Polonium: Tera-Scale Graph Mining for Malware Detection

Duen Horng Chau
Carnegie Mellon University
dchau@cs.cmu.edu

Carey Nachenberg
Symantec
cnachenberg@symantec.com

Jeffrey Wilhelm
Symantec
jeffrey_wilhelm@symantec.com

Adam Wright
Symantec
adam_wright@symantec.com

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

We present Polonium, a scalable and effective technology for detecting malware. We evaluated it with the largest anonymized file submissions dataset ever published, which spans over *60 terabytes* of disk space.

We formulated the problem of detecting malware as a large-scale graph mining and inference task, for which we construct a huge bipartite graph of almost *1 billion* nodes from our data, 48 million of which are users, and 903 million are files. Edges, each denoting a file appearing on a machine, exceeds *37 billion*. Our method for *identifying malware* is to *locate files with low reputation*.

Our Polonium algorithm computes file reputation based on the fast and scalable Belief Propagation algorithm ($O(|E|)$), which iteratively improves inference quality. With one iteration, our method attained 85% true positive rate (in detecting malware). With more iterations, the true positive rate further improves for an additional 2%, which is a significant improvement given the baseline performance is already very good.

We detail important design and implementation features of our method which enable its successful application on our dataset. We also present empirical observations on characteristics and patterns in our large billion-node graph.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – Data Mining

General Terms

Algorithms, Experimentation, Security

Keywords

Graph Mining; Malware Detection; Belief Propagation; Graphical Model; Probabilistic Inference; Large data

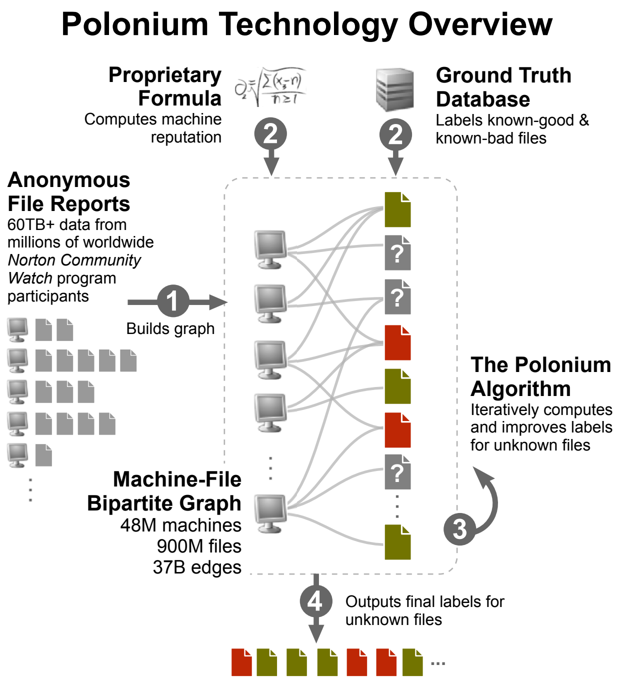
Polonium Technology Overview
Proprietary Formula
Computes machine reputation
2
Ground Truth Database
Labels known-good & known-bad files
2
Anonymous File Reports
60TB+ data from millions of worldwide *Norton Community Watch* program participants
1
Builds graph
?
The Polonium Algorithm
Iteratively computes and improves labels for unknown files
3
Machine-File Bipartite Graph
48M machines
900M files
37B edges
4
Outputs final labels for unknown files
...

Figure 1: Overview of the Polonium technology

1. INTRODUCTION

Thanks to ready availability of computers and ubiquitous access to high-speed Internet connections, malware has been rapidly gaining prevalence over the past decade, spreading and infecting computers around the world at an unprecedented rate. In 2008, Symantec, the world's leading security software provider, reported that the release rate of malicious code and other unwanted programs may be exceeding that of legitimate software applications [?]. This suggests traditional signature-based malware detection solutions will face great challenges in the years to come, as they will likely be outpaced by the number of threats being created by malware authors.

As an example, Symantec released nearly 1.8 million virus signatures (or definitions) in 2008, resulting in 200 million [?] detections per month in the field. While this is certainly a huge amount of blocked malware, Symantec estimates that a great deal more malware (so-called "zero day" malware [?]) is being generated or mutated for each victim or small number of victims, which tends to evade traditional

Technical term	Synonyms	Meaning
Malware	Bad software, malicious software, infected file	Short for malicious software, which includes computer viruses, Trojan, etc.
Reputation	Goodness, belief (when discussing the Polonium algorithm)	A measure of the goodness; can be used on *machines* and *files* (e.g., file reputation)
File	Software, application, executable, program	A software instance, typically an executable (e.g., .exe) on the user's computer reported to Symantec
Machine	Computer	A user's computer; a user can have multiple computers
File ground truth	–	File label, *good* or *bad*, assigned by human security experts
Known-good file	–	File with *good* ground truth
Known-bad file	–	File with *bad* ground truth
Unknown file	–	File with *unknown* ground truth
Positive (as in *true positive*)	–	Malware instance
True Positive	TP	Malware instance correctly identified as bad
False Positive	FP	A good file incorrectly identified as bad

Table 1: Malware detection terminology. The reader may want to refer to this table for meanings and synonyms of technical terms.

signature-based antivirus scanners. This has prompted the software security industry to rethink their approaches in detecting malware, which have heavily relied on refining existing signature-based protection models pioneered by the industry decades ago. A new, radical approach to the problem is needed.

The New Reputation-Based Approach.

Symantec introduced the new protection model that computes a *reputation* score for every application that users may encounter, and protects them from files with *poor reputation*. Good applications typically are used by many users, from known publishers, and have other attributes that characterize their legitimacy and good reputation. Bad applications, on the other hand, typically come from unknown publishers, have appeared on few computers, and have other attributes that indicate poor reputation. The application reputation is computed by leveraging tens of terabytes of data anonymously contributed by the millions of users participating in the worldwide Norton Community Watch program [**?**]. These anonymous data contain important characteristics of the applications running on their systems [**?**].

The New Polonium Technology.

At the core of the reputation-based engine is the ensemble of Machine Learning and Data Mining algorithms that sifts through huge amount of data to statistically infer the goodness of any unknown application.

In this paper, we describe Polonium, a new malware detection technology developed at Symantec that computes application reputation (Figure 1), which works in concert with Symantec's many other technologies to detect malware. Polonium stands for "***P****ropagation* ***Of*** ***L****everage* ***Of*** ***N****etwork* ***I****nfluence* ***U****nearths* ***M****alware*". We make the following contributions through Polonium:

- In Section 3, we present patterns and characteristics observed in our huge anonymized file submissions dataset, and the machine-file bipartite graph constructed from it, each of which is the largest of its kind ever published. Edges in the graph, each denoting a file appearing on a machine, exceeds *37 billion*.
- In Section 4, we propose the iterative Polonium algorithm that efficiently computes application reputation. In addition we show how domain knowledge is readily incorporated into the algorithm's operation to identify malware.
- In Section 5, we demonstrate that Polonium is fast, effective, and scalable, with experiments on the complete machine-file graph. We show that the iterative Polonium algorithm achieves a very high TPR of 84.9% (*true positive rate*, in correctly identifying malware) even with only one iteration, when measured at 1% FPR (*false positive rate*, in mislabeling good files as bad). The TPR further improves to 87.1% after 6 more iterations. Given that the baseline TPR at the first iteration is already high, this subsequent improvement is significant.
- Also in Section 5, we share the non-trivial design and implementation decisions that we made while developing Polonium, which halve both the run time and storage requirement.

To enhance readability of this paper, we have listed the malware detection terminology used in this paper in Table 1. The reader may want to return to this table throughout this paper for technical terms' meanings and synonyms used in various contexts of discussion. One important note is that we use both the words "file" and "application" to refer to an *executable* file.

2. MALWARE DETECTION & GRAPH MINING

A *malware instance* is a program that has malicious intent [**?**]. Its category of malicious code includes viruses, worms, Trojan horses, rootkits, spyware, adware, and more

[?]. While some types of malware, such as viruses, are certainly malicious, some are on the borderline. For example, some "less harmful" spyware programs collect the user's browsing history, while the "more harmful" ones steal sensitive information such as credit card numbers and passwords; depending on what it collects, a spyware can be considered malicious, or only undesirable.

The focus of our work is not on classifying software into these, sometimes subtle, malware subcategories. Rather, our goal is to come up with a new, high-level method that can automatically identify more malware instances similar to the ones that have already been flagged by Symantec as harmful and that the user should remove immediately, or would be removed automatically for them by Symantec's security products. This distinction differentiates our work from existing ones that target specific malware subcategories.

2.1 Research in Malware Detection

There has been significant research in most malware subcategories. Idika and Mathur [?] comprehensively surveyed 45 state-of-the-art malware detection techniques and broadly divide them into two categories: (1) *anomaly-based detection*, which detects malware's deviation from some presumed "normal" behavior, and (2) *signature-based detection*, which detects malware that fits certain profiles (or signatures).

There have been an increasing number of researchers who use data mining and machine learning techniques to detect malware [?]. Kephart and Arnold [?] were the pioneers in using data mining techniques to automatically extract viruses signatures. Schultz et al. [?] were one of the first who used machine learning algorithms (Naive Bayes and Multi-Naive Bayes) to classify malware. Tesauro et al. [?] used Neural Network to detect "boot sector viruses", with over 90% true positive rate in identifying those viruses, but at a 15-20% false positive rate. Also, they only had access to fewer than 200 malware samples. One of the most recent work by Kolter and Maloof [?] used TFIDF, SVM and decision trees on n-grams.

However, most research only takes into account characteristics of the malware in question, but has not taken into account those of the users who may use the malware. Our work makes explicit our strong leverage in propagating and aggregating machine reputation information for a file to infer its goodness.

Another important distinction is our unique access to a huge dataset. Most earlier works were only able to train and test their algorithms on file samples in the thousands; we have access to over 900M files, which allows us to perform testing that is more comprehensive.

2.2 Research in Graph Mining

Authority & Trust Propagation.
Finding authoritative nodes is the focus of the legendary PageRank [?] and HITS [?] algorithms; at the high level, they both consider a webpage as "important" if other "important" pages point to it. In effect, the importance of webpages are propagated over hyperlinks connecting the pages. TrustRank [?] propagates trust over a network of webpages to identify useful webpages from spam (e.g., phishing sites, adult sites, etc.) Tong et al. [?] uses *Random Walk with Restart* to find arbitrary user-defined subgraphs in an attributed graph. For the case of propagation of two or more competing labels on a graph, semi-supervised learning methods [?] have been used. Also related is the work on relational learning (see, e.g., Neville et al. [?, ?]), which aggregates features across nodes to classify movies and stocks.

Fraud Detection & Graph Mining.
The NetProbe system from Pandit et al. [?] models eBay users as a tripartite graph of *honest* users, *fraudsters*, and their *accomplices*; NetProbe uses the Belief Propagation algorithm to identify the subgraphs of fraudsters and accomplices lurking in the full graph. McGlohon et al. [?] proposed the general SNARE framework based on standard Belief Propagation [?] for general labeling tasks; they demonstrated the framework's success in pinpointing misstated accounts in some general ledger data. But they did not consider the malware detection domain.

There is also a wealth of algorithms for mining frequent subgraphs such as *gSpan*[?], the GraphMiner system [?] and related systems [?, ?, ?].

3. DATA DESCRIPTION

In this section, we familiarize our readers with the huge dataset that the Polonium technology leverages for inferring file reputation.

3.1 Source of Data

The raw submission data are contributed anonymously by the 48 million worldwide users of Norton 2008, 2009, or 2010 products, who chose to participate in the Norton Community Watch program. Thus, the collection period spans almost 3 years.

These raw data are anonymized; we have no access to personally identifiable information. They span over 60 terabytes of disk space. Symantec collects statistics on both legitimate and malicious applications running on each participant's machine—this application usage data serves as input to the Polonium system. The total number of files described in the raw data exceeds 900M.

After Symantec's teams of engineers collected and processed these raw data, we constructed a huge bipartite graph from them, with almost *one billion* nodes and *37 billion* edges. To the best of our knowledge, both the raw file submission dataset and this graph are the largest of their kind ever published. We note however, despite these large numbers, these data are only from a subset of Symantec's complete user base.

Each contributing machine is identified by an anonymized *machine ID*, and each file by a *file ID* which is generated based on a cryptographically-secure hashing function.

We try our best to show as much information about the data as possible, but there is proprietary information that we cannot disclose.

3.2 Machine & File Statistics

A total of 47,840,574 machines have submitted data about files on them. Figure 2 shows the distributions of the machines' numbers of submissions. The two modes approximately correspond to data submitted by two major versions of Norton products, where data collection mechanisms differ. Data points on the left generally represent new machines that have not submitted many file reports yet; with

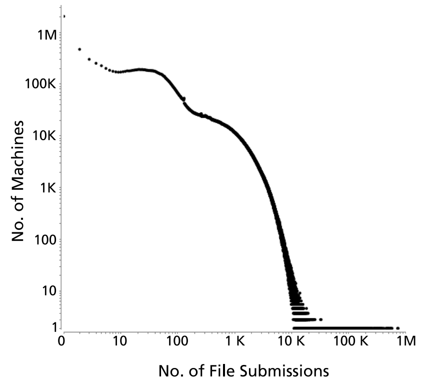

Figure 2: Machine Submission Distribution, in log-log scale.

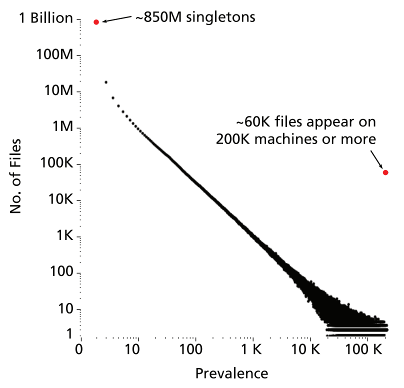

Figure 3: File Prevalence Distribution, in log-log scale. Prevalence cuts off at 200,000 which is the maximum number of machine associations stored for each file. Singletons are files reported by only one machine.

time, these points (machines) gradually move towards the right to join the dominant distribution.

903,389,196 files have been reported in the dataset. Figure 3 shows the distribution of the file prevalence, which follows the Power Law. As shown in the plot, there are about 850M files that have only been reported once. We call these files "singletons". Symantec believes that these singleton files fall into two different categories:

- Malware which has been mutated prior to distribution to a victim, generating a unique variant;
- Legitimate software applications which have their internal contents fixed up or JITted during installation or at the time of first launch. For example, Microsoft's .NET programs are JITted by the .NET runtime to optimize performance; this JITting process can result in different versions of a baseline executable being generated on different machines.

For the files that are highly prevalent, we store only the first 200,000 machine IDs associated with those files.

3.3 Bipartite Graph of Machines & Files

We generated an undirected, unweighted bipartite machine-file graph from the raw data, with almost 1 billion nodes and 37 billion edges (37,378,365,220). 48 million of the nodes are machine nodes, and 903 million are file nodes. An (undirected) edge connects a file to a machine that has the file[1]. All edges are unweighted; at most one edge connects a file and a machine. The graph is stored on disk as a binary file[2] using the *adjacency list* format. This binary file spans over 200GB.

4. THE POLONIUM ALGORITHM

In this section, we present the Polonium algorithm for detecting malware. We begin by describing the malware detection problem and enumerating the pieces of helpful domain knowledge and intuition for solving the problem.

4.1 Problem Description

Our Data. We have a billion-node graph of machines and files, and we want to label a file node as good or bad, along with a measure of the confidence in that disposition. We may treat each file as a random variable $X \in \{x_g, x_b\}$, where x_g is the *good* label (or class) and x_b is the *bad* label. The file's goodness and badness can then be expressed by the two probabilities $P(x_g)$ and $P(x_b)$ respectively, which sum to 1.

Goal. We want to find the marginal probability $P(X_i = x_g)$, or goodness, for each file i. Note that as $P(x_g)$ and $P(x_b)$ sum up to one, knowing the value of one automatically tells us the other.

4.2 Domain Knowledge & Intuition

For each file, we have the following pieces of domain knowledge and intuition, and we would like to use them to help infer the file's goodness, as depicted in Figure 4a.

Machine Reputation.

Symantec has computed a reputation score for each machine based on a proprietary formula that takes into account multiple anonymous aspects of the machine's usage and behavior. The score is a value between 0 and 1. Intuitively,

[1]A machine typically only reports a subset of all of its files (executables).

[2]This binary file is actually composed of multiple files, but to help clarify our discussion, we refer to them collectively as one file.

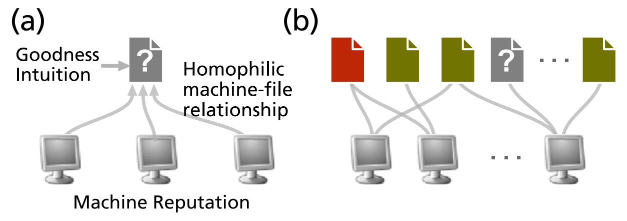

Figure 4: Inferring file goodness through incorporating (a) domain knowledge and intuition, and (b) other files' goodness through their influence on assoicated machines.

we expect files associated with a good machine to be more likely to be good.

File Goodness Intuition.

Good files typically appear on many machines and bad files appear on few machines.

Homophilic Machine-File Relationships.

We expect that good files are more likely to appear on machines with good reputation and bad files more likely to appear on machines with low reputation. In other words, the machine-file relationships can be assumed to follow homophily.

File Ground Truth.

Symantec maintains a *ground truth database* that contains large number of *known-good* and *known-bad* files, some of which exist in our graph. We can leverage the labels of these files to infer those of the unknowns. The ground truth files influence their associated machines which indirectly transfer that influence to the unknown files. This intent is depicted in Figure 4b.

The attributes mentioned above are just a small subset of the vast number of machine- and file-based attributes available at Symantec that are analyzed and leveraged to protect users from security threats.

4.3 Formal Problem Definition

After explaining our goal and information we are equipped with to detect malware, now we formally state the problem as follows:

Given:

- An undirected graph $G = (V, E)$ where the nodes V correspond to the collection of files and machines in the graph, and the edges E correspond to the associations among the nodes.
- Binary class labels $X \in \{x_g, x_b\}$ defined over some nodes in V
- Domain knowledge that may help infer label assignments

Output: Marginal probability $P(X_i = x_g)$, or goodness, for each file.

Our goal task of computing the goodness for each file over the billion-node machine-file graph is an NP-hard inference task [?]. Fortunately, the Belief Propagation algorithm (BP) has been proven very successful in solving inference problems over graphs in various domains (e.g., image restoration, error-correcting code). We adapted the algorithm for our problem. This adaptation was non-trivial, as various components used in the algorithm had to be fine tuned; more importantly, as we shall explain, modification to the algorithm was needed to induce iterative improvement in file classification.

At a high level, the algorithm infers the label of a node from some prior knowledge about the node, and from the node's neighbors. This is done through iterative message passing between all pairs of nodes v_i and v_j. Let $m_{ij}(x_j)$ denote the message sent from i to j. Intuitively, this message represents i's opinion about j's likelihood of being in class x_j. The prior knowledge about a node i, or the prior probabilities of the node being in each possible class are expressed through the *node potential function* $\phi(x_i)$ (which we shall discuss shortly). This prior probability may simply be called a *prior*.

At the end of the procedure, each file's goodness is determined. This goodness is an estimated marginal probability, and is also called *belief*, or formally $b_i(x_i)$ ($\approx P(x_i)$), which we can threshold into one of the binary classes. For example, using a threshold of 0.5, if the file belief falls below 0.5, the file is considered bad.

In details, messages are obtained as follows. Each edge e_{ij} is associated with messages $m_{ij}(x_j)$ and $m_{ji}(x_i)$ for each possible class. Provided that all messages are passed in every iteration, the order of passing can be arbitrary. Each message vector m_{ij} is normalized over j (node j is the message's recipient), so that it sums to one. Normalization also prevents numerical underflow (or zeroing-out values)[3]. Each outgoing message from a node i to a neighbor j is generated based on the incoming messages from the node's other neighbors. Mathematically, the message-update equation is:

$$m_{ij}\left(x_j\right) \leftarrow \sum_{x_i \in X} \phi\left(x_i\right) \psi_{ij}\left(x_i, x_j\right) \prod_{k \in N(i) \backslash j} m_{ki}\left(x_i\right)$$

where $N\left(i\right)$ is the set of nodes neighboring node i, and $\psi_{ij}\left(x_i, x_j\right)$ is called the *edge potential*; intuitively, it is a function that *transforms* a node's incoming messages collected into the node's outgoing ones. Formally, $\psi_{ij}\left(x_i, x_j\right)$ equals the probability of a node i being in class x_i given that its neighbor j is in class x_j. We shall explain how this function is designed to fit our problem.

The algorithm stops when the beliefs converge (within some threshold. 10^{-5} is commonly used), or a maximum number of iterations has finished. Although convergence is not guaranteed theoretically for general graphs, except for those that are trees, the algorithm often converges in practice, where convergence is quick and the beliefs are reasonably accurate. When the algorithm ends, the node beliefs are determined as follows:

$$b_i\left(x_i\right) = k\phi\left(x_i\right) \prod_{x_j \in N(i)} m_{ji}\left(x_i\right)$$

where k is a normalizing constant.

4.4 The Polonium Adaptation of BP

Now, we explain how we solve the challenges of incorporating domain knowledge and intuition to achieve our goal of detecting malware. Succinctly, we can map our domain knowledge and intuition to BP's components (or functions) as follows.

Machine-File Relationships⟶Edge Potential.

We convert our intuition about the machine-file homophilic relationship into the following edge potential shown in Table 2, which indicates that a good file is slightly more likely to be associated with a machine with good reputation than with a low-reputation one. (Similarly for bad file.) ϵ is a small value (we chose 0.001), so that the fine differences between probabilities can be preserved.

[3]Normalization may be performed only when necessary; it is not specified by the algorithm, and is only an implementation requirement

$\psi_{ij}(x_i, x_j)$	$x_i = \text{good}$	$x_i = \text{bad}$
$x_j = \text{good}$	$0.5 + \epsilon$	$0.5 - \epsilon$
$x_j = \text{bad}$	$0.5 - \epsilon$	$0.5 + \epsilon$

Table 2: Edge potentials indicating homophilic machine-file relationship. We choose $\epsilon = 0.001$ to preserve minute probability differences

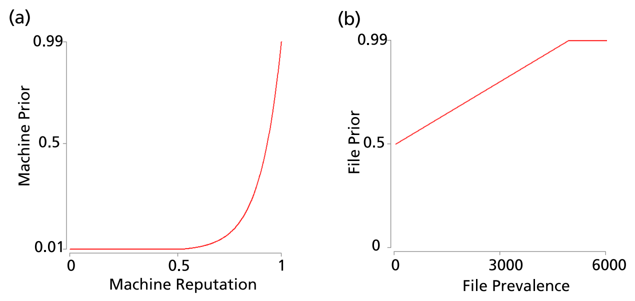

Figure 5: (a) Machine Node Potential (b) File Node Potential

Machine Reputation⟶Machine Prior.

The *node potential function* for machine nodes maps the reputation score that Symantec has computed for each machine into the machine's prior. It is an exponential mapping (see Figure 5a) of the form *machine prior* $= e^{-k \times reputation}$, where k is a numerical constant internally determined based on domain knowledge.

File Goodness Intuition⟶Unknown-File Prior.

Similarly, we use another *node potential function* to set the file prior by mapping the intuition that files that appear on many machines are typically good. Figure 5b shows the function mapping file prevalence into file prior.

File Ground Truth⟶Known-File Prior.

For known-good files, we set their priors to 0.99. For known-bad, we use 0.01[4].

4.5 Modifying The File→Machine Propagation

In standard Belief Propagation, messages are passed along both directions of an edge. That is, the same edge is associated with a *machine→file* message, and a *file→machine* message.

We explained in Section 4 that we use the homophilic edge potential (see Table 2) to propagate machine reputations to a file from its associated machines. Theoretically, we could also use the same edge potential function for propagating file reputation to machines. However, as we tried through numerous experiments – varying the ϵ parameter, or even "breaking" the homophily assumption – we found that machines' intermediate beliefs were often forced to changed too significantly, which led to an undesirable chain reaction that changes the file beliefs dramatically as well, when these machine beliefs were propagated back to the files. We hypothesized that this happens because for a machine's reputation (used in computing the machine node's prior) is a reliable indicator of machine's beliefs, while the reputations of the files that the machine is associated with are weaker indicators. Following this hypothesis, instead of propagating file reputation directly to a machine, pass it to the proprietary formula that Symantec uses to generate machine reputation, which re-compute a new reputation score for the machine. Through experiments discussed in Section 5, we show that this modification leads to iterative improvement of file classification accuracy.

In summary, the key idea of the Polonium algorithm is that it infers a file's goodness by looking at its associated machines' reputations iteratively. It uses all files' current goodness to adjust the reputation of machines associated with those files'; this adjusted machine reputation, in turn, is used for re-inferring the files' goodness.

[4]Note that no probability is ever 0, because it can "zero-out" other values multiplied with them. A lower bound of 0.01 has been imposed on all probabilities. Upper bound is, therefore, 0.99, since probabilities of the two classes add up to 1.

5. EXPERIMENTS

In this section, we show that the Polonium algorithm is scalable and effective at iteratively improving accuracy in detecting malware.

We evaluated the Polonium algorithm with the bipartite machine-file network constructed from the raw file submissions data. The graph consists of about 48 million machine nodes and 903 million file nodes. There are 37 billion edges among them, creating the largest network of its type constructed and analyzed to date.

All experiments reported were run on a 64Bit Linux (Red Hat Enterprise Linux Server 5.3) with 4 Opteron 8378 Quad Core Processors (4 x 4 = 16 total cores @ 2.4 Ghz) with 256GB of RAM, 1 TB of local storage and 60+ TB of networked storage.

One-tenth of the ground truth files were used for evaluation, and the rest were used for setting file priors.

All TPRs (True Positive Rates) reported in this section were measured at 1% FPR, a level deemed acceptable for our evaluation. Symantec uses myriads of malware detection technologies; false positives from Polonium will be further processed by those technologies, eliminating most, if not all, of them. Thus, the 1% FPR used here only refers to that of Polonium, and is independent of other technologies.

5.1 One-Iteration Results

With one iteration, the algorithm attains 84.9% TPR, for all files with prevalence four or more. A file's prevalence is the number of machines that have reported it. (e.g., a file of prevalence five means it was reported by five machines.)

Since the node beliefs are probabilities between 0 and 1, we can threshold them into the binary class of (*good* or *bad*). For example, using a threshold of 0.5, if the file belief falls below 0.5, the file is considered bad.

We generated 10000 threshold points equidistant in the range $[0, 1]$ and applied them to the file beliefs, generating 10000 pairs of TPR-FPR values. The smooth ROC curve interpolated from these 10000 points is shown in Figure 6.

We evaluated on files whose prevalence is 4 or above. For files with prevalence 2 or 3, the TPR was only 48% (at 1% FPR), too low to be usable in practice. For completeness, the overall TPR for all files with prevalence 2 and higher is 77.1% (at 1% FPR).

It is not unexpected, however, that the algorithm does not perform as effectively for low-prevalence files, because a

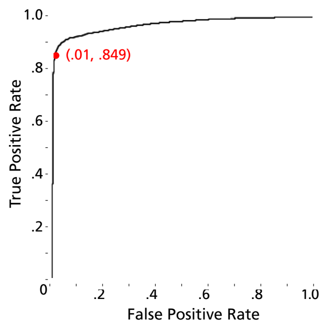

Figure 6: True positive rate and false positive rate for files with prevalence 4 and above.

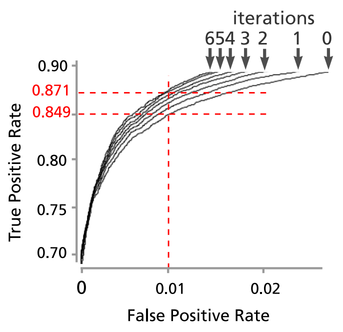

Figure 7: ROC curves of 7 iterations, showing improvements in true positive rate per iteration.

low-prevalence file is associated with few machines. Mildly inaccurate information from these machines can affect the low-prevalence file's reputation significantly more so than if the file was a high prevalence one. We intend to combine this technology with other complementary ones to tackle files in the full spectrum of prevalence.

5.2 Multi-Iteration Results

The Polonium algorithm is iterative. After the first iteration, which attained a TPR of 84.9%, we saw a further improvement of about 2.2% in TPR over 6 iterations (Figure 7), averaging at 0.37% improvement per iteration, where initial iterations' improvements are generally more than the later ones, indicating a diminishing return phenomenon. Since the baseline TPR at the first iteration is already high, this subsequent improvement is significant.

Iterative Improvements.

In Table 3, the first row shows the TPRs from iterations 0 to 6, for files with prevalence 4 or higher. The corresponding (zoomed-in) changes in the ROC curves over iterations is shown in Figure 7.

	Iteration							
Prevalence	0	1	2	3	4	5	6	%↑
≥ 4	***84.9***	*85.5*	*86.0*	*86.3*	*86.7*	*86.9*	*87.1*	2.2
≥ 8	**88.3**	88.8	89.1	89.5	89.8	90	90.1	1.8
≥ 16	**91.3**	91.7	92.1	92.3	92.4	92.6	92.8	1.5
≥ 32	**92.1**	92.9	93.3	93.5	93.7	93.9	93.9	1.8
≥ 64	**90.1**	90.9	91.3	91.6	91.9	92.1	92.3	2.2
≥ 128	**90.4**	90.7	91.4	91.6	91.7	91.8	91.9	1.5
≥ 256	**89.8**	90.7	91.1	91.6	92.0	92.5	92.5	2.7

Table 3: True positive rate (TPR, in %) in detecting malware. In the first row, TPR for all files with prevalence 4 or higher at iteration 0 is 84.9%, and it improves to 87.1% after 6 iterations. Iterative improvements are not limited to low-prevalence files.

We hypothesized that this improvement is limited to very-low-prevalence files (e.g., 20 or below), as we believed the influence they received through the propagated reputation would be comparatively larger than high-prevalence files. To find out whether this hypothesis is true, we gradually excluded the low-prevalence files, starting with the lowest ones, and observed changes in TPR. As shown in Table 3, even after excluding all files below 32 prevalence, 64, 128 and 256, we still saw improvements of over 1.5% over 6 iterations, disproving our hypothesis. This indicate, to our surprise, that the improvements are quite dispersed over the prevalence spectrum.

Goal-Oriented Termination.

As we mentioned earlier in Section 4, the Polonium algorithm's termination criterion is goal-oriented, meaning the algorithm stops when the TPR does not increase any more (at the preset 1% FPR). This is in contrast to Belief Propagation's convergence-oriented termination criterion. In our premise of detecting malware, the goal-oriented approach is more desirable, because our goal is to classify software into good or bad, at as high of a TPR as possible while maintaining low FPR – the convergence-oriented approach does not promise this; in fact, node beliefs can converge, but to undesirable values that incur poor classification accuracy.

We note that in each iteration, we are trading FPR for TPR. That is, boosting TPR comes with a cost of slightly increasing FPR. When the FPR is higher than desirable, the algorithm stops.

5.3 Scalability

We ran the Polonium algorithm on the complete bipartite graph with 37 billion edges. Each iteration took about 3 hours to complete (∼185min), with no significant differences among iterations.

Theoretically, the Polonium algorithm scales linearly with the number of edges in the graph, thanks to its adaptation of the Belief Propagation algorithm; its time complexity is $O(|E|)$. We empirically evaluated this by running the algorithm on the full graph of over 37 billion edges, and smaller billion-edge subgraphs with around 20B, 11.5B, 4.4B and 0.7B edges. We plotted the per-iteration run times for these subgraphs in Figure 8, which shows that the run time empirically achieved linear scale-up.

5.4 Design and Optimizations

We implemented two optimizations that dramatically reduce both run time and storage requirement.

The first optimization eliminates the need to store the *edge file* in memory, which describes the graph structure, by externalizing it to disk. The edge file alone is over 200GB. We were able to do this only because the Polonium algorithm did not require random access to the edges and their

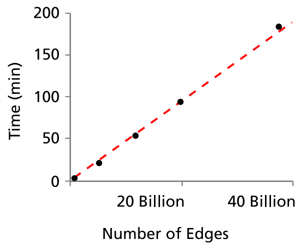

Figure 8: Scalability of Polonium. Run time per iteration is linear in the no. of edges.

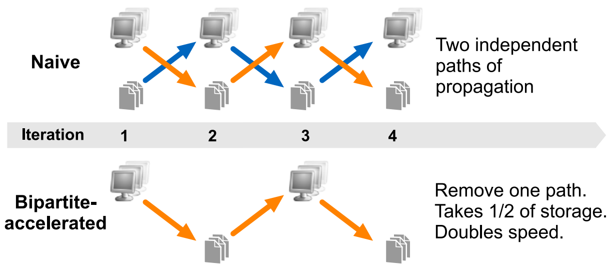
Naive
Two independent paths of propagation
Iteration
1
2
3
4
Bipartite-accelerated
Remove one path. Takes 1/2 of storage. Doubles speed.

Figure 9: Illustration of our optimization for the Polonium algorithm: Since we have a bi-partite graph ('Files' and 'Machines'), the naive version leads to two independent (but equivalent) paths of propagation of messages (orange, and blue arrows). Eliminating one path saves us half of the computation and storage for messages, with no loss of accuracy

associated messages; sequential access was sufficient. This same strategy may not apply readily to other algorithms.

The second optimization exploits the fact that the graph is bi-partite (Machines, and Files) to reduce both the storage and computation for messages by half [**?**]. We briefly explains this optimization here. Let $B_M[i,j](t)$ be the matrix of beliefs (for machine i and state j), at time t, and similarly $B_F[i,j](t)$ for the matrix of beliefs for the files. Because the graph is bi-partited, we have

$$B_M[i,j](t) = B_F[i',j'](t-1) \quad (1)$$
$$B_F[i',j'](t) = B_M[i,j](t-1) \quad (2)$$

In short, the two equations are completely decoupled, as indicated by the orange and blue edges in Figure 9. Both streams of computations seem to converge to the same values, but we can achieve exactly the same results by just following the orange arrows, eventually saving half of the effort.

6. CONCLUSIONS & DISCUSSION

In this paper, we presented Polonium, a scalable and effective technology for detecting malware.

We made the following contributions through Polonium:

- Our Polonium algorithm computes file reputation based on the fast, scalable ($O(|E|)$) Belief Propagation algorithm, which iteratively improves inference quality, which attains 84.9% TPR (at 1% FPR) with just one iteration, and further improves to 87.1% with more iterations.
- We presented empirical observations on the largest anonymized file submissions dataset ever published, which spans over *60 terabytes* of disk space. We also constructed the largest machine-file graph from the dataset and performed our evaluation on it. The graph consists of 37-billion edges among 48 million of users and 903 million files.
- We detailed important design and implementation features of our method which enable its successful application on our large dataset.

In Summary, our experience and positive results with this work has demonstrated one promising solution to the problem of detecting malware. Classifying files by propagating information from their associated machines is a revolutionary idea for malware detection. And Symantec is in a unique position to bring this approach to the users due to Symantec's unmatched installed base. Millions of Norton security product users from around the world contributed their data to Symantec which in turns help Symantec protect them from security threats at a much faster manner, reducing the needs for signatures.

We discuss our plan for future work below.

Using Larger Dataset & More Features.

In this work, we only use a *subset* of all the data collected through the Norton Community Watch program; the attributes mentioned in this paper are just a small subset of the vast number of machine- and file-based attributes available at Symantec that are analyzed and leveraged to protect the users from security threats. By considering more attributes, we may obtain even better malware detection efficacy.

Weighing in File Prevalence & Correlation.

All files are currently treated equally, no matter what their prevalence is. However, in reality, the cost of wrongly labeling a high-prevalence good file as bad has significantly higher cost than mislabeling a low-prevalence one. This points to the need to analyze our results by suitably weighing in the prevalence factor. We may also exploit the fact that some files (or applications) commonly exist together on a computer, to better estimate the reputations for these groups of files; alternative evaluation may then be performed at the *group level*, in addition to the current *file level*.

Further Optimization.

One major piece of future work is to further reduce the algorithm's execution time by parallelizing it, using the framework proposed by Gonzalez et. al [**?**].

7. ACKNOWLEDGEMENTS

Polo Chau is supported by the Symantec Research Labs Graduate Fellowship 2009–2010. We thank Zulfikar Ramzan, Adam Bromwich, and Vijay Seshadri for their helpful comments and suggestions, and Daniel Asheghian for compiling decision tree's efficacy results.