F Y E O

Security Code Review of Cosmo Burn

CosmoBurn

September 2024
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

CosmoBurn engaged FYEO Inc. to perform a Security Code Review of Cosmo Burn.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 13 - August 22, 2024, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-CosmoBurn-01 – Deadlock
- FYEO-CosmoBurn-02 – Distributor contract may rely on bad data
- FYEO-CosmoBurn-03 – IBC contract does not check funds sent into execute_send
- FYEO-CosmoBurn-04 – Backend trust required for critical logic
- FYEO-CosmoBurn-05 – Bounds / Input validation
- FYEO-CosmoBurn-06 – Loop uses check_tx_frequency, prevents iterating
- FYEO-CosmoBurn-07 – Admin can withdraw all funds
- FYEO-CosmoBurn-08 – Code clarity / optimization
- FYEO-CosmoBurn-09 – Data is silently overwritten or may not be deleted
- FYEO-CosmoBurn-10 – IBC contract appears unfinished

- FYEO-CosmoBurn-11 – Potential overflow
- FYEO-CosmoBurn-12 – Storage never cleared

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of Cosmo Burn. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/CosmoBurnOrg/cosmoburn-audit with the commit hash a515193e41a692e4846362f7b7364e8e86d0a188.

Remediations were committed with hash d93d85cf68754bcfcd223ac90605c4e6a0a33942.

Files included in the code review

```
cosmoburn/
└── contracts/
    ├── cosmoburn/
    │   └── src/
    │       ├── bin/
    │       │   └── schema.rs
    │       ├── burn.rs
    │       ├── contract.rs
    │       ├── error.rs
    │       ├── execute.rs
    │       ├── functions.rs
    │       ├── helpers.rs
    │       ├── lib.rs
    │       ├── msg.rs
    │       ├── query.rs
    │       └── state.rs
    ├── cosmoburn_funds_distributor/
    │   └── src/
    │       ├── bin/
    │       │   └── schema.rs
    │       ├── contract.rs
    │       ├── error.rs
    │       ├── execute.rs
    │       ├── functions.rs
    │       ├── helpers.rs
```

Files included in the code review

```
    │       ├── lib.rs
    │       ├── msg.rs
    │       ├── query.rs
    │       └── state.rs
    ├── cosmoburn_ibc_transfer/
    │   └── src/
    │       ├── bin/
    │       │   └── ibc_transfer_schema.rs
    │       ├── contract.rs
    │       ├── lib.rs
    │       ├── msg.rs
    │       └── state.rs
    ├── cosmoburn_tracker/
    │   └── src/
    │       ├── contract.rs
    │       ├── error.rs
    │       ├── functions.rs
    │       ├── helpers.rs
    │       ├── lib.rs
    │       ├── msg.rs
    │       ├── query.rs
    │       └── state.rs
    └── hostburner/
        └── src/
            ├── bin/
            │   └── schema.rs
            ├── contract.rs
            ├── error.rs
            ├── functions.rs
            ├── helpers.rs
            ├── lib.rs
            ├── msg.rs
            ├── query.rs
            └── state.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Cosmo Burn, we discovered:

- 3 findings with HIGH severity rating.
- 1 finding with MEDIUM severity rating.
- 2 findings with LOW severity rating.
- 6 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

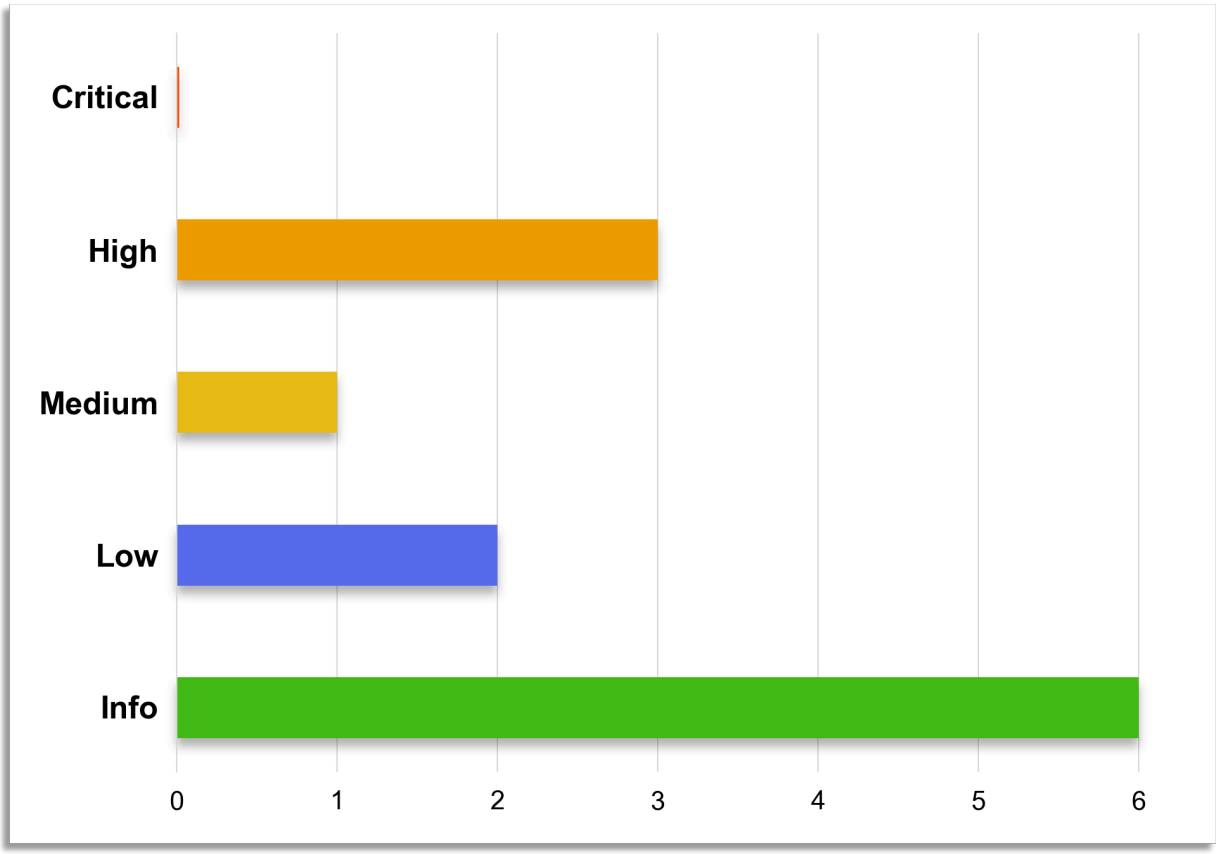

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-CosmoBurn-01	**High**	Deadlock
FYEO-CosmoBurn-02	**High**	Distributor contract may rely on bad data
FYEO-CosmoBurn-03	**High**	IBC contract does not check funds sent into execute_send
FYEO-CosmoBurn-04	**Medium**	Backend trust required for critical logic
FYEO-CosmoBurn-05	**Low**	Bounds / Input validation
FYEO-CosmoBurn-06	**Low**	Loop uses check_tx_frequency, prevents iterating
FYEO-CosmoBurn-07	**Informational**	Admin can withdraw all funds
FYEO-CosmoBurn-08	**Informational**	Code clarity / optimization
FYEO-CosmoBurn-09	**Informational**	Data is silently overwritten or may not be deleted
FYEO-CosmoBurn-10	**Informational**	IBC contract appears unfinished
FYEO-CosmoBurn-11	**Informational**	Potential overflow
FYEO-CosmoBurn-12	**Informational**	Storage never cleared

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

Cosmo Burn is a Cosmos based application that allows users to burn a selection of whitelisted tokens and receive a reward token in return. It is built with multiple CosmWasm contracts, including a burn contract, distributor contract, IBC transfer contract that leverages Neutron's cross-chain capabilities, tracker contract, and hostburner contract, the system operates on a principle where 80% of the sent tokens are permanently removed from circulation, while 20% are distributed as rewards to the token holders of the app's token. Cross-chain functionality is facilitated through an IBC transfer contract, while a backend system manages whitelisting and reward calculations. With a user-friendly, one-page application interface, the system aims to incentivize token burning by rewarding participants with ANEW tokens.

Deadlock

Finding ID: FYEO-CosmoBurn-01
Severity: **High**
Status: **Remediated**

Description

The funds distributor contract has a lock function which can be called once. It sets some data that is later required to distribute funds. For the `DistribOwnCoin` distribution type it is expected that `exch_rates` are set for each of the `whitelisted_coins`. If these are not set, the function can never execute.

Proof of Issue

File name: contracts/cosmoburn_funds_distributor/src/contract.rs
Line number: 129

```
pub fn try_lock_distrib(
    ...

    if state.lock.state == LockState::LOCKED {
        return Err(ContractError::DistributionInProgress {});
    }

    ...

    STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
        state.lock.state = LockState::LOCKED;
        ...
        state.lock.exch_rates = exch_rates;

    ...
pub fn try_step_distrib(
    deps: DepsMut,
    info: MessageInfo,
    env: Env,
    limit: Option<u32>,
) -> Result<Response, ContractError> {
    ...

    if state.lock.state != LockState::LOCKED {
        return Err(ContractError::DistributionNotLocked {});
    }

    ...

    } else if config.distrib_strategy == DistributionStrategy::DistribOwnCoin {
        let exch_rates = state.lock.exch_rates.clone().expect("exch rates not set on
lock");
        let exch_rate = exch_rates.get(coin_address).unwrap();
```

Severity and Impact Summary

User funds may be stuck indefinitely. There is however an admin function to withdraw funds and solve such a situation manually.

Recommendation

Make sure to require that an exchange rate is provided for every whitelisted coin if the distribution mode requires it.

Distributor contract may rely on bad data

Finding ID: FYEO-CosmoBurn-02
Severity: **High**
Status: **Remediated**

Description

The distributor contract uses another contract `tracker_addr` to track balances for its holders. The current user balance is used to calculate shares the user should be awarded. However, no prerequisites are checked for this token such as total supply at clearing of the distributor and any existing holders. The distribution may require several transactions to complete while tokens may be moved from one user to another in the meantime which would affect calculations as the current user balance is requested.

Proof of Issue

File name: contracts/cosmoburn_funds_distributor/src/query.rs
Line number: 70

```
let holders_response: ListHoldersResponse = deps
    .querier
    .query_wasm_smart(
        config.tracker_addr.clone(),
        &TrackerQueryMsg::GetHolders {
            from: from,
            limit: Some(limit.unwrap_or(100000)), //TODO: get holders count and
paginate
        },
    )
    .unwrap();
```

There is no specific time associated with this query.

```
let holder_balance_u128: u128 = holder_balance.into();
let coin_balance_u128: u128 = coin_balance.into();
let exch_rate_u128: u128 = (*exch_rate).into();

let supply_u128: u128 = supply.into();

// exch_rate corresponds to tracker_denom value in coin_addr in nanos
// ex: 1 tracker_denom = 0.5 coin_addr, then exch_rate = 500000
let coin_distrib_amount = (holder_balance_u128 * coin_balance_u128 * exch_rate_u128) /
(supply_u128 * 1000000);
```

Calculations will depend on the supply and an individual's balance.

```
let config = Config {
    owner: info.sender.clone(),
    admin_addr: deps.api.addr_validate(&msg.admin_addr)?.to_string(),
    tracker_addr: deps.api.addr_validate(&msg.tracker_addr)?.to_string(),
    tracker_denom: msg.tracker_denom.to_string(),
    distrib_strategy: DistributionStrategy::from_str(&msg.distrib_strategy).unwrap(),
    treasury_addr: msg.treasury_addr.to_string(),
};
```

When instantiating a distributor, it is not checked whether there are any token holders of the tracked token or if the supply is 0 or not.

```
state.lock.from = last_holder;
```

This is used as an offset to continue iterating (distributing rewards) in another transaction. Token balances could change in the meantime.

Severity and Impact Summary

When distribution happens in multiple transactions, it is possible that tokens are moved by users. This would allow then to gain more rewards than they should and would cause accounting issues.

Recommendation

Make sure to carefully craft this accounting logic in a way that does not let users manipulate the rewards.

IBC contract does not check funds sent into execute_send

Finding ID: FYEO-CosmoBurn-03
Severity: **High**
Status: **Remediated**

Description

The IBC contract does not check what funds are sent into the public `ExecuteMsg::Send` instruction. It will send any funds the user requests from its balance to whatever address the user chooses.

Proof of Issue

File name: contracts/cosmoburn_ibc_transfer/src/contract.rs
Line number: 45

```
pub enum ExecuteMsg {
    Send {// user input for the instruction
        channel: String,
        to: String,
        denom: String,
        amount: u128,
        timeout_height: Option<u64>,
    },
}

#[entry_point]
pub fn execute(
    deps: DepsMut<NeutronQuery>,
    env: Env,
    _: MessageInfo,// funds actually sent with the instruction are in the info
    msg: ExecuteMsg,
) -> NeutronResult<Response<NeutronMsg>> {
    ...

    let coin1 = coin(amount, denom.clone());// user input

    let msg1 = NeutronMsg::IbcTransfer {
        source_port: "transfer".to_string(),
        source_channel: channel.clone(),// user input
        sender: env.contract.address.to_string(),// user self
        receiver: to.clone(),// user input
        token: coin1,
```

Severity and Impact Summary

Since the burn contract also sends in fees, the IBC contract likely accumulates tokens which can then be withdrawn by anyone.

Recommendation

Make sure this contract is properly verifying token amounts.

Backend trust required for critical logic

Finding ID: FYEO-CosmoBurn-04
Severity: **Medium**
Status: **Remediated**

Description

The code does not provide insight into the way tokens are sent to the burn contract in order to be burned. This logic is done by a backend which must verify the tokens received by the burn contract and must correctly calculate the `base_amount` which represents the reward issued for burning tokens. The burn contract will then burn its own balance and send a part to the distributor contract in accordance with the data specified by the backend and an 80 / 20 split.

Proof of Issue

File name: contracts/cosmoburn/src/burn.rs
Line number: 48

```
if wtx.burner_addr == cw20_msg.sender
    && wtx.workflow_id == workflow_param.id
    && wtx.base_amount == workflow_param.base_amount
    && wtx.burn_amount == workflow_param.burn_amount
    && wtx.denom == info.sender.clone().to_string()
    && timeout > env.block.time.seconds()
{
    is_whitelisted = true;

    ...

    let total_amount: u128 = workflow_param.burn_amount.into();

    ...

    let receive_coin = Coin {
        denom: workflow.rewards_token_addr.clone(),
        amount: workflow_param.base_amount,
    };
```

The amount is provided by the backend in a whitelist instruction, which is out of scope.

Line number: 188

```
let _total_funds = info.funds.clone(); // Ignores funds send in the burn transaction

if wtx.burner_addr == info.sender
    && wtx.workflow_id == workflow_param.id
    && wtx.denom == workflow_param.denom
    && wtx.base_amount == workflow_param.base_amount
    && wtx.burn_amount == workflow_param.burn_amount
    && timeout > env.block.time.seconds()
{
    is_whitelisted = true;
```

```
    ...
    let total_amount: u128 = workflow_param.burn_amount.into();
```

This also applies to `try_burn_native`.

Line number: 329

```
let _total_funds = info.funds.clone();// Ignores funds send in the burn transaction

if wtx.burner_addr == info.sender
    && wtx.workflow_id == workflow_param.id
    && wtx.denom == workflow_param.denom
    && wtx.base_amount == workflow_param.base_amount
    && wtx.burn_amount == workflow_param.burn_amount
    && timeout > env.block.time.seconds()
{
    is_whitelisted = true;

    ...
    let total_amount: u128 = workflow_param.burn_amount.into();
```

And `try_burn_ibc`.

```
check_is_backend(&deps, info)?;

...

Wtx {
    burner_addr: burner_addr.clone(),
    denom: denom.clone(),
    burn_amount: burn_amount,
    base_amount: base_amount,
    time: env.block.time.seconds(),
    workflow_id: workflow_id.clone(),
}
```

Data is set by the backend.

The code only considers the first coin type sent in this instruction. That implies each call can only deal with one coin type. It remains unclear whether additional coins (of other types or not) indicate an error. Again, the amounts received here are not checked against anything.

```
let total_funds = info.funds.clone();
let coin_type =
state.whitelisted_coins.get(&total_funds[0].denom.clone()).unwrap().coin_type.clone();
match coin_type {
    CoinType::NATIVE => try_burn_native(deps, info, env, workflow_params),
    CoinType::IBC => try_burn_ibc(deps, info, env, workflow_params),
    CoinType::CW20 => panic!("CW20 cannot be burned directly"),
}
```

Severity and Impact Summary

Since part of this logic is out of scope and handled off chain, it remains unclear if this is implemented in a secure manner.

Recommendation

Make sure this logic is correctly implemented. It must be guaranteed that the funds were sent to the burn contract.

Bounds / Input validation

Finding ID: FYEO-CosmoBurn-05
Severity: **Low**
Status: **Remediated**

Description

Some of the input is not properly verified. Addresses and denominations should be checked to be valid and bounds should be established on all values.

Proof of Issue

File name: contracts/cosmoburn/src/execute.rs
Line number: 151

```
pub fn try_whitelist_coin(
    coin_addr: String,
    coin_type: String,
    ...
) -> Result<Response, ContractError> {

    STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
        state
            .whitelisted_coins
            .insert(coin_addr.clone().to_string(), CoinDetail {
                ...
                coin_addr : coin_addr.clone(),
```

Even though this is an admin function, this CW20 address or denom should be checked to be valid.

File name: contracts/cosmoburn/src/contract.rs
Line number: 41

```
rewards_token_addr: workflow.rewards_token_addr.clone(),

tx_cooldown_period: msg.tx_cooldown_period,
whitelist_timeout: msg.whitelist_timeout,
```

The address should be checked to be valid. Bounds should be established on the cooldown and timeout settings.

File name: contracts/cosmoburn/src/execute.rs
Line number: 109

```
burner_addr: burner_addr.clone()
```

This address is not validated.

File name: contracts/cosmoburn/src/execute.rs
Line number: 197

```
STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.tx_cooldown_period = period;
    Ok(state)
})?;
```

```
...

STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.whitelist_timeout = timeout;
    Ok(state)
})?;
```

No bounds are established.

File name: contracts/cosmoburn_funds_distributor/src/contract.rs
Line number: 42

```
tracker_denom: msg.tracker_denom.to_string(),
treasury_addr: msg.treasury_addr.to_string(),
```

These should be validated. Having an invalid treasury may lead to a loss of funds.

File name: contracts/cosmoburn_funds_distributor/src/contract.rs
Line number: 316

```
token_addr.clone().to_string(),
```

This should be validated.

File name: contracts/cosmoburn_ibc_transfer/src/contract.rs
Line number: 135

```
revision_height: timeout_height.or(Some(DEFAULT_TIMEOUT_HEIGHT)),
```

While this is called by another contract, there are no check on bounds for this timeout.

Severity and Impact Summary

Misconfiguration of some of these values can break the contract.

Recommendation

Make sure to properly verify all user input.

Loop uses check_tx_frequency, prevents iterating

Finding ID: FYEO-CosmoBurn-06
Severity: **Low**
Status: **Remediated**

Description

The loop checks a condition that can only be true once. This guarantees transaction failure if more than one item is sent. This function is also checking a specific token contract rather than an individual sender. Meaning the limit applies to the token not the users.

Proof of Issue

File name: contracts/cosmoburn/src/burn.rs
Line number: 40

```
for workflow_param in workflow_params {

    let workflow = config.workflows.get(&workflow_param.id).unwrap();

    check_tx_frequency(state.clone(), env.clone(), info.clone())?;
```

Severity and Impact Summary

This code limits the functionality of the contract.

Recommendation

Make sure to implement proper limits for token transfers.

Admin can withdraw all funds

Finding ID: FYEO-CosmoBurn-07
Severity: **Informational**
Status: **Acknowledged**

Description

The admin can withdraw funds from the contract as they please.

Proof of Issue

File name: contracts/cosmoburn/src/execute.rs
Line number: 17

File name: contracts/cosmoburn_funds_distributor/src/execute.rs
Line number: 17

```
pub fn try_withdraw(
    deps: DepsMut,
    info: MessageInfo,
    amount: Uint128,
    denom: String,
) -> Result<Response, ContractError> {
    ...

    check_is_admin(&deps, info.clone())?;
```

Severity and Impact Summary

While this might help recover funds under certain circumstances, it also opens up the possibility of an admin stealing user funds.

Recommendation

Consider the risks and benefits of this functionality.

Code clarity / optimization

Finding ID: FYEO-CosmoBurn-08
Severity: **Informational**
Status: **Remediated**

Description

Some of the code can be optimized as parts are quite convoluted, contain considerable amounts of duplicate code and sometimes it is wasteful.

Proof of Issue

File name: contracts/cosmoburn/src/burn.rs
Line number: 210

```
let mut burn_coins = Vec::new();
burn_coins.push(burn_coin.clone());

let mut dist_coins = Vec::new();
dist_coins.push(dist_coin.clone());

let mut receive_coins = Vec::new();
receive_coins.push(receive_coin.clone());

messages.push(CosmosMsg::from(BankMsg::Send {
    amount: dist_coins,
    to_address: workflow.fund_distributor_addr.clone(),
}));
```

These are only used once and can be declared inline i.e.: `vec![dist_coin]`. This applies to all 3 burn functions. Each of these also creates a new message for each iteration of the outer loop. These individual sub-messages could be combined.

File name: contracts/cosmoburn/src/burn.rs
Line number: 183

```
for ...
    whitelist_valid.push(wtx.clone());

    state.whitelisted_tx = whitelist_valid.clone();
```

The whitelist is cleaned and the state is rewritten in every iteration of the outer loop. This could instead be done once to avoid writing state again and again. Also, the `try_clean_up` function does not check if the length changed and just writes the data in any case.

The 3 burn functions share a lot of code that should be extracted into functions.

This pattern is used many times:

```
for key in ibc_transfer_amount.keys() {
    let amount = ibc_transfer_amount.get(key).unwrap();

for coin_address in coin_balances.keys() {
    let coin_balance: Uint128 = (*coin_balances.get(coin_address).unwrap()).into();
```

The key and value can be retrieved at the same time with a normal iterator: `for (coin_address, coin_balance) in coin_balances.iter()`.

This can be extracted into a function as it is duplicated a few times:

```
messages.push(match state.whitelisted_coins.get(coin_address).unwrap() {
    CoinType::NATIVE => CosmosMsg::from(BankMsg::Send {
        amount: vec![Coin {
            denom: coin_address.clone(),
            amount: Uint128::new(coin_distrib_amount.into()),
        }],
        to_address: holder.address.clone(),
    }),
    CoinType::CW20 => CosmosMsg::Wasm(WasmMsg::Execute {
        contract_addr: coin_address.clone(),
        msg: to_json_binary(&Cw20ExecuteMsg::Transfer {
            recipient: holder.address.clone(),
            amount: coin_distrib_amount.into(),
        })?,
        funds: vec![],
    }),
    CoinType::IBC => CosmosMsg::from(BankMsg::Send {
        amount: vec![Coin {
            denom: coin_address.clone(),
            amount: Uint128::new(coin_distrib_amount.into()),
        }],
        to_address: holder.address.clone(),
    }),
});
```

This is a 120+ line `if` block:

```
if !coin_balances.is_empty() && !holders.is_empty() {
    ...
```

This creates unnecessary code indentation which can be avoided by returning early for the else case instead.

There are many instances in which idiomatic rust could be used to more clearly communicate what the code does. For instance:

```
match burn_log_item.workflow_amount.get(&workflow_param.id){
    None => burn_log_item.workflow_amount.insert(
        workflow_param.id.clone(), workflow_param.base_amount.clone()
    ),
    Some(amount) => burn_log_item.workflow_amount.insert(
        workflow_param.id.clone(), amount + workflow_param.base_amount.clone()
    )
};
```

This code can be written as:

```
burn_log_item.workflow_amount
    .entry(workflow_param.id.clone())
    .and_modify(|amount| *amount += workflow_param.base_amount.clone())
    .or_insert(workflow_param.base_amount.clone());
```

Severity and Impact Summary

The code quality is somewhat bad in some parts which may negatively affect the maintenance of the codebase and may make it more prone to errors.

Recommendation

Clean code will help with the maintenance of this codebase and helps future developers easily understand it.

Data is silently overwritten or may not be deleted

Finding ID: FYEO-CosmoBurn-09
Severity: **Informational**
Status: **Remediated**

Description

Some of the state data is silently overwritten or in other cases may not be deleted at all. For instance, an admin may want to de-list a certain coin and they receive an OK, but they made a typo and did not actually de-list anything. Or maybe they wanted to add some coin but made a mistake and are now overwriting some other data which is just going to be replaced.

Proof of Issue

File name: contracts/hostburner/src/contract.rs
Line number: 93, 118

```
STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state
        .whitelisted_tokens
        .insert(token_addr.clone().to_string(),
CoinType::from_str(&token_type.to_string()).unwrap());
    Ok(state)
})?

STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.whitelisted_tokens.remove(&token_addr.clone());
    Ok(state)
})?;
```

File name: contracts/cosmoburn/src/execute.rs
Line number: 146, 181

```
STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state
        .whitelisted_coins
        .insert(coin_addr.clone().to_string(), CoinDetail {
            ...
        });
    Ok(state)
})?;

STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.whitelisted_coins.remove(&coin_addr.clone());
    Ok(state)
})?;
```

File name: contracts/cosmoburn_tracker/src/contract.rs
Line number: 78, 99

```
STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state
        .excluded_wallets
```

```
        .insert(addr.clone().to_string(), memo);
    Ok(state)
})?

STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.excluded_wallets.remove(&addr.clone());
    Ok(state)
})?;
```

File name: contracts/cosmoburn_funds_distributor/src/contract.rs
Line number: 315, 336

```
state.whitelisted_coins.insert(
    token_addr.clone().to_string(),
    CoinType::from_str(&token_type).unwrap(),
);

STATE.update(deps.storage, |mut state| -> Result<_, ContractError> {
    state.whitelisted_coins.remove(&token_addr.clone());
    Ok(state)
})?;
```

Severity and Impact Summary

Mistakes could be made and the user may not notice.

Recommendation

Consider if this implementation is in-line with expectations.

IBC contract appears unfinished

Finding ID: FYEO-CosmoBurn-10
Severity: **Informational**
Status: **Remediated**

Description

Most of the functions just store data or log debug info.

Proof of Issue

File name: contracts/cosmoburn_ibc_transfer/src/contract.rs
Line number: 158

```
#[entry_point]
pub fn sudo(deps: DepsMut, _env: Env, msg: TransferSudoMsg) -> StdResult<Response> {
    match msg {
        // For handling successful (non-error) acknowledgements
        TransferSudoMsg::Response { request, data } => sudo_response(deps, request,
data),

        // For handling error acknowledgements
        TransferSudoMsg::Error { request, details } => sudo_error(deps, request,
details),

        // For handling error timeouts
        TransferSudoMsg::Timeout { request } => sudo_timeout(deps, request),
    }
}
```

These function all end with debug messages. For instance:

```
deps.api.debug(
    format!(
        "WASMDEBUG: sudo_timeout: sudo timeout ack received: {:?}",
        req
    )
    .as_str(),
);
Ok(Response::new())
```

Severity and Impact Summary

The expected functionality is not clear as the code seems to be unfinished.

Recommendation

Make sure this contract is fully implemented.

Potential overflow

Finding ID: FYEO-CosmoBurn-11
Severity: **Informational**
Status: **Remediated**

Description

There are some possible overflows in the code that depend on inputs made outside of the scope of this review.

Proof of Issue

File name: contracts/cosmoburn/src/burn.rs
Line number: 67, 207, 348

```
let amount_80 = total_amount * 80 / 100;
```

Severity and Impact Summary

While that is unlikely to overflow, it depends on valid amounts being used by the backend.

Recommendation

Use the U128 type implemented by the cosmos sdk instead as that comes with overflow checks or use checked math.

Storage never cleared

Finding ID: FYEO-CosmoBurn-12
Severity: **Informational**
Status: **Remediated**

Description

This contract stores data but never directly cleans up. Though the `REPLY_QUEUE_ID´ works as a ring buffer of 1000 items.

Proof of Issue

File name: contracts/cosmoburn_ibc_transfer/src/state.rs
Line number: 31

```
REPLY_QUEUE_ID.save(store, id, &to_json_vec(&payload)?)?;

SUDO_PAYLOAD.save(store, (channel_id, seq_id), &to_json_vec(&payload)?)
```

Severity and Impact Summary

It appears unnecessary to store data in this fashion.

Recommendation

Consider requirements and implement the functionality accordingly.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations