dClimate Marketplace Audit by Team2 (takeda)

Recap of higher level of issues

CL-XXX: `Assets.sol` `updatePrice` could disrupt marketplace buy flow

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

As explainedin the Spec Document

> *Assets which have a cost greater than 0 should be encrypted first (off-chain) by the publisher using a symmetric key and then uploaded to IPFS. The same symmetric key used to encrypt the data should then be encrypted (off-chain) with the Provider's public key (existing on the Provider's contract) and posted to the metadata of the Asset which is also uploaded to IPFS. This allows for proxy re-encryption to be done off-chain by the Chainlink Nodes' (also known as Providers) External Adapters during the Marketplace purchasing flow.*

For each asset we have two possibility when the asset is published and added to the `Asset.sol` store:

- `price === 0` document is not encrypted
- `price !== 0` document is encrypted

In the `Marketplace.sol` because of this reason when an asset is purchased via `createOrder` it will follow a different flow.

1. `assetPrice === 0` user will not pay anything and will be able to download it directly
2. `assetPrice !== 0` user needs to pay `assetPrice` to the owner of the asset and also pay an `operatorFee` to pay for the decryption mechanism process via Chainlink

Given this premise the problem is that the `price` of an asset can be arbitrary changed after the asset publishing.

We have two possible scenarios here

- T0 asset has `price === 0`, it's **not** encrypted. At T1 price is updated to `price !== 0`. This means that when a user call `Marketplace.createOrder` it will pass through the **Marketplace Flow 2)** and the user will pay for a `operatorFee` that is not needed because the asset does not need to be encrypted. Probably the decryption operation will revert or the asset will be "corrupted" because there's nothing to decrypt
- T0 asset has `price !== 0`, **it is** encrypted. At T1 price is updated to `price === 0`. This means that when a user call `Marketplace.createOrder` it will pass through the **Marketplace Flow 1)**. While it's true that the user will not pay any price for the asset or `operatorFee`, he's still going to pay for gas for a purchase that will not be usable because it's encrypted.

Mitigation

There are many options to mitigate this problem. One possible easy solution that does not involve any changes from the `publisher` side (because remember that at any point in time the `owner` of the asset (that can change the price) can be different from the original `publisher`).

Add a new variable `originalEncrypted` to the `struct Asset` to track if the asset has been published as encrypted or not.

At this point you can act on `updatePrice` to disallow an originally encrypted asset to be update with a 0 price.
Otherwise you can use that property on the `Marketplace.createOrder` to skip the `decryption` flow if the `asset` was not originally encrypted and does not need that process.

CL-XXX: `Marketplace.sol` `Provider`'s `operatorFee` could lead to DDOS

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

While it's true that the provider should be paid for their service (decryption) the should not be able to set the `operatorFee` without any limit.

If the operator is malicius it could set an high fee and the purchase flow would be expensive (comprared to the asset price) or even impossible (user has not enough funds).

Mitigation

There are multiple solutions:

1. add a top limit to the operator fee
2. let the govern choose a general operator fee (equal for everyone)

CL-XXX: General concerns about `Providers.sol` `Publisher` logic/flow

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

Publisher have a huge role in the dClimate protocol because they are required in order to decrypt encrypted assets when the asset's price is greater than 0. There's not a requirement that a `Provider` must be the same entity of the one that publish the `Asset`.

In the current implementation, as soon as a provider has been created, the provider's owner (the one that have called the `createProvider`) have the total control on it.

Both the `governance` or the contract's ADMIN have no power on the created provider.

This mean that at any point in time the Provider could DDOS the service making all the assets associated to the provider unabled to be purchased by the end user.

These are all the possible things that could make revert a purchse tx:

- provider is disabled
- provider has not enough link funds
- operatorFee is high
- oracleOperatorFee is not enough high to cover cost
- oracleAddress is empty
- jobId is invalid

The problem is also that there's no way, at the moment to change the provider associated to an Asset. So as soon as the provider stop working, all the encrypted asset associated to it will "stop working" with the current logic/flow

Mitigation

A possible solution would be to allow the `governance` to be able to modify provider's config or enforce limits on the fees. This could be a timelocked operation that pass through a DAO vote if the main concern is transparency/power aggregation.

CL-XXX: `Assets.sol` `publish` should check both `stakeReq` and `stakeGracePeriod`

Tools/Technique: Manual
Difficulty+Impact: Medium

Details

At the moment the function is just checking if `stakeReq > 0` but it should force the user to pay the required stake amount only if also `stakeGracePeriod` is > 0.

Mitigation

Change the if statement `if (stakeReq > 0)` to `if (stakeReq > 0 &&stakeGracePeriod > 0)`

CL-XXX: `Assets.sol` Changes to `stakeGracePeriod` will have an impact on all previously created asset

Tools/Technique: Manual
Difficulty+Impact: Medium

Details

The same variable is shared between all the asset. A user may not be able to unstake although he was expecting to

Mitigation

Register the `stakeGracePeriod` of an asset when it's published.

NFTdClim.sol

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setGovernance`
- `setAssetsAddress`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setGovernance`
- `setAssetsAddress`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

CL-XXX: `governance` could be replaced with OZ `Ownable`

Tools/Technique: Manual
Difficulty+Impact: Information

Details

The current governance state variable could be replaced with the OZ `Ownable` contrct and you could leverage the `onlyOwner` modifier instead of `require(msg.sender == governance, 'DCL: !governance');` for better readibility

Mitigation

Evuluate to replace `governance` with `Ownable`

Assets.sol

CL-XXX: Remove unsuded state variable

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

Some variable are never used in the code and could be removed

- `LinkTokenInterface public link;`
- `IOracleClient public oracleclient;`

Mitigation

Remove those

CL-XXX: Variable can be declared as `immutable`

Tools/Technique: Manual
Difficulty+Impact: Informational/Gas opt

Details

The current variables are only initialized in the `constructor` and for this reason they could be declared as immutable to save gas

- `nftdclim`
- `providers`
- `paymentToken`

Mitigation

Declare those variable as `immutable`

CL-XXX: `skillScore` can be declared as uint256

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

Being the last variable of the struct it will be anyway treated as a `uint256`. There is no reason to declare it as a `uint16` because it won't save any gas

Mitigation

Declare `skillScore` as `uint256`

CL-XXX: `constructor` input variables need validation checks

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Be sure to check the validity of those input variables given that they are immutable (not setter)

- `NFTdClimAddress != address(0)`
- `_providers != address(0)`

Mitigation

Add check on user's input

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setTreasury`
- `setSkillScoreVerifier`
- `setStakeGracePeriod`
- `withdrawStake`
- `sweepSlashed`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setGovernance`
- `pause`
- `unpause`
- `setTreasury`
- `setSkillScoreVerifier`
- `setStakeReq`
- `setStakeGracePeriod`
- `updatePrice`
- `updateSkillScore`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

CL-XXX: `publish` should check both `stakeReq` and `stakeGracePeriod`

Tools/Technique: Manual
Difficulty+Impact: Medium

Details

At the moment the function is just checking if `stakeReq > 0` but it should force the user to pay the required stake amount only if also `stakeGracePeriod` is > 0.

Mitigation

Change the if statement `if (stakeReq > 0)` to `if (stakeReq > 0 &&stakeGracePeriod > 0)`

CL-XXX: `sweepSlashed` should check if `treasuryBalance` is not empty avoiding to waste gas

Tools/Technique: Manual
Difficulty+Impact: Low

Details

Avoid to waste case with SSTORE and transfer if the `treasuryBalance` value is `0`

Mitigation

Add a `require(treasuryBalance != 0, 'treasuryBalance empty');` to avoid waste of gas

CL-XXX: `withdrawStake` should check if `staked[_assetId]` is not empty avoiding to waste gas

Tools/Technique: Manual
Difficulty+Impact: Low

Details

Avoid to waste case with SSTORE and transfer if the `staked[_assetId]` value is `0`

Mitigation

Add a `require(oldBalance != 0, 'asset stake empty');` to avoid waste of gas

CL-XXX: `updatePrice` could disrupt marketplace buy flow

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

As explainedin the Spec Document

> *Assets which have a cost greater than 0 should be encrypted first (off-chain) by the publisher using a symmetric key and then uploaded to IPFS. The same symmetric key used to encrypt the data should then be encrypted (off-chain) with the Provider's public key (existing on the Provider's contract) and posted to the metadata of the Asset which is also uploaded to IPFS. This allows for proxy re-encryption to be done off-chain by the Chainlink Nodes' (also known as Providers) External Adapters during the Marketplace purchasing flow.*

For each asset we have two possibility when the asset is published and added to the `Asset.sol` store:

- `price === 0` document is not encrypted
- `price !== 0` document is encrypted

In the `Marketplace.sol` because of this reason when an asset is purchased via `createOrder` it will follow a different flow.

1. `assetPrice === 0` user will not pay anything and will be able to download it directly
2. `assetPrice !== 0` user needs to pay `assetPrice` to the owner of the asset and also pay an `operatorFee` to pay for the decryption mechanism process via Chainlink

Given this premise the problem is that the `price` of an asset can be arbitrary changed after the asset publishing.

We have two possible scenarios here

- T0 asset has `price === 0`, it's **not** encrypted. At T1 price is updated to `price !== 0`. This means that when a user call `Marketplace.createOrder` it will pass through the **Marketplace Flow 2)** and the user will pay for a `operatorFee` that is not needed because the asset does not need to be encrypted. Probably the decryption operation will revert or the asset will be "corrupted" because there's nothing to decrypt
- T0 asset has `price !== 0`, **it is** encrypted. At T1 price is updated to `price ===`

`0`. This means that when a user call `Marketplace.createOrder` it will pass through the **Marketplace Flow 1)**. While it's true that the user will not pay any price for the asset or `operatorFee`, he's still going to pay for gas for a purchase that will not be usable because it's encrypted.

Mitigation

There are many options to mitigate this problem. One possible easy solution that does not involve any changes from the `publisher` side (because remember that at any point in time the `owner` of the asset (that can change the price) can be different from the original `publisher`).

Add a new variable `originalEncrypted` to the `struct Asset` to track if the asset has been published as encrypted or not.

At this point you can act on `updatePrice` to disallow an originally encrypted asset to be update with a 0 price.
Otherwise you can use that property on the `Marketplace.createOrder` to skip the `decryption` flow if the `asset` was not originally encrypted and does not need that process.

CL-XXX: `updateSkillScore` does not check input parameters

Tools/Technique: Manual
Difficulty+Impact: Low

Details

- `_reportCID` should not be empty
- `_metaDataNFT` should not be empty
- `_skillScore` should be a value 0 <= x <= 5.000

Question: can these value `_reportCID` and `_metaDataNFT` possibly disrupt the asset if the operator is malicius?

Mitigation

Add checks on those parameters.

CL-XXX: `publish` does not check input parameters

Tools/Technique: Manual
Difficulty+Impact: Low

Details

- `_metadataCID` should not be empty
- `_metadata` should not be empty
- `_metadataNFT` should not be empty
- is there an onchain way to check if the asset is encrypted? Because based on that there should be a check also on the `_price`. If encrypted `_price` should be != 0

Mitigation

Add checks on those parameters.

CL-XXX: Changes to stakeGracePeriod will have an impact on all previously created asset

Tools/Technique: Manual
Difficulty+Impact: Medium

Details

The same variable is shared between all the asset. A user may not be able to unstake although he was expecting to

Mitigation

Register the stakeGracePeriod of an asset when it's published.

Marketplace.sol

CL-XXX: Variable can be declared as `immutable`

Tools/Technique: Manual
Difficulty+Impact: Informational/Gas opt

Details

The current variables are only initialized in the `constructor` and for this reason they could be declared as immutable to save gas

- `nftdclim`
- `paymentToken`
- `providers`
- `assets`

Mitigation

Declare those variable as `immutable`

CL-XXX: `constructor` input variables need validation checks

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Be sure to check the validity of those input variables given that they are immutable (not setter)

- `NFTdClimAddress != address(0)`
- `_providers != address(0)`
- `_assets != address(0)`

Additionally also `setOracleClient` should check if the input param `_oracleClientAddress` is `!= address(0)`

Mitigation

Add check on user's input

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setTreasury`
- `setCommission`
- `setOracleClient`
- `cancelOrder`
- `sweepCommission`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setGovernance`
- `pause`
- `unpause`
- `setTreasury`
- `setCommission`
- `setOracleClient`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

CL-XXX: `createOrder` could set the purchase status inline

Tools/Technique: Manual
Difficulty+Impact: Gas optimization

Details

When creating a new order you could directly set the status avoiding a possible secondary SSTORE

```
orders[numOrders] = Order(
  numOrders,
  block.timestamp,
  _assetId,
  assetPrice,
  msg.sender,
  assetPrice == 0 ? OrderStatus.COMPLETED : OrderStatus.PURCHASED
);
```

and remove `orders[numOrders].orderstatus = OrderStatus.COMPLETED;`

Mitigation

See example above

CL-XXX: `Provider`'s `operatorFee` could lead to DDOS

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

While it's true that the provider should be paid for their service (decryption) the should not be able to set the `operatorFee` without any limit.

If the operator is malicius it could set an high fee and the purchase flow would be expensive (comprared to the asset price) or even impossible (user has not enough funds).

Mitigation

There are multiple solutions:

1. add a top limit to the operator fee
2. let the govern choose a general operator fee (equal for everyone)

CL-XXX: `claimFees` considerations

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

A possible scenario could be in the case where `balances[assetId]` is not empty and the user transfer (sell) the `asset` ownership. there's not a real solution for this problem if not reverting the transfer at `Asset.sol` level if there are fees to be claimed but the UX would be not the best.

Other than that the best way is to make it very clear to the user on the `dClimate` website that the asset has fees to be redeemed.

Other possible optimization to double check:

- [double check please] reentrancy guard should be safe to be removed
- [double check please] `assetId <= assets.numAssets()` should be safe to be removed. if the assets.numAssets() is 10 and you ask for a claimFees(20, 10) the balances[20] would be 0 so _amount <= balances[assetId] will fail because would be 10 <= 0

Mitigation

See details

SkillscoreClient.sol

CL-XXX: Variable can be declared as `immutable`

Tools/Technique: Manual
Difficulty+Impact: Informational/Gas opt

Details

The current variables are only initialized in the `constructor` and for this reason they could be declared as immutable to save gas

- `assets`

Mitigation

Declare those variable as `immutable`

CL-XXX: `constructor` input variables need validation checks

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Be sure to check the validity of those input variables given that they are immutable (not setter)

- `_chainlinkTokenAddress != address(0)`
- `_assets != address(0)`

Mitigation

Add check on user's input

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setGovernance`

- `updateSkillScore`
- `fulfillupdateSkillScore`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setGovernance`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

Providers.sol

CL-XXX: `oracleClient` can be deleted and just use `oracleClientAddress` + cast

Tools/Technique: Manual
Difficulty+Impact: Gas Opt

Details

There is no necessity have both

```
/// @dev OracleClient Address
address public oracleClientAddress;

/// @dev OracleClient Contract
IOracleClient public oracleClient;
```

you could just store the `oracleClientAddress` and cast it to `IOracleClient(oracleClientAddress)` when needed

Mitigation

Remove `oracleClient` and cast the address when needed

CL-XXX: Variable can be declared as `immutable`

Tools/Technique: Manual
Difficulty+Impact: Informational/Gas opt

Details

The current variables are only initialized in the `constructor` and for this reason they could be declared as immutable to save gas

- `link`

Mitigation

Declare those variable as `immutable`

CL-XXX: `constructor` input variables need validation checks

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Be sure to check the validity of those input variables given that they are immutable (not setter)

- `_linkTokenAddress != address(0)`

Mitigation

Add check on user's input

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setMarketplace`
- `setOracleClient`
- `depositLink`
- `withdrawLink`
- `subtractLink`
- `setProviderOracleOperatorFee`
- `setOperatorFee`, track bad behaviour
- `setEnabled`
- `updateProviderJobOracle`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setMarketplace`
- `setOracleClient`
- `setGovernance`
- `setProviderOracleOperatorFee`
- `setOperatorFee`
- `setEnabled`
- `updateProviderJobOracle`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

CL-XXX: Disallow creation of providers from the same `msg.sender`

Tools/Technique: Manual
Difficulty+Impact: Low

Details

From the specifc doc

> *A Provider is an on-chain representation of a Chainlink Node which provides proxy re-encryption services for assets with a price > 0 on the Marketplace via External Adapters. It is expected that many Data Publishers will be Providers on the dClimate platform in order to maintain ownership (and sole decryption access) to their data.*

So providers are "asset agnostic" and there's no reason to have multiple providers from the same user (`msg.sender`). At some point in the future the `Gov` would be able to have more control on providers and allowing them to "spam-create" new providers instance will make this task more difficult.

Mitigation

Disallaow the creation of new providers from the same user checking if `msg.sender` has already created a provider.

CL-XXX: `createProvider` miss input validation

Tools/Technique: Manual
Difficulty+Impact: Low

Details

- `oracleAddress != address(0)`
- `jobId` not empty
- `name` not empty
- `firstHalfProviderPublicKey` not empty
- `secondHalfProviderPublicKey` not empty
- `oracleOperatorFee` should be a min/max fixed value based on the cost of the operation?
- `operatorFee` could lead to possible DDOS of marketplace operation if high

Note that the same concern goes into the `setOperatorFee` where the provider can set an arbitrare value for its own fee that must be paid by the marketplace buyer

Mitigation

Implement input validation

CL-XXX: `updateProviderJobOracle` miss input validation

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

- `_oracleAddress != address(0)`
- `jobId` not empty

Mitigation

Implement input validation

CL-XXX: General concerns about `Publisher` logic/flow

Tools/Technique: Manual
Difficulty+Impact: Critical

Details

Publisher have a huge role in the dClimate protocol because they are required in order to decrypt encrypted assets when the asset's price is greater than 0. There's not a requirement that a `Provider` must be the same entity of the one that publish the `Asset`.

In the current implementation, as soon as a provider has been created, the provider's owner (the one that have called the `createProvider`) have the total control on it.

Both the `governance` or the contract's ADMIN have no power on the created provider.

This mean that at any point in time the Provider could DDOS the service making all the assets associated to the provider unabled to be purchased by the end user.

These are all the possible things that could make revert a purchse tx:

- provider is disabled
- provider has not enough link funds
- operatorFee is high
- oracleOperatorFee is not enough high to cover cost
- oracleAddress is empty
- jobId is invalid

The problem is also that there's no way, at the moment to change the provider associated to an Asset. So as soon as the provider stop working, all the encrypted asset associated to it will "stop working" with the current logic/flow

Mitigation

A possible solution would be to allow the `governance` to be able to modify provider's config or enforce limits on the fees. This could be a timelocked operation that pass through a DAO vote if the main concern is transparency/power aggregation.

CL-XXX: governance and related modifier overlap with the herited AccessControl

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

governance is set as a variable to control access to administrative functions, all of which can be done using OpenZeppelin's AccessControl

Mitigation

Remove governance and onlyGovernance, use a custom role and hasRole in require statements

CL-XXX: link can be set as constant

Tools/Technique: Manual
Difficulty+Impact: Gas optimization

Details

See title, the address for link should not change and the same one can be used accross all Providers contracts on a chain

Mitigation

```
address constant LINK = ...;
```

CL-XXX: Checking allowance in depositLink()

Tools/Technique: Manual
Difficulty+Impact: Low

Details

The function requires the allowance of the sender to be sufficient before calling safeTransferFrom. The transfer will revert anyway if there is not enough allowance

Mitigation

Remove

```
require(
            link.allowance(msg.sender, address(this)) >= _amount,
            "DCL: !linkApproved"
        );
```

OracleClient.sol

CL-XXX: Variable can be declared as `immutable`

Tools/Technique: Manual
Difficulty+Impact: Informational/Gas opt

Details

The current variables are only initialized in the `constructor` and for this reason they could be declared as immutable to save gas

- `dclimateMarketplace`
- `link`

Mitigation

Declare those variable as `immutable`

CL-XXX: `constructor` input variables need validation checks

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Be sure to check the validity of those input variables given that they are immutable (not setter)

- `_chainlinkTokenAddress != address(0)`
- `_dclimateMarketplace != address(0)`

Mitigation

Add check on user's input

CL-XXX: Missing events emission

Tools/Technique: Manual
Difficulty+Impact: Information

Details

Critical functions are missing event emission

- `setGovernance`

- `setProviders`

Mitigation

Crete the event and add an `emit` instruction to those functions

CL-XXX: Avoid event spamming

Tools/Technique: Manual
Difficulty+Impact: Information

Details

You could spare some gas and eventually avoid event spammin checking if the new value is equal to the new old one and revert in that case. Spamming of events could create problems when you will need to monitor them on monitoring tools

- `setGovernance`
- `setProviders`

Mitigation

On every setter add a check like `require(_newValue != oldValue, "values are eq")` to prevent event spamming

CL-XXX: Wrong comment for `setProviders`

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

The `setProviders` function has the current comment

```
    /// @notice Function for setting the marketplace address
    /// @param _providers The Marketplace contract address
```

The function is not setting the `marketplace` address but the `providers` contract address

Mitigation

Replace `marketplace` with `providers`