SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	**Public**
Prepared for:	**Andromeda**
Prepared by:	**Sherlock**
Lead Security Expert:	**bin2chen**
Dates Audited:	**June 3 - June 23, 2024**
Prepared on:	**September 25, 2024**

Introduction

Andromeda's goal is to continuously bring more ADOs/functionality to our users and development community and beyond.

Scope

Repository: andromedaprotocol/andromeda-core

Branch: development

Audited Commit: 676c9833f0813939c0a4f8dee60fd9feb4230e01

Final Commit: ee4434ef28367ada4a9a2aa819adaca99d31de1d

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
15	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

g
bin2chen
J4X_
cu5t0mPe0
Kow
Yashar
0xR360

Issue H-1: verify_origin() previous_sender may be forged

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/45

Found by

bin2chen, g

Summary

in `AMPPkt.verify_origin()` does not verify the legitimacy of the `previous_sender` and can be specified at will, leading to security risks.

Vulnerability Detail

The `execute()` method of most current ado's can handle two types of ExecuteMsg `ExecuteMsg::AMPReceive` or `other Msg`

```
pub fn execute(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    msg: ExecuteMsg,
) -> Result<Response, ContractError> {
    let ctx = ExecuteContext::new(deps, info, env);

    match msg {
        ExecuteMsg::AMPReceive(pkt) => {
@>          ADOContract::default().execute_amp_receive(ctx, pkt, handle_execute)
        }
        _ => handle_execute(ctx, msg),
    }
}
```

If the request is for an `AMPReceive` it checks the legality of the `AMPCtx` at `execute_amp_receive()`. `execute_amp_receive()->verify_origin()`

```
   pub fn verify_origin(&self, info: &MessageInfo, deps: &Deps) -> Result<(),
↪  ContractError> {
        let kernel_address =
↪  ADOContract::default().get_kernel_address(deps.storage)?;
@>      if info.sender == self.ctx.origin || info.sender == kernel_address {
            Ok(())
        } else {
```

```
            let adodb_address: Addr =
                deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
                    contract_addr: kernel_address.to_string(),
                    msg: to_json_binary(&KernelQueryMsg::KeyAddress {
                        key: ADO_DB_KEY.to_string(),
                    })?,
                }))?;

            // Get the sender's Code ID
            let contract_info: ContractInfoResponse =
                deps.querier
                    .query(&QueryRequest::Wasm(WasmQuery::ContractInfo {
                        contract_addr: info.sender.to_string(),
                    }))?;

            let sender_code_id = contract_info.code_id;

            // We query the ADO type in the adodb, it will return an error if
↪   the sender's Code ID doesn't exist.
            AOSQuerier::verify_code_id(&deps.querier, &adodb_address,
↪   sender_code_id)
        }
    }
```

The main task is to check the legitimacy of `AMPCtx.origin` and `AMPCtx.previous_sender` There are three cases

1. sender == kernel_address -> pass (trusted by default, not malicious)
2. sender == ADO type in the adodb -> pass (trusted by default, not malicious)
3. sender == user (user submits AMPReceive directly) -> check ‘AMPCtx.origin == sender

In the third case, only `AMPCtx.origin == user` is checked and there is no restriction on `AMPCtx.previous_sender == user`. So the user can submit `ExecuteMsg::AMPReceive` and specify `previous_sender` as they wish.

Impact

If `AMPCtx.previous_sender` can be specified arbitrarily, security checks that depend on it will have security implications Example: `ExecuteContext.contains_sender()`

```
    pub fn contains_sender(&self, addr: &str) -> bool {
        if self.info.sender == addr {
            return true;
        }
```

```
        match &self.amp_ctx {
            None => false,
@>          Some(ctx) => ctx.ctx.get_origin() == addr ||
↪   ctx.ctx.get_previous_sender() == addr,
        }
    }
```

The one that currently has the ability to determine permissions using this method is `andromeda-cw721`.

```
fn execute_mint(
    ctx: ExecuteContext,
    token_id: String,
    token_uri: Option<String>,
    owner: String,
    extension: TokenExtension,
) -> Result<Response, ContractError> {
    let minter = ANDR_MINTER
        .load(ctx.deps.storage)?
        .get_raw_address(&ctx.deps.as_ref())?;
    ensure!(
@>      ctx.contains_sender(minter.as_str())
            | is_context_permissioned_strict(
                ctx.deps.storage,
                &ctx.info,
                &ctx.env,
                &ctx.amp_ctx,
                MINT_ACTION
            )?,
        ContractError::Unauthorized {}
    );
```

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/amp/messages.rs#L297

Tool used

Manual Review

Recommendation

```
    pub fn verify_origin(&self, info: &MessageInfo, deps: &Deps) -> Result<(),
↪   ContractError> {
        let kernel_address =
↪   ADOContract::default().get_kernel_address(deps.storage)?;
-       if info.sender == self.ctx.origin || info.sender == kernel_address {
+       if (info.sender == self.ctx.origin && info.sender ==
↪   self.ctx.previous_sender) || info.sender == kernel_address {
            Ok(())
        } else {
            let adodb_address: Addr =
                deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
                    contract_addr: kernel_address.to_string(),
                    msg: to_json_binary(&KernelQueryMsg::KeyAddress {
                        key: ADO_DB_KEY.to_string(),
                    })?,
                }))?;
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/552

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/552
This PR adds judgment`info.sender == self.ctx.origin && info.sender == self.ctx.previous_sender` Fixed this issue

Issue M-1: the DEFAULTVALIDATOR cannot be changed

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/18

Found by

cu5t0mPe0

Summary

`DEFAULTVALIDATOR` can not alter the default staking validator

Vulnerability Detail

sherlock docs: `can alter the default staking validator for the validator staking contract`

But in reality, validator-staking does not have a function related to `setDEFAULTVALIDATOR`. The only way to modify `DEFAULTVALIDATOR` is to call `instantiate` and reinstantiate a new `validator-staking`. This contradicts the documentation, so I consider this a Medium issue.

Impact

the default staking validator cannot be changed

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L43

Tool used

Manual Review

Recommendation

Add and modify related functions of `DEFAULTVALIDATOR`

Discussion

cu5t0mPeo

escalate The README file mentions: link.

According to Sherlock's rules: link.

Therefore, this is a medium.

sherlock-admin3

> escalate The README file mentions: link.
>
> According to Sherlock's rules: link.
>
> Therefore, this is a medium.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

J4X-98

As you voice yourself, the admin can change the default validator by re-instantiating and updating to a new version number. While it would be a nicer way to have a separate function for that, the behavior described in the contest description is implemented in code.

cu5t0mPeo

Reinstantiation will completely change the context, equivalent to redeploying a contract, which clearly does not align with the logic in the documentation. If creating a new instantiation is the solution to this problem, then I believe it is unnecessary to mention this point in the documentation at all.

cvetanovv

According to Sherlock's rules, I think it might be Medium:

"The protocol team can use the README (and only the README) to define language that indicates the codebase's **restrictions and/or expected functionality**. Issues that **break these statements**, irrespective of **whether the impact is low/unknown**, will be assigned Medium severity."

cvetanovv

Planning to accept the escalation and make this issue a valid Medium.

WangSecurity

Are there any duplicates we need to add?

WangSecurity

Result: Medium Unique

cu5t0mPeo

> Are there any duplicates we need to add?

no

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- cu5t0mPeo: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/558

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/558
This PR adds the method `execute_update_default_validator()` which modifies the `DEFAULTVALIDATOR` and can only be executed by the `owner`. Fixed this issue

Issue M-2: Permission checks will unnecessarily consume Limited uses

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/27

Found by

g

Summary

When a permission check is done with ado_contract/permissioning.rs::is_permissioned() on a Limited permission, uses are always consumed.

Vulnerability Detail

When doing a permission check, a Limited permission will always have its uses consumed.

ref: ado_contract/permissioning.rs::is_permissioned()

```
pub fn is_permissioned(
    // ... snip ...
) -> Result<(), ContractError> {
    // ... snip ...
    match permission {
        Some(mut permission) => {
            // ... snip ...
            if let Permission::Limited { .. } = permission {
                permission.consume_use();
                permissions().save(
                    store,
                    (action_string.clone() + actor_string.as_str()).as_str(),
                    &PermissionInfo {
                        action: action_string,
                        actor: actor_string,
                        permission,
                    },
                )?;
            }

            Ok(())
        }
```

In functions like is_context_permissioned() and is_context_permissioned_strict(), the permission check may be done on 2 different users. Consider the case when both the origin and the previous sender have Limited permissions. Both of their Limited uses will be consumed even when permissions for one is enough.

ref: ado_contract/permissioning.rs::is_context_permissioned()

```
pub fn is_context_permissioned(
    // ... snip ...
) -> Result<bool, ContractError> {
    let contract = ADOContract::default();

    match ctx {
        // auth is done on origin and previous_sender
        Some(amp_ctx) => {
            let action: String = action.into();
            let is_origin_permissioned = contract.is_permissioned(
                storage,
                env.clone(),
                action.clone(),
                amp_ctx.ctx.get_origin().as_str(),
            );
            let is_previous_sender_permissioned = contract.is_permissioned(
                storage,
                env.clone(),
                action,
                amp_ctx.ctx.get_previous_sender().as_str(),
            );
            Ok(is_origin_permissioned.is_ok() ||
↪   is_previous_sender_permissioned.is_ok())
```

In the cw721 ADO Contract, `is_context_permissioned()` is called for every execute handling and every mint and batch mint will call `is_context_permissioned_strict()`. This leads to minting and batch minting consuming up to 4 Limited uses across 2 addresses.

Impact

Users/Contracts with Limited permissions will unexpectedly run out of uses. A whitelisted user who notices this behavior can use up a contract's or user's limited uses by using AMP to have the target address be the origin or previous sender. This issue can also naturally occur.

Code Snippet

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/permissioning.rs#L43-L91
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/permissioning.rs#L323-L353
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/permissioning.rs#L360-L390

Tool used

Manual Review

Recommendation

Consider ensuring that the `is_permissioned()` check only consumes the Limited use of just one address. Also, it may be worth considering changing the minting functions in cw721 ADO contract to only call `is_context_permissioned_strict()` and not call `is_context_permissioned()`.

Discussion

gjaldon

Escalate

This report shows that users with Limited permissions will run out of uses sooner than expected, since the expectation is the one permissioned action will only consume 1 use and not 2-4. The issue naturally occurs and a Whitelisted user can force consumption of users with Limited uses.

sherlock-admin3

> Escalate
>
> This report shows that users with Limited permissions will run out of uses sooner than expected, since the expectation is the one permissioned action will only consume 1 use and not 2-4. The issue naturally occurs and a Whitelisted user can force consumption of users with Limited uses.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

J4X-98

This issue consist of 2 root causes:

`is_context_permissioned()` consumes a user from the origin and sender

- This is clearly intended design. If this would have not been intended, the developers would not have added the whole additional check. There is also no documentation stating that it should not consume the origins user too.

`cw721` burns too many uses

- This is a vulnerability in the `cw721` contract where the usage of permissions was incorrectly implemented. `cw721` is not in scope. So a wrong usage of the correctly working `is_permissioned` functions (that are in scope) in a different out of scope file can not be considered an issue.

cvetanovv

@gjaldon @J4X-98 @MxAxM Is `cw721` a library?

That makes `cw721` in scope according to the rules and is the opposite of what I wrote under another issue.

J4X-98

Hey @cvetanovv ,

cw721 is not a library used by any of the modules or the core contract which are in scope. It is an additional module that inherits some of the core functionalities. So the inheritance is not in the way which would put it in scope as per the rules, but the different way around.

gjaldon

@cvetanovv @J4X-98 the issue is also in `https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/permissioning.rs#L43-L91` which is a file in-scope. The issue happens in the following lines:

```
let is_origin_permissioned = contract.is_permissioned(
    storage,
    env.clone(),
    action.clone(),
    amp_ctx.ctx.get_origin().as_str(),
);
let is_previous_sender_permissioned = contract.is_permissioned(
    storage,
    env.clone(),
    action,
```

```
    amp_ctx.ctx.get_previous_sender().as_str(),
);
```

`is_permissioned()` is called on both the previous sender and the origin. Given those, the limited uses are already consumed for 2 different users.

The issue in `cw721` adds 2 more uses. So even without the issue in `cw721`, `is_permissioned()` still needs fixing. The permissioning functions are only used in the `cw721` contract and not in the contracts in-scope (`vesting` and `validator-staking`).

gjaldon

Also, the following argument has no bearing.

> This is clearly intended design. If this would have not been intended, the developers would not have added the whole additional check. There is also no documentation stating that it should not consume the origins user too.

It can also be argued that no documentation states that limited uses should be consumed for both the `origin` and the `previous_sender`.

Also, consider the scenario where `origin` and `previous_sender` are the same address (a common case). That address will get its 2 limited uses consumed to access one action. This example more clearly shows that this behavior is incorrect.

cvetanovv

I have received confirmation from the sponsor that this is not an intended design and is indeed an issue.

`is_permissioned()` is within the audit scope, so I plan to accept the escalation.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- gjaldon: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/553

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/553 https://github.com/andromedaprotocol/andromeda-core/pull/569 PR modified `is_context_permissioned` and `is_context_permissioned_strict()` to execute only once to reduce the number of times Fixed this issue

Issue M-3: Valid VFS paths with usernames can always fail validation

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/30

The protocol has acknowledged this issue.

Found by

g

Summary

VFS paths are validated across AndromedaOS with get_raw_address(). Valid VFS paths may include usernames. However, path resolution can fail for paths with valid usernames and in effect, cause validation with `get_raw_address()` to fail. This issue exists for a large subset of usernames and libraries.

Vulnerability Detail

When `get_raw_address()` is used to validate a VFS path, it queries the VFS to resolve it. It attempts to resolve the path with resolve_pathname() which eventually calls either resolve_home_path() or resolve_lib_path(). The issue exists in both `resolve_lib_path()` and `resolve_home_path()`.

When the VFS path includes a registered username or library and that username/library is also a valid address according to `deps.api.addr_validate()`, the address stored for the registered username/library will not be loaded.

ref: andromeda-vfs/src/state.rs::resolve_home_path()

```
// @audit-issue if a username is also a valid address, then the address for the
↪  registered username can never be loaded
let user_address = match api.addr_validate(username_or_address) {
    Ok(addr) => addr,
    Err(_e) => USERS.load(storage, username_or_address)?,
};
resolve_path(storage, api, parts, user_address)
```

The username/library will be used for path resolution instead of the stored address which will cause an error because a non-existent path is being loaded.

ref: andromeda-vfs/src/state.rs::resolve_path()

```
fn resolve_path(
    storage: &dyn Storage,
    api: &dyn Api,
    parts: Vec<String>,
    user_address: Addr,
) -> Result<Addr, ContractError> {
    let mut address = user_address;
    for (idx, part) in parts.iter().enumerate() {
        if idx <= 1 {
            continue;
        }
        // @audit-issue address here will be the username or library instead of
 ↪  an address. the key is non-existent
        // and will cause an error
        let info = paths().load(storage, &(address, part.clone()))?;
```

The issue can be verified by changing the `username` in the test test_resolve_home_path and then running the test with `cargo test -- test_resolve_home_path --show-output`.

```
fn test_resolve_home_path() {
    let mut deps = mock_dependencies();
-   let username = "u1";
+   let username = "username1980";
```

Impact

VFS path validation is done all over AndromedaOS. This issue will break a lot of functionality and cause a loss of funds for the valid paths that are victims of this bug. For example, the Validator Staking ADO does address validation of the recipient in execute_claim(). The recipient that fails validation can never claim their stake. In Kernel ADO, every local AMP message's recipient is validated. This means the victim paths can not receive AMP messages since they will always fail validation. The consequences of this validation issue are far-reaching in the AndromedaOS system and are just a few of the impacts caused.

Code Snippet

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/amp/addresses.rs#L63-L71
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/os/andromeda-vfs/src/state.rs#L57-L77

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/os/andromeda-vfs/src/state.rs#L92-L95
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/os/andromeda-vfs/src/state.rs#L121-L133

Tool used

Manual Review

Recommendation

When resolving the home or lib path, consider checking storage for the username or library. If it exists, then load the address for the username/library. If it does not exist, treat it is an address and validate it with `deps.api.addr_validate()`.

Discussion

gjaldon

Escalate

This is a valid issue because it shows how registered usernames will not be loaded for VFS paths with usernames like `/home/username1/app_contract`. Usernames that are also valid addresses (they will return true for `api.addr_validate()`) will not be loaded. Impact is high as explained in the Impact section.

sherlock-admin3

> Escalate
>
> This is a valid issue because it shows how registered usernames will not be loaded for VFS paths with usernames like `/home/username1/app_contract`. Usernames that are also valid addresses (they will return true for `api.addr_validate()`) will not be loaded. Impact is high as explained in the Impact section.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cu5t0mPeo

this is know issue

gjaldon

This is not a known issue. The known issue is that `/home/app_contract` is used as vfs paths for components of Apps instead of `/home/username/app_contract`. This issue exists whether path is `/home/username/app_contract` or `/home/app_contract`.

cvetanovv

@cu5t0mPeo Why do you think it is a known issue?

cu5t0mPeo

sry,I was mistaken and thought it was the same issue as the VFS path error.

cvetanovv

Watson has shown how valid VFS paths with usernames can always fail validation. This would break a lot of functionalities and lead to loss of funds.

Planning to accept the escalation and make this issue High.

WangSecurity

@gjaldon @MxAxM are there any duplicates here?

WangSecurity

Result: High Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- gjaldon: accepted

gjaldon

@WangSecurity it is unique. Thank you

J4X-98

Hey @cvetanovv @WangSecurity,

Sorry for the late remark on this, I just saw that this was validated. I do not disagree with this issues validity, this is clearly an issue. But I don't see why this was judged as high. There is no loss of funds, this issue is clearly unintended functionality.

The issue states 2 cases as impact:

1. "For example, the Validator Staking ADO does address validation of the recipient in execute_claim(). The recipient that fails validation can never claim their stake."

The impact of not being able to claim can easily be bypassed by the user, by just not passing the optional `recipient`parameter, in which case the funds will be

distributed to the `info.sender`. This address anyways has to be equal to the recipient due to earlier checks. In this case the reverting call to `get_raw_address()` will never occur.

```
let recipient = if let Some(recipient) = recipient {
    recipient.get_raw_address(&deps.as_ref())?
} else {
    info.sender
};
```

By doing this the user will still be able to retrieve his funds, so no loss of funds. -> Medium

2. "In Kernel ADO, every local AMP message's recipient is validated. This means the victim paths can not receive AMP messages since they will always fail validation."

-> This is just a functionality not working not a loss of funds -> Medium

gjaldon

Hi @cvetanovv @WangSecurity @J4X-98,

This issue leads to a loss of funds.

The Vesting ADO's execute_claim() will always fail and the `recipient` will not be able to claim any of the vested funds. All vested funds will remain stuck in the Vesting ADO because there is no way to change the configured recipient. It is only set on instantiation.

This is an example of loss of funds and there may be other parts in Andromeda where it happens because of this issue since `get_raw_address()` is used in most, if not all, parts of AndromedaOS (ADO Contracts, Modules, Core contracts).

J4X-98

Your issue states that the validator stakings is the one that can't be claimed anymore as you can see in the snippet from your issue below. That's what my first answer was based upon.

As I have shown that this is not the case, and there is no actual loss of funds, you are now stating that the Vesting's `execute_claim()` function will not work.

> The Vesting ADO's execute_claim() will always fail and the `recipient` will not be able to claim any of the vested funds. All vested funds will remain stuck in the Vesting ADO because there is no way to change the configured recipient. It is only set on instantiation.

This is also untrue as the vulnerable get_raw_address() function is never called in the vestings execute_claim() function as anyone can easily verify. So the function

will never revert due to the issue you describe. Matter of fact, the vulnerable function is never called in the whole vestings `contract.rs`.

> This is an example of loss of funds and there may be other parts in Andromeda where it happens because of this issue since `get_raw_address()` is used in most, if not all, parts of AndromedaOS (ADO Contracts, Modules, Core contracts).

To answer your second point, stating that there might be some loss of funds but not being able to prove, is not grounds for a high severity.

gjaldon

> As I have shown that this is not the case, and there is no actual loss of funds, you are now stating that the Vesting's `execute_claim()` function will not work.

I meant Vesting ADO but made a mistake and wrote Validator Staking. Anyway, I also stated the following in the report:

```
The consequences of this validation issue are far-reaching in the AndromedaOS
↪  system and are just a few of the impacts caused.
```

> This is also untrue as the vulnerable get_raw_address() function is never called in the vestings execute_claim() function as anyone can easily verify. So the function will never revert due to the issue you describe. Matter of fact, the vulnerable function is never called in the whole vestings contract.rs.

`get_raw_address()` is actually called indirectly via generate_direct_msg(). The Vesting ADO's `execute_claim()` calls `generate_direct_msg()` here.

gjaldon

Almost everywhere an `AndrAddr` is accepted `get_raw_address()` will be called to validate it.

J4X-98

You're right about the function being called via the other one, I missed that.

Nevertheless I think it's important to add that this issue in vesting is not a permanent loss of funds as the vesting contract has a migrate function which allows the owner to migrate to a version including a function that allows him to withdraw the tokens again. As the owner is also the one the funds originate from, this issue just results in him not being able to distribute the rewards to users affected by this issue. -> this is unintended functionality not loss of funds.

This is in contrast to the staking module, that does not allow migrating so there the funds can not be rescued in this way.

gjaldon

It will lead to loss of funds because will be sent to the incorrect address.

When resolving the path "/home/username1" with `get_raw_address()`, it will return `"username1"` as the address. execute_claim() will send funds to `"username1"` which is a valid address and the funds will be lost.

Function flow for `get_raw_address()` on `"/home/username1"`:

1. local_path_to_vfs_path() will return "/home/username1" since it isn't a local path.
2. vfs_resolve_path() will then be called which will call resolve_pathname() --> resolve_home_path() --> resolve_path()
3. "username1" will be passed as the user address to `resolve_path()` because it is a valid address as stated in the original report.
4. `resolve_path()` returns "username1" as the address because the "home" and "username1" parts of the path will be skipped.

==========

> This is in contrast to the staking module, that does not allow migrating so there the funds can not be rescued in this way.

Regarding Staking ADO's upgradability, not having a `migrate` function defined in Staking ADO does not mean it is not upgradeable. The `admin` only needs to be set for the contract and the new contract is the one that needs to implement the `migrate` function. Basically, all contracts in CosmWasm chains are upgradeable if an admin is set which is up to deploy ops.

From CosmWasmMigration docs:

> During the migration process, the migrate function defined in the new contract is executed and not the migrate function from the old code and therefore it is necessary for the new contract code to have a migrate function defined and properly exported as an entry_point:

J4X-98

First you stated it will revert, now you state that it will be transferred to the wrong address. Which one is the case now?

gjaldon

This issue is high severity because of the loss of funds.

"./username1" paths will lead to reverts and "/home/username1" paths will lead to transfers to the incorrect address.

The original report states:

> The consequences of this validation issue are far-reaching in the AndromedaOS system and are **just a few of the impacts caused**.

There are multiple impacts caused by the issue and are not limited to the ones I provided as stated in the original report. The report did state the impact is loss of funds.

J4X-98

But wouldn't this require a user to set a malicious name and also require the owner to set a malicious name as the recipient. I guess we can assume that if someone uses "/home/username1" as his username this is clearly a payload to everyone setting this somewhere.

gjaldon

No malicious username needs to be used. The vulnerability is naturally occurring for valid usernames.

J4X-98

Would you define "/home/username" as a non-malicious username?

gjaldon

Yes. That's only an example and there are many other usernames. Anyway, I've provided enough explanation to show that the issue exists and leads to permanent loss of funds and other impacts. I've spent too much time on this discussion already. I hope you don't mind that we end the discussion here.

cvetanovv

Judging by this comment here, I'm planning to change the severity of #41 and #30 to Medium because the funds can be rescued.

@gjaldon is right that in `CosmWasm`, contracts can be set upgradeable by the Admin at the beginning. According to the rules, the Admin is expected to take the correct action. It might also mean setting the contracts upgradeable. This is missing as QA in the Readme, so we'll stick to assuming the Admin will make the right decisions.

gjaldon

Hi @cvetanovv @WangSecurity. The loss of funds in this report is permanent because they will be sent to the incorrect address.

I explained it in this comment.

Please let me know if there is anything that is unclear.

J4X-98

The case you describe would require the user to provide a malicious username formed like a path to lose his own rewards. This scenario makes no sense to me.

I agree with @cvetanovv judging

gjaldon

> The case you describe would require the user to provide a malicious username formed like a path to lose his own rewards. This scenario makes no sense to me.

This is a high-severity issue that causes permanent loss of funds for valid usernames. There are no "malicious" usernames that need to be used. Users only need to register and use valid usernames and they end up losing funds.

I'm unsure how else to explain the issue to make it clearer for you @J4X-98.

J4X-98

I'm aware of the issue you describe. However, it requires someone to register with a username like "/home/username/" so that the resolving fails, which makes no sense unless the user wants to bypass this feature. This makes no sense for a user, because it will just result in him losing his own rewards.

gjaldon

@J4X-98 can you read the code about VFS Path resolution first before arguing here? It's a waste of time to have to explain.

When registering usernames, a user only needs to register a username like "abcde1". Now when paths are resolved, they are expected to be either just `"home"` or `"lib"` paths. Home paths are prefixed by `"~/"` or `"/home"` while lib paths are prefixed by `"/lib"`.

Code in path resolution:

```
match pathname.get_root_dir() {
    "home" => resolve_home_path(storage, api, pathname),
    "lib" => resolve_lib_path(storage, api, pathname),
    &_ => Err(ContractError::InvalidAddress {}),
}
```

The only way to use the username `"abcde1"` in a VFS path is to prefix it with `"/home"`.

Loading username:

```
fn resolve_home_path(
    storage: &dyn Storage,
    api: &dyn Api,
    pathname: AndrAddr,
) -> Result<Addr, ContractError> {
    // snip ...
    let user_address = match api.addr_validate(username_or_address) {
```

```
        Ok(addr) => addr,
        Err(_e) => USERS.load(storage, username_or_address)?, // The address
↪  linked to the username is loaded here
    };
```

So your argument below makes ZERO sense:

> However, it requires someone to register with a username like "/home/username/" so that the resolving fails, which makes no sense unless the user wants to bypass this feature.

No need to register "/home/username". They only need to register "username" and using usernames in VFS paths means the path always needs to be prefixed "/home". That's how VFS paths work in AndromedaOS.

I want to be as civil and respectful as possible and play fair, but you seem to be intentionally wasting people's time here @J4X-98.

J4X-98

Hey @gjaldon ,

I will refrain from answering your insults regarding wasting everyone's time and will try to stay factual here.

Based on your issue and the following comments, you state that if a user uses the username "username1" and a vesting is generated for him, he will not receive the tokens due to the VFS path being incorrectly resolved. In your example the vesting should be generated with the VFS path as the recipient being set.

I have adapted the testcases in the vesting module to this scenario, and the user exactly receives the tokens to the provided vfs path as you can see from the returned Send message. Could you show me exactly where the vulnerability occurs here / tokens are lost? To me it seems like the vesting system does exactly what it should.

```
fn init(deps: DepsMut) -> Response {
    let msg = InstantiateMsg {
        recipient: Recipient::from_string("/home/username1"), //Vesting is
↪  generated with the VFS path you describe as the recipient
        is_multi_batch_enabled: true,
        denom: "uusd".to_string(),
        unbonding_duration: Duration::Height(UNBONDING_BLOCK_DURATION),
        kernel_address: MOCK_KERNEL_CONTRACT.to_string(),
        owner: None,
        modules: None,
    };
```

```
    let info = mock_info("owner", &[]);
    instantiate(deps, mock_env(), info, msg).unwrap()
}

#[test]
fn test_claim_batch_single_claim() {
    let mut deps = mock_dependencies_custom(&[]);
    init(deps.as_mut());
    let info = mock_info("owner", &coins(100, "uusd"));

    let release_unit = 10;

    // Create batch.
    let msg = ExecuteMsg::CreateBatch {
        lockup_duration: None,
        release_unit,
        release_amount: WithdrawalType::Amount(Uint128::new(10)),
        validator_to_delegate_to: None,
    };

    let _res = execute(deps.as_mut(), mock_env(), info.clone(), msg).unwrap();

    deps.querier
        .base
        .update_balance(MOCK_CONTRACT_ADDR, coins(100, "uusd"));

    // Skip time.
    let mut env = mock_env();
    // A single release is available.
    env.block.time = env.block.time.plus_seconds(release_unit);

    // Query created batch.
    let msg = QueryMsg::Batch { id: 1 };
    let res: BatchResponse = from_json(query(deps.as_ref(), env.clone(),
↪   msg).unwrap()).unwrap();

    let lockup_end = mock_env().block.time.seconds();
    assert_eq!(
        BatchResponse {
            id: 1,
            amount: Uint128::new(100),
            amount_claimed: Uint128::zero(),
            amount_available_to_claim: Uint128::new(10),
            number_of_available_claims: Uint128::new(1),
            lockup_end,
            release_unit,
            release_amount: WithdrawalType::Amount(Uint128::new(10)),
```

```
            last_claimed_release_time: lockup_end,
        },
        res
    );

    // Claim batch.
    let msg = ExecuteMsg::Claim {
        number_of_claims: None,
        batch_id: 1,
    };

    let res = execute(deps.as_mut(), env, info, msg).unwrap();

    assert_eq!(
        Response::new()
            .add_message(BankMsg::Send {
                to_address: "/home/username1".to_string(), // Tokens are sent to
↪   the correct address
                amount: coins(10, "uusd")
            })
            .add_attribute("action", "claim")
            .add_attribute("amount", "10")
            .add_attribute("batch_id", "1")
            .add_attribute("amount_left", "90"),
        res
    );
    let lockup_end = mock_env().block.time.seconds();

    assert_eq!(
        Batch {
            amount: Uint128::new(100),
            amount_claimed: Uint128::new(10),
            lockup_end,
            release_unit: 10,
            release_amount: WithdrawalType::Amount(Uint128::new(10)),
            last_claimed_release_time: lockup_end + release_unit,
        },
        batches().load(deps.as_ref().storage, 1u64).unwrap()
    );
}
```

You can add this test to `contracts/finance/andromeda-vesting/src/testing/tests.rs` to verify yourself that it passes.

gjaldon

@J4X-98,

Your test case does not prove the issue does not exist. It is also incomplete since the username "username1" is not registered.

> I have adapted the testcases in the vesting module to this scenario, and the user exactly receives the tokens to the provided vfs path as you can see from the returned Send message

In your test, the tokens are sent to the address "/home/username1". When a VFS Path is resolved, it is supposed to load the address linked to a registered username and not to the actual path "/home/username1". That is exactly what this report points out as the issue.

For example:

1. Alice registers the username "imalice1" linked to the address "valid_address".
2. Alice provides the VFS Path "/home/imalice1" as recipient when claiming.
3. "valid_address" is supposed to receive the tokens but they are sent to the address "imalice1".

Does that clarify the issue?

cvetanovv

@gjaldon I spoke with the sponsor, and they confirmed that it is indeed a valid attack vector, however, it is a known issue from a previous audit. You can check out this audit - AND-42.

Because of this, I plan to invalidate the issue.

gjaldon

@cvetanovv AND-42 is different from this report. AND-42 states:

> Of note, the function resolve_home_path() in contracts/os/andromeda-vfs/src/state.rs appears to handle this possibility correctly to prevent spoofing, by first matching on a valid address before doing a username lookup:
>
> let user_address = match api.addr_validate(username_or_address) { Ok(addr) => addr, Err(_e) => USERS.load(storage, username_or_address)?, };

AND-42 states that those lines of code are correctly handling the prevention of spoofing. This report, on the other hand, points out that there is an issue in those same lines of code which leads to a vulnerability.

```
// @audit-issue if a username is also a valid address, then the address for the
↪  registered username can never be loaded
```

```
let user_address = match api.addr_validate(username_or_address) {
    Ok(addr) => addr,
    Err(_e) => USERS.load(storage, username_or_address)?,
};
resolve_path(storage, api, parts, user_address)
```

AND-42 points out that no validation prevents registering a username with an address that is not their own. However, that is no longer possible in the code in the audit scope.

This report is different since it is not about being able to register a username with an address that is not the registrant's. That is no longer possible for a user to do in the audited code. This report is about VFS Path resolution not returning the registered address for the username.

I explain the behavior here and the test case here actually confirms it.

WangSecurity

Based on this comment, isn't it Alice's mistake that she entered an incorrect data, firstly? And secondly, in that scenario the funds are still sent to her address, so the funds are not lost and she received them? I think entering one of your two addresses but receiving your funds on the another address of yours is not a loss?

And about the attack path in that comment. Firstly, you say "imalice1" is a username, but then you say it's an address, so why then not input the intended address "valid_address" initially? Moreover, it's not in the report and the two impacts explained in the report are not high severity impact, since the first leads to users not being able to claim the rewards, but the contract is upgradeable so the admin can retrieve the funds and the second about not receiving messages I believe is also medium.

WangSecurity

During the discussion in Discord with @gjaldon we reached an agreement that this indeed has to be Med, and will downgrade the severity to M in a couple of hours

Issue M-4: Calculating tax amount does not include taxes in `WasmMsg::Execute` messages

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/37

The protocol has acknowledged this issue.

Found by

g

Summary

`get_tax_amount()` fetches the amount of tax to be paid by totaling the amounts in the message transfers generated by the `on_funds_transfer()` hook. However, it only counts `CosmosMsg::Bank(BankMsg::Send)` messages and not the `WasmMsg::Execute` messages. This either fails the transfer or allows payors to skip paying taxes.

Vulnerability Detail

The issue lies in `get_tax_amount()`'s mapping logic over the messages generated by `on_funds_transfer()`.

ref: std/src/common/rates.rs::get_tax_amount()

```
pub fn get_tax_amount(
    msgs: &[SubMsg],
    base_amount: Uint128,
    remaining_amount_after_royalties: Uint128,
) -> Uint128 {
    let deducted_amount = base_amount - remaining_amount_after_royalties;
    msgs.iter()
        .map(|msg| {
            if let CosmosMsg::Bank(BankMsg::Send { amount, .. }) = &msg.msg {
                amount[0].amount
            } else {
                Uint128::zero()
            }
        })
```

Only taxes wrapped in `BankMsg::Send` are included. However, fee recipients with a `msg` set will get generated a `WasmMsg::Execute` message instead.

ref: std/src/amp/recipient.rs::generate_direct_msg()

```
Ok(match &self.msg {
    Some(message) => SubMsg::new(WasmMsg::Execute {
        contract_addr: resolved_addr.to_string(),
        msg: message.clone(),
        funds,
    }),
    None => SubMsg::new(CosmosMsg::Bank(BankMsg::Send {
        to_address: resolved_addr.to_string(),
        amount: funds,
    })),
})
```

Any taxes sent to recipients with a `msg` set will not be counted by `get_tax_amount()`.

Impact

The following effects will take place:

1. When at least one royalty is wrapped in a `WasmMsg::Execute`, `get_tax_amount()` will fail. Payment transfers for non-fungible ADO contracts *will fail* since non-fungible ADO contracts use `get_tax_amont()`.
2. When there are no royalties and at least one tax is in a `WasmMsg::Execute` msg, the tax wrapped in a `WasmMsg::Execute` message will be ignored. Payers can skip paying the `WasmMsg::Execute` taxes in these cases.

In effect, fee recipients with `msg` can not be used or it will lead to failures in non-fungible ADO contracts.

Code Snippet

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/common/rates.rs#L16-L33
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/amp/recipient.rs#L55-L59

Tool used

Manual Review

Recommendation

Consider modifying `get_tax_amount()` to count the taxes in `WasmMsg::Execute` messages.

Discussion

gjaldon

Escalate

This valid issue shows that not all taxes will be counted. Only taxes wrapped in `BankMsg::Send` will be counted even though some taxes will be wrapped in `WasmMsg::Execute`. This causes either valid payment transactions to fail or allows payers to skip paying some taxes.

std/src/common/rates.rs::get_tax_amount() is also in-scope.

sherlock-admin3

> Escalate
>
> This valid issue shows that not all taxes will be counted. Only taxes wrapped in `BankMsg::Send` will be counted even though some taxes will be wrapped in `WasmMsg::Execute`. This causes either valid payment transactions to fail or allows payers to skip paying some taxes.
>
> std/src/common/rates.rs::get_tax_amount() is also in-scope.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cvetanovv

Indeed, not all fees are counted when `get_tax_amount()` is called. So, I plan to accept the escalation and make the issue a valid Medium.

gjaldon

Thank you

WangSecurity

@gjaldon @MxAxM are there any duplicates?

WangSecurity

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- gjaldon: accepted

Issue M-5: when a validator is kicked out of the bonded validator set ,unstake funds will remain in the contract

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/41

The protocol has acknowledged this issue.

Found by

J4X_, Yashar, bin2chen

Summary

when a validator is kicked out of the bonded validator set, auto unbonding of all their delegations This portion of the funds will eventually be transferred to the contract and remain in the contract

Vulnerability Detail

in `andromeda-validator-staking` We can only get the stake funds back in the following ways

1. call execute_unstake(100)
    - UNSTAKING_QUEUE.push_back(100)
2. wait UnbondingTime , `x/staking` transfer funds to `andromeda-validator-staking`
3. call execute_withdraw_fund()
    - UNSTAKING_QUEUE.pop_front(100)
    - transfer 100 to `sender` from `andromeda-validator-staking`

but when a validator is kicked out of the bonded validator set, it will auto unbonding of all their delegations This doesn't go through the above process, it will come directly from `x/staking` transfer funds to `andromeda-validator-staking` https://github.com/cosmos/cosmos-sdk/tree/main/x/staking#validator when validator from `Bonded -> Unbonding`

Validator

..

- Unbonding: When a validator leaves the active set, either by choice or due to slashing, jailing or tombstoning, an unbonding of all their

> delegations begins. All delegations must then wait the UnbondingTime before their tokens are moved to their accounts from the BondedPool.

Impact

when a validator is kicked out of the bonded validator set This portion of the funds will eventually be transferred to the contract and remain in the contract

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L34

Tool used

Manual Review

Recommendation

in `execute_stake()` , call `ADOContract::default().add_withdrawable_token()`

Discussion

gjaldon

@cvetanovv @MxAxM @WangSecurity sorry to bother, but I just realized that this is Medium Severity because the loss of funds is not permanent. The `andromeda-validator-staking` contract is upgradable, so the contract can be upgraded so the locked funds can be recovered.

In CosmWasm, a contract is upgradeable as long as an admin is set during instantiation. This admin does not need to be set in CosmWasm contracts' code and can be set when instantiating contracts via the cosmos cli.

From the docs:

> However, if the contract is instantiated with an admin address, that account can execute migrations to update the contract's code and state.

There was also an incorrect assumption that a contract needs to define a `migrate` function in one of the duplicates. Not having a migrate function defined in the `andromeda-validator-staking` does not mean it is not upgradeable. The `admin` only needs to be set for the contract and only the new contract for `andromeda-validator-staking` needs to implement the `migrate` function.

From CosmWasmMigration docs:

> During the migration process, the migrate function defined in the new contract is executed and not the migrate function from the old code and therefore it is necessary for the new contract code to have a migrate function defined and properly exported as an entry_point:

Deploying ADO Contracts

ADO instances are deployed by the user or deployed as a bundle of components via the App ADO. The `andromeda-validator-staking` contract is an example of an ADO contract.

When deploying via App ADO

1. During instantiation of an App ADO, app components are added via `handle_add_app_component()`.
2. `handle_add_app_component()` generates instantiate messages. The admin for each component is set to the sender. This makes every ADO contract deployed by the App ADO *upgradeable*.

When user deploys ADO instances directly Contract owners and all admins are TRUSTED. So when users deploy ADO instances, they are contract owners or admins that are trusted to use values that will not cause any issues. This means they will set admin when deploying ADO contracts to make them upgradeable. Deploying instances is a function that is restricted to the owner/admin of that contract instance.

cvetanovv

Judging by this comment here, I'm planning to change the severity of #41 and #30 to Medium because the funds can be rescued.

@gjaldon is right that in `CosmWasm`, contracts can be set upgradeable by the Admin at the beginning. According to the rules, the Admin is expected to take the correct action. It might also mean setting the contracts upgradeable. This is missing as QA in the Readme, so we'll stick to assuming the Admin will make the right decisions.

cowboy0015

`withdraw_fund` function is now withdrawing the whole balance instead of manually calculating the rewards to claim. Due to various problems including old version compatibility, unable to set custom withdraw address, etc, there is a possibility where funds can be locked inside the contract. To overcome those unexpected locked funds, `withdraw_fund` is now working as an emergency as its purpose is to send rewards to the recipient set by the owner.

cowboy0015

Auto distribution did not happen for the jailed validator in e2e test. Going to upload the video demonstrating the testing process

Issue M-6: If WithdrawAddrEnabled = false, execute_claim() will fail

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/43

Found by

0xR360, J4X_, Kow, bin2chen

Summary

Currently, contracts that execute `execute_claim()` set `DistributionMsg::SetWithdrawAddress` first. If `WithdrawAddrEnabled = false`, the execution will not succeed and the `claim` will not be executed.

Vulnerability Detail

Currently the contract executes claims rewards by setting `DistributionMsg::SetWithdrawAddress` first.

```
fn execute_claim(
    ctx: ExecuteContext,
    validator: Option<Addr>,
    recipient: Option<AndrAddr>,
) -> Result<Response, ContractError> {
...
    let res = Response::new()
@>      .add_message(DistributionMsg::SetWithdrawAddress {
            address: recipient.to_string(),
        })
        .add_message(DistributionMsg::WithdrawDelegatorReward {
            validator: validator.to_string(),
        })
        .add_attribute("action", "validator-claim-reward")
        .add_attribute("recipient", recipient)
        .add_attribute("validator", validator.to_string());

    Ok(res)
}
```

If the configuration `WithdrawAddrEnabled` is changed to `false`, setting `DistributionMsg::SetWithdrawAddress` will fail! This will prevent the execution of the `claim` https://github.com/cosmos/cosmos-sdk/tree/main/x/distribution#msgsetwithdrawaddress

MsgSetWithdrawAddress

By default, the withdraw address is the delegator address. To change its withdraw address, a delegator must send a MsgSetWithdrawAddress message. Changing the withdraw address is possible **only if the parameter WithdrawAddrEnabled is set to true.**

```
func (k Keeper) SetWithdrawAddr(ctx context.Context, delegatorAddr
↪  sdk.AccAddress, withdrawAddr sdk.AccAddress) error
 if k.blockedAddrs[withdrawAddr.String()] {
  fail with "`{withdrawAddr}` is not allowed to receive external funds"
 }

 if !k.GetWithdrawAddrEnabled(ctx) {
  fail with `ErrSetWithdrawAddrDisabled`
 }

 k.SetDelegatorWithdrawAddr(ctx, delegatorAddr, withdrawAddr)
```

Impact

can't claim reward

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L231

Tool used

Manual Review

Recommendation

when set `DistributionMsg::SetWithdrawAddress` , `SubMsg` using `ReplyOn.Error`, which is ignored when this message returns an error, to avoid the whole `execute_claim` from failing!

Discussion

neko-nyaa

Escalate

#15, #67, #68 are dupes of this issue. This family of issues shows various impacts when `withdrawAddrEnabled` is set to false, and the root cause of the revert is the message `MsgSetWithdrawAddress` failing.

sherlock-admin3

> Escalate
>
> #15, #67, #68 are dupes of this issue. This family of issues shows various impacts when `withdrawAddrEnabled` is set to false, and the root cause of the revert is the message `MsgSetWithdrawAddress` failing.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cvetanovv

I agree with the escalation.

The root cause of the issue is when `withdrawAddrEnabled` is set to false. That's why I plan to duplicate all similar issues related to `WithdrawAddrEnabled = false`.

I am planning to accept the escalation and duplicate #15, #67, #68, and #52 with this issue.

WangSecurity

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- neko-nyaa: accepted

cowboy0015

`withdraw_fund` function is now working as an emergency for withdrawing funds (including locked funds due to unexpected cases)

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/559

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/559
This PR canceled the execution of `SetWithdrawAddress`. After the reward is

triggered, get back all the balance in the contract via `execute_withdraw_fund()`
Fixed this issue

Issue M-7: if Slash Validator occurs, UNSTAKING_QUEUE's unstake amount will not be accurate

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/44

Found by

J4X_, Kow, bin2chen

Summary

`UNSTAKING_QUEUE` holds `UnbondingDelegationEntry.initial_balance`. If a `Slash Validator` occurs, which actually unstake amount is `UnbondingDelegationEntry.balance`, this value will be smaller than `UnbondingDelegationEntry.initial_balance` which will cause `execute_withdraw_fund()` to fail.

Vulnerability Detail

in `andromeda-validator-staking` We can get the stake funds back in the following ways

1. call execute_unstake(100) , UnbondingDelegationEntry.initial_balance = 100 ,
    - UNSTAKING_QUEUE.push_back(100)
2. wait UnbondingTime , `x/staking` transfer funds (UnbondingDelegationEntry.balance=100) to `andromeda-validator-staking`
3. call execute_withdraw_fund()
    - UNSTAKING_QUEUE.pop_front(100)
    - transfer 100 to `sender` from `andromeda-validator-staking`

If it doesn't happen `Slash Validator` balance == initial_balance https://github.com/cosmos/cosmos-sdk/blob/207b30262fc4ae62cb6fc7c2f6df1dfaf7bc1c4d/x/staking/proto/cosmos/staking/v1beta1/staking.proto#L238

```
message UnbondingDelegationEntry {
...
  google.protobuf.Timestamp completion_time = 2
      [(gogoproto.nullable) = false, (amino.dont_omitempty) = true,
↪   (gogoproto.stdtime) = true];
  // initial_balance defines the tokens initially scheduled to receive at
↪   completion.
```

```
@>string initial_balance = 3 [
    (cosmos_proto.scalar)  = "cosmos.Int",
    (gogoproto.customtype) = "cosmossdk.io/math.Int",
    (gogoproto.nullable)   = false
  ];
  // balance defines the tokens to receive at completion.
@>string balance = 4 [
    (cosmos_proto.scalar)  = "cosmos.Int",
    (gogoproto.customtype) = "cosmossdk.io/math.Int",
    (gogoproto.nullable)   = false
  ];
...
}
```

However, happen `Slash Validator`, the actual funds received will be less than the value recorded in the 'UNSTAKING_QUEUE' record. https://github.com/cosmos/cosmos-sdk/tree/main/x/staking#slash-unbonding-delegation

> ### Slash Unbonding Delegation
>
> When a validator is slashed, so are those unbonding delegations from the validator that began unbonding after the time of the infraction. Every entry in every unbonding delegation from the validator is slashed by slashFactor. The amount slashed is calculated from the InitialBalance of the delegation and is capped to prevent a resulting negative balance. Completed (or mature) unbondings are not slashed.

Impact

If a `Slash Validator` occurs, the value of the `UNSTAKING_QUEUE` record will be less than the actual value received Resulting in

1. failure due to insufficient balance
2. blocking the normal queue behind

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L333

Tool used

Manual Review

Recommendation

when the balance is insufficient, only the balance is returned

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/551

cowboy0015

`withdraw_fund` function is now working as an emergency for withdrawing funds (including locked rewards and funds to handle unexpected cases)

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/551
This PR removes the logic of `execute_withdraw_fund()` to calculate the quantity, and directly withdraws all the balance, solving the problem of `Slash` quantity difference

Issue M-8: is_permissioned() may underflow

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/46

Found by

Kow, bin2chen, g

Summary

`is_permissioned()`, if `permissioned_action` is changed from `true` to `false`. Users who were previously set to `Permission::Limited` may be `underflowed`.

Vulnerability Detail

`is_permissioned()` is used to implement the permission check, which is implemented as follows

```
    pub fn is_permissioned(
...
        let permission = Self::get_permission(store, action_string.clone(),
↪   actor_string.clone())?;
@>      let permissioned_action = self
            .permissioned_actions
            .may_load(store, action_string.clone())?
            .unwrap_or(false);
        match permission {
            Some(mut permission) => {
                ensure!(
                    permission.is_permissioned(&env, permissioned_action),
                    ContractError::Unauthorized {}
                );

                // Consume a use for a limited permission
@>              if let Permission::Limited { .. } = permission {
@>                  permission.consume_use()?;
                    permissions().save(
                        store,
                        (action_string.clone() + actor_string.as_str()).as_str(),
                        &PermissionInfo {
                            action: action_string,
                            actor: actor_string,
                            permission,
                        },
                    )?;
```

```
                }

                Ok(())
            }
```

From the above code, we know that if the user has `Permission::Limited` , it will be reduced by 1 regardless of whether the `action` needs permission or not. This `permission.consume_use()` can be `underflow` in the following cases

1. `action1` needs permission at first, i.e. `permissioned_action=true`
2. the administrator grants `alice` `Permission::Limited` permission and `Limited.uses = 3`.
3. alice used up 3 times, `Limited.uses = 0`
4. the administrator adjusts the `action1` permissions configuration to not require permissions, i.e. `permissioned_action=false`
5. at this point `alice` wants to execute `action1`, but `is_permissioned(Alice,action1)` will revert, because `permission.consume_use()` will be executed, resulting in underflow (Limited.uses ==0 ,Limited.uses-=1)

Impact

`is_permissioned()` may underflow, causing the permission check to fail and the corresponding action to can't be executed

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/permissioning.rs#L71

Tool used

Manual Review

Recommendation

only `permissioned_action==true`, then execute `permission.consume_use()`.

```
    pub fn is_permissioned(
...
            Some(mut permission) => {
                ensure!(
```

```
                    permission.is_permissioned(&env, permissioned_action),
                    ContractError::Unauthorized {}
                );

                // Consume a use for a limited permission
+           if permissioned_action {
                if let Permission::Limited { .. } = permission {
                    permission.consume_use();
                    permissions().save(
                        store,
                        (action_string.clone() + actor_string.as_str()).as_str(),
                        &PermissionInfo {
                            action: action_string,
                            actor: actor_string,
                            permission,
                        },
                    )?;
                }
+           }
                Ok(())
            }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/553

bin2chen66

fix-reviews note: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/46 PR changed so that `permission.consume_use()` is executed only if `permissioned_action==true`. Also changed so that `consume_use()` no longer returns an error. (Secured by `local_permission.is_permissioned()`.) Fixed this issue

Issue M-9: is_permissioned() It doesn't make sense to have permissions by default after Blacklisted expires.

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/47

Found by

bin2chen, g

Summary

in `is_permissioned()`, returns `true` if `Permission::Blacklisted` has expired it is not correct

Vulnerability Detail

in `is_permissioned()` to determine if a permission is granted.

```
    pub fn is_permissioned(&self, env: &Env, strict: bool) -> bool {
        match self {
            Self::Blacklisted(expiration) => {
                if let Some(expiration) = expiration {
                    if expiration.is_expired(&env.block) {
@>                      return true;
                    }
                }
                false
            }
            Self::Limited { expiration, uses } => {
                if let Some(expiration) = expiration {
                    if expiration.is_expired(&env.block) {
@>                      return !strict;
                    }
                }
                if *uses == 0 {
                    return !strict;
                }
                true
            }
            Self::Whitelisted(expiration) => {
                if let Some(expiration) = expiration {
                    if expiration.is_expired(&env.block) {
                        return !strict;
                    }
```

```
                }
                true
            }
        }
    }
```

The current implementation returns `true` if the blacklist has expired, regardless of `strict`. The following scenarios are problematic

1. `action1` doesn't need permission at the beginning, i.e.: strict = false
2. the administrator has blacklisted `alice` for 1 month, i.e.: alice has Permission::Blacklisted
3. after some time (> 1 month)
4. the administrator changes the permissions configuration of `action1` to `action1` requires permissions, i.e.: strict = true
5. at this point `is_permissioned(alice)` returns true, and `alice` becomes permitted by default, which is not correct!

It is reasonable to return `!strict` when it expires, just like `Limited` and `Whitelisted`.

Impact

`Permission::Blacklisted` expires and returns `true`, causing users to have permissions that shouldn't have them.

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_base/permissioning.rs#L55

Tool used

Manual Review

Recommendation

```
    pub fn is_permissioned(&self, env: &Env, strict: bool) -> bool {
        match self {
            Self::Blacklisted(expiration) => {
                if let Some(expiration) = expiration {
                    if expiration.is_expired(&env.block) {
-                       return true;
+                       return  !strict;
```

```
                }
            }
            false
        }
        Self::Limited { expiration, uses } => {
            if let Some(expiration) = expiration {
                if expiration.is_expired(&env.block) {
                    return !strict;
                }
            }
            if *uses == 0 {
                return !strict;
            }
            true
        }
```

Discussion

J4X-98

Escalate

This is intended behavior. The documentation (code comments) describes the intended behavior as follows:

```
/// An enum to represent a user's permission for an action
///
/// - **Blacklisted** - The user cannot perform the action until after the
↪  provided expiration
/// - **Limited** - The user can perform the action while uses are remaining and
↪  before the provided expiration **for a permissioned action**
/// - **Whitelisted** - The user can perform the action until the provided
↪  expiration **for a permissioned action**
///
/// Expiration defaults to `Never` if not provided
#[cw_serde]
pub enum Permission {
    Blacklisted(Option<Expiration>),
    Limited {
        expiration: Option<Expiration>,
        uses: u32,
    },
    Whitelisted(Option<Expiration>),
}
```

The documentation clearly states that the intended behavior for a blacklisted user

is "The user cannot perform the action until after the provided expiration". So the user should be able to regain possibility to take actions after expiry which is the whole "vulnerability" described by this group of issues.

sherlock-admin3

> Escalate
>
> This is intended behavior. The documentation (code comments) describes the intended behavior as follows:
>
> ```
> /// An enum to represent a user's permission for an action
> ///
> /// - **Blacklisted** - The user cannot perform the action until after the
> ↪  provided expiration
> /// - **Limited** - The user can perform the action while uses are
> ↪  remaining and before the provided expiration **for a permissioned
> ↪  action**
> /// - **Whitelisted** - The user can perform the action until the provided
> ↪  expiration **for a permissioned action**
> ///
> /// Expiration defaults to `Never` if not provided
> #[cw_serde]
> pub enum Permission {
>     Blacklisted(Option<Expiration>),
>     Limited {
>         expiration: Option<Expiration>,
>         uses: u32,
>     },
>     Whitelisted(Option<Expiration>),
> }
> ```
>
> The documentation clearly states that the intended behavior for a blacklisted user is "The user cannot perform the action until after the provided expiration". So the user should be able to regain possibility to take actions after expiry which is the whole "vulnerability" described by this group of issues.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

cvetanovv

I don't see a problem here. NatSpec comments highlight that when a user is blacklisted, it is expected after a while to be able to regain the possibility to take

action after the restriction expires.

Planning to accept the escalation and invalidate the issue.

bin2chen66

> /// - **Blacklisted** - The user cannot perform the action until after the provided expiration

The way I understand this sentence is.

Suppose `action` is permissionless (`strict=false`) and can be executed by anyone. But the administrator can restrict someone from performing it, for example by blacklisting `alice` so that `alice` cannot perform the `action` until after the provided expiration.

Example: Suppose `withdrawal` doesn't need permissions, but the administrator finds out that `alice` is acting maliciously and blacklists `alice` for 1 month, so that `alice` can't withdraw funds Within this month.

But the problem is, if later the administrator decides to make the action `withdrawal` require permissions by default, i.e.: change `strict=true`.

But, since `alice` was previously blackballed and expiration, `alice` now becomes permissioned instead (Current code returns true.). This does not make sense.

So the blacklisting mechanism, like the other two `Limited` and `Whitelisted`, should expire as if it was not set, and go back to the way it was, i.e. `!strict`, instead of returning `true`.

J4X-98

Hey @bin2chen66 ,

There is no default limitation state. As you can see on both the other roles, they also return true if they are not limited by expiration/uses. So effectively blacklist goes to the "default" state after expiration.

bin2chen66

hey @J4X-98

sorry , I didn't get what you meant.

if strict=true

current code: Limited expiration return !strict = false (No permission) Whitelisted expiration return !strict = false (No permission) Blacklisted expiration return true (have permission) --> This is the key point, if it expires it should be the same as if it wasn't set up, and doesn't have permissions

J4X-98

Hey @bin2chen66 ,

All the functions only return '!strict' if the user is not allowed by either their whitelist having expired or them not having any uses left. Otherwise the 2 others also return true by default. So there seem to only be 2 intended states:

1. user is not allowed -> return '!strict'
2. User is allowed -> return 'true'

And the implementation implements this as intended.

The mediation you recommend would lead to the user still not having permissions after the blacklist expired, leading to a conflict with the description of the functionality.

bin2chen66

hey @J4X-98 Returning `true` is only good for the `strict=false` case In the case of `strict=true`, returning `true` would be a serious security issue That's what this issues mentions.and the sponsor seems to have confirmed this question: Labels :“Sponsor Confirmed "

J4X-98

Hey @bin2chen66 ,

What you recommend is returning `!strict`, which is the same behavior as when the blacklisted has not expired. This collides with the intended behavior of the user, regaining the possibility of action. The current behavior is

1. Blacklist is not expired:
   - `strict` -> return `false`
   - `!strict`-> return `true`
2. Blacklist is expired:
   - `strict`-> return `true`
   - `!strict`-> return `true`

So as a result, the user regains access to strict functions when the blacklist has expired, which adheres to the behavior described by the documentation.

Your proposed mitigation would result in the function always returning `!strict`, doesn't matter if the blacklist has expired or not, which would make the whole expiration process redundant and would not adhere to the documentation. The function is currently implemented in the way it is documented and does not lead to any vulnerabilities. You are describing intended behavior as a vulnerability, and your mitigation would break intended behavior.

gjaldon

The statement below from this comment is incorrect:

> So effectively blacklist goes to the "default" state after expiration.

The "default" state is no access and not upgraded access.

Also, the issue is that when the blacklist expires, the user's permission is upgraded to the level of a non-expired whitelisted user. A non-expired whitelisted user has access even to strict permissioned actions. It does not make sense for a blacklisted user to become whitelisted on expiration. In the NFT ADO Contract, this would mean the expired blacklisted user can arbitrarily mint NFTs like the minter role.

J4X-98

Hey, what you are missing is that a user can only be in two states as the current implementation works. These are either returning `!strict` or `true` all the time.

"The "default" state is no access and not upgraded access." -> The default state (which does not exist) you describe always returns false. So in your opinion, while a user is blacklisted, he should be able to access some functionalities that are not strict. Once the blacklist expires, the user should not be able to access anything anymore. This makes no sense and clashes with the documentation of the user regaining access once the blacklist has expired.

It might be a future improvement to change the whole system to add more states that allow for more detailed control of the user's permissions. But for the current states, the contract can choose from there are only 2, and the implementation correctly uses one of those (the restricted one) while the user is blacklisted and the other one (no restrictions) once the blacklist has expired. This confirms the documentation, so there is no vulnerability; it is just a recommendation for future improvement.

cvetanovv

I agree with @J4X-98 comments that this is expected behavior. My early decision remains to accept the escalation and invalidate the issue.

gjaldon

@cvetanovv although the comment is correct, it does not completely describe the current behavior of blacklisting.

> **Blacklisted** - The user cannot perform the action until after the provided expiration

After the blacklisting expires, the user gains whitelisted privileges. It's unclear just based on the comment if the dev team intends to increase a user's permissions to greater than a regular user once their blacklist expires.

Also, isn't the dev team fixing the issue? Doesn't that confirm it?

J4X-98

As described in multiple of the messages above there is no default behavior. There only is returning !strict or true and those are correctly used by the contract.

Repeating the same invalid argument countless times does not make it more true. The judge has already twice confirmed that this issue is invalid, I don't see a reason to continue this discussion.

gjaldon

Hi @J4X-98. Thank you for your opinions.

I presented 2 new arguments which are:

- The comment does not completely describe the current behavior of blacklisting
- The dev team is fixing the issues, which seems to confirm it

The only way to know the intended functionality is to ask the dev team, especially since it is tagged as `Will Fix`.

J4X-98

Hey, I think the comment pretty clearly describes the intended behavior "Blacklisted - The user cannot perform the action until after the provided expiration". The user should not be able to access permissioned functions while blacklisted and should be able to do so once the blacklist has expired. This is exactly what happens in the code.

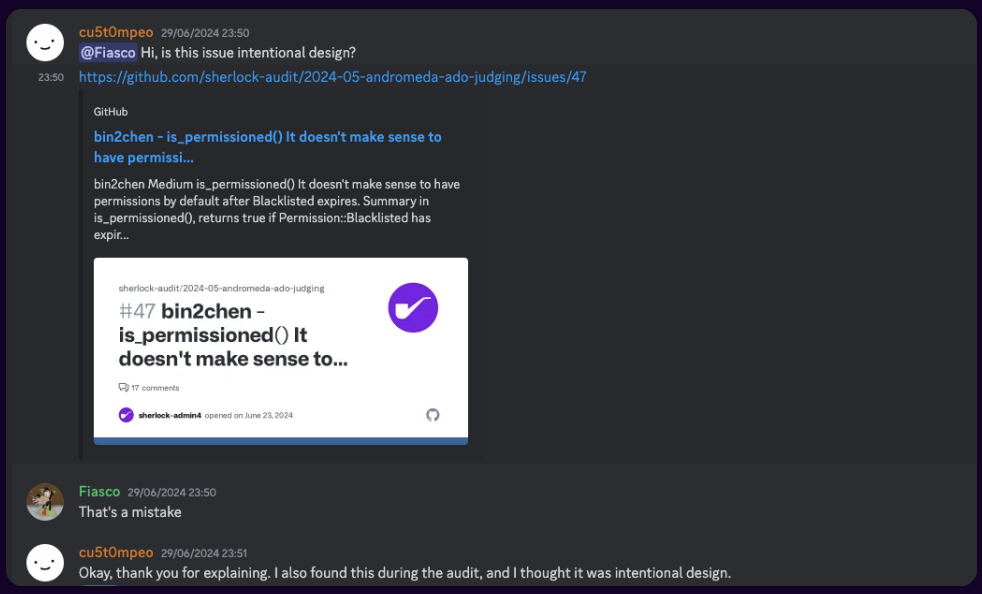
gjaldon

But does it make sense for a previously blacklisted user's privileges to be escalated to the level of a whitelisted user? The comment does not mention anything about that.

It looks like an unintended privilege escalation.

cu5t0mPeo

This seems to be a mistake. I asked the sponsor, but it was after the competition.



gjaldon

I really appreciate your input, @cu5t0mPeo.

The sponsor has confirmed this as an issue @cvetanovv. It is an unintended privilege escalation. Sorry for all the discussion. Hope that's enough to validate the issue and reject the escalation.

cvetanovv

After the sponsor confirms that this is not an intended design, I will reject the escalation and leave the issue as is.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- J4X-98: rejected

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/andromedaprotocol/andromeda-core/pull/553

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/553 This PR modifies the blacklist expiration to return `!strict` as expected Fixed this

issue

Issue M-10: execute_claim() possible loss of accuracy or even inability to retrieve funds

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/49

Found by

bin2chen

Summary

`claim_batch()` dividing and then multiplying may result in loss of precision, and in the worst case may not retrieve funds

Vulnerability Detail

`claim_batch()` is used to calculate the amount of money that can be retrieved.

```
fn claim_batch(
    querier: &QuerierWrapper,
    env: &Env,
    batch: &mut Batch,
    config: &Config,
    number_of_claims: Option<u64>,
) -> Result<Uint128, ContractError> {
    let current_time = env.block.time.seconds();
    ensure!(
        batch.lockup_end <= current_time,
        ContractError::FundsAreLocked {}
    );
@>  let amount_per_claim = batch.release_amount.get_amount(batch.amount)?;

    let total_amount = AssetInfo::native(config.denom.to_owned())
        .query_balance(querier, env.contract.address.to_owned())?;

    let elapsed_time = current_time - batch.last_claimed_release_time;
    let num_available_claims = elapsed_time / batch.release_unit;

    let number_of_claims = cmp::min(
        number_of_claims.unwrap_or(num_available_claims),
        num_available_claims,
    );

@>  let amount_to_send = amount_per_claim * Uint128::from(number_of_claims);
```

```
    let amount_available = cmp::min(batch.amount - batch.amount_claimed,
↪   total_amount);

    let amount_to_send = cmp::min(amount_to_send, amount_available);

    // We dont want to update the last_claim_time when there are no funds to
↪   claim.
    if !amount_to_send.is_zero() {
        batch.amount_claimed += amount_to_send;
        batch.last_claimed_release_time += number_of_claims * batch.release_unit;
    }

    Ok(amount_to_send)
}
```

From the code above we know that the calculation is

1. amount_per_claim = batch.amount * release_amount (release_amount is Decimal, Precision = 1e18)
2. number_of_claims = elapsed_time / batch.release_unit
3. amount_to_send = amount_per_claim * number_of_claims

i.e.: `amount_to_send = (batch.amount * release_amount / 1e18) * number_of_claims`

Since it is dividing and then multiplying, it may lead to loss of precision, even amount_per_claim = 0 Assumption: it takes 5 years to claim 1 btc, batch.amount = 1e8 btc release_unit = 1 second release_amount = 1e8 * 1e18 / 157680000(seconds) / 1e8 = 6341958396 (6341958396 percent per second, precision 1e18)

Following the existing formula, divide and multiply.

amount_to_send = (1e8 * 6341958396 / 1e18) * 157680000(seconds) = 0

If modified to multiply before dividing:

amount_to_send = (1e8 * 6341958396 * 157680000(seconds) / 1e18 = 99999999

Impact

`claim_batch()` dividing and then multiplying may result in a loss of precision, and in the worst case it may not be possible to retrieve the funds

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/contracts/finance/andromeda-vesting/src/contract.rs#L462

Tool used

Manual Review

Recommendation

In case of `WithdrawalType::Percentage`, multiply then divide Example: ‘batch.amount * number_of_claims * release_amount / 1e18

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/andromedaprotocol/andromeda-core/pull/555

bin2chen66

https://github.com/andromedaprotocol/andromeda-core/pull/555 This PR modifies `get_amount()` to return Decimal, which improves precision and solves the problem of precision loss. At the same time, when creating `execute_create_batch()`, it checks `!release_amount.get_amount(funds.amount)? .is_zero()` to avoid the value being too low Fixed this issue

Issue M-11: Attacker can freeze users first rewards

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/50

The protocol has acknowledged this issue.

Found by

J4X_, Kow, bin2chen

Summary

The `andromeda-validator-staking` contract has a vulnerability related to the staking rewards withdrawal process. If the withdrawal address is not set correctly, rewards can be unintentionally distributed to the contract itself, causing them to become stuck. This can be exploited by an attacker who can front-run the owner's first claim transaction and cause the rewards to be irretrievably sent to the contract. The impact of this issue is the loss of all rewards accrued before un-bonding.

Vulnerability Detail

The `andromeda-validator-staking` allows the owner to stake tokens to a chosen validator. The delegation will then generate staking rewards. To allow the contract owner to withdraw these rewards, the `execute_claim()` function is implemented. To be able to claim the tokens correctly, two messages have to be sent:

1. `DistributionMsg::SetWithdrawAddress` - sets the address to withdraw to the recipients address
2. `DistributionMsg::WithdrawDelegatorReward` - withdraws the rewards

If the first message is not sent, the withdrawal address is set to the delegator which in our case is the `andromeda-validator-staking` contract. When the owner calls the `execute_claim()` function directly, this leads to no issues, as the two functions are called correctly.

The issues occur as there are multiple other scenarios why rewards will be distributed besides the direct call via `DistributionMsg::WithdrawDelegatorReward`. Rewards will be distributed if a user's stake increases. The other option is that an un-bonding occurs, in which case rewards are also distributed. In total there are four scenarios why rewards will be distributed without a call to `DistributionMsg::WithdrawDelegatorReward`:

1. Owner stakes or un-stakes
2. Validator is jailed/tombstoned

3. The validator leaves the set willingly
4. Attacker stakes on behalf of the owner (which works as `execute_stake()` is not restricted)

For this case, we will only consider 2., 3., and 4. as 1. would require some owner wrongdoing. If one of these cases occurs before the owner has claimed rewards for the first time, the rewards will be sent directly to the `andromeda-validator-staking` contract. The tokens will become stuck there as the contract does not implement a way to retrieve/re-stake funds.

For the fourth scenario, a malicious attacker can intentionally abuse this and wait until the owner tries to call `execute_claim()` for the first time. When he sees the tx, he front-runs it and stakes 1 token on behalf of the owner, which will result in the owner's rewards getting sent to the `andromeda-validator-staking` contract and getting stuck. As the `SetWithdrawAddress` message will only be sent afterward, the recipient is still the `andromeda-validator-staking` contract.

Impact

The issue results in all rewards accruing before the un-bonding getting stuck in the contract and being effectively lost. As the `andromeda-validator-staking` contract does not implement a `migrate()` function, the funds can not be rescued by upgrading the contract.

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L186-L242

Tool used

Manual Review

Recommendation

We recommend mitigating this issue by setting a `withdrawal_address` when calling `instantiate()`. This withdrawal address should then be set on each call to `execute_stake()`, `execute_unstake()`, and `execute_withdraw_fund()`. This way, tokens can never be lost due to an unset withdrawal address.

Discussion

gjaldon

Escalate

This issue is invalid because the following statement is incorrect.

> The tokens will become stuck there as the contract does not implement a way to retrieve/re-stake funds.

There is functionality for the owner to withdraw funds from the Staking contract through AndromedaMsg::Withdraw. Below are the links to the relevant code that shows how the `AndromedaMsg::Withdraw` message is handled.

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L88
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/packages/std/src/ado_contract/execute.rs#L61-L64
- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/withdraw.rs#L32

There is, however, a separate and unrelated issue in `execute_withdraw()` reported here that prevents withdrawal of funds. The loss of funds is only possible if there is no Withdrawal implementation or if the withdrawal implementation does not work. The report incorrectly states there is no withdrawal functionality and does not mention any issue with withdrawal.

sherlock-admin3

> Escalate
>
> This issue is invalid because the following statement is incorrect.
>
> > The tokens will become stuck there as the contract does not implement a way to retrieve/re-stake funds.
>
> There is functionality for the owner to withdraw funds from the Staking contract through AndromedaMsg::Withdraw. Below are the links to the relevant code that shows how the `AndromedaMsg::Withdraw` message is handled.
>
> - https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L88
> - https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/packages/std/src/ado_contract/execute.rs#L61-L64

- https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/main/andromeda-core/packages/std/src/ado_contract/withdraw.rs#L32

There is, however, a separate and unrelated issue in `execute_withdraw()` reported here that prevents withdrawal of funds. The loss of funds is only possible if there is no Withdrawal implementation or if the withdrawal implementation does not work. The report incorrectly states there is no withdrawal functionality and does not mention any issue with withdrawal.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

J4X-98

Hey @gjaldon,

your escalation is based on you stating that the sentence "The tokens will become stuck there as the contract does not implement a way to retrieve/re-stake funds." is wrong. A few sentences below, you link your own issue that describes withdrawals as not being possible due to a missing implementation.

So, effectively, the issue is valid, and the mitigation works and mitigates the problem. The only critique you have is that you would have liked me to go more into detail on why withdrawals do not work. I can't see in any way how this would lead to the issue being invalidated, but I'm looking forward to hearing the judge's opinion.

gjaldon

Hello @J4X-98,

In my opinion, the finding is incomplete because it assumes that there is no way to withdraw funds even though the functionality exists. This issue would also be addressed by a fix in withdrawing of funds.

The issue is only valid if there is no way to withdraw funds, since there would be no impact if funds are withdrawable. This issue is only valid in combination with the issue I reported. Since it fails to identify the other root cause that would lead to loss of funds, I think this is invalid.

J4X-98

Hey,

The functionality of both contracts is clearly intended to distribute rewards to a preset claimant address, not let them be sent to the contract. The code clearly

tries to set a withdraw address before each time rewards are distributed. This can also be seen in the vesting functionality. However the developer forgot this for some cases in the staking module.

Withdrawing these rewards, which were incorrectly distributed to the contract, via an additional function, would be a emergency rescue, but not the intended behavior. Additionally this does not work, so there is no way to rescue these tokens, which is exactly what I described in the issue.

You're escalation is based on me not describing (the not working) emergency fix in as much detail as you would like to, which does in no way invalidate the issue. The issue is that the withdraw address is not correctly set at the start, which would lead to the intended behavior.

cvetanovv

After all, as far as I understand, the rewards can't be rescued and are lost. But I don't think it is High severity. The losses are not significant, and settling the withdrawal address will fix the problem.

The rule for High:

> Definite loss of funds without (extensive) limitations of external conditions.

I plan to reject the escalation to invalidate the issue, but I'll make it Medium severity.

J4X-98

Hey @cvetanovv ,

Setting the withdrawal address will only protect from the future loss of rewards but will not recover the originally lost funds. This will clearly lead to a loss of fund which would classify this as a high.

gjaldon

@cvetanovv the rewards can't be rescued and are lost only because the `WithdrawFunds` functionality is not working. My report describes the issue of funds not being withdrawable but it is not mentioned here. Instead, it makes an incorrect statement:

> The tokens will become stuck there as the contract does not implement a way to retrieve/re-stake funds.

The contract does implement a way to retrieve funds but it just so happens that it isn't working which is the point of the issue I reported.

Should a report be valid if its described impact (loss of funds in this case) is only possible combined with another bug that the reporter and report failed to identify? The impact is only possible when combined with a bug reported by someone else.

J4X-98

As commented before by me as an answer to you stating the same claim as now, I stated that withdrawal is not possible. You are trying to invalidate this issue based on the reason that I don't go into enough detail on why withdrawals do not work. This in now way contradicts the original issue and is your personal preference.

cvetanovv

My decision remains the same. To reject the escalation and downgrade the severity to Medium. This issue and its duplicates show how it is possible to have a loss of rewards. But because the loss is limited and the withdrawal address setting can stop the loss, I believe it is Medium severity.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- gjaldon: rejected

cowboy0015

`withdraw_funds` function is now working as an emergency to handle frozen rewards in the contract.

cowboy0015

Auto distribution did not happen for the jailed validator in e2e test. Going to upload the video demonstrating the testing process

Issue M-12: Changes of the `UnbondingTime` are not accounted for

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/54

Found by

J4X_

Summary

The `andromeda-validator-staking` contract allows the owner to stake and unstake tokens, adding unstaking entries to an `UNSTAKING_QUEUE`. The unstaking process is dependent on the `UnbondingTime` parameter of the chain, which can be changed by governance. If the `UnbondingTime` is reduced while unstakings are already queued, it can result in a denial-of-service (DoS) situation where newer entries cannot be withdrawn until older entries expire. This could lead to tokens being stuck in the contract for a significant period.

Vulnerability Detail

The `andromeda-validator-staking` contract implements a way to allow the owner of the contract to stake tokens. When the owner of the contract wants to unstake tokens again he can do this by calling the `execute_unstake()` function. The contract will then, on response from the staking module, add an entry to the `UNSTAKING_QUEUE`.

```
pub fn on_validator_unstake(deps: DepsMut, msg: Reply) -> Result<Response,
↪  ContractError> {
    let attributes = &msg.result.unwrap().events[0].attributes;
    let mut fund = Coin::default();
    let mut payout_at = Timestamp::default();
    for attr in attributes {
        if attr.key == "amount" {
            fund = Coin::from_str(&attr.value).unwrap();
        } else if attr.key == "completion_time" {
            let completion_time =
↪  DateTime::parse_from_rfc3339(&attr.value).unwrap();
            let seconds = completion_time.timestamp() as u64;
            let nanos = completion_time.timestamp_subsec_nanos() as u64;
            payout_at = Timestamp::from_seconds(seconds);
            payout_at = payout_at.plus_nanos(nanos);
        }
    }
```

```
    UNSTAKING_QUEUE.push_back(deps.storage, &UnstakingTokens { fund, payout_at
↪   })?;

    Ok(Response::default())
}
```

Once the completion time has passed, the user can now call `execute_withdraw_fund()` to withdraw the funds. The function loops over the `UNSTAKING_QUEUE` and adds all unstakings until it fins one that has not expired. Afterwards all of the found expired unstakings are payed out to the user.

```
loop {
    match UNSTAKING_QUEUE.front(deps.storage).unwrap() {
        Some(UnstakingTokens { payout_at, .. }) if payout_at <= env.block.time
↪   => {
            if let Some(UnstakingTokens { fund, .. }) =
                UNSTAKING_QUEUE.pop_front(deps.storage)?
            {
                funds.push(fund)
            }
        }
        _ => break,
    }
}
```

The completion time that the loop is based upon is the UnbondingTime which is one of the params of the x/staking module. The governance can change this parameter at any time via a MsgUpdateParamsmessage.

The issue occurs as the loop expects that for each item in it $item_n.completionTime >= item_{n-1}.completionTime$. Unfortunately this is not the case if the governance reduces the `UnbondingTime`parameter while unstakings are already queued. In that case it can occur that $item_n.completionTime < item_{n-1}.completionTime$. This will result in $item_n$ being unable to withdraw until $item_{n-1}$ has expired.

This DOS can be anwhere in the range from $0 - UnbondingTime$. For most of the targeted chains the `UnbondingTime` is set to 21 days (Incetive, Archway and Terra). While it is not a reasonable scenario that the `UnbondingTime` will be reduced to 0, a deduction of 1-2 weeks is possible. The default value of the `UnbondingTime` is only 3 days, and some other chains also user 1-2 weeks shorter unbonding times:

- Axelar - 7 days
- Bitcanna - 14 days
- Stargaze - 14 days

Based on this we can assume that a reduction by 1-2 weeks is a possible scenario. As the protocol will not only be deployed on andromeda's own chain but also on multiple other cosmos chains, the Andromeda Governance has no possibility to prevent such a change if it occurs.

Impact

The issue results in tokens getting stuck in the contract until the messages before them expire. This can result in a DOS of 1-2 weeks depending on the change of the `UnbondingTime`.

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-validator-staking/src/contract.rs#L256-L267

Tool used

Manual Review

Recommendation

We recommend adapting the loop to not break if the `payout_at > env.block.time`. Instead in that case it should just do nothing and go on to the next element.

```
loop {
    match UNSTAKING_QUEUE.front(deps.storage).unwrap() {
        Some(UnstakingTokens { payout_at, .. }) if payout_at <= env.block.time
↪   => {
            if let Some(UnstakingTokens { fund, .. }) =
                UNSTAKING_QUEUE.pop_front(deps.storage)?
            {
                funds.push(fund)
            }
        }
        _ => continue,
    }
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/551

cowboy0015

Updated the logic to removed all the expired unstaking requests. Thank you

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/551
This PR removes the logic of `execute_withdraw_fund()` to calculate the quantity, and directly withdraws all the balance, which solves the problem

Issue M-13: Staked tokens will get stuck after claim

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/57

Found by

J4X_, bin2chen

Summary

The Andromeda protocol includes a vesting functionality allowing the owner to vest and stake tokens. However, if the owner attempts to claim vested tokens while still staked, the contract only transfers the unstaked amount, resetting the `batch.last_claimed_release_time`. This results in the staked tokens being locked until they are vested again, effectively extending the vesting period and causing a denial-of-service (DoS) scenario. The duration of this DoS is dependent on the `release_unit` time set in the `execute_create_batch()` function.

Vulnerability Detail

The Andromeda protocol implements a vesting functionality. In the current implementation, the owner can vest tokens for himself, but this will be adapted to allow the owner to let tokens vest for other users. While those tokens are vesting, they can be staked by calling execute_delegate(), and the owner can withdraw their rewards by calling execute_withdraw_rewards().

The problem is that the owner/user can try to claim his tokens while they are staked. When the call occurs, the contract will take the minimum of its current balance and the tokens it should distribute. It will only transfer the minimum, not the actual amount.

```
let total_amount = AssetInfo::native(config.denom.to_owned())
    .query_balance(querier, env.contract.address.to_owned())?;

//Other calculations

let amount_to_send = amount_per_claim * Uint128::from(number_of_claims);
let amount_available = cmp::min(batch.amount - batch.amount_claimed,
↪  total_amount);

let amount_to_send = cmp::min(amount_to_send, amount_available);
```

This will result in the owner/user only receiving the tokens that are currently not staked and the `batch.last_claimed_release_time` being reset. As a result of this

the users staked tokens (which he should be able to access) will be locked until they are vested once again, resulting in extending the intended vesting period.

Exemplary scenario

An exemplary scenario can describe the vulnerability more:

1. Vesting gets generated with a total of 1.2 billion tokens, of which 100 million are distributed monthly. (effectively vesting over one year)
2. The user decides to stake 900 million of these tokens
3. At the end of the year, the user calls `execute_claim()` to claim all tokens
4. The call passes, but the user only receives 300 million tokens as the rest are staked
5. `batch.last_claimed_release_time` is set to the current date
6. If the user unstakes his tokens now, he will still need to wait another nine months to be able to retrieve them fully

Impact

If a user tries to claim his vesting while some of the tokens of that vesting are still staked, the staked tokens will become locked. The duration of this DOS is dependent on the time set in the `execute_create_batch()` function as `release_unit`. As vesting is usually done over multiple years, we can safely assume the DOS will be above seven days.

Code Snippet

```
fn claim_batch(
    querier: &QuerierWrapper,
    env: &Env,
    batch: &mut Batch,
    config: &Config,
    number_of_claims: Option<u64>,
) -> Result<Uint128, ContractError> {
    let current_time = env.block.time.seconds();
    ensure!(
        batch.lockup_end <= current_time,
        ContractError::FundsAreLocked {}
    );
    let amount_per_claim = batch.release_amount.get_amount(batch.amount)?;

    let total_amount = AssetInfo::native(config.denom.to_owned())
```

```
        .query_balance(querier, env.contract.address.to_owned())?;

    let elapsed_time = current_time - batch.last_claimed_release_time;
    let num_available_claims = elapsed_time / batch.release_unit;

    let number_of_claims = cmp::min(
        number_of_claims.unwrap_or(num_available_claims),
        num_available_claims,
    );

    let amount_to_send = amount_per_claim * Uint128::from(number_of_claims);
    let amount_available = cmp::min(batch.amount - batch.amount_claimed,
↪   total_amount);

    let amount_to_send = cmp::min(amount_to_send, amount_available);

    // We dont want to update the last_claim_time when there are no funds to
↪   claim.
    if !amount_to_send.is_zero() {
        batch.amount_claimed += amount_to_send;
        batch.last_claimed_release_time += number_of_claims * batch.release_unit;
    }

    Ok(amount_to_send)
}
```

Tool used

Manual Review

Recommendation

We recommend adapting the `claim_batch()` to revert if `amount_available < amount_to_send`. Alternatively, it could also check how many tokens are transferred and only move the `last_claimed_release_time` up by `ceil(transferred_tokens/batch.release_unit)`. This way, the user would, at max, incur a DOS of one `batch.release_unit`.

Discussion

sherlock-admin3

> escalate The owner is trustworthy, so they should not claim the amount in advance. This is a mistake on the part of the owner.

> From the README description: `Contract owner should be assumed trusted.`

You've deleted an escalation for this issue.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/554

bin2chen66

https://github.com/andromedaprotocol/andromeda-core/pull/554 PR removes `stake` logic, avoids possibility of insufficient balance This issue has been resolved

Issue M-14: Lockup of vestings or completion time can be bypassed due to missing check for staked tokens

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/58

Found by

J4X_

Summary

The vesting module in the Andromeda protocol allows multiple vestings to be created. Currently restricted to the owner, it will be extended to any user. While tokens are vesting, they can be staked to earn rewards. However, the protocol does not account for the staked tokens when claiming vestings. This allows users to withdraw staked tokens, potentially circumventing the lockup period and withdrawing tokens from other vestings that are not yet matured. This issue results in the ability to bypass vesting schedules and access locked tokens prematurely.

Vulnerability Detail

The vesting module allows for the creation of multiple vestings. This is restricted to the owner for now, but it will be extended to anyone. The current version can be used to proof lockup periods & vesting schedules to users. This is done by the owner depositing tokens into the contract and setting parameters for the vesting. While the tokens are vesting, they can be staked to a delegator to earn rewards by calling the execute_delegate() function. The vestings are tracked using batch struct.

```
pub struct Batch {
    /// The amount of tokens in the batch
    pub amount: Uint128,
    /// The amount of tokens that have been claimed.
    pub amount_claimed: Uint128,
    /// When the lockup ends.
    pub lockup_end: u64,
    /// How often releases occur.
    pub release_unit: u64,
    /// Specifies how much is to be released after each `release_unit`. If
    /// it is a percentage, it would be the percentage of the original amount.
    pub release_amount: WithdrawalType,
    /// The time at which the last claim took place in seconds.
    pub last_claimed_release_time: u64,
```

```
}
```

The problem occurs because the batches do not account for how many of their tokens were staked. As a result, the recipient can still withdraw tokens from a vesting that is currently staked. This can be seen when looking at the function handling the claiming.

```
fn claim_batch(
    querier: &QuerierWrapper,
    env: &Env,
    batch: &mut Batch,
    config: &Config,
    number_of_claims: Option<u64>,
) -> Result<Uint128, ContractError> {
    let current_time = env.block.time.seconds();
    ensure!(
        batch.lockup_end <= current_time,
        ContractError::FundsAreLocked {}
    );
    let amount_per_claim = batch.release_amount.get_amount(batch.amount)?;

    let total_amount = AssetInfo::native(config.denom.to_owned())
        .query_balance(querier, env.contract.address.to_owned())?;

    let elapsed_time = current_time - batch.last_claimed_release_time;
    let num_available_claims = elapsed_time / batch.release_unit;

    let number_of_claims = cmp::min(
        number_of_claims.unwrap_or(num_available_claims),
        num_available_claims,
    );

    let amount_to_send = amount_per_claim * Uint128::from(number_of_claims);
    let amount_available = cmp::min(batch.amount - batch.amount_claimed,
↪   total_amount);

    let amount_to_send = cmp::min(amount_to_send, amount_available);

    // We dont want to update the last_claim_time when there are no funds to
↪   claim.
    if !amount_to_send.is_zero() {
        batch.amount_claimed += amount_to_send;
        batch.last_claimed_release_time += number_of_claims * batch.release_unit;
    }

    Ok(amount_to_send)
```

```
}
```

The vulnerability leads to further issues if multiple vestings exist. In that case, the user will actually be sent tokens from one of the other vestings, which are not currently staked. This is an issue as the other vesting from which the tokens will originate might still be in its lockup period, and the tokens should not be withdrawable.

Exemplary scenario

1. `VestingA` (100 tokens) gets created with a `lockup_end` in 1 month and full claiming after that
2. User stakes all 100 tokens
3. `VestingB` (100 tokens) with `lockup_end` in 10 years is added
4. One month passes, and `VestingA` matures
5. The user does not want to wait for the completion time when unstaking his tokens from `VestingA`, so he just calls to claim `VestingA` while they are still staked
6. As it is not checked which tokens are staked, the claim passes
7. The user has effectively bypassed the completion time/lockup period.

Impact

This issue allows the recipient to circumvent the lockup duration of his vestings by withdrawing the tokens through another staked vesting.

Code Snippet

Tool used

Manual Review

Recommendation

We recommend adding the parameter `staked_tokens` to the `batch` struct.

```
pub struct Batch {
    /// The amount of tokens in the batch
    pub amount: Uint128,
    /// The amount of tokens that have been claimed.
    pub amount_claimed: Uint128,
```

```
    /// The amount of tokens that have been staked.
    pub amount_staked: Uint128, // <--- New variable
    /// When the lockup ends.
    pub lockup_end: u64,
    /// How often releases occur.
    pub release_unit: u64,
    /// Specifies how much is to be released after each `release_unit`. If
    /// it is a percentage, it would be the percentage of the original amount.
    pub release_amount: WithdrawalType,
    /// The time at which the last claim took place in seconds.
    pub last_claimed_release_time: u64,
}
```

This variable should be updated on each call to `executed_delegate()` and `execute_undelegate`. When a user tries to withdraw funds from his batch, the function must check if `amount - (amount_claimed + staked_tokens) >= tokens_to_withdraw`.

```
fn claim_batch(
    querier: &QuerierWrapper,
    env: &Env,
    batch: &mut Batch,
    config: &Config,
    number_of_claims: Option<u64>,
) -> Result<Uint128, ContractError> {
    let current_time = env.block.time.seconds();
    ensure!(
        batch.lockup_end <= current_time,
        ContractError::FundsAreLocked {}
    );
    let amount_per_claim = batch.release_amount.get_amount(batch.amount)?;

    let total_amount = AssetInfo::native(config.denom.to_owned())
        .query_balance(querier, env.contract.address.to_owned())?;

    let elapsed_time = current_time - batch.last_claimed_release_time;
    let num_available_claims = elapsed_time / batch.release_unit;

    let number_of_claims = cmp::min(
        number_of_claims.unwrap_or(num_available_claims),
        num_available_claims,
    );

    let amount_to_send = amount_per_claim * Uint128::from(number_of_claims);
    let amount_available = cmp::min(batch.amount - (batch.amount_claimed +
↪   batch.amount_staked), total_amount); // <---- Changed LOC
```

```
    let amount_to_send = cmp::min(amount_to_send, amount_available);

    // We dont want to update the last_claim_time when there are no funds to
↪   claim.
    if !amount_to_send.is_zero() {
        batch.amount_claimed += amount_to_send;
        batch.last_claimed_release_time += number_of_claims * batch.release_unit;
    }

    Ok(amount_to_send)
}
```

Discussion

cu5t0mPeo

escalate The owner is trustworthy, and there is no reason for the owner to bypass the lockup period. This is an input error on the part of the owner.

From the README description: `Contract owner should be assumed trusted.`

sherlock-admin3

> escalate The owner is trustworthy, and there is no reason for the owner to bypass the lockup period. This is an input error on the part of the owner.
>
> From the README description: `Contract owner should be assumed trusted.`

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

J4X-98

Hey,

The issue is that vestings are used as a way to restrict addresses from dumping tokens directly at the start and are intended to let tokens slowly vest over time. This is the only reason why they exist, as otherwise tokens can be directly distributed to the recipient. A user should never be able to take tokens from a vesting before its lockup-time, as it is an invariant that should never be broken. This issue clearly shows a case where this can occur. While the owner is trusted, we can assume that a owner vesting should not be claimable before the lockup

time, as it is a security guarantee to the users of the protocol. That's why the lockup time was implemented in the end.

Additionally this functionality is intended to be used for recipients and the owner, where recipients can very well act maliciously.

cvetanovv

I disagree with the escalation, and @J4X-98 describes things very well. This issue has no relation to admin trust or admin mistakes.

Planning to reject the escalation and leave the issue as is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- cu5t0mPeo: rejected

gjaldon

@cvetanovv @WangSecurity I would like to mention that this issue is similar to https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/55. I think it is invalid for the same reason which is that there is no damage to any party (recipient, owner, or protocol) because all Vesting batches belong to the same recipient.

ref: execute_claim() in Vesting ADO

```
let amount_to_send = claim_batch(&deps.querier, &env, &mut batch, &config,
↪  number_of_claims)?;
// ... snip ...
// There is only one configured recipient for the Vesting ADO and all vesting
↪  batches go to that one recipient
let withdraw_msg = config.recipient.generate_direct_msg(
    &deps.as_ref(),
    vec![Coin::new(amount_to_send.u128(), config.denom)],
)?;
```

In the example scenario, both `VestingA` and `VestingB` belong to the same Recipient, so even if `VestingA` takes the tokens from `VestingB`, there is no damage. It's like the recipient is only stealing from themself. I think the intention here was to keep the Vesting ADO simple with just one recipient so there is no need to track which tokens belong to which vesting batch.

J4X-98

This comment, as well as the arguments of you on other issues clearly ignores the fact that the vestings have a lockup duration which is a security measure that should not be circumvented in any way. It is used to prevent users from dumps etc, so if a vesting exists but the lockup period does not work, the vesting itself does not serve any purpose.

If an issue shows a way to bypass this safeguard it becomes a medium, similar to bypassing a blacklist. This has been pointed out to you by me and the judge. You seem to be missing the point that a medium can also result from the breaking of functionality or security measures not just the stealing of funds.

Additionally you mention that #55 is invalid which is simply untrue as the issue has been validated by the judge, you just escalated against it.

gjaldon

@WangSecurity @cvetanovv This is a duplicate of #55 because:

1. The root cause is the same: funds are pooled. The issue happens in the same contract, Vesting ADO, and the same function `claim_batch`.
2. The impact is the same: bypass of lockup period of vesting.
3. The fix in this report will also fix #55. The check that will be added when claiming funds `amount - (amount_claimed + staked_tokens) >= tokens_to_withdraw.` will prevent `VestingA` from getting `VestingB`'s funds in the example scenario in #55.

J4X-98

Escalations for this issue have already been resolved so it would be nice of you to respect the judges decision. Nevertheless I will answer your comment for the sake of completeness.

While the two issues are similar, the scenarios are not the same:

1. Tokens are slashed and this slashed amount is recouped from another vesting that is still in lockup. This other vesting can then never fully be withdrawn as the contract will not receive enough tokens.
2. The user stakes his tokens and then withdraws them which will pull them out of another locked up vesting.

gjaldon

Yes, the scenarios are not the same but the issues are.

Will the fix for #58 prevent the issue in #55?

cvetanovv

I agree with @gjaldon comment that #58 and #55 are duplicates.

Although there are differences between the two issues, the main rules we look at when deciding whether to duplicate them are the same.

The root cause is the same. Fixing one issue will fix the other. And the impact is the same. So I'll duplicate #58 and #55

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/554

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/554 PR removed the `stake` logic This issue has been resolved

Issue M-15: Batch creation will break if vestings are opened to recipients

Source: https://github.com/sherlock-audit/2024-05-andromeda-ado-judging/issues/59

Found by

J4X_, cu5t0mPe0

Summary

The `andromeda-vesting` contract allows the owner to create vestings (batches) for freezing tokens. The planned update will enable the recipient to claim or delegate tokens instead of the owner. However, this change introduces a conflict in the delegation process during batch creation, where the `execute_delegate()` function will check for both owner and recipient roles, causing it to always revert. This issue makes it impossible to create batches with direct delegation.

Vulnerability Detail

The `andromeda-vesting` contract allows for creating vestings, aka `batches`. The current contract is fully restricted to the `owner`. Effectively it only allows the owner to freeze his tokens in vestings to recover them later. To include some real functionality, the team plans to adapt the functionality so that the owner still creates the batches, but they can be claimed or delegated by the recipient. This is also described in the contest description:

As per my communication with the team, the only change that will occur is that the restriction for the `owner` in the claiming and delegation functions will be replaced with a restriction for the `recipient`. For the following reason, it will be impossible to create vestings with a direct delegation.

When a vesting gets created, it can only be done by the owner due to the following check

```
fn execute_create_batch(
    ctx: ExecuteContext,
    lockup_duration: Option<u64>,
    release_unit: u64,
    release_amount: WithdrawalType,
    validator_to_delegate_to: Option<String>,
) -> Result<Response, ContractError> {
    let ExecuteContext {
        deps, info, env, ..
```

```
    } = ctx;
    ensure!(
        ADOContract::default().is_owner_or_operator(deps.storage,
↪   info.sender.as_str())?,
        ContractError::Unauthorized {}
    );
```

The batch creator can pass a `validator_to_delegate_to` parameter, resulting in the vested tokens being directly staked to a validator. To do this, the `execute_create_batch()` will call the execute_delegate() function. This function is currently restricted to the owner, but will be changed to be restricted to the recipient, as based on the contest description. The problem is that in this case the delegation as well as the creation of batches will always revert as it will check `info.sender == owner` and `info.sender == recipient`.

Impact

This issue results in the creation of batches becoming impossible with a direct delegation.

Code Snippet

https://github.com/sherlock-audit/2024-05-andromeda-ado/blob/bbbf73e5d1e4092ab42ce1f827e33759308d3786/andromeda-core/contracts/finance/andromeda-vesting/src/contract.rs#L314-L317

Tool used

Manual Review

Recommendation

We recommend adapting the `execute_delegate` function to be callable by the owner or recipient instead of just the owner.

```
fn execute_delegate(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    amount: Option<Uint128>,
    validator: String,
) -> Result<Response, ContractError> {
    let sender = info.sender.to_string();
    ensure!(
```

```
        ADOContract::default().is_contract_owner(deps.storage, &sender)? ||
↪   sender ==  recipient,
        ContractError::Unauthorized {}
    );
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/andromedaprotocol/andromeda-core/pull/554

bin2chen66

fix-reviews note: https://github.com/andromedaprotocol/andromeda-core/pull/554
PR has removed `validator_to_delegate_to`
This issue has been resolved

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.