

Gaussian Process Learning for Cyber-Attack Early Warning

Jian Zhang¹, Phillip Porras¹, Johannes Ullrich²

(1) Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

(2) SANS Technology Institute
8120 Woodmont Avenue
Suite 205
Bethesda, MD 20814

Abstract

Network security has been a serious concern for many years. For example, firewalls often record thousands of exploit attempts on a daily basis. Network administrators could benefit from information on potential aggressive attack sources, as such information can help to proactively defend their networks. For this purpose, several large-scale information sharing systems have been established, in which information on cyber attacks targeting each participant network is shared such that a network can be forewarned of attacks observed by others.

However, the total number of reported attackers is huge in these systems. Thus, a challenging problem is to identify the attackers that are most *relevant* to each individual network (i.e., most likely to come to that network in the near future). We present a framework to estimate the relevance of each attacker with respect to each network. In particular, we model each attacker’s relevance as a function over the networks. Different attackers have different functions. The distribution of the functions is modeled using a Gaussian process (GP). The relevance function of each attacker is then inferred from the Gaussian process, that itself is learned from the collection of attack information. We test our framework on the attack reports in the DShield information sharing system. Experiments show that attackers found relevant to a network by our framework are indeed more likely to come to that network in the future.

1 Introduction

With the fast growth of the Internet, there has been a great increase in the number of cyber attacks. This has caused elevated concerns for network security. A recent development in improving cyber defense is to establish large-scale security information sharing systems. Such systems collect security-log data (e.g., attack information) from thousands of participating networks across

the Internet. They provide an important resource for developing global and timely assessments of emerging attack trends [1, 2, 5] and play an important role in global Internet defense.

One of the benefits a participating network can gain from such systems is proactive defense against attackers. By proactive, we mean that the network can be prepared with countermeasures or prevention methods (e.g., blacklisting the source IP of the attacker) against potential exploits and probes that have not been seen by the network itself. This is possible because of the information sharing system. An attacker may have tried its attack strategy with other networks. When detected by these networks, such attack activity is shared in a security log repository. A network that has not experienced this attack can obtain the attack information from the repository and prepare itself to fend off the attacker.

However, there is a challenging problem in building such a proactive defense system. The volume of the reported activities is huge and the number of attackers recorded is often well beyond what a typical network can take effective measures to address. Therefore, *it is essential to prioritize the attackers such that the networks needs to address only a relatively small set of high-priority attackers.*

From a network security perspective, there are several factors one needs to consider in prioritizing attackers, such as how malicious they are and whether they concern the computers in the network. (For example, an attack targeting Windows machines would be of little importance if one’s network hosts only Linux machines.) In this work, we consider a factor of the same importance for attacker prioritization, namely, the *relevance* of an attacker with respect to the network one wants to defend. Relevance can be viewed as the preference of the attacker for a network. The more the attacker prefers the network, the more likely it will target the network in the near future. (Note that

relevance is not exclusive: relevant to one network does not mean that the attacker would not be relevant to the other networks.)

The reason behind considering the relevance in prioritizing the attackers is the opportunity cost. A network has limited resources to deploy defense mechanisms against a certain number of attackers. If the resources are exhausted in defending against attackers that the network will not encounter, while they could have been used against the actual attackers, there is an opportunity cost. A good defense system should have a small opportunity cost. For example, firewalls should be loaded with a blacklist that is likely to be exercised.

Relevance is orthogonal to the other factors involved in attacker prioritization, in the sense that it can be considered separately from the others. In practice, one may rank the attackers separately considering different factors and then combine them together to form a final prioritization. For example, one may form a set of malicious attackers and then prioritize them by relevance. Alternatively, one may order the attackers by their relevance and then select the malicious ones in such order.

A commonly employed measure for ranking attackers is the prolificness of the attacker. The basic idea is that from a network’s viewpoint, if an attacker is prolific—has attacked many networks—it is more likely to come to this network, too. However, prolificness can be too simple to capture the true preference of the attackers. For example, there are attackers that choose their targeting networks more strategically, focusing on a few known vulnerable networks [4]. Such attackers are not necessarily very prolific and hence would be given low priority under the prolificness measure.

To improve the estimate on attacker relevance, we propose a probabilistic framework. We model each attacker’s relevance with respect to the networks as a function over all the networks. Different attackers have different relevance functions. The set of functions for all the attackers forms a distribution. We use a Gaussian process to model this distribution. Each individual attacker’s relevance function follows the posterior distribution of the Gaussian process given the attacker’s reported activities. Intuitively, the posterior reflects the attacker-specific relevances while the Gaussian process prior gives the global trends among all the attackers. Therefore, our relevance measure is calculated from both individual preferences and the global preference of the whole population.

We learn the Gaussian process prior from the collection of reports using an Expectation-Maximization (EM) algorithm. Because the contributed security logs contain only the activities of the attackers observed up

to the current time, to construct a Gaussian process prior that reflects the true global trends, it is necessary to consider possible future attacks. In our framework, these future attacks are accounted for using the EM algorithm, leading to a better Gaussian process that gives better estimation of the relevances.

We tested our framework on real network attack data collected by the DShield [1] information sharing system. We simulate a real application: suppose a network has the resources to address only Q attackers. We construct a list of the Q most relevant attackers. We then test how many of the attackers on the list will hit the network in the near future. We compare the list constructed by our framework with two other lists: one built using a collaborative filtering approach, and the other using the prolificness measure. Our experiments show that for the majority of the networks, our framework produces a higher hit count than the other lists. Furthermore, this advantage is consistent over time.

The rest of the paper is organized as follows. We review related work in the next section. Section 3 presents the details of our learning and inference schemes. Section 4 presents the results of our experiments, and we conclude with Section 5 and discuss future directions.

2 Related Work

Several security information sharing systems [1, 5] provide attacker notification in the form of a blacklist. Most of the blacklists are constructed using prolificness measures: an attacker is selected to be on the list according to how many networks have observed its attack. The more networks observe the attacker, the more preferred the attacker is to be on the list. Such blacklist formulation is very different from our framework.

Gaussian processes has been used in many learning scenarios [15, 14, 9]. Related areas include applications such as picture recommendation [13] and automatic music playlist construction [8]. Although the general GP model is the same, different application areas have different concerns. For example, in music playlist construction, the authors [8] are more concerned with extending the GP learned from the users’ playing experiences to completely new songs. In our case, we simplify the GP inference because of scalability concerns. Furthermore, special constraints on our application domain lead to a specific GP model with a learning process different from the previous ones.

3 Cyber Attack Early Warning

3.1 Problem Formulation We consider an information sharing system with N participating networks. We refer to the set of networks as $U = \{u_1, u_2, \dots, u_N\}$.

The system collects attack reports from the networks and shares this information with all the participants. In the collection, each report records an attack event, containing information such as the source IP of the attacker, the targeted network, the protocol involved, and other attack characteristics. We use the set of recent reports instead of the whole collection for attack early warning. We refer to this set of recent attack reports by \mathcal{C} . Each attacker is identified by its IP address. We denote by $A = \{a_1, a_2, \dots, a_M\}$ the set of attackers in \mathcal{C} where M is the number of attackers in the set. Multiple networks may observe attacks from an attacker a . We use $U(a)$ to denote this set of networks. We also use $U(\bar{a})$ to denote $U \setminus U(a)$, i.e., the networks that did not report a 's activity. Similarly, we denote by $A(u)$ the set of attackers seen by network u and by $A(\bar{u})$ the set $A - A(u)$. For a set X , we use $|X|$ to denote the size of the set.

The goal of an early warning system is to inform a network u of attackers who have not targeted u but have been observed by other networks. A simple solution is to give u the set $A(\bar{u})$. However, as we discussed above, $|A|$ as well as $|A(\bar{u})|$ are quite huge. Therefore, prioritizing the attackers such that the warnings are focused on a small set of high-priority attackers is very important. We consider attacker prioritization by relevance. We assume that the reports are preprocessed such that the attackers that pass the preprocess step are all malicious. Our concern is to prioritize them according to relevance. After prioritization, the warning system can pick a small set of highest-priority attackers in $A(\bar{u})$ to inform each network u . Note that an attacker's relevance to different networks is different. Therefore, the set of highest-priority attackers is different for different networks.

We model the relevance of an attacker a with respect to the networks as a relevance function for the attacker, i.e., a function $f^a : U \rightarrow R$ such that $f^a(u)$ for $u \in U$ is the relevance of a with respect to network u . For $u \in U(a)$, we model $f^a(u)$ as a known value (e.g., $f^a(u) = 1$) and we call it the *observed relevance*. To obtain the complete relevance function, we need to estimate $f^a(u')$ for $u' \in U(\bar{a})$. After obtaining the complete relevance function for all the attackers, for each network u , we can prioritize $a_{i_1}, a_{i_2}, \dots \in A(\bar{u})$ using the function values $f^{a_{i_1}}(u), f^{a_{i_2}}(u), \dots$. Higher function value means higher priority.

Because different attackers may have different preference on the networks, their relevance functions would be different. The set of relevance functions for the collection of attackers forms a distribution, which we model using a Gaussian process. Before we describe the details of the Gaussian process model for the relevance functions and how to learn the Gaussian process from

the reports, we review and summarize in Table 1 the notations that will be often used.

N	# of networks
M	# of attackers
U	set of networks
A	set of attackers
$f^a(u)$	attacker a 's relevance with respect to network u
$U(a)$	the set of networks that have observed attack from a
$U(\bar{a})$	the set of networks that did not observe attack from a
$A(u)$	the set of attackers observed by network u
$A(\bar{u})$	the set of attackers not observed by network u (but have been observed by other networks in U)

Table 1: Frequently Used Notations

3.2 Relevance by Gaussian Process Regression

We first give a brief introduction to Gaussian process. We refer to [10] for a detailed elaboration of the topic. A Gaussian process is a stochastic process, i.e., a collection of random variables indexed by a set. In our case, the random variables are $f(u_i)$ indexed by the set of networks $\{u_i\}$. From a function space point of view, a Gaussian process defines a distribution over a set of functions. Let $\phi(\cdot) = (\phi^1(\cdot), \phi^2(\cdot), \dots)^T$ (T is the vector/matrix transpose) be a (possibly infinite) set of basis functions. If we draw a weight vector \mathbf{w} from a Gaussian distribution, we have a distribution over $\phi(\cdot)^T \mathbf{w}$, i.e., the linear combination of the basis functions. This distribution is a Gaussian process.

A Gaussian process is fully specified by its mean function $\mu(u) = E[f(u)]$ and its kernel (variance) function $K(u, u') = E[(f(u) - \mu(u))(f(u') - \mu(u'))]$. A specific kernel function determines a specific set of basis functions for the Gaussian process. Intuitively, one may view the kernel (variance) function as an estimate of the closeness (similarity) between the networks u and u' . If the two networks are similar, the function values $f(u)$ and $f(u')$ should be close, too.

In this subsection, we assume that the mean and kernel functions of the Gaussian process for the relevance functions are given. We describe, for each attacker a , given the function values of $f^a(u)$ for $u \in U(a)$, how to estimate $f^a(u')$ for $u' \in U(\bar{a})$. In the next section, We will discuss how to learn the mean and kernel functions. Note that when limited to the N networks that we are considering, the Gaussian process is equivalent to an N -dimensional multivariate Gaussian. The

mean function becomes an N -dimensional vector and the kernel function becomes an $N \times N$ -dimensional matrix. The distribution of $f^a(u')$ is the multivariate Gaussian conditioned on the values of $f^a(u)$ for $u \in U(a)$.

To simplify notations, for a function $f^a(\cdot)$, we use \mathbf{f}^a to denote the vector $(f^a(u_1), f^a(u_2), \dots, f^a(u_N))$. We also treat the set $U(a)$ and $U(\bar{a})$ as indices. For example, given the mean vector $\boldsymbol{\mu} = (\mu(u_1), \mu(u_2), \dots, \mu(u_N))$, we use $\mu(U(a))$ to mean the vector $(\mu(u_{i_1}), \mu(u_{i_2}), \dots, \mu(u_{i_k}))$ where $u_{i_j} \in U(a)$ for $j = 1, 2, \dots, k$. For a matrix K , we use $K(u, \cdot)$ to refer to the u -th row and $K(\cdot, u)$ the u -th column. Note that $K(U(a), U(a))$ is then a submatrix indexed by $U(a)$. Given the mean and the kernel functions, $f^a(u')$ for $u' \in U(\bar{a})$ is a Gaussian random variable with mean

$$(3.1)$$

$$K(u', U(a))K(U(a), U(a))^{-1}(f^a(U(a)) - \mu(U(a))) + \mu(u')$$

and variance

$$K(u', u') - K(u', U(a))K(U(a), U(a))^{-1}K(U(a), u')$$

Although $f^a(u')$ is a random variable, for simplicity, we will treat it as a single value taken to be the mean of the random variable. Therefore, $f^a(u')$ for every $u' \in U(\bar{a})$ is computed using Eq. 3.1 to obtain the complete relevance function for the attacker a . After evaluating the complete relevance functions for all the attackers, we prioritize the attackers according to their relevances and inform each network of those attackers with the highest relevance to it.

3.3 Learning Gaussian Process A good relevance estimation requires a Gaussian process prior that reflects the intrinsic relationship between attackers and the networks as well as the relationship among the networks. We now discuss details on how to learn the mean vector and the kernel matrix for such a Gaussian process. The focus, however, is on the learning of the kernel matrix. Similar learning problem has been considered in [12, 13]. We modify the general process in [12] to meet the specific requirements of our application.

As a starting point, one may try to use the mean and the variance of the data to construct the Gaussian process. That is, for each attacker a , set $f^a(u) = 1$ for $u \in U(a)$ and $f^a(u') = 0$ for $u' \in U(\bar{a})$. We have M vectors \mathbf{f}^a and treat them as M samples from a multivariate Gaussian. The mean and variance can be estimated from these samples. This simple treatment, however, is troublesome. Recall that the relevance estimation problem is essentially to estimate $f^a(u')$ for $u' \in U(\bar{a})$. The problem would become meaningless if we had assumed $f^a(u') = 0$.

On the other hand, the collection of reports still provides some information about the kernel matrix. We want a kernel matrix that reflects the similarity between the networks to some extent. The attack reports give some clue to this similarity. When two networks both have certain types of vulnerabilities, attackers that exploit these vulnerabilities would often target both networks [7]. Hence, the overlap of attackers between networks provides a measure on the network similarity. If we associate each network with an M -dimensional attacker vector, with the i -th entry of the vector being 1 if the network has observed the i -th attacker and 0 otherwise, the normalized attacker overlap can be measured by the cosine of the two vectors. The cosines of every pair of the networks form a matrix that we call the *cosine similarity matrix* and denote by Λ . The kernel matrix of our Gaussian process prior should not be too far from this cosine similarity matrix.

There is another factor we need to consider when learning the kernel matrix. In practice, that a network belongs to $U(\bar{a})$ does not mean that the attacker a will not target the network. An attacker may target many networks. The reports, on the other hand, contains only the observed attacks in a time window up to the current time. The possible attacks an attacker may make in the near future are not accounted for in the collection of the reports. Hence, the data themselves do not capture the full relationship between the attackers and networks. For an attack early-warning framework, it is necessary to identify the trend of an attacker that includes future possibilities, in order to make good predictions. Therefore, it is necessary to construct a Gaussian process that takes future attacks into consideration.

Putting the above together, we want to learn a kernel matrix that is not too far from the cosine similarity matrix and at the same time includes possible future attacks. To achieve this, it is natural to have an iterative process as follows: starting with some initial Gaussian process, we estimate the relevance function for each attacker. We then treat these estimations as observations, and construct another Gaussian process using the set of estimated relevance functions. We can repeat these steps until arriving at a good estimate. If in the second step, the Gaussian process is obtained by maximizing likelihood, the above is a maximum-likelihood EM process. In our case, we have an additional constraint that the Gaussian process kernel should be in the neighborhood of the cosine similarity matrix. We model this constraint by a prior on the parameters of the Gaussian process, which then leads to a hierarchical model.

The hierarchical model can be described in a fashion of data generation. Two steps are involved to obtain

the relevance functions: In the first step, a pair $(\boldsymbol{\mu}, K)$ is drawn from a certain distribution. $(\boldsymbol{\mu}, K)$ defines a Gaussian process with mean vector $\boldsymbol{\mu}$ and kernel matrix K . In the second step, the relevance functions for each attackers are draw from this Gaussian process. The joint distribution is then

$$p(\boldsymbol{\mu}, K) \cdot \prod_{a=1}^M p(\mathbf{f}^a | \boldsymbol{\mu}, K).$$

We use a Normal-inverse-Wishart distribution [6, 12] as the prior for the parameter pair. The probabilities in the joint distribution are defined as the following:

$$(3.2) \quad K \sim IW(\nu, K_0^{-1})$$

$$(3.3) \quad \boldsymbol{\mu} | K \sim N(\boldsymbol{\mu}_0, \lambda^{-1} K)$$

$$(3.4) \quad \mathbf{f} | \boldsymbol{\mu}, K \sim \mathcal{GP}(\boldsymbol{\mu}, K)$$

where IW represents inverse-Wishart with $E[K] = \frac{1}{\nu - N - 1} K_0$ and $\text{mode}(K) = \frac{1}{\nu + N + 1} K_0$. The hyperparameter K_0 is set such that the prior distribution of the kernel matrix achieves maximum density at the cosine similarity matrix Λ , i.e., $K_0 = (\nu + N + 1)\Lambda$.

The learning process needs to infer the posterior of $\boldsymbol{\mu}$ and K given the set of observed relevances $\cup_{a=1}^M \{f^a(u) | u \in U(a)\}$. One approach to do inference in hierarchical models is Gibbs sampling (i.e., a Markov Chain Monte Carlo process). In one step, we sample the relevance functions according to the Gaussian process, and in the next step we draw the Gaussian process parameters according to the posterior given the samples. The sequence of the samples forms a Markov chain whose stationary distribution gives the distribution we are trying to estimate. Clearly, the MCMC estimation is quite expensive. Therefore, we use an EM process similar to the maximum-likelihood EM to learn the parameters instead.

Recall that expectation-maximization can be interpreted as follows: Given samples for the complete set of random variables, the parameters can be inferred by maximizing data likelihood. When we have samples for only some of the random variables, we can learn the parameters by maximizing the expected data likelihood, where the expectation is over the distribution of the unseen random variables, conditioned on the sample and the previously estimated parameters. In particular, let θ be the set of parameters. Let Y_o and Y_u be the random variable that can be sampled and the unseen/hidden random variable, respectively. EM iterates by

$$\theta^{(n+1)} = \text{argmax}_{\theta} E_{Y_u} [\log p(Y_o, Y_u | \theta) | Y_o, \theta^{(n)}].$$

where n indicates the step number. In our case, $\theta = (\boldsymbol{\mu}, K)$. Because we use a prior distribution on the

parameters, our situation is a little different from the standard EM. Let $\tilde{F} = \{\tilde{\mathbf{f}}^a\}$ be a set of M relevance functions formed according to Eq. 3.2 3.3 and 3.4. Instead of having data likelihood $p(\tilde{F} | \theta)$, we have the parameter posterior $p(\theta | \tilde{F}) \propto p(\tilde{F} | \theta) p(\theta)$. However, this posterior can be viewed as a penalized data likelihood and used in place of the data likelihood in the above equation. This gives the update rule for our EM learning:

$$(3.5) \quad \theta^{(n+1)} = \text{argmax}_{\theta} E[\log p(\theta | \tilde{F}) | F_o, \theta^{(n)}].$$

where F_o is the set of observed relevances, i.e., $\cup_{a=1}^M \{f^a(u) | u \in U(a)\}$.

The distribution used for the prior $p(\theta)$ is the Normal-inverse-Wishart, which is conjugate to the Gaussian process likelihood $p(\tilde{F} | \theta)$. Therefore, the parameter posterior, in other words, the penalized data likelihood $p(\theta | \tilde{F})$, has the same form as the prior. Let $\bar{\mathbf{f}}$ be the mean of the relevance functions in \tilde{F} and $S = \sum_a (\tilde{\mathbf{f}}^a - \bar{\mathbf{f}})(\tilde{\mathbf{f}}^a - \bar{\mathbf{f}})^T$. the penalized data likelihood for \tilde{F} has the form

$$p(\theta | \tilde{F}) = N(\boldsymbol{\mu}^*, (\lambda^*)^{-1} K) \cdot IW(\nu^*, (K^*)^{-1}).$$

where

$$\nu^* = \nu + M$$

$$\lambda^* = \lambda + M$$

$$\boldsymbol{\mu}^* = \frac{\lambda}{\lambda + M} \boldsymbol{\mu}_0 + \frac{M}{\lambda + M} \bar{\mathbf{f}}$$

$$K^* = K_0 + S + \frac{\lambda M}{\lambda + M} (\bar{\mathbf{f}} - \boldsymbol{\mu}_0)(\bar{\mathbf{f}} - \boldsymbol{\mu}_0)^T$$

Our task is to maximize the expectation of $p(\theta | \tilde{F})$. Note that $p(\theta | \tilde{F})$ is in the exponential family. Given the parameters θ , this probability is determined by the sufficient statistics of the data. Therefore, in the E-step, we need to obtain only the *expected* sufficient statistics. In the M-step, we search for the value of the parameter that maximizes $p(\theta | \tilde{F})$ determined by the sufficient statistics. In this case, the optimum value of the parameters is the mode of the distribution $p(\theta | \tilde{F})$.

Summing up the above, we obtain the following EM iteration for the update rule in Eq. 3.5:

E-step: for each attacker $a = 1, 2, \dots, M$, estimate the mean of its relevance function conditioned on the observed relevances:

$$\tilde{\mathbf{f}}^a = K(\cdot, U(a)) K(U(a), U(a))^{-1} (f^a(U(a)) - \boldsymbol{\mu}(U(a))) + \boldsymbol{\mu}$$

and the variance:

$$S^a = K - K(\cdot, U(a)) K(U(a), U(a))^{-1} K(\cdot, U(a))^T$$

Once we have the estimations for all attackers, we compute

$$\begin{aligned}\bar{\mathbf{f}} &= \frac{1}{M} \sum_a \tilde{\mathbf{f}}^a \\ S &= \sum_a \left(S^a + (\tilde{\mathbf{f}}^a - \bar{\mathbf{f}})(\tilde{\mathbf{f}}^a - \bar{\mathbf{f}})^T \right)\end{aligned}$$

M-step: update the mean and the kernel:

$$\begin{aligned}\boldsymbol{\mu} &= \frac{1}{\lambda + M} \left(\lambda \boldsymbol{\mu}_0 + \sum_a \tilde{\mathbf{f}}^a \right) \\ K &= \frac{1}{\Gamma + M} \left(\Gamma \Lambda + S + \frac{\lambda M}{\lambda + M} (\boldsymbol{\mu} - \boldsymbol{\mu}_0)(\boldsymbol{\mu} - \boldsymbol{\mu}_0)^T \right)\end{aligned}$$

where $\Gamma = (\nu + N + 1)$.

Note that the result of the EM learning gives us a distribution of the mean vector and the kernel. To infer the relevance function for an attacker, a Bayesian approach would integrate over the possible choice of the mean and the kernel under this distribution to obtain the relevance values. In our framework, for efficiency and simplicity considerations, we use the mode of the distribution as the point estimation of the parameters. The relevance function for each attacker is inferred using this point estimation.

However, the framework is still quite expensive even after this simplification. The bottleneck lies in the calculation of the variance for each attacker in the E-step. As with most attackers, we observe only very few attacks in the collection of reports. The relevance values of these attackers with respect to most networks need to be estimated and hence lead to the calculation of the variance. Given that the number of attackers is huge, it is very expensive to run the E-step on the full set of attackers. To overcome this problem, we use a sample of the attackers and their reported attack activities to learn the kernel.

Because the distribution of the number of observed attacks from each attacker is highly skewed, we use a stratified sampling approach to construct the sample. The whole set of attackers $\{a_i\}$ is divided into groups according to $|U(a_i)|$. We draw samples from each group with the sample size proportional to that group’s size. The EM learning is then performed on the union of the samples from all the groups. We summarize the whole process for computing the relevance in Figure 1

4 Experiments

Since we use the relevance values to prioritize (rank) the attackers, the values need to reflect only the relative relevance of the attackers and do not need to give the

1. Construct a sample of attackers using stratified sampling.
2. Use the EM algorithm in Section 3.3 to learn the mean and kernel functions of a Gaussian process from the reported activities of the attackers in the sample.
3. Use Eq. 3.1 to compute the relevance value for each attacker with respect to each network. Prioritize the attackers according to the relevance value. (High relevance means high priority.)

Figure 1: Summary of Relevance Estimate

exact probability of how likely the attackers will come to the network. To test the effectiveness of prioritizing by relevance, we simulate a real situation: Given a prioritization of the attackers, suppose a network has the resources to address only the top Q attackers. The effectiveness of the prioritization would then be measured by the number of attackers in the top Q list that actually come to the network in the near future. We view the top Q list as a blacklist of the attackers. We call the list constructed by our relevance ranking the Gaussian-Process-based BlackList (GPBL). Note that different GPBLs are produced for different networks. If an attack on the list actually comes to the network in the future, we say that there is a hit on the list. Clearly, the more hits a list experiences, the more effective it is.

To perform the test, we use the collection of attack reports in the DShield repository. We examined a collection of approximately 200 million DShield records, involving more than 1000 contributors in May 2006.

In the experiments, using data for a certain time period, we construct a list of Q attackers with the top relevance values for each network. We call this period the *training window*. We then test the list of attackers on the data from the time window following the training period. We call this period the *testing window*. Our experiments show that the choice of Q , as well as the length of the training and the testing window do not change the general results. Therefore, we use training and testing windows of length 2 days and fix Q to be 200.

To assess the effectiveness of GPBL, we compare it to both a blacklist generated using a collaborative filtering method and a blacklist generated by the commonly

used prolificness measure.

4.1 Hit Improvement One may view relevance estimation as a collaborative filtering [3, 11] problem. The networks are the items, and the attackers are the customers. An attacker’s relevance to a network then corresponds to the preference of the customers. Note that different from collaborative filtering, we are not trying to identify the items that a customer would like the most. Rather, we are trying to select the customers (the attackers) that are most likely to purchase (target) an item (network). Nevertheless, collaborative filtering techniques still apply here. We compare our GPBL to a list generated by an item-based collaborative filtering approach.

We use the cosine similarity matrix as the item similarity matrix in the collaborative-filtering approach. Let $\Lambda(u_i, u_j)$ be the cosine between the attacker vector of network u_i and network u_j . (Recall that the attacker vector for a network has the i -th entry set to 1 if the network has observed the i -th attacker and 0 otherwise.) For an attacker a , the collaborative-filtering approach would rank the attacker with respect to the network u' by $\sum_{u \in U(a)} \Lambda(u', u)$. After ranking all the attackers with respect to all the networks, a top 200 list can be formed for each network. We call this list the Collaborative-Filtering-based BlackList (CFBL).

We also construct a list by prolificness measure. The prolificness-based prioritization simply ranks attackers by the number of networks that have reported the attacker. Similarly, after ranking all the attackers, a blacklist of the most prolific attackers can be constructed for each network. We call the prolificness-based list WOL (worse offender list). Prolificness-based methods are commonly used in practice to form blacklists. Note that in our case, for a network, WOL contains only the highly prolific attackers that have not yet been seen by the network. (It is different from the ones used in practice that may contain both seen and unseen attackers.)

To compare the three types of lists, we take 20 days of data, divide into ten 2-day windows. We repeat the experiment 10 times using the i -th window as the training window and the $(i+1)$ -th window as the testing window. In the training window, we construct GPBL, CFBL and WOL. Then the three types of lists are tested on the data in the testing window.

Table 2 shows the total number of hits summed over the networks for GPBL, CFBL and WOL, respectively. It also shows the ratio of GPBL hits over that of WOL and CFBL. We see that in every window, GPBL has more hits than CFBL and WOL. Overall, GPBLs predict 20-30% more hits than CFBL and about 40-50%

more hits than WOL.

Note that there are quite large variances among the number of hits between time windows. Most of the variances, however, are not from our blacklist construction. Rather they are from the variance among the number of attackers the networks experience in different testing windows. For example, the networks may observe 500 thousand attackers in a time window. In the next time window, they may observe only 300 thousand attackers. This variance among the number of attackers causes the number of hits to vary. (The hit numbers of the three lists change in the same way.) To confirm the advantage of GPBL over the other lists, we performed paired T-tests on the hit numbers of GPBL versus the hit numbers of WOL as well as GPBL versus CFBL. It shows that GPBL has significant advantage over the other lists.

4.2 Hit Improvement Distribution We now investigate the distribution of the GPBL’s hit improvement to WOL across the networks. We consider two quantities. One is the improvement, i.e., number of hits on GPBL minus number of hits on WOL. The other is the relative improvement, i.e., the percentage of improvement over WOL’s hit number.

The distributions of the two quantities in different time windows have similar shapes. Therefore, we plot the distributions of window 4 in Figure 2 as an example. The left panel of the figure plots the histogram showing the distribution of the improvements across the networks. The x-axis indicates amount of improvement and the y-axis represents number of networks. We see that for most networks, GPBL leads to an improved hit number. Only on a few networks, GPBL predicted fewer hits than WOL. In the best case, GPBL had 126 hits and WOL had only 9. For the worst case, GPBL had 4 hits while WOL had 11 hits. (In the next section, we will discuss possible reasons why there is a small set of networks for which the GPBL hit number may be worse than that of the other lists.)

The panel on the right of Figure 2 plots the relative improvement distribution using a cumulative percentage plot. There are networks for which GPBL achieves relative improvement of more than 4000%. Instead of showing the whole distribution, we cut off the plot at relative improvement of 200. From the plot, we see that there are about 10% of networks for which the GPBLs achieve an improvement more than 200%. For about half of the networks, the GPBLs have about 20% or more hits. The GPBLs have more hits than WOL for almost 70% of the networks. Only for a few networks (less than 10%), GPBLs perform worse.

Window	WOL total hit	CFBL total hit	GPBL total hit	GPBL/WOL	GPBL/CFBL
1	6849	7605	9673	1.41	1.27
2	8192	9452	12205	1.49	1.29
3	8269	9607	12677	1.53	1.32
4	9247	10959	13429	1.45	1.23
5	8493	9704	11916	1.40	1.23
6	8836	10206	13304	1.51	1.30
7	7806	9359	11960	1.53	1.28
8	7895	9385	11844	1.50	1.26
9	8636	10354	12494	1.45	1.21
10	7914	9181	11662	1.47	1.27
Average	8214 ± 661	9581 ± 886	12116 ± 1050	1.47 ± 0.046	1.27 ± 0.036

Table 2: Hit Number Comparison between GPBL, CFBL and WOL

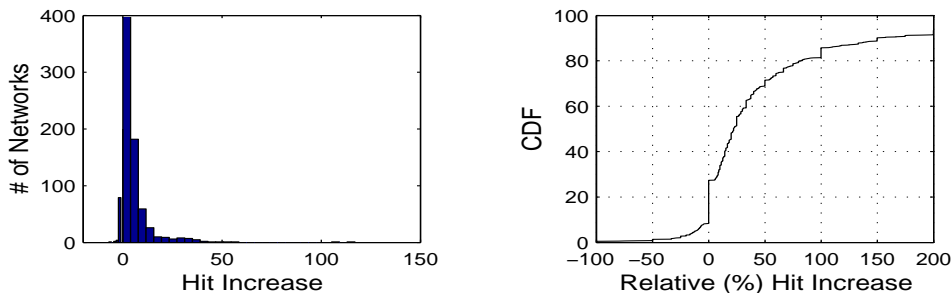


Figure 2: Distribution of GPBL Hit Improvement across Networks

4.3 Performance Consistency The results in the above experiments show that the GPBLs provide an increase in hit performance across a majority of the networks. In this experiment, we analyze the consistency of the GPBL’s performance across time windows.

We use the same 20-day DShield dataset, divided into 10 time windows. Again, we generate GPBL and WOL from data in time window i and test the lists on data in window $(i + 1)$. For each network, we compare the number of hits on GPBL to that on WOL. If GPBL has a higher or equal hit number compared to WOL, we say that the GPBL performs well for the network in that time window. Otherwise, we say that the GPBL performs worse. We define the *performance index* of a network’s GPBL as the number of windows in which GPBL performs well minus the number of windows in which it performs worse. A performance index is obtained for each network. If GPBL consistently performs better than WOL for a network, its performance index should be close to 10. If it consistently performs worse, the performance index will be close to -10. And if the performance flip-flops, the index value will be close to zero. (Note that the performance index is more sensitive than summing up the hit improvement over all the windows for each network. For some network, GPBL may

perform well in some time windows and not so well in other time windows. The sum may still show a positive improvement but the performance index will give a value much less than 10.)

Figure 3 plots the cumulative percentage of the performance index values. We see that for almost 50% of the networks, GPBL’s performance is extremely consistent. They all have a performance index value of 10. About 80% of the networks have a performance index of 6 and above, indicating that in the 10 time windows, GPBL performs worse only in 2 windows or less. Only with very few networks do we see the performance switch back and forth.

The consistency investigation sheds some light on the reason why there is a small percentage of networks for which the GPBLs (sometimes) perform worse than the other list. We observe from Figure 3 that all the consistent GPBLs have a performance index of 10. There is no GPBL that shows a performance index of -10. Hence, no network sees GPBL performing consistently worse. In fact the networks that have large positive improvement in Figure 2 consistently show such improvement in all the time windows. The networks that have negative improvement in Figure 2 show small positive improvement in some other times windows and vice versa. They are the small proportion that give the

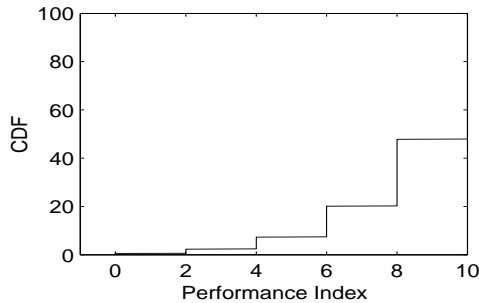


Figure 3: Cumulative Distribution of Performance Index

low performance index values.

The fact that there is no network for which GPBL performs consistently worse and that the minimal performance index is around zero indicates that there is a group of networks, for which the future attacker population can be quite random. Therefore, prediction for these networks is no better than guess: sometimes good and sometimes bad. However, as indicated by Figure 3, this group of networks is of a relatively small number. GPBL is applicable to most of the networks.

In practice, this can be very useful because the consistency of a GPBL’s performance can be used to indicate whether the attacker population of a network has the statistical pattern suitable to apply our relevance-based prioritization. In particular, an operational system can monitor the performance of the GPBL and determine whether a network should adopt it.

5 Conclusion and Future Work

Large-scale security information sharing systems have the potential to enable a network of proactive cyber defense: be prepared for attacks that have not been seen by the network but have been reported by some other networks. An important part of such a proactive defense system is to identify the unforeseen attackers that are most likely to target the network. We proposed a probabilist framework for this purpose that employs Gaussian process regression to rank the attackers according to their attack preference. Our framework also uses an EM process to learn the Gaussian process from the collections of attack reports. Experiments show that our framework is significantly more effective in identifying the possible attackers than the currently deployed method.

Potential future work may be to further improve the modeling of attacker relevances. In our framework, the relevance values are modeled by Gaussian distributions. In reality, the values are taken from a finite set or a bounded range. The Gaussian model is a rough approx-

imation to the true measure of the relevances. Although this approximation can be effective as demonstrated by our experiments, we suspect that better results can be obtained by more accurate modeling.

References

- [1] DSIELD. <http://www.dshield.org>.
- [2] Lawrence Baldwin. my-netwatchman home page. www.MyNetWatchman.com, May 2007.
- [3] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, 1998.
- [4] Zesheng Chen and Chuanyi Ji. Optimal worm-scanning method using vulnerable-host distributions. *International Journal of Security and Networks (IJSN) Special Issue on Computer & Network Security*, 2(1), 2007.
- [5] Symantec Corporation. Deepsight threat management system home page. tms.symantec.com, May 2007.
- [6] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. CRC Press, 2003.
- [7] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating Against Common Enemies. In *Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference*, October 2005.
- [8] John C. Platt, Christopher J. C. Burges, S. Swenson, C. Weare, and A. Zheng. Learning a Gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1425–1432, 2001.
- [9] Carl Edward Rasmussen. The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems (NIPS)*, pages 554–560, 2000.
- [10] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- [11] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *the 10th International World Wide Web Conference*, pages 285–295, 2001.
- [12] J. L. Schafer. *Analysis of Incomplete Multivariate Data*. CRC Press, 1997.
- [13] Anton Schwaighofer, Volker Tresp, and Kai Yu. Learning Gaussian process kernels via hierarchical Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1209–1216, 2004.
- [14] Christopher K. I. Williams and David Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.
- [15] Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In

Advances in Neural Information Processing Systems
(*NIPS*), pages 514–520, 1995.