Lecture 22 - risk transfer contract

What is a Credit Default Swap

Let's say you purchase a bond from Turkey (sovereign debt) that pays 6% per year interest. You put in $1 million on a 5 year term. So you are expecting to get $60,000.00 payed to you each year.

Only now it looks like Turkey may default on it's bonds. To cover this "risk" you want insurance that if they default that you get your million back.

This is called a "credit default swap" and you have to pay for this "insurance" or "risk transfer". You pay somebody willing to take the risk that Turkey defaults and they give you your million (the swap) if there is a "credit default". Hence the name.

Last month the going rate was 6.77% per year for a credit default swap on Turkey's sovereign debt. So instead of you getting $60,000.00 a year in income you now pay $7,700.00 a year to grantee that you get your capital returned.

This is a good indicator of a country's risk - if the credit default swaps cost more than the yield on the bond then you have a "bad risk".

Your other option is to sell the bond. Current the price for Turkey $1 million bonds is around $782,000 - so you can loose almost ¼ million - and sell now, or you can take out insurance.

Good money destroyers bad money

Turkey's internal velocity of the Lira has been drooping, while the velocity of Euro's and the Dollar in Turkey has been rising. Currently around 40% of transactions are Dollar denominated in Turkey. This is with an inflation rate of 7.83% last month - that is an annualized rate of over 100%. What this probably means is that Turkey is going to loose control of the national currency and end up being a Dollar denominated country - there are a bunch of them around the world - like the Republic of Zimbabwe.

The rule to remember is that "good money will always destroy bad money". This is totally true with crypto also. If you have good and bad crypto then all people will flee to a good crypto.

Let’s talk about a “Disintermediation” coin.

How about “beef”.

Location of Cow	% Charged	profit
cow/calf ranch	5.2%	$1.47lb in 1974, $1.52 today.
truck	8%	
backgrounder	6%	
truck	8%	
auction lot	7.5%	(takes 2% to 3% of price of cow)
truck	8%	
feed lot	5%	
truck	3%	
meat packer	4.8%	
cold storage	4.1%	
truck	7%	
warehouse	5.1%	
truck	6%	
retailer (store)	5%	$2.91 in 1974, $6.52 today (ground beef).
		$18.24 a LB for steak

Total risk is about 3% of cow dying, getting sick etc.

So add prime rate 1.2% + 3% = total risk/cost of beef should be 4.2%.

Observations:

1. The trucking companies make more than the rancher.
2. The banks make *way* more than the rancher.
3. Most costs in the supply chain have gone down.
4. The “meat packer/cold storage” industry has consolidated (5 major packers)
5. Retailers run a 1% profit margin on a guaranteed sale.
6. Trucking has gotten “less” expensive. Trucks are more fuel efficient - drivers are payed less.

How about disintermediation using a pre-purchase token. Instead of organizing the business around different layers - organize it around the product line. The cow is payed for at conception - and the entire lifetime of the cow is financed once.

Remove the “auction lot” completely. A 2-3% saving and less shipping.

My best guess is that the banks are making $308.00 per cow, the rancher is making $28.41 per cow. The backgrounder making $22.08. The retail store $8.80. Trucking makes (shipped 6 times) $76.91 or about $11 per shipment per cow. The “meat packer” making around $224.00.

Searching Events on the Chain

Ethereum uses a bloom filter to index events to transactions.

What is a “bloom” filter.

A bloom filter uses an input,

```
        eventSignature := []byte("DataSet(bytes32,bytes32)")
```

and then takes multiple hashes of this into a bit set. If all the bits are ‘1’ for the hashes then it is likely that the item can be found at that location.

Code for Searching for Events.

The contract that generates an event:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract GenEvent {
    event DataChanged(bytes32 key, bytes32 value);

    mapping (bytes32 => bytes32) public savedData;

    constructor() {}

    function setData(bytes32 key, bytes32 value) external {
        savedData[key] = value;
        emit DataChanged(key, value);
    }
}
```

The Config

```
{
        "contract_address": "0x79a2F028b5150eaDD6619fd7f7C3f92Fa41B7a2a",
        "_rinkeby_client_wss_url": "wss://rinkeby.infura.io/ws",
        "client_wss_url": "ws://127.0.0.1:8545"
}
```

The search code in .go

```
package main

// MIT Licensed -- Based on original code
// https://goethereumbook.org/en/event-read/
// the code has been fixed to work with 8.1 solc and Eth 1.10.x

import (
    "context"
    "fmt"
    "log"
    "math/big"
    "strings"

    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/accounts/abi"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/crypto"
    "github.com/ethereum/go-ethereum/ethclient"

    gen_event "github.com/Univ-Wyo-Education/S22-4010/class/lect/22/scan-event/eth/contracts"
)

// These are the values pulled in from ./cfg.json file.
type GlobalConfig struct {
    ContractAddress string   `json:"contract_address"`
    ClientWSSUrl    string   `json:"client_wss_url"`
    DbFlags         []string `json:"db_flags"`
}

var gCfg GlobalConfig
var DbOn map[string]bool = make(map[string]bool)

func main() {

    // ---------------------------------------------------------------------------------------
    // Read global config
    // ---------------------------------------------------------------------------------------
    ReadJson("cfg.json", &gCfg)
    if len(gCfg.DbFlags) > 0 {
        for _, x := range gCfg.DbFlags {
            DbOn[x] = true
        }
    }

    for _, k := range gCfg.DbFlags {
        DbOn[k] = true
    }

    if DbOn["dump-global-config"] {
        fmt.Printf("Global Config:%s\n", SVarI(gCfg))
    }

    client, err := ethclient.Dial(gCfg.ClientWSSUrl)
    if err != nil {
        log.Fatal(err)
    }

    // ---------------------------------------------------------------------------------------
    // Process contract
    // ---------------------------------------------------------------------------------------
    contractAddress := common.HexToAddress(gCfg.ContractAddress)
    query := ethereum.FilterQuery{
        FromBlock: big.NewInt(2394201),
        ToBlock:   big.NewInt(2394201),
        Addresses: []common.Address{
            contractAddress,
        },
    }

    // ------------------------------------------------------------------------------------------
    // Search Chain / Logs
    // ------------------------------------------------------------------------------------------
    logs, err := client.FilterLogs(context.Background(), query)
    if err != nil {
        log.Fatal(err)
    }

    // Generated Code -- Pull in the ABI for the contract.  contractAbi, err := abi.JSON(strings.NewReader(stri
    contractAbi, err := abi.JSON(strings.NewReader(string(gen_event.GenEventMetaData.ABI))) // xyzzy
    if err != nil {
        log.Fatal(err)
    }

    for _, vLog := range logs {
        fmt.Printf("0x%x\n", vLog.BlockHash.Hex()) // 0x3404b8c050aa0aacd0223e91b5c32fee6400f357764771d0684fa7b
        fmt.Printf("%v\n", vLog.BlockNumber)       // 2394201
        fmt.Printf("0x%x\n", vLog.TxHash.Hex())    // 0x280201eda63c9ff6f305fcee51d5eb86167fab40ca3108ec784e865

        event := struct {
            Key   [32]byte
            Value [32]byte
        }{}
        _, err := contractAbi.Unpack("DataSet", vLog.Data) // err := contractAbi.Unpack(&event, "DataSet", vLog
        if err != nil {
            log.Fatal(err)
        }

        fmt.Printf("0x%x\n", string(event.Key[:]))
        fmt.Printf("0x%x\n", string(event.Value[:]))

        var topics [4]string
        for i := range vLog.Topics {
            topics[i] = vLog.Topics[i].Hex()
        }

        fmt.Printf(topics[0]) // 0xe79e73da417710ae99aa2088575580a60415d359acfad9cdd3382d59c80281d4
    }

    eventSignature := []byte("DataSet(bytes32,bytes32)")
    hash := crypto.Keccak256Hash(eventSignature)
    fmt.Printf("%s\n", hash.Hex()) // 0xe79e73
}
```