asset_deposit_amount: Uint128::from(100_000_100u64),
        asset_withdrawn_amount: Uint128::zero(),
        deposit_timestamp: Default::default(),
        deposit_height: 42,
        lock_duration: 1_000_000,
    };

    // calc_fyfd would overflow when block height is larger than
deposit height and lock duration
    let overflow_expiration = stake.deposit_height +
stake.lock_duration + 1_u64;
```

```
    let fyfd = calc_fyfd(
        overflow_expiration,
        stake,
        Decimal::from_str("0.0000002").unwrap()
    ).unwrap();

    println!("{:?}", fyfd);
}
```

Recommendations

We recommend adding a `min` check to prevent overflow issues:

```
pub fn calc_fyfd(
    cur_height: u64,
    stake: Stake,
    locking_block_multiplier: Decimal,
) -> StdResult<Uint128> {
    let staked_blocks = Uint128::from(cur_height -
stake.deposit_height).to_decimal();
    let staking_term =
Uint128::from(stake.lock_duration).to_decimal();
    let max_fyfd = locking_block_multiplier
        * stake.asset_deposit_amount.to_decimal()
        * Decimal::from_ratio(stake.lock_duration, 1u8);
    let decayed_fyfd = (staked_blocks / staking_term) * max_fyfd;
    let decayed_fyfd = min(max_fyfd, decayed_fyfd);
    Ok((max_fyfd - decayed_fyfd) * Uint128::from(1u8))
}
```

6 - `locked_fyfd` and `locked_yfd` incorrectly calculates unlocked stake ledger

Risk Impact: Critical - **Status**: Resolved

Description

In `contracts/forge/src/contract.rs:597`, `contracts/forge/src/contract.rs:605`, and `contracts/forge/src/contract.rs:1131`, all three functions attempt to get the total locked YFD or fYFD amount by using the `lock.resolved_block != 0` filter. All resolved blocks that are not 0 are fetched to increase the total `locked_fyfd` and `locked_yfd` amount.

This function is incorrect because rather than fetching the locked balances (which are denoted by a `resolved_block` of 0), the filter is actually resolving the unlocked balances. In `contracts/forge/src/contract.rs:884-894`, unlocked stake ledgers will set the locked YFD and fYFD amounts to 0 while the resolved block will be set to the current block height.

As a result, the first two functions cause an incorrect calculation of `yfd_to_lock`, which also causes incorrect `locked_yfd` and `locked_fyfd` amounts calculated in `contracts/forge/src/contract.rs:616`. The last function would allow a proposer to claim the locked YFD tokens even though the associated yFYD amount is locked, as seen in `contracts/forge/src/contract.rs:1135`. Additionally, this attack can be leveraged by an attacker to withdraw YFD tokens and stake fYFD repeatedly to inflate their voting power

Recommendations

We recommend changing the function to filter with "`lock.resolved_block == 0`" so it fetches locked YFD and fYFD amounts.

7 - `ParameterChangeType::Activate` will not activate parameters

Risk Impact: Severe - **Status**: Resolved

Description

In `contracts/forge/src/governance_parameters.rs:643`, the `data.active` is set to `false` even if `ParameterChangeType` enum is `Activate`. This means all parameters cannot be activated even if governance tries to activate them, potentially affecting day-to-day operations because governance parameters are required to create a governance proposal. Consequently, a deactivated parameter would cause errors in `contracts/forge/src/governance_parameters.rs:473`, `502`, `531`, and `560`

Recommendations

We recommend setting `data.active` to `true`.

8 - FUNDS_DISBURSED is never saved to true

Risk Impact: Severe - **Status**: Resolved

Description

In `contracts/proposal/src/contract.rs:721` and `773`, the code would only get executed when `FUNDS_DISBURSED` is saved to true. However, it is only recorded as true in test cases, not the contract itself, as seen in `contracts/proposal/src/tests/contract.rs:577` and `contracts/proposal/src/tests/contract.rs:753`.

As a result, the full penalty will still be applied to the proposer even if the funds are disbursed, decreasing the total stake balance. The boosters will not get a full refund if there are insufficient funds.

Recommendations

We recommend setting `FUNDS_DISBURSED` to `true` when the funds are disbursed.

9 – Governance cannot modify `LockingBlockMultiplier` value

Risk Impact: Medium - **Status**: Resolved

Description

In `contracts/forge/src/governance_parameters.rs:89-91`, the max value is set to `0.000000000000000001` while the minimum value is set to `0.00005`. This is incorrect as `0.00005` is larger than `0.000000000000000001`. Ideally, the max value should always be greater than the min value. Other than that, the default value is `0.000000237823439878`, which is already higher than the max value.

As a result, governance cannot modify the `LockingBlockMultiplier` value due to a logic error occurring in `contracts/forge/src/governance_parameters.rs:825-829`.

Recommendations

We recommend switching the minimum and maximum values, ie. set the min value to `0.000000000000000001` while the max to `0.00005`.

10 – `disbursement` lacking validations

Risk Impact: Medium - **Status**: Resolved

Description

The `disbursement` function in `Claim:src/contract.rs:104` does not properly validate the user supplied `proposal_id` or asset.

Funds could be sent in a disbursement to an invalid proposal and they would be locked in the contract until a proposal with that id is created

Recommendations

We recommend validating that the `proposal_id` supplied is valid.

11 – Permissionless `MintNft` message allows an attacker to specify custom metadata

Risk Impact: Medium - **Status**: Resolved

Description

Any address may call the proposal contract's `MintNft` entrypoint. In `contracts/proposal/src/contract.rs:656`, the NFT metadata would be saved with the caller-specified `NftMetadata` struct, which contains `strategist_uri`, `booster_uri`, and `treasury_uri`. This means any caller can choose which metadata to init the NFT with, which can only be set once, preventing the admin from creating the NFT. An attacker could take advantage of this functionality to set this data before legitimate protocol users and take advantage of the trust relationship of the protocol to potentially exploit users by including malicious links.

Recommendations

We recommend checking the caller to be a privileged address to prevent unintended NFT metadata minted.

12 – Native tokens cannot be whitelisted

Risk Impact: Medium - **Status**: Resolved

Description

In the `token_whitelist_set` function in `contracts/forge/src/contract.rs:1025`, the denom variable is used to perform a `TokenInfo` query, which would only succeed if the token is CW20 token address. However, `TokenWhitelistChange::New` allows whitelisting native tokens, as seen in `line 1007`. As a result, attempting to whitelist native tokens will fail.

Recommendations

We recommend only performing the `TokenInfo` query if the token is a CW20 token. If the token is native, consider setting the decimal values to 6 directly.

13 – LOCKED_STAKE_LEDGER not updated during penalty

Risk Impact: Medium - **Status**: Resolved

Description

In `contracts/forge/src/contract.rs:838-897` of the `unlock_proposer_yfd` function, if the proposal's vote state result is `VoteState::DenyWithPenalty`, the locked YFD amount from the proposer is forced withdrawn to the treasury as seen in `line 872`. This acts as a punishment for the proposer.

However, the `LOCKED_STAKE_LEDGER` resolved block is not set to the current height, as seen in line 891. The `resolved_block` will stay as 0, which means the YFD is still locked, even when the proposer's YFD is already penalized.

Consequently, the proposer's `active_proposals` will be decreased permanently. As seen in `contracts/forge/src/contract.rs:1341`, the `LOCKED_STAKE_LEDGER` will get fetched as `active_proposals`, which will be checked in line 1345.

Recommendations

We recommend unlocking the penalizer's `LOCKED_STAKE_LEDGER` during `unlock_proposer_yfd` to prevent the penalized proposal from being counted as an active proposal.

14 – Anyone can start proposals in the proposal contract

Risk Impact: Medium - **Status**: Resolved

Description

In `contracts/proposal/src/contract.rs:301` and `302`, the `PayDeveloper` and `ReleaseVault` message allows permissionless execution. Ideally, both messages should only be executed by the developer themselves.

If not, anyone can “fake” the developer and try to pay the developer money or release a malicious vault with a malicious code id. Additionally, attackers can keep creating proposals to prevent real proposals that the developer needs to create from passing through.

We consider this a medium-severity issue because it needs to pass through governance.

Recommendations

We recommend only allowing the developer to call it.

15 – Governance parameter's minimum and maximum values are not validated

Risk Impact: Low - **Status**: Acknowledged

Description

In `contracts/forge/src/governance_parameters.rs:620`, the `ProposalType::Parameter` allows for governance to make changes to the values of the governance parameters. While the internal minimum and maximum validations are enforced for each individual parameter, there are no logic validations with the associated min/max values between different parameters.

For example, when changing the value of `MinLockTime`, the final value should be validated to be lower or equal to the value of `MaxLockTime`. Without this validation, an error would occur in `contracts/forge/src/contract.rs:1072`.

Recommendations

We recommended adding validation checks to parameters that require this logical validation. For example, the value of `MinLockTime` should be validated to be lower than the value of `MaxLockTime` that is set and vice versa. The same applies to the values of `VaultDeveloperMin` and `VaultDeveloperMax`, as well as `VaultBoosterDepositMin` and `VaultBoosterDepositMax`.

16 – `VaultRelease ProposalType` is not properly validated in `create_proposal`

Risk Impact: Low - **Status**: Acknowledged

Description

The `create_proposal` function in `contracts/forge/src/contract.rs:351` does not properly validate the `VaultRelease ProposalType`. Specifically, the `instantiate_msg` should be validated to ensure that the proposal can successfully be finalized.

Recommendations

We recommend validating the `ProposalType::VaultRelease` type before a proposal is created in `create_proposal`.

17 – Proposal justification link is always empty

Risk Impact: Low - **Status**: Resolved

Description

In `contracts/forge/src/contract.rs:551`, when caching the data for a reply, the `justification_link` is set as an empty string. This is problematic because the users that provided a valid input for the `msg.justification_link` parameter will be overridden and shown empty despite having submitted a valid input.

Recommendations

We recommend keeping the `msg.justification_link` parameter in `line 551` as the user's input, instead of setting it to an empty string.

18 – Users can still vote if an emergency vote is completed

Risk Impact: Low - **Status**: Resolved

Description

The `finalize_vote` function in `contracts/cw20-vote/src/contract.rs:375` finalizes an emergency vote and allows it to be executed before the expiration as long as the quorum is met. Currently, all voting messages only check if the vote is canceled or expired. This means that users can still cast their votes even if the emergency vote is finished, ie. `VoteState` is mutated to `Affirm` and others.

As a result, this could cause misleading voting results to occur. Suppose many users vote `Deny` after the emergency vote ends, users will see the voting state as `Affirm` but the total number of denied votes will be higher, causing confusion for users.

Recommendations

We recommend ensuring all voting executions check that the voting state is `VoteState::InProgress`.

19 – `query_funds` will fail for fully funded proposals when the proposal state is `Affirm`

Risk Impact: Low - **Status**: Resolved

Description

In `contracts/proposal/src/contract.rs:1340`, `proposal.num_payments` is being divided in the `distributable_stake` function. If the `num_payments` value is 0, a division by zero error will occur.

The `num_payments` value will become 0 if the vault proposer decides to fully fund their project as seen in `contracts/forge/src/vault_proposal.rs:155`. This means that any function that tries to call `distributable_stake` for fully funded projects would fail due to a division by 0 error.

As a result, the `query_funds` function in `contracts/proposal/src/contract.rs:1376` calls the `distributable_stake` function if the current proposal state is `Affirm`. This means that `query_funds` will always fail when the proposal state is Affirm.

Recommendations

We recommend only handling the division by zero error gracefully.

20 – `query_votes` does not include `affirm_with_penalty` votes

Risk Impact: Low - **Status**: Resolved

Description

The `query_votes` function in `contracts/cw20-vote/src/contract.rs:496` does not show `affirm_with_penalty` in the `VotesResponse` struct.

Recommendations

We recommend adding `affirm_with_penalty` to the `VotesResponse` struct.

21 – Empty `address_whitelist` will block proposal creation

Risk Impact: Low - **Status**: Resolved

Description

In `contracts/forge/src/contract.rs:208`, the `create_vault_proposal` and `create_proposal` functions require the sender to be the Proposer or Strategist role, and their address must be saved into `ADDRESS_WHITELIST`. If `msg.address_whitelist` is empty or no users contain the above roles, the forge contract cannot be executed.

We consider this a low-severity issue because it can only be caused by the forge contract admin.

Recommendations

We recommend checking `msg.address_whitelist` contains both `Proposer` and `Strategist` roles.

22 – `AddressWhitelistChange::Change` does not remove old roles

Risk Impact: Low - **Status**: Acknowledged

Description

In `contracts/forge/src/contract.rs:978`, the roles are appended instead of being overwritten. This means that if `AddressWhitelistChange::Change` is called with a new set of roles, the roles are appended and not removed. Due to the append function, the final roles would be both `Strategist` and `Booster`.

As a result, this forces governance to call `AddressWhitelistChange::Remove` and `AddressWhitelistChange::New` so that the changing of roles will work correctly.

Recommendations

We recommend overwriting the roles instead of appending them in `line 978`.

23 – Liquidated Vault contract specifies incorrect contract name

Risk Impact: Informational - **Status**: Resolved

Description

The `CONTRACT_NAME` in the liquidated vault contract is incorrectly set to `"stub-vault"`. This may be misleading as that name refers to another governance contract.

Recommendations

We recommend updating the `CONTRACT_NAME` in `contracts/liquidated_vault/src/contract.rs:12`.

24 – Token contract has default name

Risk Impact: Informational - **Status**: Resolved

Description

In `token:src/contract.rs:23` the name is specified as `"crates.io:cw20-base"` this should be updated to a more descriptive name of the contract.

Recommendations

We recommend updating the contract's name.

25 – `do_transfer` should return a descriptive error message rather than an overflow error

Risk Impact: Informational - **Status**: Resolved

Description

The `do_transfer` function in `contracts/cw20-vote/src/contract.rs:123` transfers a set amount from the sender's balance to the recipient. This function is invoked when the voting execute messages are called. The checked subtraction on `line 133` will return an `OverflowError` when a sender votes with an amount greater than their balance. Rather than returning a generic `OverflowError`, it would be best to return a descriptive error noting that the sender attempted to transfer a balance greater than what they had.

Recommendations

We recommend returning a descriptive error that notes that the sender does not have the appropriate balance to vote or transfer the amount specified.

26 – Proposal contract uses outdated code id values

Risk Impact: Informational - **Status**: Acknowledged

Description

All governance values are retrieved from `get_governance_values` and saved into the `GOVERNANCE_VALUES` storage state. This means that if there were a governance parameter change proposal that modifies the values, the related parameter in the proposal contract would not get updated. Parameters such as `VoteCodeId` and `NftCodeId` should follow the latest governance values.

Suppose a vulnerability is discovered in the code id and all contracts are migrated. The proposal contract still uses the old code id, once the contract is uploaded, the attacker can exploit it.

Recommendations

We recommend receiving dynamic governance values for `VoteCodeId` and `NftCodeId`.

Revision Notes

This was the design intent to use governance as it was at the start of the vault proposal.

27 – Max name limit can be bypassed with `AddressWhitelistChange::Change`

Risk Impact: Informational - **Status**: Resolved

Description

In `contracts/forge/src/contract.rs:976` of the `address_whitelist_set` function, the `name` input parameter should be validated to be below a 40 letter max character limit.

Recommendations

We recommend adding validations similar to the one in `contracts/forge/src/contract.rs:207` and `209`.

Document control

Version	Date	Approved by	Changes
0.1	12th December 2022	Vinicius Marino	Document Pre-Release
0.2	07th February 2023	SCV-Team	Revisions
1.0	08th February 2023	Vinicius Marino	Document Release

Appendices

A. Appendix – Risk assessment methodology

A qualitative risk assessment is performed on each vulnerability to determine the impact and likelihood of each.

Risk rate will be calculated on a scale. As per criteria Likelihood vs Impact table below:

	Rare	Unlikely	Possible	Likely
Critical	Medium	Severe	Critical	Critical
Severe	Low	Medium	Severe	Severe
Moderate	Low	Medium	Medium	Severe
Low	Low	Low	Low	Medium
Informational	Informational	Informational	Informational	Informational

LIKELIHOOD

- Likely: likely a security incident will occur;
- Possible: It is possible a security incident can occur;
- Unlikely: Low probability a security incident will occur;
- Rare: In rare situations, a security incident can occur;

IMPACT

- Critical: May cause a significant and critical impact;
- Severe: May cause a severe impact;
- Moderate: May cause a moderated impact;
- Low: May cause low or none impact;
- Informational: May cause very low impact or none.

B. Appendix – Report Disclaimer

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts SCV-Security to perform a security review. The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The content of this audit report is provided “as is”, without representations and warranties of any kind, and SCV-Security disclaims any liability for damage arising out of, or in connection with, this audit report.