Detecting Ponzi Schemes on Ethereum: Towards Healthier Blockchain Technology
W. Chen et al.

Sajjad Heydari

Nov 2020

1 Introduction

2 Case Study

3 Data Gathering
- Account Features
- Code Features

4 Experiment
- Model
- Evaluation
- Results
- Feature Analysis
- Comparison With Previous Experiments
- Application

5 Discussion

Ponzi Scheme

- Fraudulent investment operation
- Return of older investments is paid by new investments
- Many participants will lose most of their invested money

Ponzi Scheme

- Fraudulent investment operation
- Return of older investments is paid by new investments
- Many participants will lose most of their invested money
- This paper studies Ponzi schemes on Ethereum Smart Contracts
- Extracts features from user accounts and operation codes
- Builds a classifier

Smart Ponzi Scheme

- Ponzi schemes implemented using smart contracts
- Smart contracts are automatically enforced and can not be terminated on blockchain
- This gives confidence in continuously paying back the investors

Ethereum Recap

- Ethereum runs on EVM Byte Code
- An assembly like system consisting of opcodes[1]
- Smart contracts are created by sending a special transaction to zero-account
- Some creator publish high level codes publicly, but they don't have to
- They could be triggered by anyone until the creator kills them

[1] such as PUSH 1, ISZERO, CALLDATASIZE

1 Introduction

2 Case Study

3 Data Gathering
- Account Features
- Code Features

4 Experiment
- Model
- Evaluation
- Results
- Feature Analysis
- Comparison With Previous Experiments
- Application

5 Discussion

Rubixi

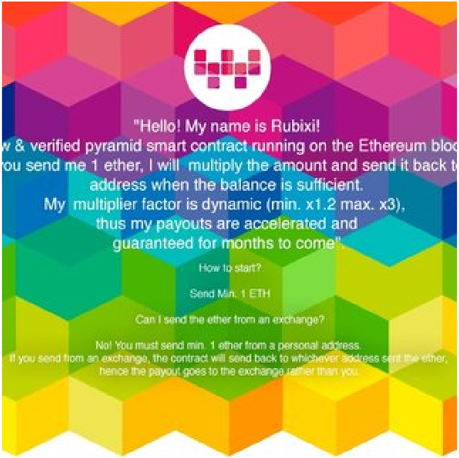


Figure 1: Rubixi (source: Bitcoin Talks)

Rubixi Contd.

- Promise of 20% to 200% profit
- 112 participants
- Only 22 made a profit
- With two participants taking more than 40% of the incomes

1 Introduction

2 Case Study

3 Data Gathering
- Account Features
- Code Features

4 Experiment
- Model
- Evaluation
- Results
- Feature Analysis
- Comparison With Previous Experiments
- Application

5 Discussion

Features

- **Account Features** looking at Ether flow, Ether amount, etc.
- **Code Features** looking only at the decompiled byte code

Features

- **Account Features** looking at Ether flow, Ether amount, etc.
- **Code Features** looking only at the decompiled byte code
- Gathered from Etherscan.io

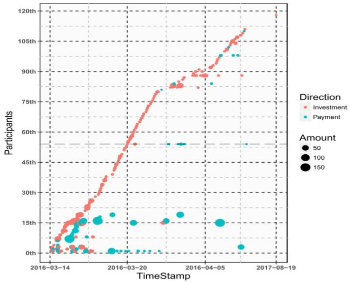
Account Features

- Ether transfer pattern
- Some accounts receive more counts of payment than investment
- Balance of contract is low to maintain image of fast and high returns

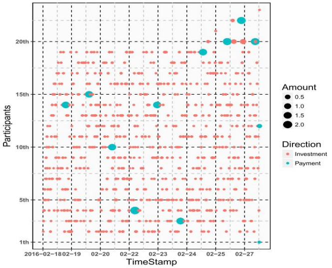
Ether Flow Graph

- Focus on a contract
- Display transactions overtime between the contract and participants
- Color coding to disambiguate direction

Ether Flows



(a) Ether Flow of Rubixi



(b) Ether Flow of LooneyLottery

Figure 2: Comparison of Ether flows between a Ponzi scheme and a lottery system

Observation #1 Ether Flow

- Payment happens after investment
- Investments don't have followed payments
- Some participants have more payment

Key Account Features

- **Known rate**, proportion of receivers who have invested before payment
- **Balance** of the contract
- **Number of investments**
- **Number of payments**
- **Paid rate**, proportion of investors who have at least one payment
- **Number of max payment**, the maximum count a participant received payment
- **Difference Index**

Difference Index

Create difference vector $v[i] = n_i - m_i$, difference between investment and payment

$$D_ind = \begin{cases} 0 & \text{if } v = 0 \text{ or } p \leq 2; \\ \text{skewness of } v & \text{otherwise} \end{cases} \tag{1}$$

Account Features Stats

Ponzi Scheme contracts							
	Kr	Bal	N_inv	N_pay	D_ind	Pr	N_max
Mean	0.89	4.65	56.84	92.49	-1.04	0.62	36.12
Median	1.00	0.26	17.00	21.00	-0.65	0.66	11.00
Sd	0.29	15.51	119.41	204.71	1.95	0.30	94.36
Non-Ponzi Scheme Contracts							
	Kr	Bal	N_inv	N_pay	D_ind	Pr	N_max
Mean	0.49	22319.60	653.44	540.74	-0.51	0.43	237.95
Median	0.50	0.10	6.50	4.00	0.00	0.40	2.00
Sd	0.43	187549.23	3986.45	2195.42	6.05	0.41	1095.08

Figure 3: Account Features Stats

Code Features

- EVM bytecodes are in binary format
- Opcodes could be extracted by decompiling the bytecodes
- The authors want to explore the opcode frequency

Opcode Frequency

Figure 4: Opcode frequency clouds of Rubixi(left) and LooneyLottery(right)

For readability, PUSH, DUP and SWAP are removed

JUMPI vs. JUMP, TIMESTAMP

1 Introduction

2 Case Study

3 Data Gathering
- Account Features
- Code Features

4 Experiment
- Model
- Evaluation
- Results
- Feature Analysis
- Comparison With Previous Experiments
- Application

5 Discussion

Experiment

- **Objective**: Determine whether a smart contract is running Ponzi scheme
- **Input**: Account and/or code features
- **Output**: Probability of the smart contract being a Ponzi scheme
- **Justification** As shown in previous section, Ponzi schemes have distinguishing features in both account features and code features

XGBoost[2]

- $f(x) = \omega_{q(x)}, \omega \in R^T, q : R^d \rightarrow \{1, 2, ..., T\}$
- ω is the leaf tree weight and q is the tree structure

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

[2] Extreme Gradient Boosting

Model

- Avoids overfitting by design
- Objective function: $Obj(\theta) = L(\theta) + \Omega(\theta)$
- $L(\theta) = \sum_i [y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i})]$, is the logistic loss
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$, measures tree complexity

Evaluation

- 80% training and 20% testing
- 5-fold cross validation
- **Precision** $\frac{\text{true positive}}{\text{true positive}+\text{false positive}}$
- **Recall** $\frac{\text{true positive}}{\text{true positive}+\text{false negative}}$
- **F-Score** $2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$

Results

Features	Precision	Recall	F-Score
Account	0.74	0.32	0.44
Opcode	0.90	0.80	0.84
Account + Opcode	0.94	0.81	0.86

Table 1: Feature Performance Comparison

Observation #2 Performance

- **Account features did not perform good**

Observation #2 Performance

- **Account features did not perform good**
- This could be due to experimental smart contracts
- Or smart contracts that have no transactions
- Or we need more account features

Observation #2 Performance

- **Account features did not perform good**
- This could be due to experimental smart contracts
- Or smart contracts that have no transactions
- Or we need more account features
- **Opcode features performance is much better**

Observation #2 Performance

- **Account features did not perform good**
- This could be due to experimental smart contracts
- Or smart contracts that have no transactions
- Or we need more account features
- **Opcode features performance is much better**
- They can be used independently to find Ponzi schemes

Feature Importance

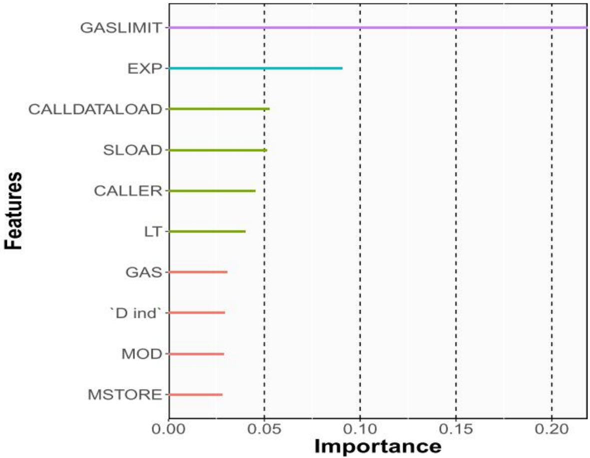


Figure 6: Feature Importance Among the Ten Most Significant Features

Opcode description

Opcode	Description
GASLIMIT[3]	Get the block's gas limit
EXP	Exponential operation
CALLDATALOAD	Get input data of current environment
SLOAD	Load word from storage
CALLER	Get caller address
LT	Less-than comparison
GAS	Get the amount of available gas
MOD	Modulo remainder operation
MSTORE	Save word to memory

Table 2: Important Opcode and Their Descriptions

[3] Was used in Oracle's API

Comparison With Previous Experiments

- Compared with Bartoletti et al., that compares similarity across bytecode files
- 45 out of 54 (83%) were correctly marked, (9 errors)

Comparison With Previous Experiments

- Compared with Bartoletti et al., that compares similarity across bytecode files
- 45 out of 54 (83%) were correctly marked, (9 errors)
- Two were mislabeled by Bartoletti et al.
- Two had irregular interactions (only with creator, and to unknown addresses)
- Two had only one transaction with amount larger than zero, the rest were only function calls
- Two had reverse payment order, the anterior participants receive payments later than posteriors
- The last one looks like Ponzi, but has more payments than investments so it is hard to conclude

Application

- The model could be used to estimate the number of smart Ponzi schemes on Ethereum
- They gathered all contracts created before May 7, 2017, totaling to 280,704
- Detected 386 smart Ponzi Schemes (including the previously verified)
- Estimately $434 (= 386 \times$ precision/recall) smart Ponzi schemes exist, i.e., 0.15%
- The latest studies claimed only 191, so the problem is more serious than previously thought

Probability of the Detected Schemes

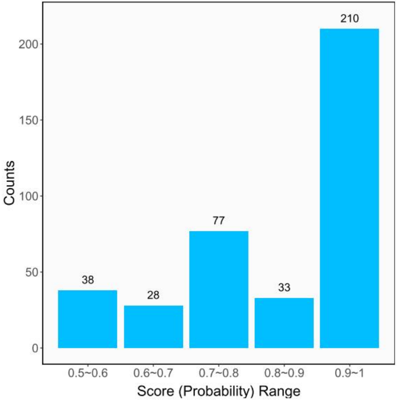


Figure 7: Probability of the Detected Schemes

1 Introduction

2 Case Study

3 Data Gathering
- Account Features
- Code Features

4 Experiment
- Model
- Evaluation
- Results
- Feature Analysis
- Comparison With Previous Experiments
- Application

5 Discussion

Future Works

- Extending ground truth and improving the classification results
- Cover other kinds of scams
- Provide a platform to evaluate and score every smart contract and gather user reports

Review

- Full of novel ideas (Ether Flow Graph, Difference Index, etc.)
- Great visualization
- Typos in two different links and references

Questions for the audience

- Why didn't the author use the source code directly?
- Can this classifier detect schemes across multiple contracts?

QA

Thank you!