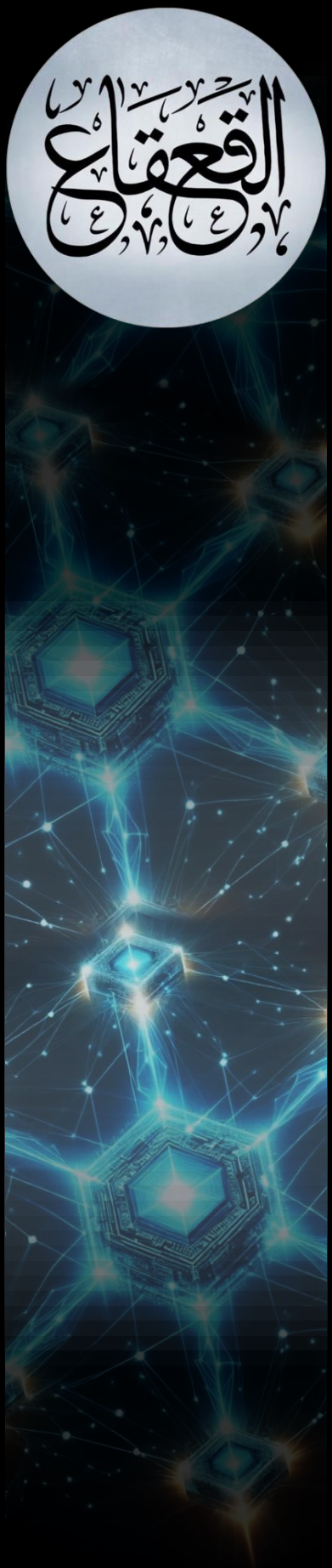
Al-Qa'qa’



TakaDAO

ReferralGateway Auditing Report

Auditor: Al-Qa'qa’

14 December 2024

Table of Contents

1 Introduction

1.1 About Al-Qa'qa'

Al-Qa'qa' is an independent Web3 security researcher specializing in smart contract audits. Success in placing top 5 in multiple contests on code4rena and sherlock.
In addition to smart contract audits, he has moderate experience in core EVM architecture, geth.

For security consulting, reach out to him on Twitter - @Al_Qa_qa

1.2 About TakaDAO

TakaDAO is A decentralized Insurance Protocol, that takes the traditional insurance process one step further. People Can contribute to different DAOs, which represents different insurance types like Life Insurance, Health insurance, etc... And put a given amount of money in a given DAO, that makes them eligible for taking funds, if they prove they deserve them.

1.3 Disclaimer

Security review cannot guarantee `100%` the protocol's safety. In the Auditing process, we try to identify all possible issues, and we cannot be sure if we missed something.

Al-Qa'qa' is not responsible for any misbehavior, bugs, or exploits affecting the audited
code or any part of the deployment phase.

And change to the code after the mitigation process, puts the protocol at risk, and should be audited again.

1.4 Risk Classification

Severity	Impact:High	Impact:Medium	Impact:Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.4.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption to the protocol's functionality
- Low - Funds are **not** at risk

1.4.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align or little-to-no incentive

2 Executive Summary

2.1 Overview

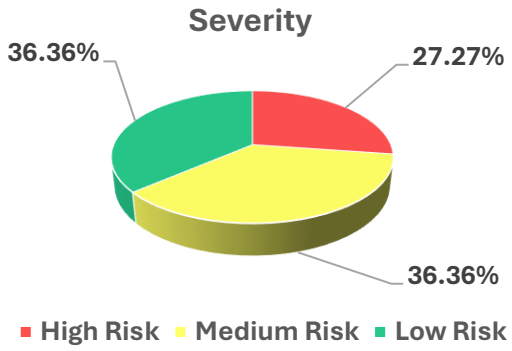
Project	**TakaDAO**
Repository	takasure.sc
Commit Hash	986e61d7e25209e675bf86dbf2edbf871c052247
Mitigation Hash	f872b84a10becd316992e7eb0a04cbb7e12dece4
Audit Timeline	30 Oct 2024 to 06 Nov 2024

2.2 Scope

- contracts/referrals/ReferralGateway.sol

2.3 Issues Found

Severity	Count
High Risk	3
Medium Risk	4
Low Risk	4



3 Findings Summary

ID	Title	Severity
H-01	Referral rewards are being deducted even with no referrer	High
H-02	The refunding process deducts discounts from the member without giving them to the operator as well as repoll fees	High
H-03	Refunding process will result in Draining of the Gateway contract	High
M-01	`setUsdcAddress()` will make the protocol insolvable	Medium
M-02	The Protocol is incompatible with a Single member joining multiple DAOs	Medium
M-03	Members will be prevented from joining launched DAOs if they join unlaunched ones	Medium
M-04	Referrers will not earn their rewards for more than one DAO contribution	Medium
L-01	`updateLaunchDate()` is not checking `launchDate` value	Low
L-02	No event emitting when firing critical functions	Low
L-03	`payContribution()` doesn't check for existing DAO	Low
L-04	Rewards are not resetting to zero when distributing to referrers	Low
I-01	`joinDAO()` do not check the contribution of the member	Info
I-02	No need for `parentRewardsByLayer` anymore	Info

4 Findings

4.1 High Risk

4.1.1 Referral rewards are being deducted even with no referrer

context: ReferralGateway.sol#L354-L355

Description:
When referrals are allowed for the DAO, we are reducing the fees goes to the developers by `5%`, which will then be distributed to the referrer who referred that referee (member).

The address of the referrer is provided by the caller (member/referee) itself, if it is `address(0)`, then this member came alone and no one referred him to the program.

The problem is that the amount sent to the referrer is deducted from fees without checking if that member is the referee of not. So in case of a member that came alone, we will subtract `5%` of the amount, which will still be sent by the caller, and not given to developers.

ReferralGateway.sol#L354-L355

```
    if (DAO.referralDiscount) {
>>      uint256 toReferralReserve = (contribution * REFERRAL_RESERVE) / 100;
>>      finalFee -= toReferralReserve;
        if (parent != address(0) && isMemberKYCed[parent]) {
            ...

            finalFee = _parentRewards(
                msg.sender,
                contribution,
>>              toReferralReserve,
                finalFee
            );
        }
    }
```

The problem here is that the amount we will distribute to the referrer (referrer tree) is only distributed if there is an actual referrer address using `_parentRewards()` but without it, the deducted funds (5%) from the contribution will still stuck in in the gateway contract.

In the referral system, we are distributing `5%` at max.

- If we distributed < 5% | increase `referralReserveBalance` global variable
- If we distributed > 5% | take from `referralReserveBalance` if existed of decrease the fees

ReferralGateway.sol#L603-L609

```
    function _parentRewards( ... ) internal returns (uint256) {
        address currentChildToCheck = _initialChildToCheck;
```

```
        uint256 newReferralReserveBalance = referralReserveBalance +
_toReferralReserve;
        uint256 parentRewardsAccumulated;

        for (int256 i; i < MAX_TIER; ++i) {
            ...
            parentRewardsAccumulated += parentReward;
            ...
        }

        if (newReferralReserveBalance > parentRewardsAccumulated) {
>>          newReferralReserveBalance -= parentRewardsAccumulated;
        } else {
            uint256 rewardFromReserve = parentRewardsAccumulated -
newReferralReserveBalance;
>>          _currentFee -= rewardFromReserve;
>>          newReferralReserveBalance = 0;
        }

        referralReserveBalance = newReferralReserveBalance;

        return _currentFee;
    }
```

So the members with no referrer will will pay `5%` (toReferralReserve) and this amount will get sent by the member to the contract, and not get transferred to protocol Devs, nor will it be transferred to parents, nor `referralReserveBalance` will increase by that amount.

This will result in a Stuck of funds in the contract.

Recommendations:
Don't reduce the amount if there is no referrer. Or you can increase `referralReserveBalance` balance by the amount of contribution.

4.1.2 The refunding process deducts discounts from the member without giving them to the operator as well as repoll fees

context: ReferralGateway.sol#L504

Description:
When a given DAO is created, members can contribute to that DAO and put funds in it. The process of contributing is done by paying an amount, and there can be discounts in case of pre-join, referrals, etc...

In case the DAO fails to launch, users can refund their money. In refunding process, they take the money they contributed with but the discount they gain is getting deducted from there balance.

ReferralGateway.sol#L504

```
    function refundIfDAOIsNotLaunched(address member, string calldata tDAOName)
external {
        ...
```

```
        uint256 feePaid = prepaidMember.contributionBeforeFee -
prepaidMember.contributionAfterFee;
        uint256 discountReceived = prepaidMember.discount;

>>      uint256 amountToRefund = prepaidMember.contributionBeforeFee - feePaid -
discountReceived;

        ...
    }
```

The problem here is that the discount the user is given is not paid by him as contribution, but gets decreased from the balance, as `contributionAfterFee` store to total fees taken from the user (is intended to be taken), which is `27%`.

ReferralGateway.sol#L373-L380

```
    function payContribution( ... ) external returns (uint256 finalFee, uint256
discount) {
        ...
        if (DAO.preJoinEnabled) {
            ...
            PrepaidMember memory prepayer = PrepaidMember({
                tDAOName: tDAOName,
                member: msg.sender,
>>              contributionBeforeFee: contribution, // Input value, we need it
like this for the actual join when the DAO is deployed
>>              contributionAfterFee: contribution - fee, // Without discount, we
need it like this for the actual join when the DAO is deployed
                finalFee: finalFee,
                discount: discount
            });

            ...
        } else { ... }
    }
```

So, in case the DAO didn't launch all discounts made to users will get deducted from their original balance, and not get transferred to operator. Since this discount is being taken from the operator's fees himself, they should get transferred back to the operator.

Proof of Concept:

- `UserA` contributed with `100 USDC` in a given DAO
- He is in a prejoin and referrals are not activated
- He received `10%` discount, i.e 10 USDC
- He transferred `90 USDC` to the contract
- contributionBeforeFee = `100 USDC`
- contributionAfterFee = 100 - (100 * 27%) = `73 USDC`
- finalFee = 17% * 100 USDC = `17 USDC`
- discount = 10% * 100 = `10 USDC`
- `UserA` sent `90 USDC`
- send finalFee to the operator (15 USDC), with 2% remains for repool
- Now our contract state is:
    - Gateway balance is: 90 - 15 = 75 USDC

    - 2 USDC are for repooling and the other 73 USDC is the contribution of that member
- DAO didn't launch
- `UserA` made a refund
    - feePaid = contributionBeforeFee - contributionAfterFee = 100 - 73 = `27 USDC`
    - amountToRefund = contributionBeforeFee - feePaid - discount = 100 - 27 - 10 = `63 USDC`
    - sent `63 USDC` to that member
- Know our contract will have 75 - 63 = 12 USDC
    - 2 USDC are for repooling (which will get lost) as the launch didn't occur
    - 10 USDC left the discount the user was given as prejoin, but this money gets deducted from his recovered balance
    - Funds will get stucked in the contract

Recommendations:

- Recover discounts made for the user by sending them to the operator
- implement a function that recovers repool funds in case DAO didn't launch

4.1.3 Refunding process will result in Draining of the Gateway contract

context: ReferralGateway.sol#L514-L516

Description:
When creating a DAO, members contribute to that DAO, but the DAO launching didn't occur for reasons. There is a way for members to refund the money they contributed.

The problem is that the refunding process will result in Paying members with money more than they paid, results in Gateway contract balance, which is the balance for DAOs, this balance will get drained as they pay money exceeds what was but by the members.

When a given member contributes to a given DAO, the amount he paid is calculated as following.

- `contributionBeforeFee`: all the amount he paid
- `contributionAfterFee`: The actual balance of that member after deducting fees
- `feeToOperator`: The amount of fees goes to Operator
- `discount`: The discount that member receives (discount is taken from the fees)

ReferralGateway.sol#L378-L391

```
    function payContribution( ... ) external whenNotPaused nonReentrant returns
(uint256 finalFee, uint256 discount) {
        ...
        if (nameToDAOData[tDAOName].preJoinEnabled) {
            ...
1:          amountToTransfer -= discount;
            ...
2:          usdc.safeTransferFrom(msg.sender, address(this), amountToTransfer);
```

```
3:          usdc.safeTransfer(operator, finalFee);

            nameToDAOData[tDAOName].prepaidMembers[msg.sender].member =
msg.sender;

nameToDAOData[tDAOName].prepaidMembers[msg.sender].contributionBeforeFee =
contribution;

nameToDAOData[tDAOName].prepaidMembers[msg.sender].contributionAfterFee =
                contribution -
                fee;
            nameToDAOData[tDAOName].prepaidMembers[msg.sender].feeToOperator =
finalFee;
            nameToDAOData[tDAOName].prepaidMembers[msg.sender].discount =
discount;
            ...
        } else { ... }
    }
```

The amount the member paid gets decreased by the `discount`. So if he contributed with `100 USDC` and received `10%` discount, he will pay `90 USDC`. Then, the fees are getting transferred to operator.

The problem is that when making the refund process, we are giving the member his original contribution `contributionBeforeFee` by subtracting discount he received.

ReferralGateway.sol#L514-L516

```
    function refundIfDAOIsNotLaunched(address member, string calldata tDAOName)
external {
        ...
        uint256 discountReceived =
nameToDAOData[tDAOName].prepaidMembers[member].discount;

>>      uint256 amountToRefund = nameToDAOData[tDAOName]
            .prepaidMembers[member]
            .contributionBeforeFee - discountReceived;
        ...
    }
```

This will result in refunding the user with amount exceeds his actual contribution balance without Fees

Proof of Concept:

- `UserA` contributed to `DAO 1`
- `UserA` contributed with `100 USDC`
- `UserA` received `10%` discount (prejoin) and fees gets deducted by `10%` too
- `UserA` transferred `90 USDC` to the Gateway
- `15 USDC` gets transferred to operator and `2 USDC` left for repoll
- Gateway has `75 USDC` balance at total
- DAO didn't launch
- `UserA` refunded his money
- amountToRefund = contributionBeforeFee - discount = 100 - 10 = `90 USDC`

- The refund will either fail because of no balance, or success by taking money from another member (contract balance)

Recommendations:
Use `contributionAfterFee` instead of `contributionBeforeFee`. But this will leave the discount balance stucked in the contract as stated in `H-02`. So, there are two possible solutions.

1. Is to implement the mitigation in `H-02` which uses `contributionAfterFee` and transfer discount to operators
2. Send to the user `contributionAfterFee`, without making him pay for the discount.

NOTE: this will still make repool money (2%) stuck as stated in `H-02`

4.2 Medium Risk

4.2.1 `setUsdcAddress()` will make the protocol insolvable

context: ReferralGateway.sol#L538-L540

Description:
This function is used to change the USDC address, which is the address used by the protocol, and firing this function will introduce massive problems.

ReferralGateway.sol#L538-L540

```
    function setUsdcAddress(address _usdcAddress) external onlyRole(OPERATOR) {
        usdc = IERC20(_usdcAddress);
    }
```

`USDC` is getting transferred to the contract via `payContribution()`. Simply changing this value will make the OLD tokens stuck in the contract, as the address gets changed.

Since the contract is Upgradeable, there will be a way to rescue funds (locked USDC tokens for the old address). However, the method itself will make the protocol fill in a situation where the old contributions money will get stuck.

I don't know the exact purpose of this function, but if we want to change the token we use to pay like changing `USDC` to `USDT`, then we should do the following things.

- transfer all USDC tokens to ourselves (let's say `10K`)
- change address
- transfer the same amount to the contract (give contract `10K` USDT)

This will make the protocol survive the DOS scenario, that will occur because of no tokens.

Another thing is that the new token should have the same decimals.

Recommendations:
Either removing the function used to change `USDC` address which can still occur using the upgradability feature, which will be handled more accurately. Or implement the

mechanism of Taking all Tokens before changing the address of USDC, providing the same amount we took using new token.

4.2.2 The Protocol is incompatible with a Single member joining multiple DAOs

context: BenefitMultiplierConsumer.sol#L80

Description:
There can be more than one DAO and users (members) can contribute in more than one of these DAO.

When contributing in a DAO, we are requesting the consumer, which will make an API function call using ChainLink functions to complete the KYC process, and join DAO terms, etc...

ReferralGateway.sol#L397-L398 | ReferralGateway.sol#L570

```
    function payContribution(
        uint256 contribution,
        string calldata tDAOName,
        address parent
    ) external returns (uint256 finalFee, uint256 discount) {
        ...
        if (DAO.preJoinEnabled) {
            ...

            prepaidMembers[msg.sender][tDAOName] = prepayer;

            // Finally, we request the benefit multiplier for the member, this to
have it ready when the member joins the DAO
>>          _getBenefitMultiplierFromOracle(msg.sender);

            emit OnPrepayment(parent, msg.sender, contribution);
        } else { ... }
    }
// --------------
    function _getBenefitMultiplierFromOracle(address _member) internal {
        string memory memberAddressToString =
Strings.toHexString(uint256(uint160(_member)), 20);
        string[] memory args = new string[](1);
        args[0] = memberAddressToString;
>>      bmConsumer.sendRequest(args);
    }
```

In `BenefitMultiplierConsumer`, we store the request in `memberToRequestId`. But the problem is that we only rely on the address of the sender.

BenefitMultiplierConsumer.sol#L80

```
    function sendRequest(
        string[] memory args
    ) external onlyRole(BM_REQUESTER_ROLE) returns (bytes32 requestId) {
        ...
>>      memberToRequestId[args[0]] = lastRequestId;
        requestId = lastRequestId;
```

```
    }
```

We store our members using only their addresses in a string format. So, in case the same user (member) makes a contribution to one DAO and then makes another one to another DAO, the mapping of the second DAO contribution `requestId` will override the first DAO `requestId`.

The `TakasurePool` contract (DAO Address), is checking for joining using that mapping where when joining a DAO we call `TakasurePool::prejoins()`, which checks whether the user was already requested or not using the BMF Consumer,
using `TakasurePool::_getBenefitMultiplierFromOracle()`

TakasurePool.sol#L831

```
    function _getBenefitMultiplierFromOracle( ... ) internal returns (uint256
benefitMultiplier_) {
        string memory memberAddressToString =
Strings.toHexString(uint256(uint160(_member)), 20);

        // First we check if there is already a request id for this member
>>      bytes32 requestId = bmConsumer.memberToRequestId(memberAddressToString);

        if (requestId == 0) { ... } else {
            // If there is a request id, we check if it was successful
>>          bool successRequest = bmConsumer.idToSuccessRequest(requestId);

            if (successRequest) { ... } else { ...  }
        }
    }
```

As we can see in `_getBenefitMultiplierFromOracle()`, we are retrieving a requestId, and with that ID, the dAO determines the member's status. However, this will make all DAOs that the member contributed to. It relies on the last request only, not the actual request the Consumer receiver for a given DAO.

Proof of Concept:

- `userA` contributed to `DAO 1`
- request fulfillment success
- `userA` contributed to `DAO 2`
- request fulfillment success
- `userA` contributed to `DAO 3`
- request fulfillment failed
- All three DAOs will rely on the last request which receives an error. preventing the user from joining The three DAOs, instead of just preventing him from joining `DAO 3`

There are other scenarios like changing the JS code
using `setBMSourceRequestCode()` which will make Old DAOs rely on the new results from the new JavaScript code instead of the old one, which can introduce several issues on How DAOs organize the Insurance process

Recommendations:
Instead of storing the mapping using only the member address, use it like nested

mapping using the member address with the DAO name. This will make each DAO have its own requests instead of having requests override each other.

4.2.3 Members will be prevented from joining launched DAOs if they join unlaunched ones

context: ReferralGateway.sol#L514

Description:
Users (members) can contribute in different DAOs, when contributing in DAOs, the BMF consumer checks whether they need a KYC process or not, and in case of needed KYC process, it is done, and then the KYC provider can set that member as KYCed member.

In order for the members to joinDAOs, there are two conditions. The member should be KYCed member, and the DAO should be launched.

ReferralGateway.sol#L461-L469

```
    function joinDAO(address newMember, string calldata tDAOName) external
nonReentrant {
        // Initial checks
>>      require(isMemberKYCed[newMember], ReferralGateway__NotKYCed());
        ...
        require(
            DAO.DAOAddress != address(0) && DAO.launchDate <= block.timestamp,
            ReferralGateway__tDAONotReadyYet()
        );
        ...
    }
```

Some DAOs can't launch, and in this case members can refund there contributions calling `refundIfDAOIsNotLaunched()`.

The problem is that in case of refunding a member when DAO is not launched, we are resetting his KYC process.

ReferralGateway.sol#L514

```
    function refundIfDAOIsNotLaunched(address member, string calldata tDAOName)
external {
        ...
>>      isMemberKYCed[member] = false;

        usdc.safeTransfer(member, amountToRefund);

        emit OnRefund(tDAOName, member, amountToRefund);
    }
```

So, in case one member contributed in more than on DAO, and one of them didn't launch, refunding process will make that member unKYCed, preventing him from joining other DAOs he contributed to.

Proof of Concept:

- `UserA` contributed to `DAO 1`
- `User A` is now a KYC member
- `UserA` contributed to `DAO 2`
- `User A` is already a KYC member, no need to request a KYC from him
- `DAO 2` launch date came before `DAO 1` launch date
- `DAO 2` didn't launch
- `UserA` refunded his contribution from `DAO 2`
- `UserA` is not a KYCed member now
- `DAO 1` get launched successfully
- `UserA` tried to join the DAO, but the tx reverted as he is not a KYC member

Note in the case of refunding there is no restriction that the caller is the member himself, where anyone can put the member's address and make the refund process, making him an unKYCed member. So even in case users know that thing, the issue can still occur.

Recommendations:\

- Store the DAOs members have joined in in an array structure, Set, mapping, etc...
- In case of refunding when DAO is not launched, check that the member is KYCed, and has joined that specific DAO, using the array from the first step
- Remove that DAO from the array of DAOs that the member contributes too
- In case the array becomes empty, you can unKYCed that member

4.2.4 Referrers will not earn their rewards for more than one DAO contribution

context: ReferralGateway.sol#L425

Description:
For members to make the KYC process, they join and contribute to a given DAO, then they can do the KYC process.

The KYC process is set only one time, if the user is KYCed, he don't need to make the KYC process again.

In the new referral system, rewards are only distributed to referrers when KYCing the referee, so in case the referee was already KYCed, this will make referrers unable to take their rewards as it is only distributed when KYCing a member, which occuar only one time, for all DAOs.

ReferralGateway.sol#L425

```
    function setKYCStatus( ... ) external whenNotPaused notZeroAddress(child)
onlyRole(KYC_PROVIDER) {
        ...
>>      require(!isMemberKYCed[child], ReferralGateway__MemberAlreadyKYCed());
        ...
        address parent = childToParent[child];
```

```
        for (uint256 i; i < uint256(MAX_TIER); ++i) {
            ...
>>          usdc.safeTransfer(parent, parentReward);

            emit OnParentRewarded(parent, layer, child, parentReward);
        }
        ...
    }
```

Recommendations:
Implement a function that allows referrers to withdraw their referral rewards manually.

4.3 Low Risk

4.3.1 `updateLaunchDate()` is not checking `launchDate` value

context: ReferralGateway.sol#L233-L242

Description:
When updating the launchDate, we should check that the new launch date is in the future not the past, as done when creating the DAO using `createDAO`.

ReferralGateway.sol#L209

```
    function createDAO( ... ) external {
        require(bytes(DAOName).length > 0, ReferralGateway__MustHaveName());
        require(
            !(Strings.equal(nameToDAOData[DAOName].name, DAOName)),
            ReferralGateway__AlreadyExists()
        );
>>      require(launchDate > block.timestamp,
ReferralGateway__InvalidLaunchDate());
        ...
    }
```

ReferralGateway.sol#L233-L242

```
    // @audit missing checking launchDate value come in the future
    function updateLaunchDate( ... ) external onlyDAOAdmin(tDAOName) {
        require(
            nameToDAOData[tDAOName].DAOAddress == address(0),
            ReferralGateway__DAOAlreadyLaunched()
        );
        nameToDAOData[tDAOName].launchDate = launchDate;
    }
```

Recommendations:
Check that the new `launchDate` is valid (in the future not the past)

```
diff --git a/contracts/referrals/ReferralGateway.sol
b/contracts/referrals/ReferralGateway.sol
index 9b00b9d..f69bd86 100644
--- a/contracts/referrals/ReferralGateway.sol
+++ b/contracts/referrals/ReferralGateway.sol
```

```
@@ -238,6 +238,7 @@ contract ReferralGateway is
             nameToDAOData[tDAOName].DAOAddress == address(0),
             ReferralGateway__DAOAlreadyLaunched()
         );
+        require(launchDate > block.timestamp,
ReferralGateway__InvalidLaunchDate());
         nameToDAOData[tDAOName].launchDate = launchDate;
     }
```

4.3.2 No event emitting when firing critical functions

context: ReferralGateway.sol

Description:
There are no events emits when doing important actions. This includes: `createDAO()`, `launchDAO()`, `setUsdcAddress()`. And other less important functions like: `updateLaunchDate()`, `switchReferralDiscount()`, `enableRepool()`

This will make the analysis process and gathering information harder. And since our contract relies on an off-chain mechanism, events are one of the important things off-chain will use to collect data, related to active DAOs, and other information.

Recommendations:
Add the appropriate events when calling these functions.

4.3.3 `payContribution()` doesn't check for existing DAO

context: ReferralGateway.sol#L324-L329

Description:
When a given member pays a contribution, there is no check whether the DAO existed or not.

ReferralGateway.sol#L324-L329

```
    function payContribution(
        uint256 contribution,
        string calldata tDAOName,
        address parent
    ) external returns (uint256 finalFee, uint256 discount) {
        tDAO memory DAO = nameToDAOData[tDAOName];
        require(
            contribution >= MINIMUM_CONTRIBUTION && contribution <=
MAXIMUM_CONTRIBUTION,
            ReferralGateway__ContributionOutOfRange()
        );
        ...
    }
```

This will allow users to join for DAOs that are not yet existed. which is not intended, as the DAO should be created or launched in order for users to contribute in.

Since the logic for joining is still not implemented, it will not make a critical issues in the current time. But after implementing fired joining, the problem will occur.

Recommendations:
Check that the DAO is created, before paying contributions.

```
diff --git a/contracts/referrals/ReferralGateway.sol
b/contracts/referrals/ReferralGateway.sol
index bf9eb80..5812c6b 100644
--- a/contracts/referrals/ReferralGateway.sol
+++ b/contracts/referrals/ReferralGateway.sol
@@ -331,6 +331,7 @@ contract ReferralGateway is
             contribution >= MINIMUM_CONTRIBUTION && contribution <=
MAXIMUM_CONTRIBUTION,
             ReferralGateway__ContributionOutOfRange()
         );
+        require(bytes(DAO.name).length > 0, ReferralGateway__MustHaveName());

         uint256 fee = (contribution * SERVICE_FEE_RATIO) / 100;
         finalFee = fee;
```

4.3.4 Rewards are not resetting to zero when distributing to referrers

context: ReferralGateway.sol#L443-L447

Description:
When contributing to a given DAO that supports referral system, referrers rewards increase.

ReferralGateway.sol#L673-L679

```
    function _parentRewards( ... ) internal returns (uint256, uint256) {
        ...
        for (int256 i; i < MAX_TIER; ++i) {
            ...
>>          nameToDAOData[_tDAOName]
                .prepaidMembers[childToParent[currentChildToCheck]]
                .parentRewardsByChild[msg.sender] += parentReward;

>>          nameToDAOData[_tDAOName]
                .prepaidMembers[childToParent[currentChildToCheck]]
                .parentRewardsByLayer[uint256(i + 1)] += parentReward;

            ...
        }
        ...
    }
```

When that member who another one referred gets KYCed, rewards are distributed to referrers. But when distributing rewards, we do not reset the value to zero; it stays the same.

ReferralGateway.sol#L443-L447

```
    function setKYCStatus(
        address child,
        string calldata tDAOName
```

```
    ) external whenNotPaused notZeroAddress(child) onlyRole(KYC_PROVIDER) {
        ...
        for (uint256 i; i < uint256(MAX_TIER); ++i) {
            ...
>>          uint256 parentReward = nameToDAOData[tDAOName]
                .prepaidMembers[parent]
                .parentRewardsByChild[child];

>>          usdc.safeTransfer(parent, parentReward);

            emit OnParentRewarded(parent, layer, child, parentReward);
            ...
        }
        ...
    }
```

Since we use a hierarchical referral system, the rewards are increasing. But according to the current implemented logic.

- Each DAO has its members
- single member can contribute only once

This will make the value of `parentRewardsByChild` only increased one time, and distributing rewards will also occur one time, as it only occurs for KYCed use.

But since there is a refund process in case of failed launching, and we use `delete` operation to delete the member, then setting it's KYC status to false. Our mappings will still not getting deleted. making the reward for `parentRewardsByChild` still existed.

ReferralGateway.sol#L518

```
    function refundIfDAOIsNotLaunched(address member, string calldata tDAOName)
external {
        ...
>>      delete nameToDAOData[tDAOName].prepaidMembers[member];

>>      isMemberKYCed[member] = false;
        ...
    }
```

So, if a given DAO gets created and launch process gets postponed, and the refunding process occurs by contributions. But after a while, the contribution process opened again, and the DAO was launched. These users will request a KYC process again. making the rewards distributed twice.

Recommendations:
Reset the rewards to zero when distributing them to referrers.

4.4 Informational Findings

4.4.1 `joinDAO()` do not check the contribution of the member

context: ReferralGateway.sol#L465-L476

Description:
When member join the DAO, there is no check if they were contributors or not, there is just a check that the member is KYCed. But if the member is KYCed, this doesn't mean he is contributed to that DAO, as there may be more than one DAO. Or maybe contributed members can get KYCed.

ReferralGateway.sol#L465-L476

```
    function joinDAO( ... ) external whenNotPaused nonReentrant {
        // Initial checks
        require(isMemberKYCed[newMember], ReferralGateway__NotKYCed());

        require(
            nameToDAOData[tDAOName].DAOAddress != address(0) &&
                nameToDAOData[tDAOName].launchDate <= block.timestamp,
            ReferralGateway__tDAONotReadyYet()
        );

        // Finally, we join the prepaidMember to the tDAO
        ITakasurePool(nameToDAOData[tDAOName].DAOAddress).prejoins( ... );
        ...
    }
```

Recommendations:
Check that the member is a contributor by checking
his `contributionBeforeFee` and/or `contributionAfterFee`.

4.4.2 No need for `parentRewardsByLayer` anymore

context: ReferralGateway.sol#L50

Description:
When member contributes to a given DAO with referrer, referrer rewards was stored using two mappings `parentRewardsByChild` and `parentRewardsByLayer`.

In the new design, rewards are getting transferred only when KYCing the referee, using `parentRewardsByChild`, and there is no need for `parentRewardsByLayer` variable in the rest of the contract.

Recommendations:
remove `parentRewardsByLayer` mapping from PrepaidMember Struct.