BreachRadar: Automatic Detection of Points-of-Compromise

Miguel Araujo* Miguel Almeida† Jaime Ferreira† Luis Silva† Pedro Bizarro†

Abstract

Bank transaction fraud results in over $13B annual losses for banks, merchants, and card holders worldwide. Much of this fraud starts with a Point-of-Compromise (a data breach or a "skimming" operation) where credit and debit card digital information is stolen, resold, and later used to perform fraud. We introduce this problem and present an automatic Points-of-Compromise (POC) detection procedure. BreachRadar is a distributed alternating algorithm that attributes to the different possible locations a probability of being compromised. We implement this method using Apache Spark and show its linear scalability in the number of machines and transactions. BreachRadar is applied to two datasets with *billions* of real transaction records and fraud labels where we provide multiple examples of real Points-of-Compromise we are able to detect. We further show the effectiveness of our method when injecting Points-of-Compromise in one of these datasets, simultaneously achieving over 90% precision and recall when only 10% of the cards have been victims of fraud.

1 Introduction

FICO estimates European losses to fraud at €1.6 billion in 2014 [7] and global fraud in 2013 was estimated at $13.9 billion [16]. If the cost of lost merchandise as well as redistribution costs are included, fraud is estimated to account for 1.32% of the revenue for the average merchant in the US and it's higher if the merchant operates globally [14].

A Point-of-Compromise (POC) is a (physical or virtual) location of the payment network, such as an ATM or a point-of-sales terminal, that processed or collected payment information and that was compromised by fraudsters. In a classical scenario, the victim's card is swiped in a small rogue device (possibly installed in an ATM or vending machine, or used by malicious employees whenever the card leaves the owners' sight at a bar, restaurant or gas station) that reads and stores the magnetic stripe data which is then, e.g., sold and written on a new cloned card. However, this is not the only scenario: data breaches are also common POCs which might occur at the merchant or even payment-processor level.

Given the increase in both the number of data breaches and in the number of cards affected (Target's 2013 data breach alone exposed an estimate of 40 million cards [13]), early and accurate detection of POCs is not only vital for fraud prevention, but could also lead to a decrease in the expected loss from these breaches, reducing their frequency. The timely discovery of a Point-of-Compromise could prevent the fraudulent use of other cards compromised at the same location and early detection could prevent thousands of fraud cases.

As an example, Figure 1a illustrates the weekly savings when BreachRadar is applied to one of the two real datasets we explore. By automatically canceling cards that were used in locations marked as potentially compromised, and even after assuming a $10 reissue cost per card, our system is able to prevent over $2 million USD in credit-card fraud in a period of just 6 weeks.

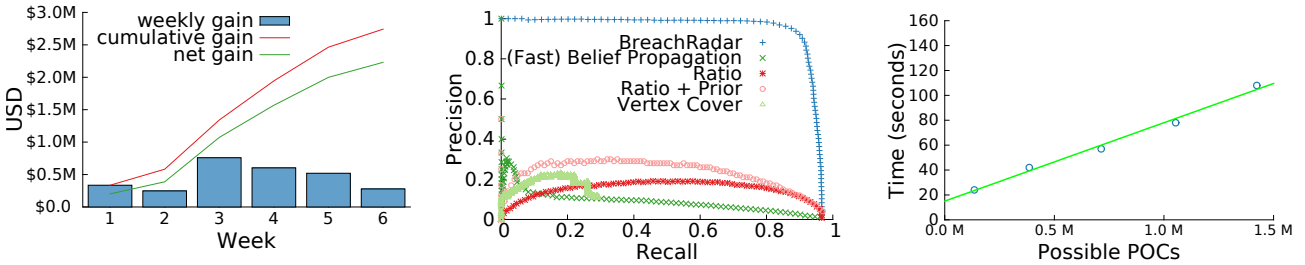
The main contributions of this paper can be summarized as follows:

1. **Point-of-Compromise Problem.** We formulate a novel and important POC detection problem.
2. **Effectiveness.** BreachRadar accurately identifies *Points-of-Compromise*, achieving over 90% precision and recall when only 10% of the stolen cards have been used in fraud (Figure 1b).
3. **Distributed POC-Detection algorithm.** We provide a scalable distributed algorithm for POC detection in big datasets (Figure 1c).

Further Applications. While we focus on the identification of *Points-of-Compromise* in bank transactions, there are other domains where BreachRadar could help identify malicious or abnormal activity. We invite the reader to consider any situation where individual nodes might trigger abnormal behavior in their neighbors. Consider anti-virus applications and machine-file bipartite graphs: given malware symptoms in some of the machines, a small set of files that exist in common could be formulated as *Point-of-Compromise* detection problem. Similarly, quick identification of food-poisoning sources or of faulty parts in the car industry can be formulated under this setting.

*Carnegie Mellon University and INESC-TEC.
maraujo@cs.cmu.edu .

†Feedzai.
miguel.almeida@feedzai.com, jaime.ferreira@feedzai.com, luis.silva@feedzai.com, pedro.bizarro@feedzai.com



(a) **Significant savings,** over a period of 6 weeks (red), even assuming a $10 reissue cost per card (green).

(b) Almost perfect **precision and recall, significantly higher than competing methods.**

(c) **Linear speed-up** with number of possible Points-of-Compromise.

Figure 1: BreachRadar is **fast and effective**.

2 Background and Related Work

While apparently a simple problem, several reasons compound to make the automatic detection of POCs a challenge to naive approaches: a) the variety of *Points-of-Compromise*, e.g., database breaches, card skimming devices, etc; b) the variety of time granularities, e.g. database breaches compromise months of transactions, while an employee skimming cards might do it for a single day; c) the lack of ground-truth labels, as fraudulent charges reports do not identify the origin of the compromise and d) the scale of the problem, as datasets with billions of transactions with millions of possible *Points-of-Compromise* are common.

2.1 Summary Table 1 characterizes the most relevant methods described in this section. We analyze a method's ability to *find Points-of-Compromise* and to *scale* at least quasilinearly with the number of transactions that need to be processed. We consider that a method has proper *risk assessment* if it doesn't believe that more transactions to safe merchants reduce the probability that the card might have been stolen at a single compromised location. We consider methods to be *Blame-aware* if they acknowledge that cards are likely stolen only once, so they should not significantly contribute to an increased POC likelihood of multiple locations. We consider a method to be *Confidence-aware* if it incorporates the idea that more evidence improves the confidence of a POC label[1]. Finally, a method is capable of *Early Detection* if it shows high recall even when only a small percentage (e.g., 10%) of the cards at a location are *fraud-cards* (a card with at least one fraudulent transaction).

[1] As an example, we are more confident that a location has been compromised if 200 out of 600 cards who transacted there were victims, than if 3 out of 6 were.

2.2 Real-time Fraud Detection While not able to identify *Points-of-Compromise*, state-of-the-art fraud detection solutions merge statistical, machine learning and data mining tools in order to create models that estimate the fraud probability of individual transactions in real-time. For further information on this orthogonal problem, we refer the reader to specific literature [4, 5].

2.3 Points-of-Compromise Simple metrics are unable to provide an appropriate measure for the likelihood of a point being the origin of a compromise. Ranking locations by the number of *fraud-cards* with which they interacted does not work, as many merchants process many transactions and thus interact with a high number of *fraud-cards*. The ratio of *fraud-cards* shouldn't be used either, as the majority of the locations have small numbers of transactions and high ratios (by chance) do not imply a compromised location.

Current systems for *Point-of-Compromise* detection are typically hindered by these issues. Absolute number of *fraud-cards* and *fraud-card* ratios are commonly used [11, 18], perhaps coupled with time-windows [22] to restrict the set of transactions considered. Arbitrary thresholds that indicate whether a merchant was compromised need to be defined, but suggestions have been made that supervised classification algorithms could also be trained, after information about which merchants were in fact compromised is obtained [8]. A different approach suggests comparing recent fraud-rates at each merchant with their historical fraud-rate and flagging outlier deviations [24].

2.4 Guilt-by-association Semi-supervised learning techniques, in particular graph-based methods such as Label Propagation, could be used to label nodes as *compromised* or not-*compromised* in the network; labeled nodes *influence* neighbors according to an Homophily Matrix which establishes whether nearby nodes tend to

Table 1: Comparison of BreachRadar with other methods. Properties are described in Section 2.

	BreachRadar	Ratio	Ratio + Prior	Belief Propagation	Vertex Cover	Real-time Detection
Finds POCs	✓	✓	✓	✓	✓	
Scalability	✓	✓	✓	✓		✓
Risk assessment	✓	✓	✓		✓	
Blame-aware	✓				✓	
Confidence-aware	✓		✓			
Early Detection	✓					

have similar or opposite labels. Some variations extend Label Propagation to incorporate label confidence and prior information, which could be used when only positive labels (fraud) are observed [21].

However, both methods have the underlying assumption that more connections to innocent nodes imply a smaller likelihood that the node has been compromised; this idea is at the core of guilt-by-association algorithms: similarly to how connections to fraudulent nodes increase the probability of a node being fraudulent, then connections to safe locations decrease this probability. For this problem, we know that the opposite is true: connections to innocent merchants do not compensate for the fact that a connection to a compromised location exists. Our results showing Belief Propagation's low performance corroborate this intuition.

2.5 Vertex Cover This problem can also be formulated as a vertex cover problem in bipartite graphs: given a set of cards who were victim of fraud, we would like to identify the smallest subset $\mathcal{S}$ of adjacent nodes (i.e., merchants) so that every card who has been a victim has at least one adjacent location in $\mathcal{S}$. Unfortunately, this formulation is NP-hard[2]. Nevertheless, in Section 6.5 we evaluate a greedy approximation: on each iteration, we consider as compromised the location with the highest number of adjacent *fraud-cards*, remove them from the bipartite graph and repeat.

3 The POC Problem

We assume the point-of-view of a payment network or card issuer with visibility over the majority of the transactions of a set of cards, some of which have been identified as *fraud-cards* (these are typically canceled and reissued). Loosely speaking, our goal is to automatically identify a small set of locations that many *fraud-cards* have in common.

We represent the set of transactions as a bipartite graph with cards on one side and *possible Points-of-Compromise* on the other.

A *possible Point-of-Compromise* is a feature that a subset of transactions have in common, such as a specific point-of-sale terminal, a store identifier or a merchant name (i.e. all the stores from a corporation). In practice, we would also like to incorporate time as a feature as data breaches and skimming devices temporally correlate transactions. For example, in Section 6, we consider *terminal-week* pairs as *possible Points-of-Compromise*, but many other possibilities (or combinations thereof) are admissible. Edges connect two nodes if there is a transaction between a given card and a given bucket (we use these names interchangeably).

3.1 Problem Definition Lets consider $\mathcal{C}$ to be the set of all cards and $\mathcal{B}$ to be the set of all possible POCs. $\mathbf{G} = (\mathcal{C} \cup \mathcal{B}, E)$ is the bipartite graph and for every edge $(i, j) \in E \Rightarrow i \in \mathcal{C}$ and $j \in \mathcal{B}$.

We will always use index i to represent cards and index j to represent possible POCs. $\mathbf{N}_i$ is the set of neighboring possible POCs of card i and $\mathbf{N}_j$ is the set of neighboring cards of possible POC j; n_i and n_j will represent their respective size.

$f : \mathcal{C} \to \{0, 1\}$, part of our input, is a function indicating whether a given card $c \in \mathcal{C}$ was a victim of fraud or not.

$\mathbf{B}$ is a $|\mathcal{C}| \times |\mathcal{B}|$ matrix where b_{ij} is the probability that j is the POC responsible for i being a *fraud-card*, or the blame which card i attributes to possible POC j.

Using this notation (summarized in Table 2), the POC detection problem can be formulated as:

Problem 1. ***(Spotting Points-of-Compromise)***

- ***Given:*** *A graph* $\mathbf{G} = (\mathcal{C} \cup \mathcal{B}, E)$ *and fraud labels* $f : \mathcal{C} \to \{0, 1\}$.
- ***Find:***
  - $\boldsymbol{\theta} : \mathcal{B} \to [0, 1]$*, where* θ_j *is the probability that* j *is a Point-of-Compromise;*
  - $\mathbf{B} : \mathcal{C} \times \mathcal{B} \to [0, 1]$*, where* b_{ij} *is the blame that card* i *assigns to possible POC* j*.*

[2]This can be easily shown through reduction to Minimum Set Cover, one of Karp's 21 NP-complete problems [9].

Table 2: Notation, symbols and definitions

Symbol	Definition
$\mathcal{C}$	Set of all cards
$\mathcal{B}$	Set of possible POCs
$\mathbf{G}$	Graph of cards and possible POCs
$\mathbf{N}_i$	Set of possible POCs neighbors of card i
$\mathbf{N}_j$	Set of cards neighbors of possible POC j
n_i	$\lvert\mathbf{N}_i\rvert$
n_j	$\lvert\mathbf{N}_j\rvert$
f_i	Boolean indicating if i is a fraud-card
$\boldsymbol{\theta}$	Vector of POC probabilities.
$\mathbf{B}$	Blame matrix
$\mathbf{b_i}$	A row of matrix $\mathbf{B}$
b_{ij}	A cell of matrix $\mathbf{B}$

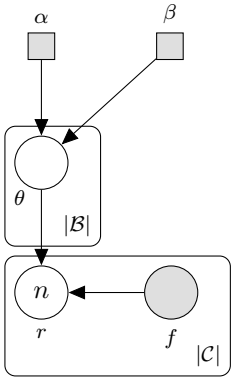


Figure 2: **Plate notation of the probabilistic graphical model.** The blames b_{ij} are a direct function of $\boldsymbol{\theta}$ and $\mathbf{f}$ and are omitted for clarity.

4 POC-detection Algorithm

We describe a novel algorithm for the identification of *Points-of-Compromise* following a simple Bayesian inference approach. We adopt the principle that predictions are inherently uncertain and require more evidence in order to increase their confidence, and assume that cards are compromised at a single location and should influence each other towards agreeing on the (locally) most likely mutual *Point(s)-of-Compromise*.

4.1 A POC Hierarchical Model We start by assuming that whether a location has been compromised is represented by p_j, a Bernoulli random variable whose success probability θ_j is taken from a Beta prior.

From the card perspective, we assume that each *fraud-card* has an associated categorical distribution r_i of size n_i and probability vector $\mathbf{b_i}$, where each element b_{ij} of $\mathbf{b_i}$ is linearly proportional to the respective θ_j compromise probability. This means we are implicitly assuming that the probability of a card blaming a possible POC is linearly proportional to the probability of it being a POC.

This model can be formally defined as follows:

$$\theta_j \sim Beta(\alpha, \beta) \tag{4.1}$$

$$b_{ij} = \begin{cases} \frac{\theta_j}{\sum_{k \in N_i} \theta_k} & \text{if } f_i = 1 \text{ and } j \in \mathbf{N_i} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

$$r_i \sim Categorical(n_i, \mathbf{b_i}) \tag{4.3}$$

whose corresponding graphical model in plate notation can be seen in Figure 2. The only hyper-parameter of this formulation is the Beta distribution which is encoded using α and β. Intuitively, α and β control how much evidence we need to be confident that a location has really been compromised.

The inputs of this model are the sets $\mathbf{N}_i$ (the locations with which each card interacted) and the boolean indicators f_i on whether a card has been a victim. Note the direct relationship between the problem definition and this formulation: the probability that a location j has been compromised can be obtained directly from the expected value of p_j ($E[p_j] = \theta_j$) and the blames attributed by the different cards are encoded in the row-normalized matrix $\mathbf{B}$.

In the following, we will describe an alternating algorithm to simultaneously find $\boldsymbol{\theta}$ and $\mathbf{B}$.

4.2 From Blames to POC Probabilities Lets suppose that we knew $\mathbf{B}$, i.e., we knew how much blame each card attributes to each possible POC. Ideally, we would then like to find $\boldsymbol{\theta}$ that maximizes $P[\boldsymbol{\theta}|\mathbf{B}]$, as our model relates the probability of being compromised with the blames attributed.

We defined that θ_j comes from a $Beta(\alpha, \beta)$ distribution, therefore we know that:

$$P[\theta_j; \alpha, \beta] = \frac{\theta_j^{\alpha-1}(1-\theta_j)^{\beta-1}}{B(\alpha, \beta)}, \tag{4.4}$$

where the beta function $B(\alpha, \beta)$ is simply the normalization constant that ensures that the total probability integrates to 1.

Let $z_j = \sum_i b_{ij}$ be the sum of all the blames assigned to possible POC j. z_j follows a Beta-Binomial distribution and we know that

$$P[z_j|\theta_j; n_j] \propto \theta_j^{z_j}(1-\theta_j)^{n_j - z_j} \tag{4.5}$$

and, from Bayes' theorem, the posterior distribution equals

$$\begin{aligned} P[\theta_j|z_j; \alpha, \beta, n_j] &\propto P[z_j|\theta_j; n_j]P[\theta_j; \alpha, \beta] \propto \\ &\propto \theta_j^{z_j+\alpha-1}(1-\theta_j)^{n_j+\beta-z_j-1} \end{aligned} \tag{4.6}$$

This means that the posterior probability distribution of θ_j is defined as another Beta distribution that can be parametrized as $Beta(z_j + \alpha, n_j - z_j + \beta)$ and with expected value

$$E[\theta_j] = \frac{z_j + \alpha}{n_j + \alpha + \beta} \tag{4.7}$$

We can think of this expected value as a ratio of the blames (z_j) to the total number of cards that used this possible POC (n_j), with additional terms α and β that represent, respectively, "virtual" *fraud-cards* that used this location (α) and "virtual" non-*fraud-cards* that used this location (β). Depending on the $Beta(\alpha, \beta)$ prior chosen, two possible POCs with the same fraud-cards ratio will have their probability adjusted to match our confidence on how far away from the prior distribution they are. The ratio $\frac{\alpha}{\alpha+\beta}$ represents the expected probability that a random location is compromised, while the magnitude of α and β encode our confidence on this prior.

4.3 From POC Probabilities to Blames Following a similar line of reasoning, lets suppose that we knew $\boldsymbol{\theta}$ and that we would like to find $\mathbf{B}$, i.e. we would like to know the probability that a card will blame each possible POC, given their respective compromise probabilities.

As mentioned in Section 4.1, we assume that blaming probabilities are linearly proportional to compromise probabilities. Therefore, the blames matrix $\mathbf{B}$ can be found by directly using Equation 4.2.

4.4 An Alternating Algorithm We previously defined the probability that a location was compromised based on the blames assigned to it, and defined the blame assigned by a card to a possible POC given the compromise probabilities of all the locations in Equation 4.2. This tight coupling suggests an alternating algorithm in which one updates blames and POC probabilities in succession until convergence.

We initialize blames as uniformly distributed across all neighboring possible POCs, and proceed by updating the POC probabilities and blames in sequence. We check for convergence using the l_1 norm of the difference of successive POC probability estimations. Algorithm 1 describes this procedure and, as we will see in Section 5, one of its advantages is being easily parallelizable.

4.5 Convergence Given its many applications, such as k-means clustering or expectation-maximization methods, the convergence of alternating optimization algorithms is a well studied problem and it is known to work well in a surprising number of cases [3]. In general, one cannot guarantee global convergence but local convergence tends to be very fast. By using the dataset and hyper-parameters described in Section 6, we show empirically that our implementation of Algorithm 1 converges exponentially fast in Figure 3.

Algorithm 1 BREACHRADAR

Require: $\mathbf{N}$, neighbors of each card
Require: $\mathbf{n}$: number of cards on each possible POC
Require: ϵ: convergence threshold
Require: α, β: prior parameters

$\mathbf{B} \leftarrow$ UniformBlames()
$\boldsymbol{\theta} \leftarrow$ UpdatePOCProbabilities($\mathbf{B}$, $\mathbf{n}$, α, β)
repeat
  $\mathbf{B} \leftarrow$ UpdateBlames($\boldsymbol{\theta}$, $\mathbf{N}$)
  $\boldsymbol{\theta_{prev}} \leftarrow \boldsymbol{\theta}$
  $\boldsymbol{\theta} \leftarrow$ UpdatePOCProbabilities($\mathbf{B}$, $\mathbf{n}$, α, β)
until $\|\boldsymbol{\theta} - \boldsymbol{\theta_{prev}}\|_1 < \epsilon$
return $(\boldsymbol{\theta}, \mathbf{B})$

function UPDATEBLAMES($\boldsymbol{\theta}$, $\mathbf{N}$)
  for every card i **do**
    $sum \leftarrow \sum_{k \in N_i} \theta_k$
    for every bucket $j \in N_i$ **do**
      $b_{ij} \leftarrow \frac{\theta_j}{sum}$
  return B

function UPDATEPOCPROBABILITIES($\mathbf{B}$, $\mathbf{n}$, α, β)
  for every bucket j **do**
    $z_j \leftarrow \sum_i b_{ij}$
    $\theta_j \leftarrow \frac{z_j + \alpha}{n_j + \alpha + \beta}$
  return $\boldsymbol{\theta}$

5 Distributed POC-detection

Both stages of the alternating optimization algorithm are parallelizable if we assume a message-passing model of computation, as used in Pregel [15] and other large-scale graph processing systems. We assume that both nodes and edges can store information which is shared and updated until the whole system converges; in our case, edges contain information relative to blames ($\mathbf{B}$) while possible-POC nodes contain their respective *Point-of-Compromise* probability.

Under this new model of computation, the differences to Algorithm 1 can be summarized in Algorithm 2.

Implementation. In order to obtain an highly efficient parallel solution, this implementation was devel-

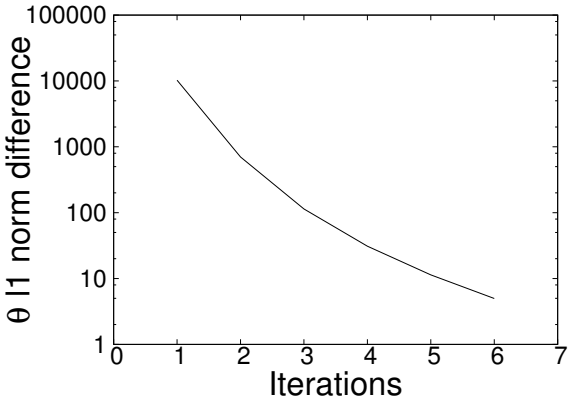


Figure 3: **Exponentially fast convergence -** notice the log scale in the y-axis.

oped using Apache Spark [23], a MapReduce engine that enables in-memory computation. Spark is well suited for machine learning algorithms as its in-memory model doesn't force sequential stages to synchronize data to disk.

In particular, we rely on Spark's GraphX [20] module which overlays an abstraction for graph-parallel computation that allows message passing and aggregation.

6 Results

We answer the following questions:

Q1. **Effectiveness -** How accurately and how early can we detect *Points-of-Compromise* in reality? What are the trade-offs?

Q2. **Scalability -** How does our method scale in terms of the size of the network and in terms of the number of machines available?

Q3. **Fraud-cards precision and recall -** How much of the fraud that is reported can be explained through the identification of *Points-of-Compromise*?

Q4. **Discoveries -** Can we identify real and validated *Points-of-Compromise* in real data?

6.1 Experimental Setup BreachRadar was applied to two datasets provided by different sources, each with over one billion transactions, 0.4 million cards and 2 million fraudulent transactions that cover more than one year. The data is very unbalanced with the percentage of fraud in accordance with industry averages, between 0.01% and 0.1% [6]. Due to privacy concerns, results in this section do not indicate the corresponding dataset.

We created possible POCs corresponding to *terminal-week* pairs and removed multi-edges and all possible POCs that interacted with less than 5 *fraud-cards*, as they could not be confidently labeled *Points-of-Compromise* under any circumstance. After this pre-processing stage, there were at least 1.5 million terminal-week pairs that had to be considered in each dataset. Results here reported correspond to a $\alpha = 0.2$ and $\beta = 15$ prior which provides a significant assumption that a random *terminal-week* is not compromised.

Algorithm 2 Distributed BreachRadar

G.POCs.onNewMessage(*blame*)
  $z = z + blame$
  $\Theta = \frac{z+\alpha}{n_j+\alpha+\beta}$

G.Cards.onNewMessage(θ)
  $sum = sum + \theta$

function UpdateBlames($\boldsymbol{\theta}$, **N**)
  for each POC j in G **do in parallel**
    for each Card i in N_j **do in parallel**
      j.sendMessage(i, θ)
  // After all messages are aggregated.
  for each Edge e in G **do in parallel**
    e.blame = $\frac{e.POC.\theta}{e.card.sum}$

function UpdatePOCProbabilities(**G**)
  for each Edge e in G **do in parallel**
    e.sendMessage(e.POC, e.blame)

Table 4: Overview of the two datasets used. Specific values masked for privacy.

	#cards	#trxs	#buckets	#fraud trxs
Dataset1	$> 10^5$	$> 10^9$	$> 10^6$	$> 10^6$
Dataset2	$> 10^5$	$> 10^9$	$> 10^6$	$> 10^6$

6.2 Empirical Evidence and Fraud Prevented We collected significant empirical evidence demonstrating our ability to find real POCs, both data breaches and terminals that we suspect were victims of skimming. The list includes tobacco machines equipped with credit card readers and other general vending machines where the percentage of *fraud-cards* is as high as 80%.

Data breaches are easier to validate as they are often reported in the news; a non-exhaustive list of POCs we automatically detected along with a sample news report can be found in Table 3. We were also able to identify POCs whose first report occurred more than 6 months after the last transaction have available in the dataset.

We evaluated the amount of fraud that could be

Table 3: Several Points-of-Compromise identified in one of the datasets have also been mentioned in news reports.

Merchant	Source	Summary
Schnucks	*ComputerWorld* [19]	A supermarket chain where a breach exposed 2.4M cards.
NoMoreRack.com	*Reuters* [17]	An online retailer with over $340 million in sales annually, probes likely card breach.
Jetro Cash & Carry	*DataBreaches* [1]	A data breach allowed attackers to access private data in cards used over a one month period in this chain.
Bashah's Family of Stores	*BankInfoSecurity* [10]	A supermarket chain tied to the compromise of hundreds of cards.
Buy.com	*Yahoo Finance - Money Talks* [2]	Hundreds of online shoppers reported fraudulent charges on their credit cards after making a purchase at this online marketplace.

prevented if cards of likely-compromised POCs were automatically reissued. Figure 1a shows the gains obtained when BreachRadar is evaluated in a 6 weeks period. Over $2 million USD in savings would be possible when reissuing all cards that interacted with POCs with an expected compromise probability above 10%, even if we assume a $10 reissue cost per card. 17% of the cards reissued would have been victims of fraud and 95% of these would be first-time victims.

6.3 Accuracy and Early Detection Due to the lack of ground-truth regarding which locations are effectively *Points-of-Compromise*, we evaluate the precision and recall of BreachRadar through the injection of synthetic POCs in real data. We evaluate BreachRadar along two dimensions:

1. Probability that a card is a victim of fraud after using a compromised location (p). This can be viewed as a proxy for how early our method is able to detect compromised cards, as detection gets naturally easier as the number of victims increases.

2. Presence of noise in the set of *fraud-cards*, i.e., we randomly mark additional cards as victims, although fraud cannot be attributed to any of the POCs.

In each experiment, we define a set of POCs and vary a probability (p) that their transactions will steal the corresponding card. Based on this new *fraud-cards* list, we then obtain a new list of possible *Points-of-Compromise*. Table 5 shows the number of *fraud-cards* and *possible Points-of-Compromise* as the probability of the card being stolen increases, when no noise is included.

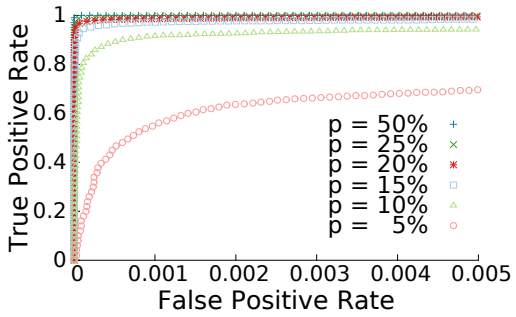
Figures 4a and 4b show the receiver operating characteristic (ROC) and "precision vs recall" curves for the different compromise probabilities. Note that we are able to simultaneously achieve high precision and recall for relatively small compromise probabilities; we achieve over 90% precision and recall even when only 10% of the cards have been victims of fraud.

Table 5: Impact of the infection probability on the number of *fraud-cards* and possible Points-of-Compromise.

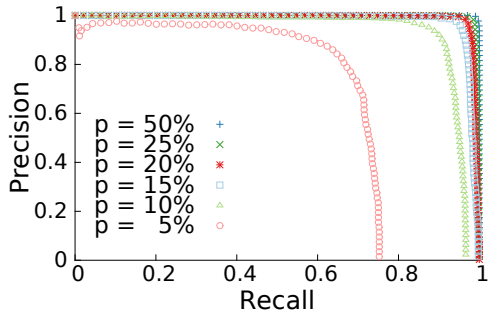
Probability (p)	#fraud-cards	#poss. POC
5%	29 326	104 185
10%	57 593	263 115
15%	84 833	450 609
20%	110 955	662 967
25%	136 896	892 119
50%	257 609	2 168 733

We also analyze the impact of noise in the effectiveness of our method: we double the number of *fraud-cards* by randomly selection additional cards. These are cards that do not have a corresponding POC in the data, even though they were marked as victims. As before, Figures 4c and 4d show the ROC and "precision vs recall" curves for the different compromise probabilities. Using the same scenario of a 10% probability of cards being compromised as example, note that we are still able to achieve about 75% recall maintaining 50% precision, even though we are simultaneously considering very aggressive levels of cards mislabeled as *fraud-cards* and early detection.

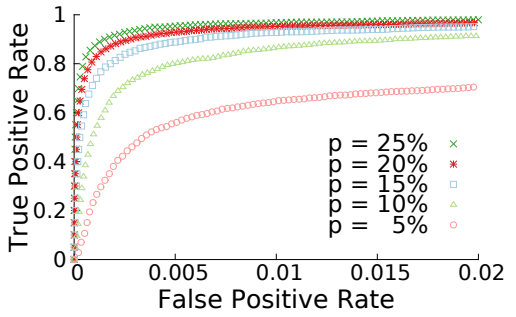
6.4 Scalability Scalability experiments are performed using the data described in Table 5. We show BreachRadar linear scalability on the number of possible POCs (Figure 1c) and analyze the runtime of its Spark implementation when changing the number of machines available in a small cluster of 6 quad-core machines (Figure 5). The number of cores in each machine does not provide any advantage, as disk input-output is the bottleneck of our system, not processing power.



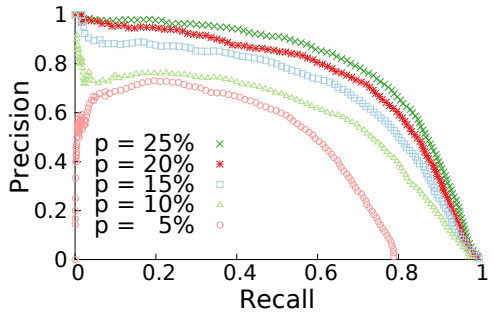
(a) Receiver Operating Curve: notice the **very low false positive rate**.



(b) **High precision and recall, even with low stealing probability (p).**



(c) Receiver Operating Curve: **low false positive rate, with 100% noise**.



(d) **Even with 100% added noise, high precision and recall.**

Figure 4: Accuracy with varying probability of a card being a victim of fraud. (Top row: without noise. Bottom row: 100% additional fraud-cards as noise.)

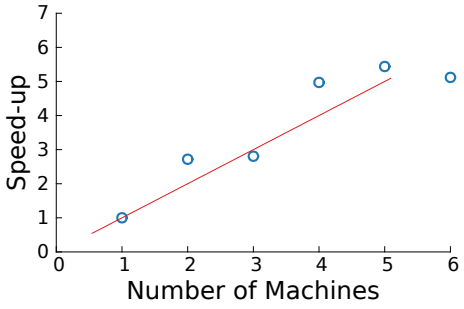


Figure 5: **Good distributed speed-up.**

6.5 Comparison We compare the precision and recall of our method to (1) FABP - Fast Belief Propagation [12]; (2) a greedy approximation of Vertex Cover described in Section 2; (3) the ratio as proxy for POC probability, as commonly used by previous methods and (4) the ratio metric combined with the best prior found for BREACHRADAR, in order to reduce its false positive rate.

FABP [12] is a fast approximation to Belief Propagation with low sensitivity to input parameters. We assigned a high prior belief (0.5) to *fraud-card* nodes, a low prior belief (-0.1) to non-*fraud-card* nodes and a neutral belief (0.0) to possible POC nodes. We then decreasingly sort the possible POCs by their final belief, creating the corresponding precision vs recall curve. The curve for the other methods was created similarly, based on the ordering of the possible POCs that they explicitly define.

We compare all methods on the non-noise dataset when considering a stealing probability of 10%.

As can be seen in Figure 1b, our method significantly improves over all alternatives. Reasons have been detailed in previous sections, but can be summarized as a combination of appropriate priors and the focused blame of the *fraud-cards*. Vertex Cover's result shows that focused blames are not sufficient, while the ratio with prior's result shows that removing false positives with a small amount of *fraud-cards* is not enough either.

6.6 Reproducibility We make available the dataset used in the comparison experiments described in Section

6.3, when requested by email to the authors.

7 Conclusion

We present, as far as the authors know, the first distributed procedure for the automatic detection of *Points-of-Compromise* in transaction networks. We achieve highly accurate results through the implementation of an in-memory algorithm that updates POC probabilities and blame scores alternatingly, and we have demonstrated surprising empirical evidence in a real dataset. Our main contributions are the following:

1. **Point-of-Compromise Problem.** We formulate a novel and important POC detection problem.
2. **Effectiveness.** BreachRadar accurately identifies *Points-of-Compromise*, achieving over 90% precision and recall when only 10% of the stolen cards have been used in fraud (Figure 1b).
3. **Distributed POC-Detection algorithm.** We provide a scalable distributed algorithm for POC detection in big datasets (Figure 1c).

References

[1] Restaurant depot/jetro cash & carry customers' credit cards hacked. DataBreaches.net, December 2011.

[2] B. Ballenger. Rakuten.com customers reporting credit card fraud. finance.yahoo.com, June 2013.

[3] J. C. Bezdek and R. J. Hathaway. Some notes on alternating optimization. In *Advances in Soft Computing—AFSS 2002*, pages 288–300. Springer, 2002.

[4] R. J. Bolton and D. J. Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002.

[5] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications*, 41(10):4915–4928, 2014.

[6] European Central Bank. Fourth report on card fraud. Technical report, European Central Bank, 2015.

[7] Fair Isaac Corporation. How europe's card fraud is evolving. Insights White Paper, 2015.

[8] G. Forman. Determining point-of-compromise. US Patent US 20050055373 A1, March 2005.

[9] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[10] T. Kitten. Bashas' breach exposes security flaws. BankInfoSecurity.com - The Fraud Blog, February 2013.

[11] V. F. Klebanoff. Method and system for assisting in the identification of merchants at which payment accounts have been compromised. US Patent US 8473415 B2, August 2009.

[12] D. Koutra, T.-Y. Ke, U. Kang, D. H. P. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Machine Learning and Knowledge Discovery in Databases*, pages 245–260. Springer, 2011.

[13] B. Krebs. The target breach, by the numbers. KrebsOnSecurity.com, May 2014.

[14] LexisNexis Risk Solutions. Merchants contend with increasing fraud losses as remote channels prove especially challenging. LexisNexis True Cost of Fraud Study, September 2015.

[15] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[16] Financial Technology Partners Research.

[17] A. Roy. Online retailer nomorerack.com probes likely card breach-report. Reuters, March 2013.

[18] K. P. Siegel, R. A. Paynter, R. L. Grossman, C. Brown, C. R. Byce, T. Dwyer, and A. Chen. System and method for identifying a point of compromise in a payment transaction processing system. US Patent US 8473415 B2, 2013 June.

[19] J. Vijayan. Schnucks supermarket chain struggled to find breach that exposed 2.4m cards. ComputerWorld, April 2015.

[20] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[21] Y. Yamaguchi, C. Faloutsos, and H. Kitagawa. Socnl: Bayesian label propagation with confidence. In *Advances in Knowledge Discovery and Data Mining*, pages 633–645. Springer, 2015.

[22] S. Yan. System and method for detecting account compromises. US Patent US 8600872 B1, December 2013.

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[24] S. M. Zoldi, L. Wang, L. Sun, and S. G. Wu. Mass compromise/point of compromise analytic detection and compromised card portfolio management system. US Patent 7761379 B2, July 2010.