EURASIP Journal on Information Security
a SpringerOpen Journal

RESEARCH Open Access

MUSE: asset risk scoring in enterprise network with mutually reinforced reputation propagation

Xin Hu[1*], Ting Wang[1], Marc Ph Stoecklin[2], Douglas L Schales[1], Jiyong Jang[1] and Reiner Sailer[1]

Abstract

Cyber security attacks are becoming ever more frequent and sophisticated. Enterprises often deploy several security protection mechanisms, such as anti-virus software, intrusion detection/prevention systems, and firewalls, to protect their critical assets against emerging threats. Unfortunately, these protection systems are typically 'noisy', e.g., regularly generating thousands of alerts every day. Plagued by false positives and irrelevant events, it is often neither practical nor cost-effective to analyze and respond to every single alert. The main challenges faced by enterprises are to extract important information from the plethora of alerts and to infer potential risks to their critical assets. A better understanding of risks will facilitate effective resource allocation and prioritization of further investigation. In this paper, we present MUSE, a system that analyzes a large number of alerts and derives risk scores by correlating diverse entities in an enterprise network. Instead of considering a risk as an isolated and static property pertaining only to individual users or devices, MUSE exploits a novel *mutual reinforcement principle* and models the dynamics of risk based on the interdependent relationship among multiple entities. We apply MUSE on real-world network traces and alerts from a large enterprise network consisting of more than 10,000 nodes and 100,000 edges. To scale up to such large graphical models, we formulate the algorithm using a distributed memory abstraction model that allows efficient in-memory parallel computations on large clusters. We implement MUSE on Apache Spark and demonstrate its efficacy in risk assessment and flexibility in incorporating a wide variety of datasets.

Keywords: Risk scoring; Enterprise network; Reputation propagation

Introduction

Mitigating and defending against ever more frequent and sophisticated cyber attacks are often top priorities for enterprises. To this end, a plethora of detection and prevention solutions have been developed and deployed, including anti-virus software, intrusion detection/prevention systems (IDS/IPS), blacklists, firewalls, and so on. With these state-of-the-art technologies capturing various types of security threats, one would expect that they are very effective in detecting and preventing attacks. In reality, however, the effectiveness of these systems often fall short. The increasingly diversified types of cyber attacks, coupled with increasing collection of applications, hardware configurations, and network equipments, have made the enterprise environment extremely 'noisy'. For example, IDS/IPS systems regularly generate over 10,000 alerts every day. Majority of them turn out to be false positives. Even true alerts are often triggered by low level of threats such as brute-force password guessing and SQL injection attempts. Although the suspicious nature of these events warrants the reports by IPS/IDS systems, their excessive amount often only made the situation even more noisy.

Digging into the haystack of alerts to find clues to actual threats is a daunting task that is very expensive, if not impossible, through manual inspection. As a result, most enterprises practically only have resources to investigate a very small fraction of alerts raised by IPS/IDS systems. Vast majority of others are stored in a database merely for forensic purposes and inspected only after significant incidents have been discovered or critical assets have

*Correspondence: huxin@us.ibm.com
[1]Security Research Department, IBM T.J. Watson Research Center, 1101 Kitchawan Rd, Yorktown Heights, NY, USA
Full list of author information is available at the end of the article

already been severely damaged, e.g., security breaches and data leakage. However, even those small fraction of true positive alerts (e.g., device compromise, virus infection on a user's computer) are often too voluminous to become security analysts' top priority. In addition, these alerts are often considered as low level of threats and pose far less risks to the enterprises comparing with more severe attacks, such as server compromise or sensitive data leakage. Evidently, a more effective solution is to better understand and rank potential risks associated with these alerts so that analysts can effectively prioritize and allocate resources for alert investigation.

In a typical enterprise environment, as shown in Figure 1, there are different sets of entities: servers, devices, users, credentials, and (high-value) assets (e.g., databases, business processes). The connections between these entities represent their intuitive relationships; for example, a user may own multiple devices, a device may connect to different types of servers and internal databases, a device may be associated with multiple user accounts with different credentials, etc. We note that the reputation of these entities provide valuable indicators into their corresponding risks and are important factors to rank various security incidents associated with these entities e.g., IPS/IDS alerts and behavior anomalies. More importantly, an entity's reputation and the risk it may produce are not restricted to each individual entity. In fact, multiple entities are often tied together in a mutually reinforcing relationship with their reputation closely interdependent with each other.

In this work, we develop MUSE (**M**utually-reinforced **U**nsupervised **S**coring for **E**nterprise risk), a risk analysis framework that analyzes a large amount of security alerts and computes the reputation of diverse entities based on various domain knowledge and interactions among these entities. Specifically, MUSE models the interactions with composite, multi-level bipartite graphs where each pair of entity types (e.g., a user and a device) constitute one bipartite graph. MUSE then applies an iterative propagation algorithm on the graphic model to exploit the mutual reinforcement relationship between the connected entities and derive their reputation and risk score simultaneously. Finally, with the refined risk scores, MUSE is able to provide useful information such as ranking of low-reputation entities and potential risks to critical assets, allowing security analysts to make an informed decision as to how resources can be prioritized for further investigation. MUSE will also provide greater visibility into the set of alerts that are responsible for an entity's low reputation, offering insights into the root cause of cyber attacks.

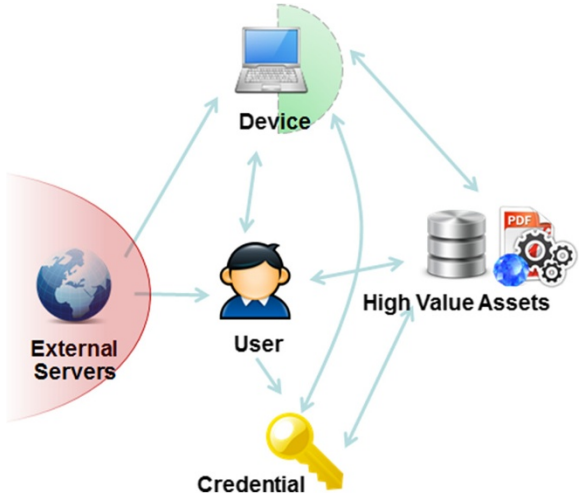

Figure 1 Entities in a typical enterprise network.

The main contributions of this work include: 1) a mutual reinforcement framework to analyze the reputation and the risk of diverse entities in an enterprise network, 2) a scalable propagation algorithm to exploit the networking structures and identify potential risky entities that may be overlooked by a discrete risk score, 3) a highly flexible system that can incorporate data sources in multiple domains, 4) implementation of MUSE that takes advantage of recent advances in distributed in-memory cluster computing framework and scales to very large graphic models, and 5) evaluations with real network traces from a large enterprise to verify the efficiency and efficacy of MUSE.

Risk and reputation in a multi-entity environment

In the following sections, we will present the design and the architecture of MUSE. We will first formulate the problem of risk and reputation assessment in enterprise network and then discuss specific domain knowledge and intuition that are crucial for solving the problem.

Problem formulation

In a typical enterprise environment, there are multiple sets of connected entities. Specifically, we consider five distinct types of entities as depicted in Figure 1: users $\mathcal{U}$, devices $\mathcal{D}$, credentials $\mathcal{C}$, high value assets $\mathcal{A}$, and external servers $\mathcal{S}$. These entities are often related in a pairwise many-to-many fashion. For example, a device can access multiple external servers. A user may own several devices e.g., laptops and workstations, while one device (e.g., server clusters) can be used by multiple users.

We model the interconnection between entities as a composite bipartite graph $G = (V, E)$, schematically shown in Figure 2. In G, vertices $V = \{\mathcal{U}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \mathcal{S}\}$ represent various entities. Edges $E = \{\mathcal{M}_{\mathcal{DS}}, \mathcal{M}_{\mathcal{DU}}, \mathcal{M}_{\mathcal{DA}}, \mathcal{M}_{\mathcal{UC}}, \mathcal{M}_{\mathcal{DC}}\}$ of bipartite graphs represent their

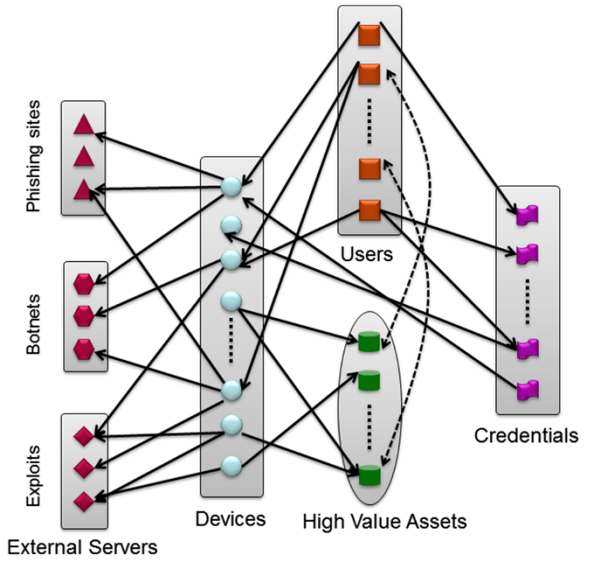

Figure 2 Network of interactions among multiple entities.

relationships, where $\mathcal{M}_{\mathcal{DS}}$ is the $|\mathcal{D}|$-by-$|\mathcal{S}|$ matrix containing all the pairwise edges, i.e., $\mathcal{M}_{\mathcal{DS}}(i,j) > 0$ if there is an edge between device d_i and external server s_j. The value of $\mathcal{M}_{\mathcal{DS}}(i,j)$ denotes the edge weight derived from the characteristics of the relationship, such as the number of connections, the number of bytes transmitted, duration, and so on. Similarly, $\mathcal{M}_{\mathcal{DS}}$, $\mathcal{M}_{\mathcal{DU}}$, $\mathcal{M}_{\mathcal{DA}}$, $\mathcal{M}_{\mathcal{UC}}$, and $\mathcal{M}_{\mathcal{DC}}$ are the matrices of the pairwise edges representing the association of their respective entities.

Next, we define the risk and the reputation of different entities more precisely. We treat each entity as a random variable X with a binary class label $X = \{x_R, x_{NR}\}$ assigned to it. Here, x_R is a risky (or bad) label, and x_{NR} is a non-risky (or good) label. A probability distribution P is defined over this binary class, where $P(x_R)$ is the probability of being risky and $P(x_{NR})$ is the probability of being non-risky. By definition, the sum of $P(x_R)$ and $P(x_{NR})$ is 1. We use this probabilistic definition because it encompasses a natural mapping between $P(x_{NR})$ and the general concept of reputation, i.e., an entity with a high probability of being good (or non-risky) is expected to have high reputation. In addition, it accepts different types of entities to incorporate specific domain knowledge into the reputation computation considering their respective characteristics, e.g.,

- **External server reputation** $\boldsymbol{p_s} = P_s(x_{NR})$ indicates the server's probability of being malicious and infecting the clients connecting to it. Notice that a low reputation p_s means high probability of being malicious.
- **Device reputation** $\boldsymbol{p_d} = P_d(x_{NR})$ represents the probability that a device may have been infected or compromised and thus under control of adversaries.
- **User reputation** $\boldsymbol{p_u} = P_u(x_{NR})$ indicates how suspiciously a user behaves, e.g., an unauthorized access to sensitive data.
- **Credential reputation** $\boldsymbol{p_c} = P_c(x_{NR})$ denotes the probability that a credential may have been leaked to the adversaries and thus making any servers associated with the credential vulnerable.
- **High-value asset reputation** $\boldsymbol{p_a} = P_a(x_{NR})$ denotes the asset's probability of being risky; for instance, a confidential database being accessed by unauthorized users, exfiltration of sensitive data, etc.

As there is a natural correlation between reputation and risk (e.g., less reputable entities generally pose high risks), we define an entity's risk as $P(x_R) = (1-P(x_{NR}))$ weighted by the importance of the entity, such that a high-value asset will experience a large increase in its risk score even with a small decline in its reputation. With these definitions, the goal of MUSE is to aggregate large amounts of security alerts, determine the reputation of each entity by exploiting their structural relationships in the connectivity graph, and finally output a ranked list of risky entities for further investigation. In the next section, we will describe the mutual reinforcement principle [1] that underlies MUSE.

Mutual reinforcement principle

The key observation of MUSE is that entities' reputation and risk are not separated; instead, they are closely correlated and interdependent. Through interacting with each other, an entity's reputation can impact on the risk associated with its neighbors, and at the same time, the entity's risk can be influenced by the reputation of its neighbors. For example, a device is likely to be of low reputation 1) if the server it frequently visits are suspicious or malicious e.g. Botnet C&C, Phishing, or malware sites, 2) if the users using the device have bad reputation, and 3) if the credentials used to log into the device have high risks of being compromised, leaked, or even used by an unauthorized user. Similarly, a credential's risk of being exposed will increase if it has been used by a less reputable user and/or on a device that exhibits suspicious behavior patterns. Along the same line, a user will have low reputation if she owns several low-reputation devices and credentials. Last but not least, a high-value asset or the sensitive data stored in internal databases are likely to be under a significant risk e.g., data exfiltration if they have been accessed by multiple low-reputation devices that also connect to external malicious servers. We describe these mutually dependent relationships more formally in our multi-layer mutual reinforcement framework, using the following set

of equations governing the server reputation p_s, device reputation p_d, user reputation p_u, credential reputation p_c, and high-value asset reputation p_a.

$$
\begin{aligned}
p_d &\propto \omega_{ds} \sum_{d\sim s} m_{ds} p_s + \omega_{du} \sum_{d\sim u} m_{du} p_u + \omega_{dc} \sum_{d\sim c} m_{dc} p_c \\
p_u &\propto \omega_{du} \sum_{d\sim u} m_{du} p_d + \omega_{uc} \sum_{u\sim c} m_{uc} p_c \\
p_c &\propto \omega_{uc} \sum_{u\sim c} m_{uc} p_u + \omega_{dc} \sum_{d\sim c} m_{dc} p_d \\
p_a &\propto \omega_{da} \sum_{d\sim a} m_{da} p_d + \omega_{ua} \sum_{u\sim a} m_{ua} p_u + \omega_{ca} \sum_{c\sim a} m_{ca} p_c,
\end{aligned}
$$

where $d \sim s$, $d \sim u$, etc. represent edges connecting device d with server s and user u, etc., $\propto$ means 'proportional to', ω_{ij} indicates the weights associated with edges and reputation types, and m_{ij} is the value in the connectivity matrices. Next, we exploit this mutual reinforcement principle in the bipartite graph network to simultaneously estimate the reputation and the risk using the propagation algorithm described in the next section.

Reputation propagation algorithm

Specifically, we employ the principle of belief propagation (BP) [2] on the large composite bipartite graph G to exploit the link structure and efficiently compute the reputations for all entities. Belief propagation is an iterative message passing algorithm on general graphs and has been widely used to solve many graph inference problem [3], such as social network analysis [1], fraud detection [4], and computer vision [5].

BP is typically used for computing the marginal distribution (or so-called 'hidden' distribution) for the nodes in the graph, based on the prior knowledge (or 'observed' distribution) about the nodes and from its neighbors. In our case, the algorithm infers the probabilistic distribution of an entity's reputation in the graph based on two sources of information: 1) the prior knowledge about the entity itself and 2) information about the neighbor entities and relationship between them. The inference is accomplished by iteratively passing messages between all pairs of entities n_i and n_j. Let $m_{i,j}$ denote the 'message' sent from i to j. The message represents i's influence on j's reputation. One could view it as if i, with a certain probability of being risky, passes some 'risk' to j. Additionally, the prior knowledge about i (e.g., importance of the assets and a user's anomalous behavior) is expressed by *node potential function* $\phi(i)$ which plays a role in determining the magnitude of the influence passed from i to j. In details, edge $e_{i,j}$ is associated with message $m_{i,j}$ (and $m_{j,i}$ if the message passing is bi-directional). The outgoing message from i to neighbor j is updated at each iteration based on the incoming messages from i's other neighbors and node potential function $\phi(i)$ as follows.

$$
m_{i,j}(x_j) \leftarrow \sum_{x_i \in \{x_R, x_{NR}\}} \phi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{k,i}(x_i) \tag{1}
$$

where $N(i)$ is the set of i's neighbors, and $\psi_{i,j}$ is the *edge potential* which is a transformation function defined on the edge between i and j to convert a node's incoming messages into its outgoing messages. Edge potential also controls how much influence can be passed to the receiving nodes, depending on the properties of the connections between i and j (e.g., the number of connections and volume of traffic). $\psi(x_i, x_j)$ is typically set according to the transition matrix shown in Table 1, which indicates that a low-reputation entity (e.g. a less reputable user) is more likely to be associated with low-reputation neighbors (e.g. compromised devices). The algorithm runs iteratively and stops when the entire network is converged with some threshold T, i.e., the change of any $m_{i,j}$ is smaller than T, or a maximum number of iterations are done. Convergence is not theoretically guaranteed for general graphs; however, the algorithm often does converge for real-world graphs in practice. At the end of the propagation procedure, each entity's reputation (i.e. marginal probability distribution) is determined by the converged messages $m_{i,j}$ and the node potential function (i.e. prior distribution).

$$
p(x_i) = k \phi_i(x_i) \prod_{j \in N(i)} m_{j,i}(x_i); \quad x_i \in \{x_R, x_{NR}\} \tag{2}
$$

where k is the normalization constant.

Incorporating domain knowledge

One of the major challenges in adopting a BP algorithm is to properly determine its parameters, particularly, the node potential and the edge potential function. In this section, we briefly discuss how we leverage the available data sources in a typical enterprise network and incorporate specific domain knowledge (unique to each entity type) to infer the parameters.

Characteristics of external servers $\mathcal{S}$

We develop an intrusion detection system that leverages several external blacklists to inspect all the HTTP traffic. It flags different types of suspicious web servers to which internal devices try to connect; this which allows us to

Table 1 Edge potential function

$\psi(x_i, x_j)$	$x_i = x_{NR}$	$x_i = x_R$
$x_j = x_{NR}$	$0.5 + \epsilon$	$0.5 - \epsilon$
$x_j = x_R$	$0.5 - \epsilon$	$0.5 + \epsilon$

assign the node potential function according to the maliciousness of the external servers. Specifically, we classify suspicious servers into the following five types:

- *Spam websites*: servers that are flagged by external spam blacklists like Spamhaus, SpamCop, etc.
- *Malware websites:* servers that host malicious software including virus, spyware, ransomware, and other unwanted programs that may infect the client machine.
- *Phishing websites:* servers that try to purport to be popular sites such as bank sites, social networks, online payment, or IT administration sites in order to lure unsuspecting users to disclose their sensitive information e.g., user names, passwords, and credit card details. Recently, attackers started to employ more targeted spear phishing attacks which use specific information about the target to increase the probability of success. Because of its potential to cause severe damage, we assign a high risky value to its node potential.
- *Exploit websites:* servers that host exploit toolkits, such as Blackhole and Flashpack, which are designed to exploit vulnerabilities of the victims' web browsers and install malware on victims' machines.
- *Botnet C&C servers:* are connected with bot programs to command instructions, update bot programs, or to extrude confidential information. If an internal device makes an attempt to connect to any known botnet C&C servers, the device is likely to be compromised. In addition to blacklists (e.g., Zeus Tracker), we also design models to detect fast fluxing and domain name generation botnets based on their distinct DNS request patterns.

Using the categorization of suspicious servers, we determine initial node potential values according to the severity of their categories. We assign $(\phi(x_R), \phi(x_{NR})) = (0.95, 0.05)$ for the high-risk types, such as botnets and exploit servers. For the medium-risk (Phishing and malware) and low-risk (Spam) types, we assign $(0.75, 0.25)$ and $(0.6, 0.4)$, respectively.

Characteristics of internal entities $\mathcal{D}, \mathcal{U}, \mathcal{C}, \mathcal{A}$

For internal entities, e.g., devices, users, credentials, and assets, rich information can be obtained from the internal asset management systems and IPS/IDS systems. Available information include device's status (e.g., OS version, patch level), device behavior anomalies (e.g., scanning), suspicious user activities (e.g., illegal accesses to sensitive data, multiple failed login attempts), and credential anomalies (e.g., unauthorized accesses). For instance, from the IPS system deployed in our enterprise network, we are able collect over 500 different alert types, most of which are various attack vectors such as SYN port scan, remote memory corruption attempts, bruteforce logon, XSS, SQL injection, etc. Based on these information, we adjust node potential for the internal entities by assigning a severity score (1 to 3 for low, medium, and high-risk alerts)to each type of suspicious activities exemplified above and summing up the severities of all suspicious activities associated with an entity i to get total severity S_i. Since an entity i may be flagged multiple times for the same or different types of suspicious behaviors, to avoid being overshadowed by a few outliers, we transform the aggregated severity score using the sigmoid function

$$P_i = \frac{1}{1 + \exp(-S_i)}$$

The node potential for i is then calculated as $(\phi(x_R), \phi(x_{NR})) = (P_i, 1 - P_i)$. The key benefit of using a sigmoid function is that if no alerts have been reported for an entity (e.g. $S_i = 0$), its initial node potential will automatically be set to $(0.5, 0.5)$ i.e. equal probability of being risky and non-risky, implying that no prior information exist for the particular entity.

Although the parameters in MUSE require some level of manual tuning by domain experts, it is valuable to security analysts in several aspects. First, the output of MUSE is the ranking of high-risk entities whose absolute risk values are less important. As long as the parameters are assigned based on reasonable estimation of the severities of different types of alerts, MUSE will able to derive a ranking of entities based on their potential risks, thus providing useful information to help analysts prioritize their investigation. Second, MUSE offers the flexibility to incorporate diverse types of entities and thus can be easily adapted in a wide variety of other domains. Finally, it is possible to automatically learn the appropriate parameter values through machine learning techniques, provided that proper labeled training sets are available. We leave this as our plan for future exploration.

Scale up propagation algorithm to big data

Another major challenge of applying BP algorithm in a large enterprise network is the scalability. Even though BP itself is a computationally efficient algorithm: the running time scales quadratically with the number of edges in the graph, for large enterprises with hundred of thousands nodes and edges, the computation cost can become significant. To make the MUSE practical for large-scale graphic models, we observe that the main computation in belief propagation is *localized*, i.e. message passing is performed between only a specific node and its neighbors. This means that the computation can be efficiently parallelized and distributed to a cluster of machines.

One of the most prominent parallel programming paradigms is MapReduce, popularized by its open-source

implementation of Apache Hadoop [6]. MapReduce framework consists of the map and the reduce stage that are chained together to perform complex operations in a distributed and fault-tolerant fashion. However, MapReduce is notoriously inefficient for *iterative* algorithms where the intermediate results are reused across multiple rounds of computations. Due to the lack of abstraction for leveraging distributed memory, the only way to reuse data across two MapReduce jobs is to persist them to an external storage system e.g. HDFS and load them back via another Map job. This incurs substantial overheads due to disk I/O and data synchronization which can dominate the execution times. Unfortunately, the BP algorithm underlying MUSE is a typical example of iterative computation where the same set of operations i.e. message update in Equation 1 are repeatedly applied to multiple data items. As a result, instead of MapReduce, we leverage a new cluster computing abstraction called Resilient Distributed Datasets (RDDs) [7] that achieves orders of magnitude performance improvement for iterative algorithms over existing parallel computing frameworks[a].

RDDs are parallel data structures that are created through deterministic operations on data in stable storage or through transformations from other RDDs. Typical transformations include map, filter, join, reduce, etc. The major benefit of RDDs is that it allows users to explicitly specify which intermediate results (in the form of RDDs) they want to reuse in the future operations. Keeping those persistent RDDs in memory eliminates unnecessary and expensive disk I/O or data replication across iterations, thus making it ideal for iterative algorithms. To abstract BP algorithm into RDDs, our key observation is that the message update process Equation 1 can be more efficiently represented by RDDs on an induced *line graph* from the original graph which represents the adjacencies between edges of original graph. Formally, given a directed graph $G = (V, E)$, a directed *line graph* or *derived graph* $L(G)$ is a graph such that:

- each vertex of $L(G)$ represents an edge of G. We use following notations to denote vertices and edges in G and $L(G)$: let $i, j \in V$ denote two vertices in the original graph G, we use (i, j) to represent the edge in G and the corresponding vertex in $L(G)$
- two vertices of $L(G)$ are adjacent if and only if their corresponding edges share a common endpoint ('are incident') in G and they form a length two directed path. In other words, for two vertices (i, j) and (m, n) in $L(G)$, there is an edge from (i, j) to (m, n) if and only if $j = m$ in the original G.

Figure 3 shows the conversion of original graph G to its directed line graph. Since an edge $(i, j) \in E$ in the original graph G corresponds to a node in $L(G)$, the message passing process in Equation 1 is essentially an iterative updating process of a node in $L(G)$ based on all of this node's adjacent nodes. On each iteration, each node in $L(G)$ sends a message (or influence) $m_{i,j}$ to all of its neighbors and at the same time, it updates its own message based on the message it received from the neighbors. This can be easily described in RDDs as follows:

Algorithm 1 Message passing algorithm using RDDs

```
1: // Load line graph L(G) as an RDD of (srcNode, dstNode) pair
2: links = RDD.textFile(graphFile).map(split).persist()
3:
4: // load initial node potential function in original graph G as (node, φ_i) pairs
5: potentials = RDD.textFile(potentialFile).map(split).persist()
6:
7: messages = // initialize RDD of messages as (Node, m_{i,j}) pairs
8:
9: for iteration in xrange(ITERATIONS):
10:        // Build an RDD of (dstNode, m_src) pairs with messages sent by all node to dstNode
11:     MsgContrib = links.join(messages).map(
12:                        lambda(srcNode, (dstNode, m_src): (dstNode, m_src))
13:
14:        // Multiplication of all incoming messages by dstNode
15:     AggContrib = MsgContrib.reduceByKey(labmda (m_1, m_2): m_1 * m_2)
16:
17:   // Get New updated Messages for the next iteration
18:     messages = potentials.join(AggContrib).map(
19:                        labmda(dstNode, (φ_i, m_agg )): φ_i * ψ_{i,j} * m_agg)
20:
21: //After iterations, compute final belief according to Eq. 2
22: belief = potentials.join(messages).mapValues(lambda (φ_i, m_agg): k * φ_i * m_agg)
23:
24: // and save to external storage
25: belief.saveAsTextFile("Beliefs.txt")
```

The above algorithm leads to the RDD lineage graph in Figure 4. On each iteration, the algorithm create a new *messages* dataset based on the aggregated contributions *AggContrib* and *messages* from previous iteration as well as the static *links* and *potential* datasets that are persisted

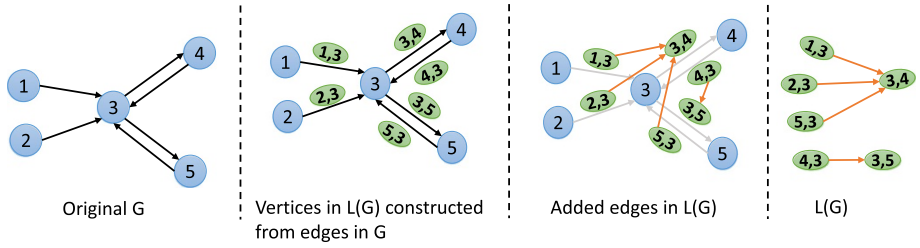

Figure 3 Converting a directed graph G to a directed line graph.

in memory. Keeping these static datasets and intermediate *messages* in memory and making them readily available in the subsequent iterations avoids unnecessary I/O overhead and thus significantly speed up the computation.

Evaluation

Datasets

We evaluated MUSE with datasets collected from multiple data sources in the entire USNorth IBM network. The data sources included DNS messages from local DNS servers, network flows from edge routers, proxy logs, IPS alerts, and HTTP session headers (for categorization of websites). The size of the raw data per day was about 200 GB and the average data event rates are summarized in Table 2.

Experiment results

We first evaluate MUSE using data collected over 1 week period of time. The resulting graph consists of 11,790 nodes and 44,624 edges. The number of different entity types[b] are shown in Table 3.

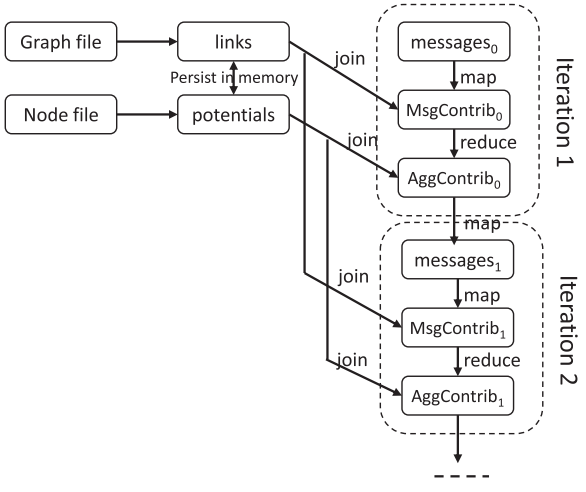

Figure 4 Linage graph for BP algorithms.

We applied MUSE to the graph, and our algorithm converged at the 5th iteration. We manually inspected top ranked entities (i.e., with higher $P(x_R)$) in each entity type and were able to confirm that they were all suspicious or malicious entities including infected devices, suspicious users, etc. Here, we show one example of user reputation among our findings. We selected five top risky users based on the output of MUSE. Figure 5 shows their risk values at each iteration. Note that all the users started with neutral score (0.5, 0.5), meaning that these users had not been flagged by anomalous behaviors. However, due to their interaction with low-reputation neighbors, their associated risks increased. Further investigation showed that user332755 owned five devices which made 56 times of connections to spam websites, four times of connections to malware websites, and two times of connections to exploit websites during our monitoring period. user332755 inherited low reputation from his neighbors including the user's devices, causing his risk to quickly rise to the top.

We also measured the running time of MUSE at each iteration against different size of the graph in terms of the number of edges. Experiments are performed in a server blade with 2.00 GHz Intel(R) Xeon(R) CPU and 500 G memory (the memory usage of MUSE is less than 1 G). The experiment results are shown in Figure 6. From the figure, one can see that BP is efficient in handling small-to-medium-sized graphic models. Even for 1 week worth of traffic data, MUSE is able to finish each iteration in less than 1 min. However, we also notice that the running

Table 2 Average traffic rate for IBM US North network

Data type	Data rate
Firewall logs	950 M/day
DNS messages	1,350 M/day
Proxy logs	490 M/day
IPS/IDS events	4 M/day
Overall:	2.5 billion events/day

Table 3 Number of different entities for one week data

Server					Device	User	Assets
Spam	Malware	Phishing	Exploit	Botnet			
5,500	124	10	26	16	3,823	2,191	100

time quadratically increases with the number of edges, which can be a bottleneck to handle large graphs as we will demonstrate in the next section.

Scalability of MUSE using RDDs

To compare the scalability of non-parallelized BP algorithm with that of the distributed version using RDDs abstraction, we collected half month worth of network traffic to stress test MUSE. The resulting graph consists of 24,706 nodes and 123,380 edges. The number of different entities are listed in Table 4. As a baseline benchmark, we ran the non-distributed version of MUSE against this large dataset for five iterations on the same server blade. The overall experiment took 1,641 s to finish and each iteration on average required 328 s.

We implement a distributed version of MUSE on Apache Spark [8] which is the open source implementation of RDDs. We deploy Spark on our blade center with three blade servers. We vary the number of CPU cores available for the Spark framework from 10 to 30 and submit the same workload to it. Figure 7 illustrates the comparison results. From the figure, we can notice that MUSE is able to leverage RDDs' in-memory cluster computing framework to achieve 10× to 15× speed up. For instance, with 30 CPU cores, MUSE is able to complete five iterations in 104 s with each iteration requiring less than 20 s. Although the algorithm does not scale linearly with the number of cores due to fixed I/O and communication overhead, the results demonstrate that with moderate hardware configuration, MUSE is scalable and practical for large enterprise networks.

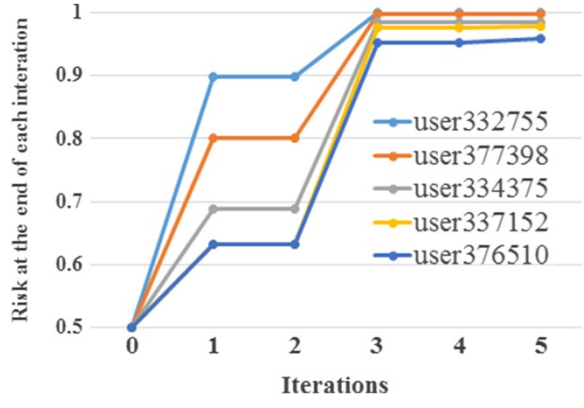

Figure 5 Risk scores of top five risky users at the end of each iteration.

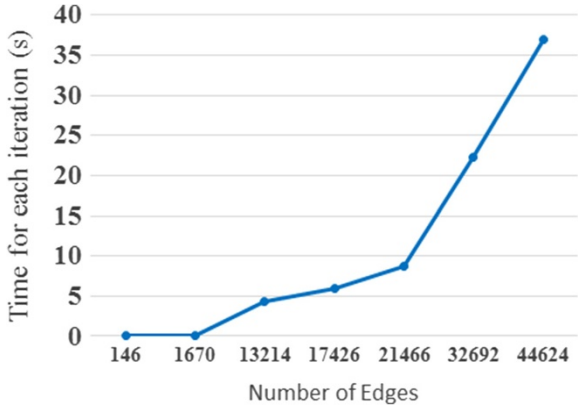

Figure 6 Running time for each iteration with varying sizes of graphs.

Related work

With the cyber threats rapidly evolving towards large-scale and multi-channel attacks, security becomes crucial for organizations of varying types and sizes. Many traditional intrusion detection and anomaly detection methods are focused on a single entity and applied rule-based approaches [9]. They were often too noisy to be useful in practice [10]. Our work is inspired by the prominent research in the social network area that used a link structure to infer knowledge about the network properties. Previous work demonstrated that social structure was valuable to find authoritative nodes [11], to infer individual identities [12], to combat web spam [13], and to detect security fraud [14]. Among various graph mining algorithms, the belief propagation algorithm [2] has been successfully applied in many domains, e.g., detecting fraud [4], accounting irregularities [15], and malicious software [3]. For example, NetProbe [4] applied a BP algorithm to the eBay user graph to identify subgraphs of fraudsters and accomplices.

Conclusion

In this paper, we proposed MUSE, a framework to systematically quantify and rank risks in an enterprise network. MUSE aggregated alerts generated by traditional IPS/IDS on multiple data sources, and leveraged the link structure

Table 4 Number of different entities for half a month data

Server					Device	User	Assets
Spam	Malware	Phishing	Exploit	Botnet			
8,924	171	103	33	22	10,809	4,527	116

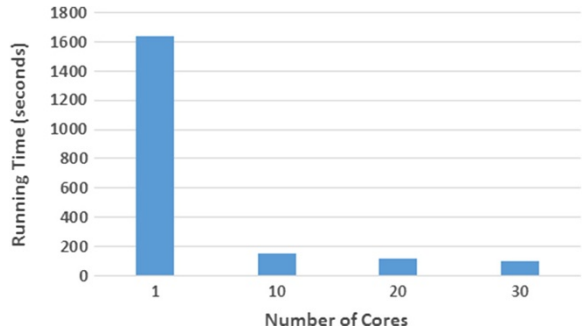

Figure 7 Comparison of running time between non-distributed MUSE and distributed version implemented using RDDs.

among entities to infer their reputation. The key advantage of MUSE was that it derived the risk of each entity not only by considering its own characteristics but also by incorporating the influence from its neighbors. This allowed MUSE to pinpoint a high-risk entity based on its interaction with low-reputation neighbors, even if the entity itself was benign. By providing risk rankings, MUSE helps security analysts to make an informed decision on allocation of resources and prioritization of further investigation to develop proper defense mechanisms at an early stage. We have implemented and tested MUSE on real world traces collected from large enterprise network, demonstrating that the efficacy and scalability of MUSE.

Endnotes

[a]10x-100x speedup as compared to Hadoop [8].

[b]Due to the privacy issues, we were not able to include authentication logs to incorporate user credentials in our experiments.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defense, or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

Author details
[1]Security Research Department, IBM T.J. Watson Research Center, 1101 Kitchawan Rd, Yorktown Heights, NY, USA. [2]IBM Zurich Research Lab, Saumerstrasse 4, 8803 Ruschlikon, Switzerland.

Received: 11 August 2014 Accepted: 28 October 2014
Published online: 18 November 2014

References

1. J Bian, Y Liu, D Zhou, E Agichtein, H Zha, in *Proceedings of WWW '09*. Learning to recognize reliable users and content in social media with coupled mutual reinforcement (ACM New York, 2009)
2. JS Yedidia, WT Freeman, Y Weiss, in *Exploring Artificial Intelligence in the New Millennium, Volume 8*. Understanding belief propagation and its generalizations (Morgan Kaufmann Publishers Inc., San Francisco, 2003), pp. 236–239
3. DH Chau, C Nachenberg, J Wilhelm, A Wright, C Faloutsos, in *SIAM International Conference on Data Mining*. Polonium: tera-scale graph mining and inference for malware detection, (2011)
4. S Pandit, DH Chau, S Wang, C Faloutsos, in *International Conference on World Wide Web*. Netprobe: a fast and scalable system for fraud detection in online auction networks (ACM New York, 2007)
5. PF Felzenszwalb, DP Huttenlocher, Efficient belief propagation for early vision. Int. J. Comput. Vis. **70**, 41–54 (2006)
6. AS Foundation, Apache Hadoop. http://hadoop.apache.org/
7. M Zaharia, M Chowdhury, T Das, A Dave, J Ma, M McCauley, MJ Franklin, S Shenker, I Stoica, in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing (USENIX Association San Jose, 2012), pp. 2–2
8. AS Foundation, Apache Spark Lightning-fast cluster computing. https://spark.apache.org/
9. P Garcia-Teodoro, J Diaz-Verdejo, G Maciá-Fernández, E Vázquez, Anomaly-based network intrusion detection: techniques, systems and challenges. Comput. Secur. **28**, 18–28 (2009)
10. J Viega, *Myths of Security*. (O'Reilly Media, Inc, Sebastopol, 2009)
11. JM Kleinberg, Authoritative sources in a hyperlinked environment. J. ACM. **46**(5), 604–632 (1999)
12. S Hill, F Provost, The Myth of the double-blind review?: Author identification using only citations. ACM SIGKDD Explorations Newsl. **5**(2), 179–184 (2003)
13. Z Gyöngyi, H Garcia-Molina, J Pedersen, in *Intl Conference on Very Large Data Bases*. Combating Web Spam with Trustrank (VLDB Endowment Toronto, 2004)
14. J Neville, O Simsek, D Jensen, J Komoroske, K Palmer, H Goldberg, in *ACM Conference on Knowledge Discovery and Data Mining*. Using relational knowledge discovery to prevent securities fraud (ACM New York, 2005)
15. M McGlohon, S Bay, MG Anderle, DM Steier, C Faloutsos, in *ACM Conference on Knowledge Discovery and Data Mining*. SNARE: a link analytic system for graph labeling and risk detection (ACM New York, 2009)

doi:10.1186/s13635-014-0017-1
Cite this article as: Hu *et al.*: **MUSE: asset risk scoring in enterprise network with mutually reinforced reputation propagation.** *EURASIP Journal on Information Security* 2014 **2014**:17.