Predicting Cyber Threats with Virtual Security Products

Shang-Tse Chen
Georgia Tech
schen351@gatech.edu

Yufei Han
Symantec Research Labs
Yufei_Han@symantec.com

Duen Horng Chau
Georgia Tech
polo@gatech.edu

Christopher Gates
Symantec Research Labs
Chris_Gates@symantec.com

Michael Hart
Symantec Research Labs
Michael_Hart@symantec.com

Kevin A. Roundy
Symantec Research Labs
Kevin_Roundy@symantec.com

ABSTRACT

Cybersecurity analysts are often presented suspicious machine activity that does not conclusively indicate compromise, resulting in undetected incidents or in costly investigations into the most appropriate remediation actions. There are many reasons for this: deficiencies in the number and quality of security products that are deployed, poor configuration of those security products, and incomplete reporting of product-security telemetry. Managed Security Service Providers (MSSP's), which are tasked with detecting security incidents on behalf of multiple customers, are confronted with these data quality issues, but also possess a wealth of cross-product security data that enables innovative solutions. We use MSSP data to develop *Virtual Product*, which addresses the aforementioned data challenges by predicting what security events would have been triggered by a security product if it had been present. This benefits the analysts by providing more context into existing security incidents (albeit probabilistic) and by making questionable security incidents more conclusive. We achieve up to 99% AUC in predicting the incidents that some products would have detected had they been present.

CCS CONCEPTS

• Security and privacy → Intrusion/anomaly detection and malware mitigation; **• Information systems → Data mining**;

KEYWORDS

Virtual Product, semi-supervised matrix factorization

ACM Reference format:
Shang-Tse Chen, Yufei Han, Duen Horng Chau, Christopher Gates, Michael Hart, and Kevin A. Roundy. 2017. Predicting Cyber Threats with Virtual Security Products. In *Proceedings of Annual Computer Security Applications Conference, San Juan, Puerto Rico, USA, December 4, 2017 (ACSAC '17),* 11 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Security products often are primed to detect certain threats extremely well. In other contexts, they will generally provide less than conclusive or no evidence of attacks. This motivates a defense in depth strategy that advocates for deploying multiple kinds of security devices to provide the most robust defense. Clearly it is infeasible to deploy every single security product to maximally protect every single device in an organization. Cybersecurity analysts, therefore, must contend with suboptimal context regarding potential attacks because the products that are providing telemetry are not well suited for a potential attack. Their confidence in either pursuing or ignoring a potential compromise is often less than ideal.

The key to improving detection rates in this environment is to learn from the vast amounts of telemetry produced by the prevalent defense-in-depth approach to computer security, wherein multiple security products are deployed alongside each other, producing highly correlated alert data. By studying this data, we are able to accurately predict which security alerts a product would have triggered in a particular situation, even though it was not deployed. A representative

Product Type	Alert Description (Event)
Gateway	TCP Urgent Data Enforcement
Gateway	TCP anomaly
Gateway	TCP Out of Sequence
Gateway	ICMP Echo Request
Windows	Cryptographic operation
Windows	Attempt to unprotect auditable protected data
Windows	Logon attempt using explicit credentials
Windows	Key file operation
Windows	Filter Manager Event 1
Windows	Attempt to register a security event source
Windows	Attempt to unregister a security event source
Windows	Special privileges assigned to new logon
Windows	A privileged service was called
Windows	A network share object was accessed
Firewall	TCP Connection
Firewall	UDP Connection
Proxy	TCP Cache Hit
Proxy	TCP Cache Miss: Non-Cacheable Object

Table 1: A long list of inconclusive alerts generated in a real incident of a machine infected by the infamous *Zbot Trojan*. These alerts overwhelm a cybersecurity analyst, and do not help answer important questions such as: *Is this machine compromised? How severe is the attack? What actions should be taken?* Our technique, *Virtual Product*, correctly predicts the presence of the infamous Zbot Trojan, which would have been identified by an AV product, had it been installed.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ACSAC '17, December 4, 2017, San Juan, Puerto Rico, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

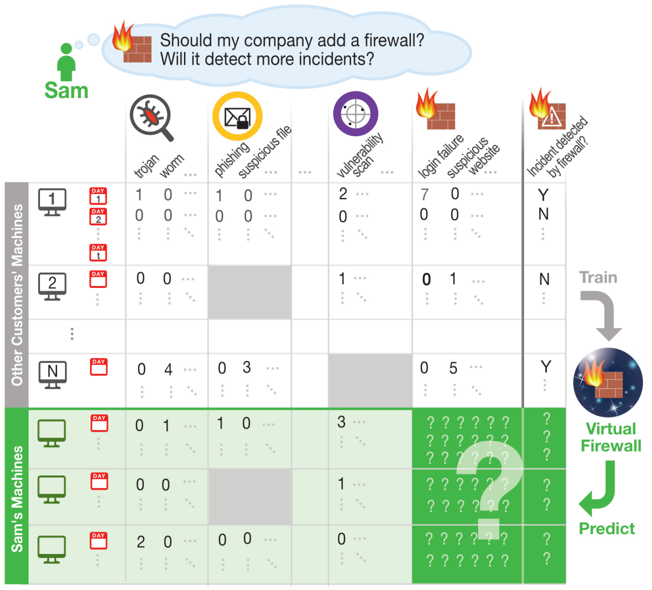

Figure 1: *Virtual Product* helps our user Sam discover and understand cyber-threats, and informs deployment decisions (e.g., add firewall?) through semi-supervised non-negative matrix factorization on telemetry data from other users (with firewalls deployed). In the data matrix, each row represents a machine-day, and each column a security event's occurrences. Missing events from undeployed products are shown as gray blocks. The last column indicates if the firewall has detected an incident. Our *virtual* firewall serves as a proxy to the actual product and predicts the output Sam may observe (dark green block) if he deploys it.

example is shown in Table 1, wherein security alerts produced by several products hint at the possibility of a security problem, but do not present conclusive evidence. Our models, however, are able to correctly predict the presence of the Zeus (also known as the Zbot) trojan, as the cause of the anomalous system and network behavior on the machine.

We introduce and formulate the novel problem of *Virtual Product*, the first known attempt to predict the security events and high-severity incidents that would have been identified by a product if it had been deployed. Given sufficient data from many organizations deploying different sets of security products, we posit it should be possible to predict the events that would have been reported by additional security products that were not deployed. This analysis benefits from the observations that many security products detect the same threats, and that attacks are typically automated and therefore proceed in predictable sequences of behavior.

Figure 1 shows how *Virtual Product* works. We formulate incident data as a large matrix. Each row, called a *machine-day*, tracks all of the security events that were observed on a particular machine, on a given date. Although many entries will be empty since machines are at most protected by a handful of products, we can predict the likely events that would have been triggered by those products that were not deployed. The security officers can then hopefully make a more informed decision about the trade off of cost and value of what other security products would provide. For the analyst, *Virtual Product* enriches each incident (i.e., row) with more context to understand the severity of the threat posed by the observed activity. Our work makes the following contributions:

- **Novel Idea of Virtual Product.** We introduce the problem of simulating a security product's individual security events and the security incidents that these events would have raised, had it actually been deployed. We formulate techniques by which the security data managed by Security Incident and Event Managers (SIEM's) and Managed Security Service Providers (MSSP's) on behalf of multiple products can be used for this purpose (see Table 2 for definitions of these terms).
- **Effective Approach.** We provide a practical implementation for this problem by adapting semi-supervised non-negative matrix factorization techniques, which simultaneously addresses the problem of security incident and event prediction for absent products, with high accuracy.
- **Impact to Security Industry.** Our Virtual Product model will impact the security industry by increasing company security at significantly reduced costs. We are working towards making *Virtual Product* events and security incidents available to customers of an MSSP. By deploying *Virtual Product* on behalf of customers, we provide a new way for them to experience the potential benefits of security products without deploying them, allowing them to make more informed purchasing decisions.

To enhance readability of this paper, we have listed the terminology used in this paper in Table 2. The reader may want to return to this table throughout this paper for technical terms' meanings and synonyms used in various contexts of discussion. We proceed by discussing related work in Section 2, and present our proposed *Virtual Product* model in Section 3. We evaluate the performance of our algorithm in Section 4. Next, we discuss the expected impact of *Virtual Product* and concrete deployment plans studies in Section 5. Finally, we discuss our findings and conclude in 6.

2 RELATED WORK

There has been growing interest in applying machine learning and data mining techniques to detect cyber-threats, such as malicious files [2, 23], malicious websites [12, 19], and online fraudulent behaviour [16], using approaches that range from Naive Bayes [20], to neural networks [3], decision trees [24], to large-scale graph-based inference [2, 23]. In contrast to prior work, instead of predicting cyber-threats directly, we formulate and tackle the novel *Virtual Product* problem of predicting how a security product would work in a customer's specific environment, had it been deployed. Not only do we predict the incident triggering behaviour of a product, but also reconstruct all the security events it detects, tackling both tasks simultaneously using matrix factorization methods. To the best of our knowledge, *Virtual Product* addresses a novel problem, one that provides additional context and detection capabilities by predicting the incidents and individual security events that would be provided by security products had they been deployed.

Matrix factorization [10, 14, 22] exploits latent features of a data matrix by decomposing the matrix into a series of low-rank factor matrices. These factor matrices, though additional constraints can be enforced on them, are learnt by minimizing a generalized Bregman divergence [5, 22] between the original data matrix and

Technical term	Meaning
Virtual product	A machine learning model used to reconstruct a real security product's behavior
Machine-day	A machine on a particular day
Security event	A description of activity recorded by a security product, not necessarily malicious, e.g., login failure
Incident	A serious security threat, evidenced by one or more events, warranting attention, e.g., unblocked malware
SIEM	Security Incident and Event Managers, which manage security events produced by products, that they analyze to detect and report security incidents
MSSP	Managed Security Service Providers, which run a SIEM on behalf of multiple customers

Table 2: Terminology used in this paper

dot product of the low-rank factor matrices. Matrix factorization has been popularly used in collaborative filtering [13, 18, 21] of highly sparse user-item rating records to predict users' preferences and recommend unrated products. Document clustering is another well-studied research domain that uses matrix factorization. A common approach is to apply non-negative matrix factorization (*NMF*) [6, 11] on sparse bag-of-words features of documents and group documents using the derived non-negative factors [4, 6, 11]. Supervision in the context of matrix factorization, introduces class separating structure into the factor matrices, which enforces linear separation of classes in the linear projection of data [13, 17]. The objective function of the factorization has been designed to enforce specific properties of the latent projected data. Previous efforts on supervised and weakly supervised matrix factorization can be found in [1, 9, 13, 25]. Most of them focus on decomposing densely valued data matrices to improve clustering accuracy. Our algorithm extends and adapts prior work to the classification problem of predicting attacks by reconstructing missing signals on extremely sparse data.

3 PROPOSED MODEL: VIRTUAL PRODUCT

Given a security product P, our *Virtual Product* model aims to detect and categorize incidents for customers who have not deployed P. We formulate the construction of *Virtual Product* as a classification problem, training it on machine-day observations collected from machines that have deployed P. The training process learns the functional mapping between event occurrence patterns of other products and the incident class labels reported by P. During testing, the *Virtual Product* model takes as input the observed event occurrence patterns from products except P, and produces incident detection and categorization results.

A main challenge for *Virtual Product* is in training and applying it with incomplete event occurrence patterns as input. Events may be missing either because their corresponding products are not deployed, or due to data corruption at the telemetry data collection process.

To address this issue, we propose a semi-supervised non-negative matrix factorization method (*SSNMF*) as a core computation technique for *Virtual Product*. It extracts an unified discriminative feature representation of the event occurrence records from both the training and testing datasets. We conduct incident detection and categorization in *Virtual Product* by feeding the learned feature representations as input to any standard supervised classifiers. *Virtual Product* denotes the process of conducting *SSNMF* on event occurrence data, followed by training a supervised classifier on the output of *SSNMF*.

Another contribution of *Virtual Product* is to estimate event occurrence patterns that are missing from the observed data. It is helpful for security analysts to understand relations between event occurrence profiles and reported security incidents. The *SSNMF* well matches this requirement, as it is intrinsically equipped with the capability of reconstructing the missing event values through inner product between U and V.

3.1 Semi-Supervised Non-negative Matrix Factorization (SSNMF)

We use a non-negative data matrix $X \in R^{N \times M}$ to denote the aggregation of both training and testing event occurrence data. Each row in X, noted as $X_{i,:}$ denotes occurrence counts of different events around a machine-day. Without loss of generality, the first N_1 rows of X belong to the training event occurrence data. They are equipped with corresponding incident class labels reported by the target product P. The remaining $N - N_1$ rows of X are the testing data corresponding to event occurrence data collected from customers' machines without P deployed.

Non-negative Matrix Factorization reconstructs a non-negative data matrix $X \in R^{N \times M}$ using the dot product of two non-negative factors $U \in R^{N \times k}$ and $V \in R^{M \times k}$. As shown in Equation 1, the latent factors are learned by minimizing the reconstruction error on the observed events in our data.

$$U, V = \min_{U,V \geqslant 0} \|X - UV^T\|_o{}^2 \tag{1}$$

The norm $\|\|_o$ indicates the aggregated reconstruction error on the observed entries of X. Each row in U, $U_{i,:}$ represents the linear projection of $X_{i,:}$, which formulates a new feature representation of machine-day observations in a low-dimensional space. Column vectors of V are the projection bases spanning the projection space.

To integrate supervision information into the matrix factorization process, we introduce a class-sensitive loss into the objective function of matrix factorization, in order to force machine-day observations of different classes to be separated from each other in the projected space. Equation 2 and Equation 3 give the formulation of the discriminative loss functions defined for binary and multi-class classification scenarios, respectively.

$$\begin{aligned} F(\hat{Y}, U, W) = -\sum_{i=1}^{N} \hat{Y}_i \log \frac{1}{1 + \exp(-U_{i,:} W^T)} \\ + (1 - \hat{Y}_i) \log \frac{1}{1 + \exp(U_{i,:} W^T)} \end{aligned} \tag{2}$$

$$F(\hat{Y}, U, W) = -\sum_{i=1}^{N}\sum_{j=1}^{C} \hat{Y}_{i,j} \log \frac{\exp(U_{i,:}W_{j,:}^T)}{\sum_j \exp(U_{i,:}W_{j,:}^T)} \quad (3)$$

where $W \in R^{1\times K}$ stores the regression coefficients. $\hat{Y}_i$ represents the class label of each machine-day observation. For labeled machine-days, $\hat{Y}_i$ it either 1 or 0, depending on whether $X_{i,:}$ belongs to positive or negative class. For unlabeled machine-days, $\hat{Y}_i$ represents any plug-in estimator of probabilistic confidence of $X_{i,:}$ belonging to positive class. In the multi-class version of the loss function, C denotes the number of classes in the labeled dataset. As a result, W becomes a $R^{C\times K}$ matrix. Each row in W corresponds to the regression coefficients for each class. $\hat{Y}_{i,j}$ of labeled data is defined following one-hot encoding scheme. For unlabeled data, $\hat{Y}_{i,j}$ represents the probabilistic class membership of each $X_{i,:}$. U is the common factor shared by both matrix factorisation in Equation 1 and the class-sensitive loss function defined in Equation 2 and 3. This design guarantees the feature representation U preserves the class separating structure of the training data.

$\hat{Y}$ for unlabeled data can be initialized using external oracles with probabilistic output, such as gradient boosting and logistic regression. In this work, we treat $\hat{Y}$ as one variable to learn and estimate it by jointly optimizing the objective function with respect to U, V, W and $\hat{Y}$. We assume that unlabeled data points with similar profiles are likely to share similar soft class label $\hat{Y}$. By enforcing such assumption to the objective function design, we explicitly inject supervised information into the projection of both labeled and unlabeled machine-day observations.

The objective function of *SSNMF* is shown in Equation 4

$$\begin{aligned} U, V, W, \hat{Y} = &\underset{U,V\geq 0, W, 1\geq \hat{Y}\geq 0}{arg\,min} \quad \|X - UV^T\|_o{}^2 + \alpha F(\hat{Y}, U, W) \\ &+ \beta Tr(\hat{Y}^T L \hat{Y}) + \gamma(\|U\|^2 + \|V\|^2) + \rho\|W\|^2 \\ &s.t. \hat{Y}_i = Y_i \text{ if } X_{i,:} \text{ is labeled} \end{aligned} \quad (4)$$

The constraint in the objective function requires strict consistency between $\hat{Y}$ and the true class labels on labeled machine-day observations. L is the graph laplacian matrix defined based on K-nearest neighbor graph of the whole data matrix X. Minimizing the trace function $Tr(\hat{Y}^T L\hat{Y})$ propagates the confidence of class membership from true class label of labeled machine-days to unlabeled machine-days. It embeds class-separating information into the projection U of unlabeled machine-days. Regularization terms $\gamma(\|U\|^2 + \|V\|^2)$ and $\rho\|W\|^2$ are added to prevent over-fitting.

3.2 Optimization Algorithm

We use coordinate descent to optimize Equation 4. During each iteration, U, V, W and $\hat{Y}$ are updated alternatively. One of the four variables are updated while all the others are fixed. Iterations continue until the objective value cannot be further improved. U, V are updated using multiplicative update [11], which is popularly applied in *NMF*. Equation 5 gives the formulations of multiplicative update of U and V

$$\begin{aligned} U^{t+1} &= U^t \odot \frac{[(X\odot M)V]^+ + [(UV^T \odot M)V]^- + \alpha[\hat{Y}W]^+ + \alpha[PW]^-}{[(X\odot M)V]^- + [(UV^T \odot M)V]^+ + \alpha[\hat{Y}W]^- + \alpha[PW]^+ + \gamma U} \\ V^{t+1} &= V^t \odot \frac{(X\odot M)^T U}{(UV^T \odot M)^T U + \gamma V} \end{aligned} \quad (5)$$

where $[A]^+ = (|A| + A)/2$ and $[A]^- = (|A| - A)/2$. P is the output from the sigmoid function (binary classification) $P = \frac{1}{1+\exp -(UW^T)}$ or the softmax function (multi-class classification) $P_j = \frac{\exp(UW_{j,:}^T)}{\sum_{j=1}^{C}\exp(UW_{j,:}^T)}$. The operation $\odot$ indicates hadamard product between matrices. M is a entry-wise weight matrix. $M_{i,j} = 1$ if the entry $X_{i,j}$ is observed, and $M_{i,j} = 0$ otherwise.

Updating $\hat{Y}$ consists of two components. For one aspect, the learning of$\hat{Y}$ is based on supervision information propagation. For the other aspect, estimates of $\hat{Y}$ depends on the output from the sigmoid or softmax function, which encodes the retraction from data reconstruction penalty in the objective function. Equation 6 and Equation 7 define how to estimate $\hat{Y}$ in binary and multi-class classification scenarios:

$$\begin{aligned} \hat{Y}_i &= Y_i \quad \text{if} \quad X_{i,:} \quad \text{is labeled} \\ \hat{Y}^{t+1} &= \hat{Y}^t \odot \frac{\alpha \log(1+\exp(UW^T)) + 2\beta S\hat{Y}}{\alpha\log(1+\exp -(UW^T)) + 2\beta D\hat{Y}} \end{aligned} \quad (6)$$

$$\begin{aligned} \hat{Y}_i &= Y_i \quad \text{if} \quad X_{i,:} \quad \text{is labeled} \\ \hat{Y}^{t+1} &= \hat{Y}^t \odot \frac{\alpha\log\hat{P} + 2\beta S\hat{Y}}{2\beta D\hat{Y}} \end{aligned} \quad (7)$$

where $\hat{P}_{i,j} = \frac{\exp(U_{i,:}W_{j,:}^T)}{\sum_{j=1}^{C}\exp(U_{i,:}W_{j,:}^T)}$. S is weight matrix of K-nearest neighbor graph. D is a diagonal matrix with $D_{i,i}$ defined as $\sum_{i=1}^{N} S_{i,j}$.

By removing terms without W in Equation 4, the left terms of the objective function formulate a $L2$-penalised logistic regression with soft class labels $\hat{Y}$ and training data points in the projected space U. Therefore, learning W given U and $\hat{Y}$ fixed can be performed through iterative gradient descent until convergence. We found that the number of iterations can be dramatically reduced by choosing the step size of gradient adaptively using AdaGrad [8], as shown in Equation 8:

$$W^{t+1} = W^t + \lambda \frac{G_{W^t}}{\sqrt{\sum_{t'=1}^{t-1} G_{W^{t'}}}} \quad (8)$$

where G_W is the gradient of Equation 4 with respect to W.

When U, V and W converge, $U_{1:N_1,:}$ and $U_{N_1:N,:}$ can be used as low-dimensional feature representations of the training and testing data, respectively. We then train a logistic regression on the row vector space of U to conduct incident detection and categorization in *Virtual Product*. Note that we are not restricted to logistic regression. we choose it due to its simplicity and probabilistic decision output. Despite its simplicity, it produces superior performances thanks to the learned feature representation U, as reported in the experimental study.

Dataset	#Machine-days	#Detected Incidents	#Events	Sparsity level	#Incident Type
FW1: Firewall 1	4506	770	1011	98%	10
FW2: Firewall 2	9254	3093	1927	99%	12
FW3: Firewall 3	4477	1274	2019	98%	10
EP1: Endpoint Protection 1	18983	4128	2409	99%	30
EP2: Endpoint Protection 2	8006	904	988	97%	5

Table 3: Summary of the training datasets (Jul-Sept) for the top five products that detect the most incidents.

4 EVALUATION

4.1 Data Collection

Our evaluation uses telemetry data sent from a leading Managed Security Service Provider (MSSP), which supports roughly 80 security products from different vendors. Customers send telemetry from their deployed products to the MSSP, which analyzes the telemetry to identify and report incidents. Due to space constraints, we show the results of the top five products that detect the most incidents: three firewalls (*FW1*, *FW2*, *FW3*) and two endpoint protection products (*EP1*, *EP2*).

To evaluate our approach's prediction performance for a specific product P, we derived an anonymized dataset from the telemetry data. This derived dataset consists of data contributed by the machines that have deployed P, which allows us to extract ground truth labels. When performing prediction, we do not use any events from P. In other words, we pretend that product P is not deployed and hide all its events.

The dataset is represented as a N-by-M matrix X (see Figure 1). Each row $X_{i,:}$ is an instance that represents a machine-day. Each column $X_{:,j}$ is a feature that corresponds to the number of occurrences of a event from different products except P. To prevent numerical overflow during computation, we take the logarithm of each event occurrence count in X. Since events relevant to an incident may not appear within a single machine-day, when counting occurrences, we consider the period that spans three days before and after the machine-day. Note that our task is not to predict an incident before it happens, so we also collect events observed after the machine-day. To prevent duplicate and similar instances, we only use machine-days from the same machine that are at least one week apart from other machine-days.

The matrix is extremely sparse, and each machine-day typically only has a few observed events. Events may be missing if their corresponding products are not deployed. They may also be caused by data corruption when the products report them. To avoid machine-days with zero or very few observed events, which are nearly impossible to perform prediction, we filter out all machine-days with fewer than 20 observed events.

There are two sets of labels associated with each machine-day, one for binary classification of whether there is an incident, and the other one for multi-class classification of the incident type. The positive and negative machine-days are collected as follows. For each incident reported in our system, we label a machine-day as **positive** through counting event occurrences, as mentioned above. For *negative* machine-days, we use the same set of machines (as when collecting positive instances). A machine-day is labeled **negative** if no incidents have been detected by any products within a one-month period (15 days before to 15 days after). This binary label definition is used in experiments, in order to evaluate the capability of the proposed method for detecting malicious incidents. We also include a multi-class definition of incident labels. The multi-class incident label denotes multiple categories of detected incidents, valued as $\{-1, 1, 2, ...C\}$, where C is the number of incident categories, and -1 means "no incident".

Dataset	#Machine-days	#Detected Incidents	Sparsity level
FW1	3090	355	98%
FW2	6515	2830	98%
FW3	2660	253	97%
EP1	8222	2377	98%
EP2	2275	754	98%

Table 4: Summary of validation datasets (Oct-Dec).

The same data collection process is performed over two independent time periods. The first dataset was collected from July to September in 2016 — we call this the **training** dataset (summarized in Table 3), on which we conduct cross-validation test to verify theoretical validity of the algorithmic design in Section 4.4 and evaluate reconstruction performances in Section 4.3. The second dataset was collected from October to December in 2016 — we call this the **validation** dataset (summarized in Table 4). We use the *training* dataset to tune the parameters of our model, then apply it on the *validation* dataset to evaluate incident classification accuracy in real-world applications

In Table 3, we also show the sparsity level of each product's dataset, to highlight how sparse the data matrices are in our study. The sparsity level of a given data matrix is defined as the fraction of unobserved entries in the matrix.

4.2 Experiment Setup and Overview

Our experiment consists of three parts.

(1) In Section 4.3, benefiting from matrix factorization, the proposed method estimates count values of security events produced by those products whose events were withheld from the dataset. The reconstructed event counts will later help us determine whether a security incident would have been raised by the events produced by the withheld product. Since security incident are formulated as collections of relevant events,

reconstructing the missing events is essential for incident reproduction based on the occurrence pattern of the corresponding incident. Furthermore, the individual events provide important insights and context into the nature of the security incident, which frequently enable improved triage and remediation of the incident. We evaluate our proposed event-reconstruction model by measuring reconstruction error between ground truth event counts and our estimated values.

(2) In Section 4.4, we evaluate the performance of our proposed method for detecting security incidents. This output of Virtual Product's methods allows us to build a incident detector based on incomplete event information. The test is conducted on the training data matrices and the incident labels of all five products. Both binary and multi-class incident labels are produced for this test.

(3) In Section 4.5, we investigate the computational complexity and empirical scalability of our proposed *Virtual Product* model, noted as *VP*.

4.3 Evaluation on Reconstruction Accuracy

We validate in this section that the proposed model can compensate for missing events. We investigate the reconstruction performance of *Virtual Product* with respect to the occurrence counts of withheld event observations. Reconstruction capability is a key function of the proposed model, since knowing which events were responsible for triggering a predicted security incidents is essential to the understanding of that incident, and to its remediation. Accordingly, we evaluate the reconstruction capability of the proposed method. We randomly select 50% of the observed entries and take the event count of these entries as ground truth. After that, we hold out the ground truth counts and apply our matrix factorization method to derive the estimated count values of the masked entries. To measure reconstruction accuracy we use the R-squared score between the ground truth and the estimated values. To remove randomness introduced by sampling, we repeatedly sample the observe entries 10 times. The average and standard deviation of the derived R-squared scores from different sampling rounds are used as a comprehensive evaluation metric of the reconstruction performance.

The average and standard deviation of R-squared scores derived on the five datasets are shown in Table 5. We also provide a statistical summary of our reconstruction results in Table 5. In addition, for each dataset, we count the percentage of the entries in which the reconstructed event occurrence counts are larger than 50% of the corresponding ground truth occurrence count values, as noted as $Percentage$ in Table 5. This statistical summary provides an intuitive understanding on the reported reconstruction accuracy. In practical use, if the reconstructed occurrence count of a given event is close enough to its ground truth, the reconstruction is precise enough to estimate whether this event was triggered by the corresponding product. The results show that *Virtual Product* is able to reconstruct event occurrence patterns with precision for the security products. As seen in the results, almost all masked event occurrence patterns are perfectly recovered through the matrix factorization process embedded using our proposed method. As we will see in Section 4.6, these recovered security events enable machine learning models to perform improved incident detection. The true value of this work, however, is perhaps best illustrated by the case studies shown in Section 5.1.

	R-squared Score		Percentage	
Dataset	Mean	Std	Mean	Std
FW1	0.8493	0.0036	0.9819	0.0009
FW2	0.7300	0.0032	0.9834	0.0002
FW3	0.8408	0.0023	0.9858	0.0007
EP1	0.7356	0.0013	0.9801	0.0001
EP2	0.8193	0.0014	0.9832	0.0004

Table 5: Performance of reconstruction on all five datasets

4.4 Evaluation: Incident Detection and Categorization

We perform 10-fold Monte Carlo cross-validation, where each randomly samples 70% of the machine-days from the training dataset collected from July to September in 2016. The remaining 30% is left for testing.

We set up a baseline model (shorthand: **LR**) by training a logistic regression classifier directly on the event count matrix X, with missing entries filled with zeros. In our approach (shorthand: **VP**), we train a logistic regression classifier on the low-dimensional feature representation of X produced by *Virtual Product* model.

The purpose of introducing the baseline model is two-folds. Firstly, we use the results from the baseline model to further validate our initial assumption: it is possible to predict the events that would have been reported by additional security products that were not deployed. The baseline model conducts classification using only the observed events from deployed products. No reconstructed event information is embedded. Therefore, if the baseline method can detect or categorize incidents with an acceptable accuracy, we have strong reason to believe the proposed *Virtual Product* model can perform even better by incorporating the reconstructed event counts into the classifier design. Secondly, we aim to conduct a fair comparative study for our proposed methodology, though we note that the baseline model is not a comparison to prior art, as *Virtual Product* addresses a novel problem of not only predicting the incidents but also recovering the associated security events. The objective function of *SSNMF*, used in *Virtual Product*, can be roughly understood as construction of a logistic regression classifier on the projected space of the original data. This comparative study aims to verify the benefits gained from the algorithmic design of *Virtual Product* for classification with missing features.

To allow fine-grained comparison, we compute the mean and standard deviation of the *Area-Under-Curve* (AUC) and the *True Positive Rate* (TPR) across 10 folds, and display them in Table 6 and Table 7 respectively. As we can see in the two tables, both the baseline and the proposed *Virtual Product* method present good classification performances over *training* datasets of all five security products. It indicates that counts of events collected from different organisations are able to predict occurrence of incidents that would have been reported by undeployed products. In a further step, the result unveils consistently superior incident detection precision of the

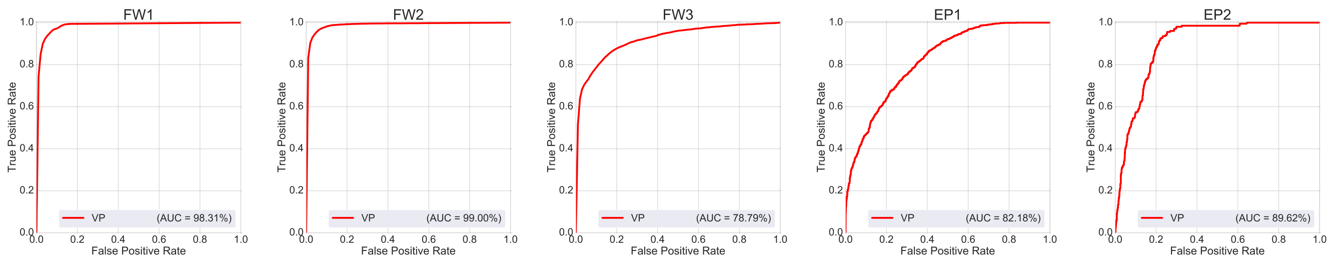

Figure 2: Averaged ROC curves from 10-fold cross-validation of *Virtual Product* on our top five product datasets.

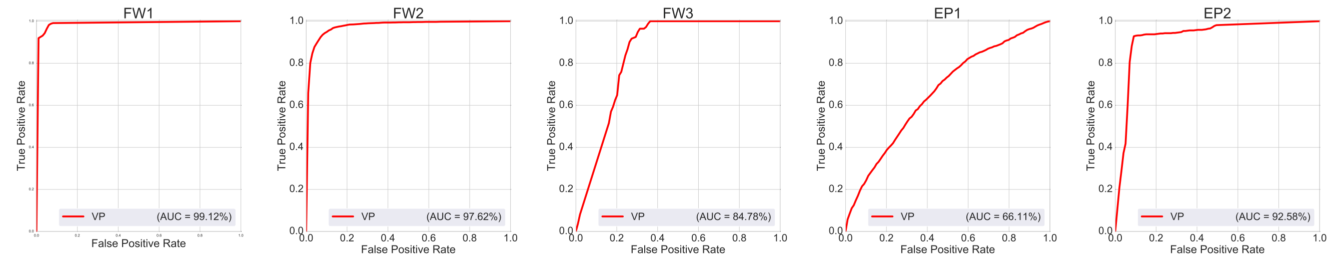

Figure 3: ROC curves of the Virtual Product model evaluated using the validation datasets of the five products.

	VP AUC		LR AUC	
Dataset	Mean	Std	Mean	Std
FW1	**0.9831**	0.0041	0.9695	0.0055
FW2	**0.9900**	0.0018	0.9810	0.0029
FW3	**0.9200**	0.0070	0.8761	0.0131
EP1	**0.8218**	0.0066	0.8076	0.0072
EP2	**0.8962**	0.0083	0.8306	0.0164

Table 6: Our approach (VP) detects security incidents with high accuracies (AUCs) across all five datasets, outperforming the baseline model (LR).

	VP TPR		LR TPR	
Dataset	Mean	Std	Mean	Std
FW1	**0.9724**	0.0114	0.9661	0.0078
FW2	**0.9820**	0.0057	0.9810	0.0074
FW3	**0.7879**	0.0157	0.7608	0.0228
EP1	**0.5200**	0.0175	0.5016	0.0268
EP2	**0.5897**	0.0293	0.5663	0.0399

Table 7: True positive rate (TPR) of incident detection on all five data sets at 10% false positive rate (FPR). Our approach (VP) outperforms the baseline (LR)

proposed *Virtual Product* model over the baseline method across the *training* data sets of different products. Figure 2 shows the average ROC curve and AUC derived from the cross-validation test, offering a global and intuitive view of incident detection performances

	VP	LR
FW1	**0.9927**	0.9910
FW2	**0.9425**	0.9338
FW3	0.8005	**0.8043**
EP1	**0.7501**	0.7220

Table 8: Average F1 scores of incident categorization on our datasets. We do not include McAfee because over 99% of detected incidents belong to one single incident type.

over *training* datasets of different products using the proposed *Virtual Product* model. All obtained results support the design of the proposed *Virtual Product* method. Embedding matrix completion into classification helps to extract correlation between events observed events of different products, which increases available information to boost classification precision.

Additionally, test on the validation datasets follows a standard training-testing process of machine learning models in real-world applications. Classification model built with the *training* dataset collected within the precedent time period is used to detect incidents on the *validation* dataset formulated within the current time slot.

Interestingly, as shown in Figure 3, incident detection result using the proposed *Virtual Product* model presents consistent high detection accuracy over *validation* datasets of different products. The reported detection accuracy confirms the robustness of the proposed *Virtual Product* model.

As described in Section 3, the proposed *Virtual Product* can be seamlessly extended for incident categorization, which classifies detected incident at a finer scale. Without major modification, the proposed *Virtual Product* is able to achieve both incident detection and categorization (multi-class classification) at the same time. Table 8 shows the average F1-score of incident categorization on *training* datasets of different products using *Virtual Product*. As we can see, *Virtual Product* can achieve almost perfect incident categorization on the *FW1* and *FW2* datasets. In the *EP2* dataset, over 99% of detected incidents belong to a single incident type. Severe class imbalance makes any classifier built on the dataset statistically unstable, so we chose not to include the *EP2* dataset in the experimental study of incident categorization. The categorization precisions on *EP1* and *FW3* are relatively lower. This is mainly due to class imbalance among different incident categories in these two datasets, particularly in the case of the *EP1* training dataset, for which nearly half of the 30 incident types are minority classes. Each of these minority classes contains fewer than 10 machine-day observations, which increases the difficulty of categorization. The impact of class imbalance is also confirmed by the baseline *LR* method. Nevertheless, even in this extreme situation, the proposed *Virtual Product* still obtains improvements compared to the baseline model.

In general, all experimental results in this section verify the validity of the design of the proposed *Virtual Product*. By jointly conducting matrix factorization and discriminative model learning, the proposed model makes full use of inter-event correlation to compensate information missing due to the extremely sparse data structure. As a result, it provides a good reconstruction of the classification boundary from highly incomplete event occurrence data.

4.5 Evaluation of Computational Cost

Time Complexity Analysis. The training of *Virtual Product*'s model consists of two parts. First, we construct a k-nearest neighbor (K-NN) graph in an offline manner. Nearest neighbor searching generally requires a cost of $O(N^2M)$, which is quadratic to the size of the dataset. Since the machine-day event count data is high-dimensional and highly sparse, we use an approximate K-NN method [7] tailored for sparse data. This reduces the cost of K-NN searching to $O(DNlogN)$ in the worst case, where D is the number of feature dimensions. Next, we perform multiplicative updates in $O(TNMk) + O(TNCk)$ time, where T is the number of iterations and k is the dimension of the projected representation U. The total cost of the proposed model is therefore at most $O(DNlogN) + O(TNMk) + O(TNCk)$. For all five datasets, we observed that 200 iterations ($T = 200$) were sufficient to achieve convergence.

Empirical Scalability. We conducted experiments to study how our proposed model scales with increasing volumes of data. We report the average training runtime of the proposed *Virtual Product* model on *EP1* and *FW2* (across 10 runs). Our machine is a 64-bit Linux laptop (Ubuntu 14.0) with an Intel Core i7 quad-core CPU running at 2.5GHz, 16GB RAM and 500GB disk. The *Virtual Product* model is implemented in Python 2.7, with API provided in python scientific computing packages, numpy 1.13 and scikit-learn 0.18.1. *EP1* and *FW2* datasets contain approximately $19k$ and $9k$ machine-days, representing real-world medium-scale applications. To study large-scale scenarios, we enlarge *EP1* and *FW2* datasets by 10 folds by replicating real machine-days contained in the original datasets. The enlarged *EP1* and *FW2* datasets (we call them Large *EP1* and Large *FW2*) contain $190k$ and $90k$ machine-days respectively. Table 9 shows the average runtimes of our proposed model across the datasets.

Dataset	#Machine-days	Runtime (minutes)
FW2	9,254	10
EP1	18,983	25
Large *FW2*	92,540	45
Large *EP1*	189,830	57

Table 9: Average virtual product training times (over 10 runs).

Our MSS service currently monitors about 80 products. Since each virtual product can be trained independently from each other, we can easily speed up overall computation through parallelization (e.g., distributed computation using Spark; more discussion in Section 5.2). For purposes of the evaluations performed in this section, we were able to train *Virtual Product* in under an hour, even on a commodity computer of modest power with an unoptimized software implementation.

4.6 Improvement in Analyst Response Predictions

As additional evidence to support the utility of *Virtual Product*, we measure event reconstruction's ability to improve the accuracy of a model that internal Managed Security Services analysts use in determining whether to publish incidents to customers or suppress them as false positives. We use a recent version of this model that is trained with no interaction or influence from *Virtual Product*, and whose primary task is to recommend whether incidents should be published to customers, or suppressed.

We took the *FW1* dataset and removed all events from *FW1*, while keeping the events of other devices. We call this dataset X_{none}. We take X_{none} and create a new dataset X_{top2} from it, for which we include the top two predicted events for *FW1*. We choose the two events with the highest predicted instance count, normalized by the average instance count for that event. Although we could expand the number of predicted events beyond two, we believe that the simplicity afforded by this heuristic will include the most salient missing context versus the complexity of determining which predicted events to include.

Our model achieved 94.7% accuracy on X_{none} and 97.6% on X_{top2}. The 2.9% improvement in accuracy results in halving the error rate. Although the accuracy on X_{none} is quite good already, we must consider that the model is used to increase the productivity of security analysts. The median salary for an analyst is $90.1K US dollars [15] and therefore making them individually more efficient is desirable.

5 IMPACT AND DEPLOYMENT

This section will illustrate how *Virtual Product* empowers MSSP customers to identify and security incidents by adding much needed context to make more confident incident response decisions. To provide concrete illustrations of this, we present two case studies and discuss other areas of expected impact. We then proceed to a discussion of our current efforts, and future plans to integrate

Product	Event Description
Seen Indicators	
Proxy	Suspicious connection
FirewallA	WebVPN Authentication Rejected
FirewallA	WebVPN session created
FirewallA	WebVPN session terminated
FirewallA	WebVPN session deleted
FirewallA	WebVPN session started
FirewallA	WebVPN Authentication success
FirewallA	SSL handshake completed
FirewallA	Teardown TCP connection
FirewallA	TCP connection
FirewallA	Session disconnected
IPS	SQL Query in HTTP Request
IPS	RookIE/1.0 malicious user-agent string
IPS	Angler exploit kit exploit download attempt
IPS	Known malicious user agent - mozilla
IPS	HiKit initial HTTP beacon
IPS	TeamViewer remote administration tool outbound connection attempt
Router	Flow session close
Top Predicted Primary Indicators	
FirewallB	Windows Executable
FirewallB	Malicious File
FirewallB	SQL Injection Attempt
FirewallB	Phishing Webpage
FirewallB	RIG Exploit Kit
FirewallB	Windows DLL
FirewallB	Heartbleed Malformed OpenSSL Heartbeat
FirewallB	Microsoft Indexing Service UTF-7 Cross-Site Scripting Vulnerability
~~FirewallB~~	~~Microsoft IIS HTR Request Parsing Buffer Overflow Vulnerability~~
FirewallB	/etc/passwd Access Attempt

Table 10: Virtual Product correctly predicts that FirewallB would have detected an incident, and 10 of its top 11 predicted alerts coincide with the alerts that actually occurred, yielding a clearer picture of the artifacts involved in the attack, and the vulnerabilities used. The incorrect prediction is shown in strikeout font.

Virtual Product into the infrastructure used by our Managed Security Services.

5.1 Case Studies and Impact

As in Table 1, in this section we present two additional real-world incidents and the event predictions identified by *Virtual Product* for these incidents as examples of its positive impact on the incident response process.

Example 1. One of our customers, whom we will call Alice, has an important server that is protected by many network security products, as shown in Table 10. What value is FirewallB providing? Let us imagine that FirewallB is not deployed. Alice observes several suspicious events output from the deployed products. FirewallA detects an HTTP beacon from the HiKit exploit kit and the proxy also detects visits to suspicious websites. No incident was generated by these security products, indicating that without the evidence from FirewallB, the remaining events are insufficiently threatening to warrant attention. Based on evidence from the "virtual" FirewallB, however, Alice finds that there is likely an incident, with 95% confidence.

Product	Event Description
Seen Indicators	
Firewall	Bad TCP Header length
Firewall	P2P Outbound GNUTella client request
Firewall	wu-ftp bad file completion attempt
Firewall	DNS zone transfer via TCP detected
Firewall	SNMP possible reconnaissance, private access udp
Firewall	ICMP PATH MTU denial of service attempt
Firewall	FTP format string attempt
Firewall	SMTP expn root
Firewall	SMTP vrfy root
Firewall	Server netcat (nc.exe) attempt
Firewall	philboard_admin.asp auth bypass attempt
Firewall	SSLv2 Challenge Length overflow attempt
Firewall	OpenSSL KEY_ARG buffer overflow attempt
Firewall	proxystylesheet arbitrary arbitrary command attempt
Firewall	Oracle ONE JSP src-code disclosure attempt
Firewall	JBoss admin-console access
Firewall	RevSlider information disclosure attempt
Firewall	Accellion FTA arbitrary file read attempt
Firewall	Apache Tomcat directory traversal attempt
Firewall	Apache non-SSL conn. to SSL port DoS attempt
Firewall	Windows NAT helper components tcp DoS attempt
Firewall	Multiple SQL injection attempts
Firewall	Bash CGI environment variable inject attempt
Firewall	Suspicious .tk dns query
Firewall	Suspicious .pw dns query
Firewall	ColdFusion admin interface access attempt
Firewall	Windows Terminal server RDP attempt
Firewall	Suspicious DNS request for 360safe.com
Gateway	Connectra Request Accepted
Gateway	ICMP: Timestamp Request
Gateway	Possible IP spoof
Router	Admin Authentication Failed
Top Predicted Primary Indicators	
AV	CVE-2012-4933 ZENWorks Asset Mgmt Exploit
AV	Post-Compromise PHP Shell Command Execution
AV	CVE-2015-1635 OS attack, HTTP.sys Remote Code Execution Exploit

Table 11: An attack on a webserver is obviously underway, but was it successful? Virtual Product correctly predicts, with 99.9% confidence, that not only a deployed AV product would detect attacks on the machine, but predict successful infection of the system.

Product	Event Description
Seen Indicators	
Firewall	Microsoft Windows 98 User-Agent string
Firewall	SMTP: Attempted response buffer overflow
Windows	Encrypted data recovery policy was changed.
Windows	A cryptographic self test was performed.
Windows	Cryptographic operation.
Windows	MSI Installer
Windows	Key file operation.
Windows	A logon was attempted using explicit credentials.
Windows	An attempt was made to reset an account's password.
Windows	Special privileges assigned to new logon.
Windows	System audit policy was changed.
Windows	A user account was changed.
Windows	A security-enabled local group was changed.
Windows	An account failed to logon
Proxy	TCP Cache Miss: Non-Cacheable Object
Gateway	Connectra Request Accepted
Top Predicted Primary Indicators	
AV	Bloodhound.Exploit.170

Table 12: There are indications of possible ransomware activity, but how did the attack appear on the machine in the first place? Virtual Product correctly indicates that a malicious spreadsheet (detected as Bloodhound.Exploit.170) was at fault, a method by which the Locky RansomWare has been known to propagate.

To further understand the cause of the potential incident, Alice takes a deeper look at FirewallB's predicted events, which include malicious Windows executables, SQL injection attempts, a visit to a phishing webpage, and attacks on several recognized vulnerabilities. This additional telemetry gives Alice clarity on the used avenues of attack, which she can use to prioritize patching updates to prevent a recurrence of the attack. It also suggests possible data leaks through SQL injection and visits to phishing websites, enabling Alice to take action that could prevent a serious data breach.

For this particular incident, 11 events were triggered by the actual FirewallB product, and we list the top 11 reconstructed events identified by *Virtual Product*. These predictions are prioritized by dividing the events' reconstructed instance count by the average instance count for that event, which is akin to TF-IDF normalization in statistical language model. In actual deployment, *Virtual Product* users can customize its confidence thresholds based on whether they wish *Virtual Product* to provide only highly confident event reconstructions or a broader list that is more likely to include erroneous predictions, but that may include valuable information that would otherwise have been suppressed.

Example 2. In some cases, while existing security events may make it quite obvious that an attack has taken place, they may leave a vital question unanswered, *Was the attack successful?*. This is a vital question, since most webservers are constantly exposed to attacks, and yet most attacks do not succeed in compromising the machine, both because the machine is often not vulnerable to the attempted attack, and because the network devices that report attack events are often able to block them. Table 11 illustrates such an example, in which Virtual Product is able to determine that an AV product would have detected a serious incident with 99.9% probability. The reconstructed AV events further indicate that the attack is very likely to have been successful, and they give further insight into the nature of the predicted attack.

Example 3. Virtual Product is often able to provide context that outlines appropriate remediative and preventative actions. In the product events seen in Table 12, an observant analyst may see hints of a possible Ransomware attack, but the initial method of attack is not clear. Virtual Product correctly indicates that a malicious spreadsheet was at fault, a method by which the Locky Ransomware has been known to propagate, and therefore, reveals a possible social engineering campaign that the company's security department should investigate.

As is evident in these three case studies, and in the case study shown in Table 1, *Virtual Product* helps security analyst by providing context that helps them answer vital questions, such as: Is this machine compromised or just displaying unusual behavior? Was the attack that I see on this machine successful? How should I go about cleaning up this infected machine? How can I prevent a recurrence of a similar attack on this or other machines in my environment? By answering these questions for MSSP customers, Virtual Product significantly facilitates the security analyst's core tasks.

5.2 Deployment

We are currently working towards delivering an initial version of this technology to our Managed Security Services Product (MSSP), which will run on the Amazon Web Services platform. At present, we process data in batches, because telemetry data is uploaded every 15 minutes from our Security Operations Centers.

To integrate virtual product into the existing customer interface, we both introduce entirely new security incidents that are identified on the basis of Virtual Product's missing signal detection, and enrich existing incidents with additional context from virtual products. Our current system is a hybrid of components that are coupled with services that publish and subscribe to various streaming pipelines. Because of the flexibility that will be afforded by cloud platforms, we will schedule and provision resources to perform matrix completion and will leverage the existing pipelines for incident generation and enrichment. The interactions that customers and MSS analysts have with *Virtual Product* will be fully captured, as at present, allowing us to tune the parameters of our algorithm.

6 CONCLUSIONS AND DISCUSSION

We have presented *Virtual Product*, a novel technology that allows us to predict events from devices that are not currently deployed. Our evaluation shows that *Virtual Product* can significantly improve our ability to detect incidents. The business value of *Virtual Product* affects multiple levels of the enterprise. Cybersecurity analysts can leverage *Virtual Product* to enrich events coming from machines to make a better determination if and what kind of attack is being conducted. For security officers, *Virtual Product* empowers these decision makers to make informed purchasing decisions based on the additive value of potential products. If this technology is broadly adopted, it could create pressure on security product vendors to focus more on differentiation through actual capability and not through naming conventions. Future applications of this technology include providing product recommendations to our customers, particularly if we can perform "attack forecasting" to identify the likely attacks a customer would experience and how well they are defended and detected by existing products.

REFERENCES

[1] Lei Zhang Changhu Wang, Shuicheng Yan and Hongjiang Zhang. 2009. Non-Negative Semi-Supervised Learning. In *Proceedings of the 12th AISTATS*. 575–582.

[2] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. 131–142.

[3] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 3422–3426.

[4] Jiawei Han Deng Cai, Xiaofei He and Thomas S. Huang. 2011. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 33, 8 (2011), 1548–1560.

[5] Inderjit S. Dhillon and Suvrit Sra. 2005. Generalized nonnegative matrix approximations with Bregman divergences. In *In: Neural Information Proc. Systems*. 283–290.

[6] Tao Li Ding Chris and Wei Peng. 2006. Nonnegative Matrix Factorization and Probabilistic Latent Semantic Indexing: Equivalence Chi-Square Statistic and a Hybrid Method. In *Processings of the 21st AAAI Conference on Artificial Intelligence*.

[7] Ting Huang Jingjing Wang Zengyou He Jijie Wang, Lei Lin. 2010. Efficient K-Nearest Neighbor Join Algorithms for High Dimensional Sparse Data. *arXiv:1011.2807* (2010).

[8] Elad Hazan John Duchi and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[9] Alexandros Nanopoulos Josif Grabocka and Lars Schmidt-Thieme. 2012. Classification of Sparse Time Series via Supervised Matrix Factorization. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*. 928–934.

[10] K.R.Gabriel and S. Zamir. 1979. Lower rank approximation of matrices by least squares with any choice of weights. *Technometrics* 21, 4 (1979), 489–498.

[11] Daniel D. Lee and H. Sebastian Seung. 2000. Algorithms for Non-negative Matrix Factorization. In *In NIPS*. MIT Press, 556–562.

[12] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1245–1254.

[13] Jason Rennie Nathan Srebro Nati and Tommi Jaakkola. 2004. Maximum Margin Matrix Factorizations. In *Proceesings of Advances in Neural Information Processing Systems (NIPS) 17*.

[14] Nathan Srebro Nati and Tommi Jaakkola. 2003. Weighted Low-Rank Approximations. In *In 20th International Conference on Machine Learning*. 720–727.

[15] United States Department of Labor. 2017. Information Security Analysts. (2017). https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm

[16] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*. 201–210.

[17] Francisco Pereira and Geoffrey Gordon. 2006. The Support Vector Decomposition Machine. In *Proceedings of the 23rd International Conference on Machine Learning*. 689–696.

[18] Steffen Rendle and Lars Schmidt-Thieme. 2008. Online-updating Regularized Kernel Matrix Factorization Models for Large-scale Recommender Systems. In *Proceedings of the 2008 ACM Conference on Recommender Systems*. 251–258.

[19] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. 2017. Malicious URL Detection using Machine Learning: A Survey. *arXiv preprint arXiv:1701.07179* (2017).

[20] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. 2001. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 38–49.

[21] Ajit P. Singh and Geoffrey J. Gordon. 2008. Relational Learning via Collective Matrix Factorization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*. ACM, New York, NY, USA, 650–658. DOI:http://dx.doi.org/10.1145/1401890.1401969

[22] Ajit P. Singh and Geoffrey J. Gordon. 2008. A Unified View of Matrix Factorization Models. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*. 358–373.

[23] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1524–1533.

[24] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. 2007. IMDS: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1043–1047.

[25] Songfang Huang Peng Han Yulei Niu, Zhiwu Lu and Ji-Rong Wen. 2015. Weakly Supervised Matrix Factorization for Noisily Tagged Image Parsing. In *Proceedings of International Joint Conference on Artificial Intelligence 2015*. 3749–3755.