COBrA - Fair COntent Trade on the BlockchAin

Project report

Alessandro Pagiaro

June 2018

1 Introduction

Cobra is a Content Management System. It is composed of several different modules:

- The catalog, that is responsible for searching and managing the access right
- The contents, the contracts that extend the `BaseContentManagement` contract, are responsible for distributing the content to the users that require the accesses. [1]

2 Catalog

In order to illustrate the `Catalog` contract, I provide some use cases that explicit the logic of the contract.

2.0.1 Publish a content

Given a ”content address”, the address at which a contract that inherits from `BaseContentManagement` is deployed, you can invoke the function `publishContent(address _addr)`.

This function checks that the content has no already catalog address set and the `views` variable is set to 0. This prevents some malicious attacks that can be exploited by changing those controls variables (`views` and `catalog`). If those checks are passed, the function pushes the content into the `contentsName` array that stores all the names of the contents, and saves the content address into the `mapping(bytes32 => address) name2address`.

Gas cost In my test the deploying and publishing of a content cost 103546 GAS units. Considereing the gas cost equal to 16Gwei [2], the total cost is around

[1] In test scenario is used the `BaseContentManagement` itself to deploy the contents.

[2] This cost is assumed in future conversion in this report

0.70 EUR.

2.0.2 Get premium subscription

A user, that would have to buy a premium subscription, have to invoke the `buyPremium()` function with `premiumCost` wei as the value of the transaction. This function checks that the value paid is correct, then set the `premiumEndBlockNumber` mapping for this specific user to the `block.number + premiumTime`.

I'd like to prefer to store the `premiumEndBlockNumber` instead of `premiumStartBlockNumber` in order to manage the case when a user extends the premium subscription before the expiration of the previous one.

Consume a content as premium user After receiving a premium subscription, in order to access to a content, a user has to retrieve the content address, invoking the `name2address` getter that is generated due to the `public` visibility of the mapping. Given that address, a user has to invoke, to the content contract, the function `getContentPremium()` that checks if the user is a premium user and then returns the content.

To avoid a malicious user to get the access right on all contents in order to consume the content after the subscription expiration, for the premium user the granting access function and the consuming function is fused into the `getContentPremium()` [3].

2.0.3 Buy content access without premium subscripion

In order to consume a content, a non-premium user has to buy the access right invoking the function of the catalog `grantAccess(bytes32 _content)`. This function set a content mapping variable to `true` for the user that sends the transaction associated, in fact the access right are stored by the content itself. Given the access right, a user has to retrive the content address (using the public mapping of the Catalog `name2address`) and then, invoke the content function `getContent` that checks for the access right and returns the content. The function `getContent`, moreover, invokes the function `Catalog.consumeContent(bytes32 _name)`. This function is a handler function that increase the catalog `totalViews` variable used into the `Catalog.goodbye()` function (the selfdestruct wrapper) in order to avoid a cycle into this function and trigger, if it necessary, the payment cycle. [4]

Gas cost The cost for executing this functionalities, composed by the function `grantAccess(_c)` and `Content.getContent()`, is 62527 Gas units (value computed as mean of 100 runs), that is around 0,43EUR. This cost may appear

[3] Since there are right access for all the contents, this function check only if a user has a valid premium subscription

[4] The gas cost for this cycle is paid by the unlucky user that trigger the magic view number. This problem will be resolved in the project exam.

high but this cost is influenced by an internal function to manage views counts and other instances variabile.

If a user doesn't like to pay so much Ether for getting content, with a premium subscription can save money since, given a premium subscription, the consuming content costs only 37680 Gas (due to the fact that the view are not incresed), that is around 0,25EUR.

2.1 Searching a content

In order to search a content you have several functions:

- `getContentList()`, that returns all the content's name
- `getNewContentList(uint _n)`, that returns all the _n most recent content added
- `getLatestByGenre`, that iterates over the `contentsName`, accesses to the content and checks its genre. It may require a big execution gas cost but, in general, is executed on the local node so doesn't cost ETHER.
- `getMostPopularByGenre(uint _g)`, returns the most popular content of the genre _g. Genre _g is expressed as `enum` type into the `BaseContentManagement` and is passed as basic type `uint` as a the parameter and is casted later.

2.2 Gift something

In order to gift a premium subscription, a user can simply call the `giftPremium(address _user)` function. This function performs the same control of the `buyPremium` but sets the user `_user` as the beneficiary of the premium features.

In order to gift a content access, the function to be invoked `function giftContent(bytes32 _contentName, address _userAddr)` that is similar to the `giftPremium(address _user)`.

3 The BaseContentManagement

The contract base for the content is the `BaseContentManagement`. This contract implements all the function that is required to support the communication between the user and the content's access right.

The `Genre` is expressed as enumeration, the `author` as a string, since it can be long (more than 32 bytes). The Content stores also its views, used by Catalog to distribute the final balance and the access right (set by the catalog using `Content.grantAccess(_user)`, restricted by the modifier `onlyCatalog()`). In order to block malicious users, I've tried to implement some controls on the content when it is published (already explained in the previous section), these controls invoke the `setCatalogAddress` that is a function that saves the catalog address into the `catalog` variable. If the catalog is already set, the function returns without modifing the state. This prevents some malicious user to change the inner state of the content.

3.1 Get the money earned

Since the `Catalog` pays the `Content` and not the address that deployed the Content, the functions `seeBalance()` and `withdraw()` are added to the content. This function can be used to retrieve the money from the `Content` when the Catalog destruct itself.

4 Consideration on implementation

Gas consumed between getContent and search for a content I've tested several implementations before to choose the actual one. I prefer to do not store some variable into the catalog in order to decrease the access time to the most popular content since the loops in the view are free and the function view are useful only for the user and are not called inside other functions.

Bytes32 instead of String When a content is searched, it is returned as bytes32 instead of the address. This choice is done because the bytes32 is more significative of string and, moreover, this statically allocated space cost less gas then the string.

The user identifies the contract from its name, and can retrieve the content address using the function `Catalog.name2address(bytes32 _name)`, an automatic getter that is generated by Solidity since the `name2address` variable is public. The content address is returned without any controls on the access right because the content consuming itself implements the controls, so having the address of the content doesn't give any security issuee that permits to retrieve the content without cheching the Catalog access rights.

4.1 Parameter of the system

A cost for a content is 0.002 ETH, that is, at the current change, 1EUR. The premium cost is 100 times a single content cost. The premium subscription duration is 10000 blocks, that, with the average mining time equal to 14 seconds it is 40 hours.

The views counter is `uint64`, since, if Youtube can use 64 bits for storing its views counter, my project will not exceed this number, absolutely.

The payment to the content is trigged every 1000 total views (w.r.t. Catalog consumeContent, not related to Content). The payment is performed into the `Catalog.consumeContent()` function and reset the local view counter for every content in the same function that uses for send ETH to them [5].

The premiumTime is measured in block height since a timestamp can be manipulated by the miners but block height is defined by the network and cannot be changed.

[5] A content has two variable, one used for counting the total views, one for the view not yet paid

4.2 Test

The catalog is tested with a battery of test that can be run with `truffle test`. They require to have Ganache installed and running (or the config file have to be edited in order to use truffle develop blockchain). The test requires around 4 minutes to be completed and it will deploy 100 content in the catalog and performs 125 pay to view accesses and 100 premium accesses. An output is provided with the testresult.txt (due some random access your output will be different).