SRI International

April 12, 2007

Highly Predictive Blacklists

Jian Zhang and Phillip Porras
first.lastname@sri.com
SRI International

Johannes Ullrich
SANS Institute

This material is based upon work supported through the U.S. Army Research Office under the Cyber-TA Research Grant No. W911NF-06-1-0316. The views expressed in this document are those of the author(s) and do not necessarily represent the official position of the sponsor.

Abstract

We propose a radically different approach to source address blacklist formulation, which we call *highly predictive blacklisting.* We present a probabilistic attacker ranking algorithm for blacklist formulation for use in centralized collaborative log sharing infrastructures, such as the DShield.org security log repository. Our objective is not simply to identify a global list of prolific attackers, which is among the most widely used strategies today. Rather, we construct a custom blacklist per contributor that reflects the most probable set of attackers that will attack the target contributor over a prediction window that may last several days into the future. Our attacker rank equation, inspired by hyperlink document link analysis, prioritizes candidate blacklist entries based on the how frequently they are observed by other peer contributors who share significant attacker overlap with the target blacklist user. Through an examination of real DShield datasets we conduct a comparative assessment of the highly predictive blacklist strategy versus three competing blacklist formulation methods. Our results show that highly predictive blacklist entries consistently yield much higher attacker hit rates for the vast majority of contributors in the repositories. In addition, the hit rate quality of these blacklists can last multiple days into the future. We discuss a practical implementation of our highly predictive blacklist algorithm, which we have posted to the DShield.org website for use by DShield log contributors.

Contents

Chapter 1

Introduction

For nearly as long as we have been detecting malicious activity in networks, we have been compiling and sharing *blacklists* to identify and filter the most prolific perpetrators. Source blacklists are a fundamental notion in collaborative network protection. Many blacklists focus on a variety of illicit activity. Network and email address blacklists have been around since even the earliest stages of the Internet [5, 16]. However, as the population size and personal integrity of Internet users have continued to grow in inverse directions, so too has grown the popularity and diversity of blacklisting as a strategy for self-protection. Recent examples include source blacklists to help networks detect and block the most prolific port scanners and attack sources, SPAM producers, and phishing sites, to name a few [6, 7, 15].

Today, sites such as DShield.org not only compile *global worst offender lists* (GWOLs) of the most prolific attack sources, they regularly post firewall parsable filters of these lists to help the Internet community fight back [15]. DShield represents a centralized approach to blacklist formulation, with more than 1000 contributors providing a daily perspective of the malicious background radiation that plagues the Internet [14, 18]. The published GWOL captures a snapshot of those class C subnets whose addresses have been logged by the greatest number of contributors. Another common practice is for a local network to create its own *local worst offender list* (LWOL) of those sites that have attacked it the most. LWOLs have the property of capturing repeat offenders that are indeed more likely to return to the local site in the future. However, LWOLs are by definition completely *reactive* to new encounters with previously unseen attackers. On the other hand, while the GWOL strategy has the potential to inform a local network of highly prolific attackers, it also has the potential to provide a subscriber with a list of addresses that will simply never be encountered.

We propose a radically different approach to blacklist formulation in the context of large-scale log sharing repositories, such as DShield. Our objective is to construct a customized blacklist per repository contributor that reflects the most probable set of addresses that may attack the contributor over a *prediction window* that may last several days. We refer to our algorithm as the *highly predictive blacklist* (HPB) strategy. Under the HPB strategy, for every contributor, we enumerate all sources of reported attackers and assign each of them a ranking score relative to its probability to attack the contributor in the future. The ranking score is based on observation of the particular attacker's past activities, as well as the collective attack patterns exhibited by all other attackers in the alert repository. This is another key difference between our HPB algorithm and the other blacklist strategies. In the compilation of GWOL and LWOL or their like, each blacklist entry is selected solely based on its own attack history. In contrast, our HPB strategy takes a collective approach. HPB attacker selection is influenced by both an attacker's own attack patterns and the full set of all attack patterns found within the dataset. In particular, we consider the correlations among the contributors introduced by the collection of attackers, i.e., the amount of attacker overlap between each pair of contributors. Our ranking score cares not only how many contributors have reported the attacker but also who gave the reports. It favors attackers reported by many contributors that are also correlated (have many common attackers) with the contributor under consideration. The choice of contributor correlation for use in our collective attacker ranking algorithm is inspired by recent work of Katti et al. [9].

We tested our HPB strategy using more than 600 million DShield log entries produced by more than 1000 independent contributors from April to June 2006. We generate HPBs using DShield datasets and contrast perfornamce to that of corresponding GWOLs and LWOLs. Our results show that for most contributors (more than 80%), HPBs entries exhibit much higher hit counts over a multiday prediciton window than both GWOL and LWOL. In the best case, one HPB of length 200 successfully predicted 195 attacks in comparison to only two addresses from the GWOL. We further compared HPB with a *hybrid* blacklist consisting of half GWOL and half LWOL. Our experiment shows a similar result: the HPB algorithm exhibits a higher hit count for most of the contributors. Our experiments also show that HPB's performance is consistent over time, and these advantages remain stable across various list lengths and predict windows.

Our contributions are the following. We present an alternative approach to blacklist formulation, which is substantially different from that of the well-established worst offender list strategy. We view the amount of hits on the blacklist as an additional metric to quantify the degree to which a given blacklist is exercised in protecting a site from unwanted connections. We propose a collective approach for ranking attackers that considers not only individual attackers' past activity but also the attack patterns shown by the collection of attackers. To compute these rankings, we present a novel system based on Google's PageRank [3]. We also propose the use of various attacker severity metrics to perform final blacklist entry selection and present a practical implementation of our highly predictive blacklist algorithm, which we have now posted to the DShield website for use by all DShield log contributors.

This technical report is organized as follows. We summarize common network address blacklisting practices in Section 2. In Section 3 we present the general intuition of our approach, and discuss how it differs from traditional methods of blacklisting. Section 4 presents the details of our predictive score algorithm, and Section 5 presents our experimental results on extensive assessments of HPBs versus other competing blacklist generation strategies. We suggest methods for generating a final blacklist from HPBs by down-selecting using various attacker severity metrics in Section 6. Section 7 describes the DShield implementation of our HPB system, and Section 8 summarizes our key conlusions.

Chapter 2

The Basics of Source Address Blacklisting

Today, source address blacklisting a set of known-bad IP addresses is a common technique applied to defend a given network from the worst of the worst. Different approaches are used to compile these blacklists. Most often, such blacklists are either based on local data (e.g., data from an Intrusion Detection System (IDS)), or they use data collected from larger sets of sensors. More recently, researchers have proposed peer-based attack sharing strategies to create fast reaction blacklists that might help in combating malware epidemics [13, 17].

Blacklists derived from local data are inherently reactive, in that they cannot include sources that the local site has not yet encountered. Local blacklisting relies on the assumption that it is possible to identify reconnaissance activity, and to then block the sources of such activity before they are able to launch actual attacks. This may work in some cases, but in many cases the attack is launched from a different source than the source of the probes, or no probing takes place at all (an attack is launched blindly, such as in the case of MySQL Slammer [4]). Even though these sources that launch the attack may have been seen by other networks, they are new to the particular local network. A reactive local blacklist, therefore, would not be able to defend against them.

One may alleviate this problem by using global blacklists derived from data supplied by a large set of contributors across the Internet. A common approach to global blacklist formulation is to simply populate the blacklist with the most prolific offenders, i.e., sources that have attacked the greatest number of targets. However, this strategy may miss certain significant attackers that prefer to attack the same set of networks that have been proven to provide vulnerable hosts in the past. These attackers are not necessarily very prolific, as they focus on the known vulnerable networks. The particular attacks may change, but attackers will probe the same networks as new vulnerabilities are incorporated into the attacker's tool set. This behavior is in particular common in bots. Bots typically include a range of attack tools that are controlled from a central location, like an IRC server or web server. With any new tool made available to the bot, the same networks tend to be rescanned using this new vulnerability.

More importantly, significant pragmatic limitations are inherent in all source address blacklisting strategies. Packet filtering devices are able to hold only a limited number of rules. Depending on the device, this filter set size may range from a few dozen to a few hundred. Thus, the cost of blacklist filters is not just measured in compute cycles and the potential for accidental blocking of good traffic, but in the *opportunity cost* in exhausting the firewall filter set with entries that are unlikely to be exercised. The central goal of our highly predictive blacklist system is to provide a filter set that will be exercised with much higher probability than other contemporary blacklists while amending the aforementioned problems.

Chapter 3

Toward a Predictive Blacklisting Strategy

With today's wide deployment of Internet sensors and collaborative log repositories, there is greater potential than ever to develop blacklists that truly provide a global and adaptive perspective of emerging attack sources. This richness in data provides not only the foundation to produce an effective blacklist, but to open a new way to think about how blacklists might be formulated.

The problem of formulating blacklists has certain similarity to the *recommendation system* problem. Recommendation systems have been intensively studied and widely adopted by many commercial websites [1, 2, 11, 12]. A website compiles a collection of its users' activities such as their purchase or browsing behaviors. The recommendation system then processes this data set and predicts each individual user's future activity. The predictions are based on both the individual user's history and the preferences induced by the activities of the *whole collection of users*. The prediction is presented to the user as a recommendation or is used by the website to prepare resources for the anticipated activity. A typical recommendation system can be found on Amazon.com, where it provides suggestions, such as "users who have purchased book X also find book Y highly appealing."

Both blacklist-formulation systems and recommendation systems depend on history data to make predictions. However, the traditional way of generating a blacklist does not utilize the full information in the history data. For example, GWOL assumes the more prolific the attacker, the more likely it will attack in the future. This prediction uses only information about the particular attacker, i.e., its target diversity. It does not consider the patterns introduced by the other (possibly similar) attackers.

Suppose we have a collection of history data on the attacks made by a large set of sources. Consider two attackers A_1 and A_2 in this collection that both attacked 20 targets. Our task is to assist network C in deciding which of the attackers, A_1 or A_2, C should choose to block with higher priority. With the traditional GWOL approach, because the two sources attacked the same number of targets, we will treat them equally. Such decision making does not fully consider the information contained in the data collection. If we utilize the full information, we may be able to make a more intelligent decision. For example, let $targets(A_1)$ denote the set of targets attacked by A_1 and $targets(A_2)$ the set of targets attacked by A_2. Let us assume $targets(A_1)$ and $targets(A_2)$ have no intersection. Suppose after examining the collection of attack data, we find that many sources that attacked some networks in $targets(A_1)$ also came to C but very few that attacked networks in $targets(A_2)$ hit C. In other words, C shares many common (overlap) attackers with networks in $targets(A_1)$ but not those in $targets(A_2)$. A recommendation system that utilized this information would then suggest that C put higher priority on A_1 than A_2.

Therefore, similarity among the targets determined by their common attackers can be as useful in making blacklists as the individual attacker's own history. We are interested in the relationship between target networks X and Y in the form of "many attackers who target X often also target Y". Note that this relationship is transitive. Let $X \sim Y$ denote the attacker overlap relationship between X and Y. If $X \sim Y$ and $Y \sim Z$ holds, then when we formulate a blacklist for Z, we assert that an attacker who comes to X should be given some favor comparing to an attacker that has no connection with Z.

We use a graph to represent the above overlap relation (i.e., the "$\sim$" relationship) among networks. An attacker's activity can be modeled as a random walk through our overlap relations graph. Using the attacker's history and such a random walk, we explore a new scoring system for selecting entries for customized contributor blacklists. For each attacker and each target network, the system produces a score that is proportional to the probability estimation of the attacker attacking the network. The blacklist for a network then consists of the attackers that have the top scores for that network.

Chapter 4

A Common-Source-Based Address Scoring Algorithm

In describing our Highly Predictive Blacklisting algorithm for formulating source address blacklists, we start with a basic algorithm. Although it lacks consideration for certain details, the basic algorithm carries the essence of our HPB strategy. We then extend the basic algorithm to a full algorithm by adding the considerations for the omitted details. Finally we briefly discuss the complexity of our algorithm.

4.1 Basic Scoring Algorithm

The heart of our blacklisting system is the scoring algorithm. In formulating a blacklist for a contributor, it assigns scores to each attacker that are proportional to our estimation that the attacker will attack the contributor. Here, attackers represent class C addresses. We will use the terms “attacker” and (attack) “source” in an interchangeable way. The contributors supply data to the log repository. They are the victims of the attacks reported in the repository. We will also use the terms “contributor” and “victim” interchangeably. We use s and v to denote the source and the victim of an attack, respectively. Our algorithm generates a customized blacklist per contributor (victim). We use $R^s(v)$ to denote the score for attacker s with respect to a victim (blacklist consumer) v. $R^s(v)$ is a sum of two ranking scores: $RP^s(v)$ and $RI^s(v)$. $RP^s(v)$ is an estimation of s's attack probability given 1) s's past activities involving other victims, and 2) information on similarities among victims compiled over a collection of attack data. $RI^s(v)$ is the estimation of s's attack probability based on s's previous activity involving only v itself. We first describe how we compute $RP^s(v)$ and then we continue to the calculation of $RI^s(v)$.

It has been shown that there are correlations among attack victims [9]. In a collection of Internet-scale attack activities, we often can see that not all the attack victims are equal. For some pairs of victims, attacks that targeted one victim tend to also attack the other. For other pairs, there is no such relationship. We refer to the pair of victims in the former case as *correlated victims*. Among the correlated pairs of victims, the strength of correlation may differ from pair to pair. We illustrate this using a running example in Table 4.1. We will use this example throughout the description of our system.

	v_1	v_2	v_3	v_4	v_5
s_1	X	X			
s_2		X			
s_3	X	X		X	
s_4		X	X		
s_5			X		
s_6			X	X	
s_7					X

Table 4.1: Attack Table

In this example, we have five victims (v_1 to v_5) and seven attack sources (s_1 to s_7). We will refer to Table 4.1 as an *attack table*. The rows of the table represent the sources, and the columns represent the victims. An "X" in an attack table cell indicates that the corresponding source has reportedly attacked the corresponding victim. Victims may be attacked by multiple sources. For a pair of victims v_i and v_j, the set of sources that attack v_i may overlap with the set of sources that attack v_j. For now, we will treat the amount of such overlap as the strength of correlation between victims v_i and v_j. (This is simply for illustration. Our algorithm uses a value modified from such overlap to indicate the correlation strength, which we will discuss later.) For example, v_1 and v_2 may be said to share strong correlation (two overlaps) relative to v_1 and v_3 and v_5 who share zero overlaps.

To estimate probabilities of which an attacker may attack a victim in the future, it is natural to consider the correlations among the victims. In our running example, although s_2 and s_7 have attacked the same number of victims, from the viewpoint of v_1, we can make the estimation that s_2 is more likely to attack than s_7, because s_2 has attacked v_2, which is closely correlated with v_1. On the other hand, s_7 attacked v_5, which has no correlation with v_1. For better estimation of the attack probabilities, one may use information beyond "direct" correlation. For example, consider the source s_5 and compare it to s_7. Both sources attacked only one victim. None of their victims have a correlation with v_1. However, for v_1, s_5 and s_7 are not equal. One may view the correlation as a *transitive* relationship. Notice that v_2 correlates with v_1, and v_3 correlates with v_2. A path $v_3 \sim v_2 \sim v_1$ connects s_5 with v_1. Using this form of reasoning, one can estimate that s_5 is more likely to attack v_1 than s_7.

We model the correlation relationship between victims by a *correlation graph*. The correlation graph is a weighted directed graph $G = (V, E)$. The nodes in the graph are the victims, i.e. $V = \{v_1, v_2, \ldots\}$. There is an edge from node v_i to node v_j if v_i is correlated with v_j. The weight on the edge is proportional to the strength of this correlation. Figure 4.1 shows the correlation graph for our running example. (We will discuss how we obtain the edge weight later.) We can use the correlation graph to make estimations on attack probabilities. Suppose we have an estimation on source s's probability of attacking victim v_i. Following the outgoing edges of v_i, a fraction of this probability can be distributed to the neighbors of v_i in the graph. Each neighbor receives a share of this probability that is proportional to its strength of correlation with v_i (i.e., proportional to the weight of the edge from v_i to that neighbor.) Suppose v_j is one of these neighbors in the correlation graph. A fraction of the probability received by v_j is then further distributed, in the similar fashion, to its neighbors. The propagation of probability continues until the estimations for each victim reach a stable state.

Such a probability-propagation process can be simulated by a random walk on the correlation graph: a source walks on the correlation graph, going from one node to another by

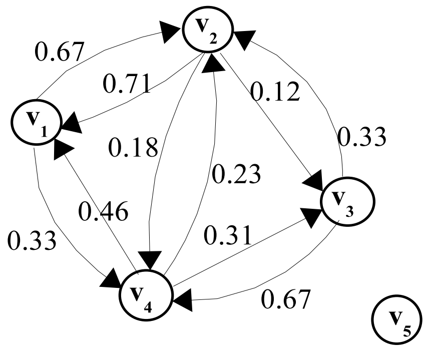

Figure 4.1: Correlation Graph of Our Running Example

following the edges in the graph. When at a node, the source chooses an outgoing edge to follow with a probability proportional to the weight on that edge (i.e., proportional to the strength of correlation between the two victims connected by the edge). Similar types of random walk have been applied in other probability propagation and estimation systems, for example, Google's PageRank [3]. The PageRank system ranks webpages using a score that is related to the probablity that a particular webpage may be visited. Based on the PageRank system, we develop an algorithm to estimate attack probabilities given a victim v and an observed set of attackers. The estimates then determine the value of $RP^s(v)$ for each attacker s in the set.

We first give a brief description of the PageRank system. Let $PR(i)$ be the ranking score for webpage i. Let $LT(i)$ be the set of webpages that contain links to page i. Denote by $NL(i)$ the number of outgoing links on page i. The PageRank score is determined by

$$PR(i) = (1 - \alpha) + \alpha \sum_{j \in LT(i)} \frac{1}{NL(j)} PR(j)$$

where $\alpha < 1$ is a damping factor that decides how far the probability from a node can propagate.

We modify this ranking function to estimate the attack probabilities. Let $P^s(v)$ be an estimate proportional to the total probability that s attacks v. Let W_{ij} be the correlation strength from victim v_j to v_i. We define

$$P^s(v_i) = B^s(v_i) + \alpha \sum W_{ij} \cdot P^s(v_j) \tag{4.1}$$

where $B^s(v_i)$ is an initial estimation based on whether s attacks v_i in the attack table. Given a fixed source s, we will have an equation in the same form as Eq. 4.1 for every victim v_i. The sets of $P^s(v_i)$ and $B^s(v_i)$ form vectors, which we denote by $\mathbf{P}^s$ and $\mathbf{B}^s$ respectively. (We use boldface for vectors and matrices and normal font for scalar values.) The set of equations for different v_i can then be expressed in a matrix form:

$$\mathbf{P}^s = \mathbf{B}^s + \alpha \mathbf{W} \mathbf{P}^s \tag{4.2}$$

Given the vector $\mathbf{B}^s$ and the matrix $\mathbf{W}$, the system of linear equations in Eq. 4.2 can be solved to obtain the values $P^s(v_i)$. The source s's first score for victim v_i, $RP^s(v_i)$, is then defined to be $RP^s(v_i) = P^s(v_i) - B^s(v_i)$. (We need $RP^s(v_i)$ to be a score that predicts future attacks. $B^s(v_i)$ represents attacks in the past. We therefore remove $B^s(v_i)$ from $P^s(v_i)$ to obtain the ranking score $RP^s(v_i)$.) For each source s, we will have a system of linear equations in the same form as Eq. 4.2. Solutions to each system of equations give us the corresponding source's ranking scores for each victim v_i.

The vector $\mathbf{B}^s$ and the matrix $\mathbf{W}$ are two important components in Eq. 4.2. We now describe how to construct them from the attack tables. Since $B^s(v_i)$ reflects whether s attacks v_i in the attack table, we can simply define it to be the following:

$$B^s(v_i) = \begin{cases} \text{1, if } s \text{ attacked } v_i; \\ \text{0, otherwise}. \end{cases} \tag{4.3}$$

The entry $W_{i,j}$ in matrix $\mathbf{W}$ reflects the strength of correlation between victims i and j. For simplicity, in our previous illustration, we used the amount of overlap between sources that attacked v_i and that attacked v_j as the measure of the correlation between the two victims. We call this the *simple correlation*. Figure 4.2 shows a simple correlation matrix for our running example. (We have five victims in the running example. Therefore, the correlation matrix is a 5×5 matrix. Entry (j, i) is the strength of correlation from victim i to j.)

$$\begin{pmatrix} 0 & 2 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4.2: Simple Correlation Matrix of Our Running Example

The actual measure of correlation needs to consider more factors. For example, consider the following two cases. In Case 1, victim v_i sees attacks from 500 sources and v_j sees 10 sources. Five sources attack both v_i and v_j. In Case 2, there are also five common sources. However, v_i sees 100 sources while v_j sees 10. Although the number of overlapping sources is the same in both cases, the strength of correlation is different. Because in the first case, v_i sees more sources, it is expected that there should be more overlap. Therefore, the five common sources in Case 1 should be equivalent to just one common source in Case 2. Similarly, if v_i sees 100 sources but v_j sees five, and there are still five common sources. These five common sources should be counted as 10 comparing to those in Case 2.

We use a statistical measure to standardize the amount of overlaps. Assume that there are a total of N sources. Let n_i be the number of sources seen by v_i, n_j the number seen by v_j, and n_{ij} the number of common sources. The probability for v_j to see a source is n_j out of N. If v_i and v_j are statistically independent, among the n_i sources seen by v_i, we expect to see $n_i \cdot \frac{n_j}{N}$ common sources. We define the standardization factor (std) to be $std(i,j) = \frac{n_{ij}}{n_i \cdot \frac{n_j}{N}}$, which is essentially the ratio of the actual overlap over the expected overlap. Our standardized correlation is then obtained by scaling the simple correlation using the standardization factor, i.e. standardized correlation between v_i and v_j is $std(i,j) \cdot n_{ij}$. Figure 4.3 shows the standardized correlation matrix for our running example.

The matrix $\mathbf{W}$ in Eq. 4.2 is obtained by normalizing the columns of the standardized correlation matrix. In Figure 4.1, we show the correlation graph for our running example. The weight on the edge from node i to node j is the value of the entry (j,i) in the matrix $\mathbf{W}$.

$$\begin{pmatrix} 0 & 3.5 & 0 & 1.75 & 0 \\ 3.5 & 0 & 0.583 & 0.875 & 0 \\ 0 & 0.583 & 0 & 1.167 & 0 \\ 1.75 & 0.875 & 1.167 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4.3: Standardized Correlation Matrix of Our Running Example

So far we have described how to compute the score $RP^s(v)$. Recall that a source s's ranking score for a victim v is the sum of two scores $RP^s(v)$ and $RI^s(v)$. We now continue to the computation of $RI^s(v)$. $RI^s(v_i)$ reflects s's attack probability based on its previous activities that target v_i. We calculate $RI^s(v_i)$ using a formula similar to Eq. 4.1. In particular, we have

$$RI^s(v_i) = \hat{B}^s(v_i) + \alpha \hat{W}_{ii} RI^s(v_i) \tag{4.4}$$

The coefficients $\hat{W}_{ii}$ reflects the probability that a previous attacker of v_i will come back. We calculate this value in the following way. Let T_1 and T_2 be two consecutive time windows. We estimate $\hat{W}_{ii}$ by the fraction of sources that attacked v_i in T_1 that also attack in T_2. Similar to $B^s(v_i)$, $\hat{B}^s(v_i)$ reflects our observation on s's previous attack on v_i. Therefore, it can also be determined using Eq. 4.3.

With Eq. 4.1 and Eq. 4.4, we can compute the two scores $RP^s(v)$ and $RI^s(v)$, respectively. Our final ranking score for source s with respect to victim v is then $R^s(v) = RP^s(v) + RI^s(v)$. We also described how the coefficients in the equations can be determined. Putting these together, we have a basic system to rank attackers. Our final system is an extension of this basic system that incorporates more detailed considerations on the sources' past activities.

4.2 From Basic to Full Algorithm

The extensions discussed here were derived from our experiences in iteratively applying our ranking system to the DShield repository. In doing so we discovered various patterns in contributor log production that we now reflect into our scheme.

The first extension is a modification of the vector $B^s(v_i)$ in Eq. 4.1, which is defined by Eq. 4.3. This definition considers whether a source s attacks victim v but does not reflect when the attack happens. For example, when we predict a source's attack, we should take into consideration the case in which the source's last attack occurred yesterday from the case where the attacker was last observed 10 days ago. In the former case, the attacker may be still active while in the latter case, it may not. To reflect this, we modify $B^s(v_i)$ by multiplying it with a decay factor $d^s(v_i) = 2^{-k}$, where k is the delta between the time v_i last appeared and the start of the time window for which we plan to make our prediction.

The second extension modifies $\hat{B}^s(v_i)$. We consider the number of times that source s has attacked v_i as well as the time when these attacks happened. Suppose s made t attacks to v_i. The i-th attack happened at time $-k_i$. Note that we consider the beginning of the prediction time window as time zero. Since we are counting backward into the history, we put a minus sign in front of k_i. The new value of $\hat{B}^s(v_i)$ is then $\hat{B}^s(v_i) = \log(\sum_1^t 2^{-k_i})$. In the case where $t = 0$ (i.e., s did not attack v in the past) we set $\hat{B}^s(v_i)$ to zero. Recall that $\hat{B}^s(v_i)$ is an initial estimation based on s's previous attack on v_i. Here we assert that a source that has attacked 500 times in the past is not 5 times more probable to come back (to the same victim) than a source that has attacked 100 times. Therefore, in the new value for $\hat{B}^s(v_i)$, we use a logarithmic form of the decayed sum instead of the sum itself. This reflects the observation that two very frequent attackers share a quite close probability of coming back.

Notice that we make different modifications to $B^s(v_i)$ and $\hat{B}^s(v_i)$. We observed that an attacker that has attacked victim v_i many times in the past is not necessarily more likely to attack another victim v_j. Therefore, we considered the number of previous attacks on v_i in formulating $\hat{B}^s(v_i)$, but not for $B^s(v_i)$. It also shows that the amount of attack probability derived from the number of past attacks does not propagate. This is the main reason that our final ranking score is a sum of two scores: one that is related to the attack probability propagated from other victims and the other focusing on the local attack probability.

Finally, we extend the attack table to include time information. The entries in the basic attack table are boolean values, indicating whether source s attacked victim v. We use sets as entries in the extended attack table, i.e., the set of attacks from s to v that happened at different times. Let T_{ij} and T_{ik} be two entries in the extended attack table T. When we measure the attack overlap between v_j and v_k, we will include the intersection of the two sets T_{ij} and T_{ik}. For example, with the basic attack table T^B, if $T^B_{ij} = T^B_{ik} = 1$, we add 1 to the attack overlap. With the extended attack table, we add $|T_{ij} \cap T_{ik}|$ to the overlap. One may construct the extended attack table using different granularity. For example, the set T_{ij} may include every individual attack event. Alternatively, one may discretize the time by hours and aggregate attacks (from s_i to v_j) that happened within an hour into a single attack in T_{ij}. Coarse granularity may be preferred because the attack table is used only to obtain the correlation matrix, which then affects the value of $RP^s(v)$. A granularity at hourly level is equivalent to asserting that if s attacked victims i and j within one hour, the two attacks are correlated. Otherwise, they are not. Because of this, one should not distinguish individual attacks, but rather should use coarse granularity at hourly or even daily intervals.

Applying the above extensions to our basic system, we obtain our final source ranking system. We summarize the whole system in Algorithm 1.

4.3 Complexity Analysis

Because our HPB algorithm is constructed from collections of alert data, we need to go through the data collection at least once. Hence, there is always the amount of complexity that is linear to the size of the alert data. That is, let $N(data)$ be the number of alerts in the data collection; we have a minimum complexity of $O(N(data))$. Our discussion will focus on other complexity incurred by the algorithm besides this linear-time requirement.

We denote by $N(s)$ and $N(v)$ the number of sources and victims in the data collection, respectively. Recall that victims are the contributors that supply data to the alert collection; $N(v)$ is actually the number of such contributors. We can expect $N(v)$ to be in the order of thousands. Unfortunately, $N(s)$ is much larger and can be in the tens of millions. We obtain the coefficients in Eq. 4.2 and Eq. 4.4 by going through the collection of alert data and doing simple accounting. For example, the weight matrix $\mathbf{W}$ used in Eq. 4.2 requires the most work to construct. To obtain this matrix, we record every overlapped attack while going through the alert data and then perform standardization and normalization. The latter steps require us to go through the whole matrix, which results in $O(N(v)^2)$ complexity.

Besides going through the data, the most time consuming step in the ranking process is the computation that solves the linear equations in Eq. 4.2. At first glance, because for each source s, we have a linear system determined by Eq. 4.2, it seems that we need to solve $N(s)$ linear systems. This can be expensive as $N(s)$ is very large. A further investigation shows that the solution to Eq. 4.2 is $(\mathbf{I} - \alpha\mathbf{W})^{-1} \cdot \mathbf{B}^s$ where $\mathbf{I}$ is the identity matrix. While $\mathbf{B}^s$ is different per source s, the term $(\mathbf{I} - \alpha\mathbf{W})^{-1}$ is the same for all s. Therefore, we need to compute it only once, which requires $O(N(v)^3)$ time by brute force or $O(N(v)^{2.376})$ using more sophisticated methods [**?**]. Because $\mathbf{B}^s$ is sparse, once we have $(\mathbf{I} - \alpha\mathbf{W})^{-1}$, the total time to obtain the ranking scores for all the sources and all the victims is $O(N(v) \cdot N(data))$. Assuming $N(v)^2$ is much smaller than $N(data)$, the total complexity to generate HPBs is $O(N(v) \cdot N(data))$. Note that this captures the complexity to generate HPBs for all the victims. For a data set that contains a billion records contributed by a thousand sensors, generating a thousand HPBs only requires several trillion operations (additions and multiplications). This can be easily handled by modern computers. In fact, in our experiments, with $N(data)$ in the tens of millions and $N(v)$ on the order of one thousand, it takes less than 30 minutes to generate all the HPBs on an Intel Xeon 3.6 GHz machine.

Algorithm 1: Generate HPB for victim v

```
HPB_GEN(BL_Length, $v$)
(1)   foreach source s
(2)       $RP^s \leftarrow$ RANK_SCORE1(v, s)
(3)       $RI^s \leftarrow$ RANK_SCORE2(v, s)
(4)       $R^s \leftarrow RP^s + RI^s$
(5)   Sort $s$ in descending order according to their rank score $R^s$.
(6)   return BL_Length of $s$ with top $R^s$

RANK_SCORE1(v,s)
(1)   Obtain attack overlap from the attack table;
(2)   Generate the standardized correlation matrix;
(3)   Generate the final correlation matrix $\mathbf{W}$;
(4)   Construct $\mathbf{B}^s$;
(5)   Solve linear system in Eq. 4.2;
(6)   return $RP^s(v)$

RANK_SCORE2(v,s)
(1)   Estimate $\hat{W}_{ii}$;
(2)   Calculate $\hat{B}^s$;
(3)   Solve Eq. 4.4
(4)   return $RI^s(v)$
```

Chapter 5

Comparative Analysis of Blacklisting Strategies

To test our proposed HPB algorithm, we have conducted a battery of experiments using the DShield.org firewall and IDS log repository. We examined a collection of approximately 600 million DShield records, collected from more than 1700 contributors between April and June 2006. The prerequisite for our HPB algorithm to produce meaningful predictive results is to have it first compute correlation relationship among the contributors. In this sense, HPB blacklist production is not applicable to contributors who have submitted very few reports (DShield has contributors who hand-select contributions to the repository, providing very few alerts from which to compute a correlation relationship). We therefore exclude those contributors who we find effectively have no correlation with the wider contributor pool or simply have too few alerts to produce meaningful results. For this analysis, we found that we could compute correlation relationships for 1,088 contributors.

In the experiments, we generate HPB using data for a certain time period and then test the HPB on data from the time window following this period. We call the period used for producing HPB the *training window* and the period for testing the *prediction window*. In practice, the training period represents a snapshot of the most recent history of the repository used to formulate the HPB of a contributor who is then expected to use the HPB for the length of the prediction window. The sizes of these two windows are not necessarily equal.

We will first describe experiments that use 5-day lengths for both the training window and the prediction window.

As a baseline for comparing the performance of the HPB method, we compute its performance relative to the standard DShield-produced Global Worst Offender List (GWOL) [15]. In addition, we compare our HPB performance to that of local worst offender lists (LWOLs), which we compute individually for all contributors in our comparison set. For the purpose of our comparative assessment, we fixed the length of all three competing blacklist strategies to exactly 200 entries. However, after we present our comparative performance results, we will then continue our investigation by analyzing how window size and blacklist length affect HPB's predictive performance.

5.1 Hits Improvement

Data in the prediction window are used to test how many sources that are on a blacklist actually hit the contributor. We call this the *hit number* ($\mathcal{H}$) for that blacklist. For each contributor, a blacklist that produces a high hit number is said to provide stronger predictive power than a blacklist that produces a lower hit number. We compare the hit numbers produced by our HPB algorithm to those of GWOL and LWOL over our 1088 DShield contributors. The hit numbers of the three types of blacklists vary from contributor to contributor. Table 5.1 lists the mean and median of the hit numbers over our 5-day prediction window interval for our three competing blacklist strategies.

	Mean	Median
HPB	59	47
LWOL	49	36
GWOL	31	30

Table 5.1: Summary of Hit Numbers

We now break down the performance results of these three blacklist strategies on the individual contributor. For each contributor v, there is a hit number for HPB, GWOL, and LWOL. Let $\mathcal{H}^{HPB}(v)$, $\mathcal{H}^{GWOL}(v)$, and $\mathcal{H}^{LWOL}(v)$ be the respective hit numbers.

To compare HPB to another blacklist, we define two quantities for each contributor. One measure is the *Improvement Value* (IV), which we define as the HPB hit number minus the hit number on the other list. That is, the IV over GWOL for contributor v is $\mathcal{H}^{HPB}(v) - \mathcal{H}^{GWOL}(v)$, and the IV over LWOL is $\mathcal{H}^{HPB}(v) - \mathcal{H}^{LWOL}(v)$. A positive IV means HPB has produced a better hit number than the other list, while a negative IV means HPB performs worse.

A second comparative measure is the *Hit Number Ratio* (HR), which is simply the ratio of an HPB hit number over the other blacklist hit number. For example, the HR over GWOL for v is $\frac{\mathcal{H}^{HPB}(v)}{\mathcal{H}^{GWOL}(v)}$. If the other list hit number is zero we define HR to be the HPB hit number, and if both hit numbers are zero we set HR to one. An HR of value 1 means the HPB predicts the same amount of hits as the other list. The larger the HR, the better the HPB's prediction performance. In our experiments, we find that IV and HR differ from contributor to contributor. We plot the distribution of these values when comparing HPB to the other blacklists.

In Figure 5.1, we compare HPB to GWOL. The left panel of the figure plots the distribution of the IV values for the 1088 contributors. The x-axis represents IV values and the y-axis represents the number of contributors that produce the corresponding IV. The dash-dotted line indicates $x = 0$ or no improvement, where anything to the left of this line represents worse HPB performance. We see that for most contributors, the IV is positive. HPBs have more hits than GWOL. The largest IV reaches 193. For this contributor, the GWOL has very few hits while almost all sources in the HPB were found to have attacked in the prediction window.

The two panels on the right of Figure 5.1 plot the HR (ratio of HPB's hit number over GWOL's hit number) distribution. The distribution is highly skewed. Therefore, we separate the largest 5% of HR values from the rest of 95% and plot the two sets of values in different ways. The upper right panel shows the distribution of the top 5% of HR values in the histogram. (The dash-dotted line in this panel indicates $x = 1$.) We see that for all the contributors in this panel, HPB achieves an HR at least 6 times the HR of GWOL. The largest ratio is 138. In the lower right panel, we plot the cumulative distribution function (CDF) for the rest of the HRs. (Again, in this panel, the dash-dotted line shows $x = 1$.) For

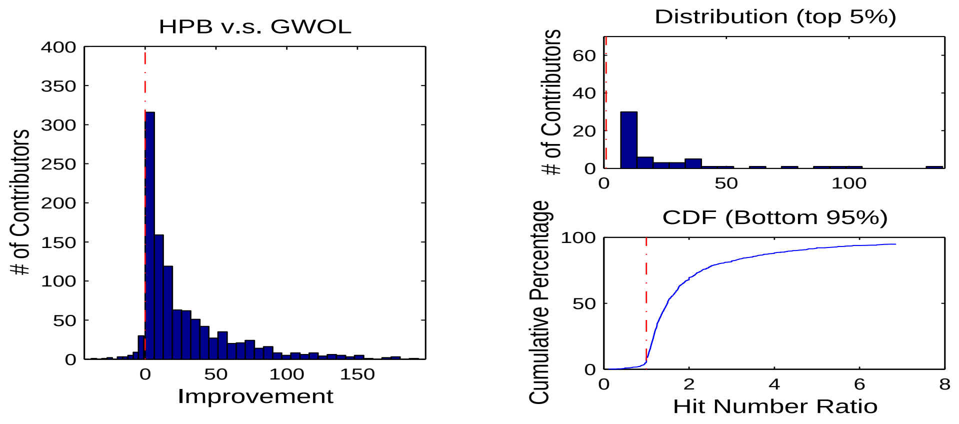

Figure 5.1: Hit Number Comparison of HPB and GWOL

each value x of HR, CDF plots the percentage of HRs that is below x. The x-axis represents HR values, and y-axis represents percentage. A point (x, y) on the CDF curve indicates that there are y percent HRs that have values smaller than x. (Or there are $100 - y$ percent HRs with values larger than x.) From the CDF plot, we see that for only a very small percentage (less than 10) contributors, HPB performs worse than GWOL. For about 30% of the contributors, HPB doubles the hit number. For half the contributors, HPB improved the hit number by more than 50%. For about 70% of the contributors, HPB has 20% more hits than GWOL.

In Figure 5.2 we compare HPB hit numbers to those of LWOL. The data are plotted in the same way as in Figure 5.1. Overall, HPB demonstrates a performance advantage over LWOL. The IV and HR values also exhibit similar distribution. However, comparing Figures 5.2 and 5.1, we see that HPB's advantage over LWOL is not as large as its advantage over GWOL. For example, from the CDF in the lower right panel of Figure 5.2, we see that at the 50% (median) level, the HPB has about 20% more hits than LWOL, less than the 50% it achieved with GWOL.

In preparing this analysis, one apparent observation is that the GWOL and LWOL strategies produce very complementary lists of attackers. That is, by definition GWOL focuses on prolific Internet-wide IP sweepers, while LWOL focuses on sources that have specifically concentrated their attacks (i.e., on the LWOL owner in particular). Because the two

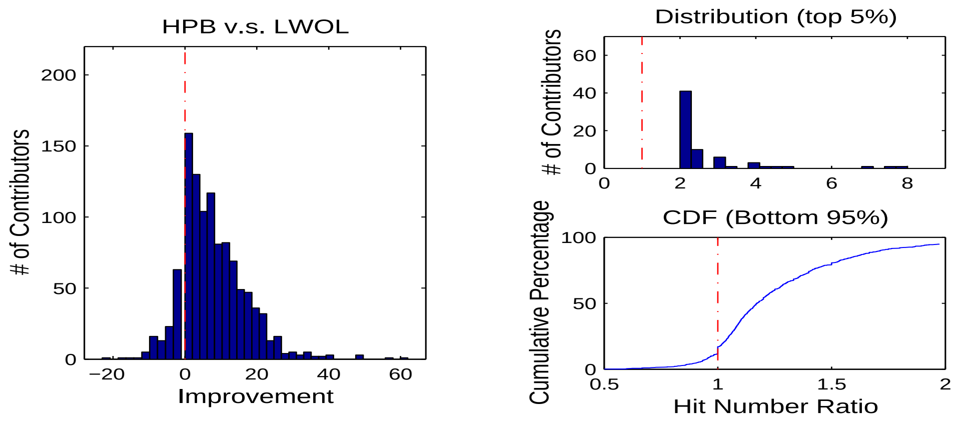

Figure 5.2: Hit Number Comparison of HPB and LWOL

lists have different focuses, one may wonder whether HPB outperforms GWOL and LWOL because HPB considered both types of sources while GWOL and LWOL target only an individual type.

We believe this is part of the reason that HPB has more prediction power. However, in studying this question we find that the HPB algorithm produces much more than a naive combination of GWOL and LWOL. To illustrate this point, we now compare the hit numbers of our HPB method to that of a *hybrid* blacklist. Recall that for the purpose of our performance assessment, we have set all blacklists to a fixed length of 200 entries. We now consider a hybrid blacklist that is composed of the top 100 sources of the GWOL and fill the remainder of entries with the top non-overlapping sources from the LWOL. Figure 5.3 plots the comparison of HPB against this new hybrid list. The figure shows that for most of the contributors, HPB performs better than the hybrid list.

5.2 Prediction of New Attacks

One clear motivating assumption in secure collaborative defense strategies is that participants have the potential to prepare themselves from attacks that they have not yet encountered. We will say that a *new attack* occurs when a contributor produces a DShield log entry from a source that this contributor has never before reported. In this experiment, we

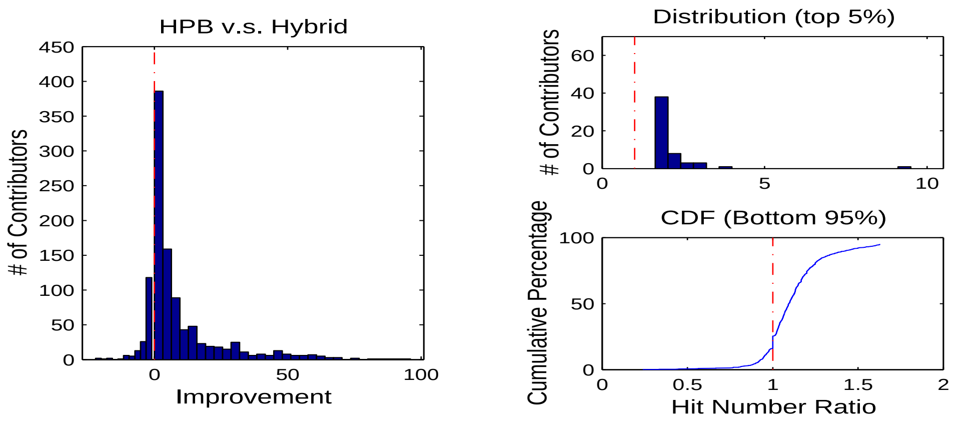

Figure 5.3: Hit Number Comparison of HPB and A Hybrid Blacklist

show that HPB analysis provides contributors a potential to predict more new attacks than GWOL. (LWOL is not considered, since by definition it *only* includes attackers that are actively hitting the LWOL owner.) For each contributor, we construct two new HPB and GWOL lists with equal length of 200 entries, such that no entries have been reported by the contributor during our training window. We call these lists HPB-local (HPB minus local) and GWOL-local (GWOL minus local), respectively. Figure 5.4 compares HPB-local and GWOL-local on their ability to predict on new attack sources for the local contributor. These hit number plots demonstrate that HPB-local provides substantial improvement over the predictive value of GWOL.

5.3 Performance Consistency

The results in the above experiments all show that the HPB algorithm provides an increase in hit number performance across the majority of all contributors. We then ask the following question: is the HPB predictive performance consistent for a given contributor over time? In this experiment, we investigate this performance consistency question.

We use a 45 days DShield data. We divide the 45 days into nine time windows, $T_0, T_1, \ldots, T_8$. We generate blacklists from data in time window T_{i-1} and test them on data in T_i. For each contributor v, we compare HPB with GWOL and obtain eight IV val-

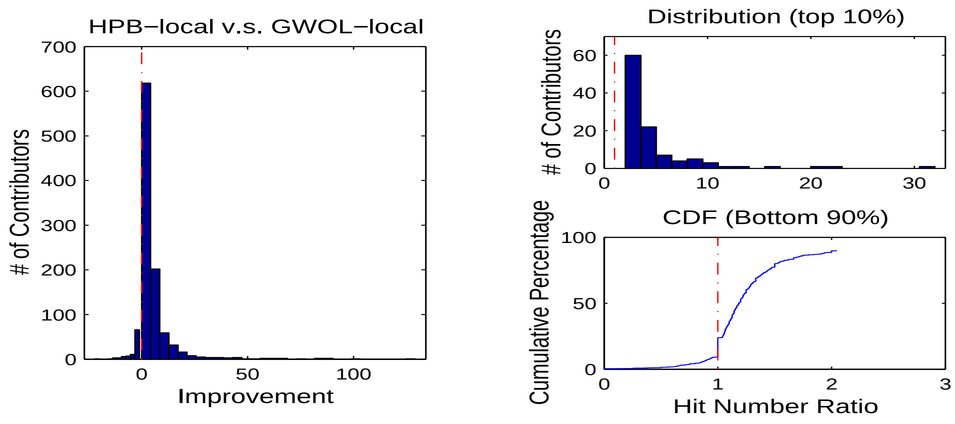

Figure 5.4: HPB-local Predicts More New Attacks Than GWOL-local

ues for window T_1 to T_8. We denote them $IVs(v) = \{IV_1(v), IV_2(v), \ldots IV_8(v)\}$. We then define a consistency index (CI) for each contributor. If $IV_i(v) \geq 0$, we say that HPB performs well for v in window i. Otherwise, we say that HPB performs worse. CI is the absolute value of the difference between the number of windows in which HPB performs well and the ones in which HPB performs poorly, i.e., $CI(v) = abs(|\{p \in IVs(v) : p \geq 0\}| - |\{p \in IVs(v) : p < 0\}|)$, where $abs(x)$ is the absolute value of x. (Note that we focus on HPB's performance consistency in this experiment. For a particular contributor, HPB may perform well or worse. However, as long as it perform well (worse) for all the 8 windows, we say that it is consistent. Therefore, we use the absolute value in the definition of CI. Being consistent does not necessarily mean that HPB predicts more hits.) Clearly, if HPB has a consistent performance along time for contributor v, $CI(v)$ should be close to 8. Otherwise, if the performance of HPB flip-flops, its CI value will be close to zero. Figure 5.5 plots the CDF of HPB's CI values with different contributors. We see that for more than 80% of the contributors, HPB's performance is extremely consistent. They all have CI value 8. For more than 90% of the contributors, HPB demonstrates fairly good consistency. Only with very few contributors, the performance switches back and forth. Further investigation shows that flip-flop contributors involve cases where HPB and GWOL have hit numbers in very close proximity. In such a case, a small change in hit number can switch HPB from good to bad.

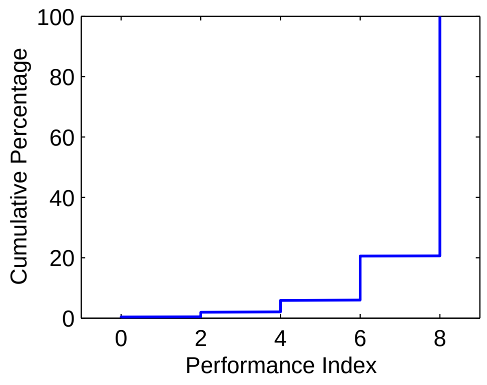

Figure 5.5: Cumulative Distribution of Consistency Index

The consistency result is quite useful. It implies that given a few cycles of computing HPB and GWOL for a given user, we can provide an informed recommendation for the user as to which list he or she should adopt over a longer term. That is, our experiment suggests that if HPB performed well in the past month, it will tend to continue performing well over at least several subsequent weeks. Therefore, the users may benefit from HPB in this case.

5.4 Blacklist Length

In this experiment, we vary the length of the blacklists to be 50, 100, 200, 500, and 1000. We then compare the hit numbers of HPB, GWOL, and LWOL. Because in all the experiments, the IV and HR values for different contributors display similar distributions, we will simply plot the medians of the hit numbers of HPB, GWOL, and LWOL when we present the results. In Figure 5.6, for each choice of list length, we plot the median of the hit numbers for HPB, GWOL, and LWOL across the contributors. The result shows that HPB has an advantage for all choices of length value.

5.5 Training and Prediction Window Sizes

We now investigate how far into the future the blacklists can make good predictions and how different training window sizes affect HPB's hit numbers. The former helps to determine

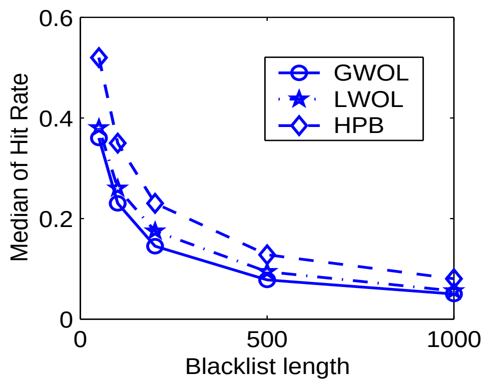

Figure 5.6: Hit Numbers of HPB, GWOL, and LWOL with Different Lengths

how often we need to recompute the blacklist, and the latter helps to select the right amount of history data as the input to our system. The left panel of Figure 5.7 shows the median of the hit number of HPB, GWOL, and LWOL on day $1, 2, 3, \ldots, 31$ in a large prediction window. All lists are generated using data from a 5-day window prior to the prediction window. We see that for all blacklists, the number of hits decreases along time. HPB always has an advantage during the whole prediction window. (There is a dip around day 5 because the total number of attacks reported by DShield is significantly less on that particular day. Nevertheless, HPB still performs better in this case.) The right panel of Figure 5.7 plots hit-number medians for five HPBs that are generated using training windows of size 2, 5, 10, 20, and 30 days. The hit numbers are obtained in a 5-day prediction window. We see that the hit numbers are roughly the same for HPBs generated with training windows of different sizes. At first glance, this seems strange. But it turns out to be a natural property of our HPB algorithm. Two major factors affect the generation of HPBs. One is the correlation pattern among victims (contributors). Previous work [9] has shown that the victim correlation patterns are quite stable. They remain the same for a long time. Therefore, the correlation graph obtained from different training windows may be very similar. The other more important factor is that we used an exponential decay function in computing B and $\hat{B}$ in Eq. 4.1 and Eq. 4.4. Attackers that have been dormant for longer time periods will be assigned extremely small scores and therefore are effectively excluded from consideration for HPBs.

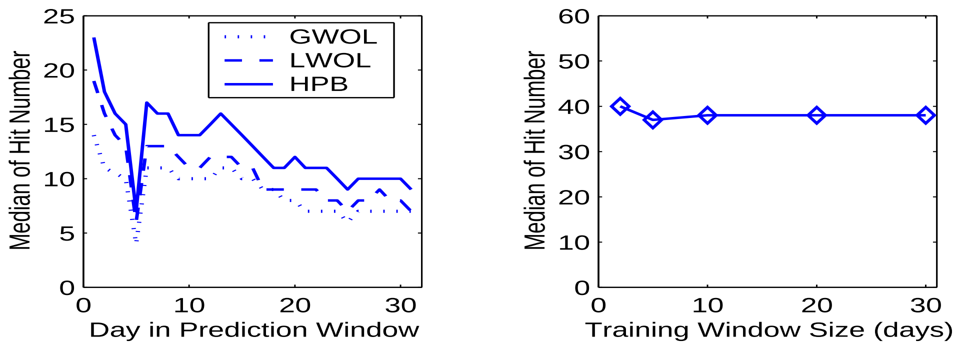

Figure 5.7: Effect of Training Window and Prediction Window Size on HPB's Hit Number

Chapter 6

Examining Selection Strategies for Final Blacklist Publication

No matter what the blacklist formulation strategy, one must recognize that most perimeter boundaries will have a finite filter set size. In practice we observe that these filter sets may range from a few dozen to several hundred entries. Thus, when deciding which entries to incorporate into a perimeter defense, the opportunity cost for incorporating poor-performing rules can be significant. Poor-performing rules may have several interpretations, depending on one's perspective. Poor-performing rules may include rules that exhibit

poor timing: filtering rules that are inserted well after an attacker has had time to saturate the network with unwanted communications

poor hit rate: rules that identify malicious sources that are rarely if ever encountered by the blacklist user

low severity: rules that filter sources whose traffic patterns would suggest benign rather than malicious communications (e.g., selecting to filter web crawlers rather than sources that are actively targeting known malware ports).

With respect to *poorly timed* blacklist entries, worst offender lists generally suffer from the fact that a source does not achieve candidacy into the list until it has produced a sufficient mass of communication. GWOL rules have the tendency to incorporate only those sources that have already achieved mass proliferation. LWOLs are particularly limited by poor timing, as they are entirely reactive to attackers that are actively pounding the LWOL network with bad communications. Figure 5.6 has already quantified HPB's ability to successfully identify new attackers.

With respect to poor *hit rate*, LWOLs generally achieve a much higher hit rate than GWOLs. We interpret this result to follow a pattern similar to Newton's first Law of Motion. An examination of LWOL hit rates suggests that an attacker who is sending unwanted communications to a target will tend to continue sending unwanted communications to the target. With respect to our experimentation with the Hybrid strategy, we observe that with regard to the DShield dataset, the Hybrid hit rate is typically bounded below the best-performing list, GWOL or LWOL.

Regarding *low severity* blacklist entries, we believe one potential key selection criterion in formulating a final blacklist could be that of evaluating entries relative to the apparent benignness or aggressiveness of their historical traffic patterns. With respect to evaluating the *benignness* of a potential blacklist entry, this question is particularly relevant in the case of global blacklist publication. In practice, formulation of the DShield blacklist requires special care to avoid including benign sources that are accidentally logged by many contributors. For example, web crawlers and Internet measurement services must be regularly removed by hand prior to the publication of the DShield GWOL. While such filtering should also be considered for the other lists, such care is arguably less important as these lists are constructed individually per contributor network. Unlike the GWOL that may be incorporated by tens of thousands of users, an accidental inclusion of a benign address in the LWOL or HPB would not produce an unwanted large-scale impact.

Prioritizing *aggressiveness* in network communication patterns has the potential to provide an important selection criterion in final blacklist publication. Here we assume that most perimeter defenses would like to achieve as much overlap as possible in incorporating highly malicious entries and entries that also have a high probability of being exercised. With respect to DShield-derived blacklist entries, the DShield entries provide enough context per log entry to support at least three metrics for developing attack severity heuristics. In proposing these metrics we will borrow criteria that have already been proposed in the

Candidate Address	Source Count	DPort Count	Weighted Average	Target Port N-Gram
024.044.038.*	92	3	3.0	[1434-udp 38566-tcp 1434-tcp]
222.091.041.*	86	5	2.8	[1434-udp 32459-udp 8672-udp 445-tcp 1434-tcp]
084.016.230.*	8	2	2.8	[80-tcp 22-tcp]
219.081.144.*	12	8	2.2	[13102-udp 139-tcp 3127-tcp 445-tcp 25-tcp 6662-tcp ...]
218.092.241.*	98	4	1.8	[3621-udp 1433-udp 1433-tcp 6192-udp]
210.021.119.*	9	2778	1.0	[20056-tcp 59990-tcp 56333-tcp 47196-tcp 5543-tcp ...]
204.094.057.*	43	1	4.0	[445-tcp]
071.123.126.*	52	2	4.0	[1434-udp 1434-tcp]
066.190.000.*	39	3	3.2	[80-tcp 445-tcp 443-tcp]
065.214.044.*	56	3	3.0	[53-udp 80-tcp 1026-udp]
059.114.213.*	19	5	2.8	[139-tcp 32459-udp 3127-tcp 9963-udp 445-tcp]
061.216.052.*	24	7	2.8	[135-tcp 139-tcp 6883-udp 8672-udp 3127-tcp 445-tcp ...]
058.052.129.*	115	4	2.5	[1434-udp 32459-udp 6567-udp 1434-tcp]
219.116.071.*	3	2	2.5	[1434-udp 32606-tcp]
061.229.070.*	13	7	2.4	[135-tcp 139-tcp 445-tcp 25-tcp 12198-tcp 38566-udp ...]
060.176.088.*	78	7	2.3	[1434-udp 35496-udp 32459-udp 445-tcp 1434-tcp ...]
216.127.253.*	57	6	1.5	[6346-tcp 38566-tcp 445-tcp 49200-udp 12198-udp 30614-tcp]
058.215.065.*	54	948	1.4	[968-tcp 844-tcp 1086-tcp 1065-tcp 897-tcp 250-tcp ...]
219.129.216.*	68	981	1.1	[4504-tcp 281-tcp 6701-tcp 897-tcp 822-tcp 6452-tcp ...]
061.129.051.*	65	6611	1.0	[20383-tcp 21577-tcp 8147-tcp 22343-tcp 22578-tcp ...]
218.007.120.*	83	2329	1.0	[56333-tcp 59990-tcp 23831-tcp 10846-tcp 53833-tcp ...]

Table 6.1: Example HPB Blacklist Candidates

context of dynamic signature generation [10] and malicious port scan analysis [8] for use in similar threat-level prioritization:

- *Source Diversity:* a measurement of the number of unique contributors that have logged the source address. A higher measurement implies greater IP address range sweeping.
- *Target Port Diversity:* a measurement of the number of unique target ports reported against this attacker. A higher measurement implies greater IP address range sweeping.

- *Weighted Target Port Scan Metrics:* a weighted aggregate or average measurement of unique destination, where greater weight may be based on known vulnerable ports (e.g., ports well associated with exploit usage), less weight placed on network services UDP/TCP port ranges, and lesser weight placed on application ports above 1024.

Table 6.1 illustrates a set of attacker addresses that were selected for one contributor blacklist by the HPB strategy (a subset of entries that were uniquely identified by the HPB in one example blacklist calculation). Each row identifies a unique candidate source LAN for possible inclusion into the contributor's blacklist. Column 2 indicates the number of other contributors (within DShield's full contributor pool) who also reported the candidate source address during the training window (the higher this number, the more aggressively this address is sweeping the Internet address space). Column 3 indicates the number of unique target ports that the candidate address has been observed connecting to among all contributors (one contributor has port scanned more than 2700 unique target ports). Column 4 reports the average weighted port scan value for the candidates. With respect to the vulnerable port list metric used in Table 6.1, we used the following set of target ports:

Vulnerable Port Set: 53-tcp, 42-tcp, 80-tcp, 135-tcp, 139-tcp, 445-tcp,559-tcp, 1025-tcp, 1433-tcp, 14434-tcp, 2082-tcp, 2100-tcp, 2745-tcp, 2535-tcp, 3127-tcp, 3306-tcp, 3410-tcp, 5000-tcp, 5554-tcp, 6101-tcp, 6129-tcp, 0-tcp, 11768-tcp, 15118-tcp, 27374-tcp, 65506-tcp, 53-udp, 69-udp, 137-udp, 1434-udp, 4444-tcp, 9995-tcp, 9996-tcp, 17300-tcp, 3140-tcp, 903-tcp

We envision that the application of a severity-based entry selection process could be incorporated at any of several stages in the blacklist formulation process. One possibility is to incorporate attacker severity assessment as a prefiltering step that occurs in selecting which sources are considered during the HPB training window. Another possibility is to allow all sources to be considered during the training window, but to incorporate a severity metric into the HPB scoring algorithm itself. Finally, we could also incorporate a severity-based entry selection as a final processing step in deciding which entries of a candidate HPB list of size N will be published to the final delivered blacklist of size M, where M <

N. For example, using a combination of the metrics suggested above, one could prioritize source addresses that appear to exhibit malware characteristics (i.e., the example use of such metrics is already documented in [8, 10]).

Chapter 7

The DShield Implementation

DShield began providing a global worst offender blacklist early from its inception, in the form of a text file. The text file was designed to be human readable as well as easy to parse by simple scripts. A number of open source as well as commercial firewalls use this list. The highly predictive blacklist will be offered in a very similar form. To provide this customized blacklist, a credential exchange occurs (discussed below), followed by a release of a URL containing the contributor's updated HPB list. HPBs are calculated daily for the entire contributor base, not on demand (avoiding potential DoS conditions).

A DShield user is provided with an account to the DShield website in order to review reports. To ease the automatic retrieval of a user's HPB blacklist, we will not require the user to log in via our standard web-based procedure. Instead, the user can generate a unique token. This random hexadecimal sequence will be appended to the URL and identify the user. This token has a number of advantages over using the user's username and password. For example, the user may still change the password without having to change the information used by automated scripts retrieving the blacklist.

To provide further protection of the integrity and confidentiality of the HPB blacklist, it will be offered via https. A detached PGP signature can be retrieved in case https is not available or not considered sufficient to prove the authenticity of the list.

The HPB blacklist itself uses a simple tab delimited format. The first column identifies the network address, and the second column provides the netmask. Additional columns can be used to provide more information about the respective offender, like type of attacks seen, name of the network, and country of origin. Initially, such additional columns are intended for human review of the blacklist. Comments may be added to the blacklist. All comments start with a # mark. The following is a sample HPB blacklist template:

```
# DShield Customized HPB Blacklist
# created 2007-01-19 12:13:14 UTC
# Sor userid 11111
# Some rights reserved, DShield Inc.,
#
# Creative Commons Share Alike License
# License and Usage Info: http://www.dshield.org/blocklist.html
#
1.1.1.1    255.255.255.0      test network
2.2.2.2    255.255.255.0      another test: network does not exist
# End of list
```

The URL for DShield's current GWOL remains the following URL:

`https://feeds.dshield.org/block.txt.`

The PGP signature for the current list is available from the following URL:

`http://feeds.dshield.org/block.txt.asc.`

The user will request the personalized HPB using the following URL:

`http://feeds.dshield.org/hpblock6de8e...a6a.txt.`

The respective PGP signature will be retrieved using the following URL:

`http://feeds.dshield.org/hpblock6de8e...a6a.txt.asc.`

Blocklists will be updated once a day or whenever processing load allows. The files retrieved by the user are static. The processing to generate a blocklist is too large to be done on demand. Using static files will prevent an obvious DoS condition by downloading the same file multiple times. Instead of using a PGP signature, we may opt to use HTTPS. However, HTTPS will protect only the data on the wire. It will not protect the pregenerated list on the server.

Chapter 8

Conclusion

We offer a new argument to help motivate the field of secure collaborative data sharing, by demonstrating that people who collaborate in blacklist formulation can produce *highly predictive blacklists.* We introduce a blacklist formulation algorithm that is based on an extension of Google's PageRank link analysis. Experimenting on a large corpus of real DShield data, we demonstrate that HPB has higher attacker hit rates, better *new attacker* prediction quality, and long-term performance stability. Furthermore, we show that such advantage exists for different blacklist lengths, as well as a variety of prediction window sizes. We also suggest methods for assessing the severity of source attacker behavior to prioritizing the sources on the list. Our HPB algorithm has been developed into a real-world application, which is now posted to the DShield.org website for all repository contributors to use.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] 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.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[4] US CERT. CERT Advisory CA-2003-04: MS SQL Server Worm. `http:/www.cert.org/advisories/CA-2003004.html`, 2003.

[5] Mark Humphrys. The Internet in the 1980s. `http://www.computing.dcu.ie/~humphrys/net.80s.html`, 2007.

[6] Google Incorporated. List of Blacklists. `http://directory.google.com/Top/Computers/Internet/Abuse/Spam/Blacklist%s/`, 2007.

[7] Google Incorporated. Live-Feed Anti-Phishing Blacklist. `http://sb.google.com/safebrowsing/update?version=goog-black-url:1:1`, 2007.

[8] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[9] S. Katti and B. Krishnamurthy. Collaborating Against Common Enemies. In *Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference*, October 2005.

[10] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, pages 271–286, 2004.

[11] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Recommendation systems: A probabilistic analysis. *Journal of Computer and System Sciences*, 63:42–61, 2001.

[12] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[13] M. Locasto, J. Parekh, A. Keromytis, and S. Stolfo. Towards collaborative security and P2P intrusion detection. In *Proceedings of the 2005 IEEE Workshop on Information Assurance and Security*, June 2005.

[14] P. Ruoming, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, October 2004.

[15] Johannes Ullrich. DShield Global Worst Offender List. `https://feeds.dshield.org/block.txt`.

[16] Wikipedia. The first DNS Blacklist. `http://en.wikipedia.org/wiki/DNSBL`, 2007.

[17] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the DOMINO overlay system. In *Proceedings of Network and Distributed Security Symposium*, June 2004.

[18] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence . In *Proceedings of ACM SIGMETRICS*, June 2003.