A Novel Worm Detection Model Based on Host Packet Behavior Ranking

Fengtao Xiao[1], HuaPing Hu[1,2], Bo Liu[1], and Xin Chen[1]

[1] School of Computer Science, National University of Defense Technology, Chang Sha, 410073
myfri2001@126.com, boliu615@yahoo.com.cn, cx917@21cn.com
[2] The 61070 Army Fu Zhou, Fu Jian, 350003 China
howardnudt@yahoo.com.cn

Abstract. Traditional behavior-based worm detection can't eliminate the influence of the worm-like P2P traffic effectively, as well as detect slow worms. To try to address these problems, this paper first presents a user habit model to describe the factors which influent the generation of network traffic, then a design of HPBRWD (Host Packet Behavior Ranking Based Worm detection) and some key issues about it are introduced. This paper has three contributions to the worm detection: 1) presenting a hierarchical user habit model; 2) using normal software and time profile to eliminate the worm-like P2P traffic and accelerate the detection of worms; 3) presenting HPBRWD to effectively detect worms. Experiments results show that HPBRWD is effective to detect worms.

Keywords: worm detection, behavior based worm detection, user habit model.

1 Introduction

Computer worms have become a serious threat to the Internet in recent years. Many researches [1] [2] have found that well designed worms can infect the whole Internet in a few minutes. Now, there exist two kinds of technologies on detecting worms: signature-based technology and behavior-based technology. Signature-based worm detection (SBWD) [3-6] can detect worms in real time, but it is a kind of worm-specific and passive technology. Although widely used in commercial AV software, signature-based worm detection cannot deal with the new coming and polymorphic or metamorphic worms effectively. Behavior-based worm detection (BBWD) [7-14] in itself is a kind of statistics method, so it is less effective in real time than SBWD, but on the other hand, BBWD is independent of packet content so that it can exceed signature-based in the ability of detecting unknown worms and polymorphic and metamorphic worms. BBWD has also its drawbacks. One of them is that it is difficult to distinguish between normal traffic and abnormal traffic. For example, in recent years, P2P software has been widely used. Our previous work [17] has showed that the widely used P2P software nowadays has the worm-like traffic behavior more or less.

In our opinion, the difficulties of detecting worms lie in four points: 1) propagation speed of worms is getting much faster so that the detection time left become less; 2) the application of polymorphic or metamorphic technology makes the

R. Meersman and Z. Tari (Eds.): OTM 2008, Part II, LNCS 5332, pp. 969–986, 2008.

signature-based worm detection less effective or completely fail; 3) the emergence of worm-like traffic such as P2P traffic makes the network-behavior based worm detection influenced with high false positives; 4) many work on BBWD is effective on fast spreading worms, but tends to miss the detection of slow worms or has high false positives.

In this paper, we mainly focus on BBWD. To try to address the four points mentioned above, a worm detection system based on host packet behavior ranking (HPBRWD) is presented. Its contributions to the four difficulties above lie in four points: 1) Using behavior based method to try to solve the second problem above; 2) Using normal software and time profile generated by HPBRWD to eliminated the influence of worm-like P2P traffic; 3) Using normal software profile and worm behavior profile to reduce the detection time; 4) detecting slow worms through the cumulative effect of HPBRWD.

The remainder of this paper is organized as follows: section 2 introduces the user habit model on Internet access; section3 overviews the design of HPBRWD; section4 discuss several key design issues and our solutions; section5 describe our prototype implementation and evaluations; section6 reviews related work. Finally, section7 makes concluding remarks with future work.

2 User Habit Model on Internet Access

Work [7] [8] have presented a definition for user habit on Internet access. According to that definition, user habit is influenced by user hobbies, characters and limitations of using internet. The definition is good, but it is lack in further description, so some important information is missed, such as habitual software. In our opinion, user habit is influenced with multi-level factors. Fig.1 lists the factors which influence the user habit.

In this model, user habit is described at three layers: user representation layer, use representation layer and network representation layer. To understand this classification, we can try to answer these questions: 1) what are the motivations for a user to access the internet? 2) Given these motivations, a user must find a way to satisfy these motivations under certain limitations. So how can he accomplish this? 3) Information is located in the internet, so what on earth is the representation of two layers above on the network packets?

We can see later that the three layers are just designed to answer the three questions above. At the same time, we can also see that each layer can separately describe the user habit, but they are in different levels.

User representation layer: this layer tries to answer the first question, that is to say, to describe what information or knowledge a user wants to acquire. User's characters, interests or hobbies, unexpected factors and other factors are included in this layer. Here other factors mean the customary access because of group behavior etc, for example, if you are impressionable, then once your colleagues tell you something interesting in the Internet, you will perhaps be attracted to visit it right now.

Use representation layer: this layer is used to answer the second question, that is to say, to describe what tools we are using and what limitations we will have to get the information. From fig.1 we can see there are two elements we shall pay attention to: Time and Software:

1) Time means that the time limitation to access the Internet. It includes starting time limitation and ending time limitation. Detail to say, this limitation has two meanings:

a) The time slots which can be used to access the internet. For example, a user in a demanding corporation can only use the Internet during non-working time which is the time limitation for this user. In such a case, the time limitation always has fixed start-time or end-time or both;

b) Unexpected changes of time limitation in length or start-time and end-time. This case is usually the result of user representation layer. For example, when a world-cup final is held, football fans will tend to spend more time using the Internet and browsing relative websites for further information about football. In such a case, the total time spent on Internet, the start-time or end-time of using Internet are always unexpected, but on the user representation layer side, the unexpected changes are in fact the result of user's interest on football.

2) Software means that the tools used to acquire the information needed. Different choosing of software will directly affects the network packets generated. For example, a user wants to watch world-cup final through the Internet webcast. There exist two kinds of software at least. One is the direct webcast on the official website or some web portals, such as www.sina.com; the other is using P2P software, such as ppstream or pplive etc. We can find that the two methods are greatly different in the destination IP address selection and the traffic of communication.

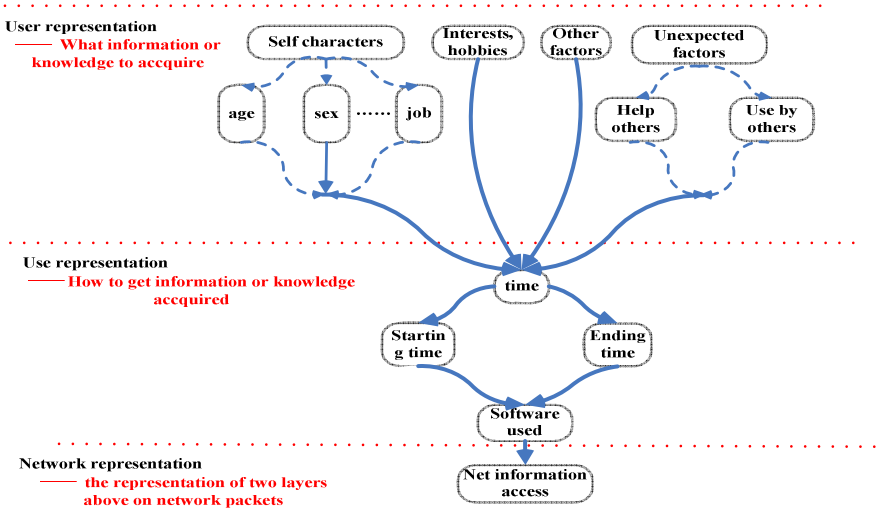

Fig. 1. User habit model

Network representation layer: this layer is used to answer the third question, that is to say, to describe the network packets created during a user accesses internet. These packets can be seen as the results of the two higher layers.

From this model we can see that to distinguish normal user network behavior and worm behavior, we must pay attention to both the use representation layer and network representation layer. But unfortunately, traditional behavior-based worm detection

only values at network representation layer. In fact, network representation layer is the least stable compared with the two other layers. But to tell the truth, network representation is the easiest to describe and use on detecting worms.

In this paper, we will try to present a novel worm detection algorithm through combining the information of user representation layer and use representation layer. Detail to say, we will try to setup time and software profile automatically to improve worm detection.

3 Overview of HPBRWD Design

3.1 Design Goals and Principles

The design goals of HPBRWD include three points: 1) detecting worms as early as possible; 2) eliminating worm-like P2P traffic; 3) detecting both fast worms and slow worms.

To achieve such goals, HPBRWD is designed to be host-based so as to get extra information such as normal software profile and normal time profile easily. Information about normal software and normal time profile will be detailed addressed later.

We monitor the software used on the host and the traffic from and to the host. Based on the information, we dynamically setup an initial profile about normal software and corresponding ports used by the software automatically.

Profile of normal time slots is dynamic created and HPBRWD has a default profile which sets all the time slots usable. After the steps above, HPBRWD begins to monitor in real-time.

The principles underlying the HPBRWD design are as follows:

1) Connecting different IPs with the same destination port within a time slot is suspicious. For a fast worm, it needs to propagate as quickly as possible, so the slot will be short. HPBR is designed to not be limited to a short time slot, so it can not only detect fast worms, but also it is effective to detect slow worms.
2) Same messages sent to different nodes construct a tree or chain. This behavior is also considered to be suspicious. Up to now, except some P2P software, Normal traffic will not have this behavior. For example, if a host receives a message on port A, then it sends to port A on another host the same or similar message, then we can consider this process is suspicious.
3) Software not in the normal software profile and having one or both of the two behaviors above is suspicious.
4) Packets' timestamp not in the normal time profile is suspicious.

With the emergence of P2P software, we can find that the first and the second principles are not the unique principles of computer worms. But our previous work [17] has showed that based on HPBR, we can eliminate most or whole worm-like P2P traffic.

3.2 HPBRWD Architecture and Flow of Control

Figure 2 shows the modules of HPBRWD. HPBRWD is host-based, so all the modules are located on just one host. There are six modules in HPBRWD:

1) Profile setup module. This module focuses on setting up profiles of normal software and normal time slots.

2) Packet capture module. This module focuses on capturing the packets to and from the host. Different from general packet capture process, the traffic created by software in the normal software profile will not be captured.

3) Packet preprocess module. This module preprocesses the packets to create formatted packets denoted as meta-access information (MI). These formatted packets will be the data source for HPBR detector module.

4) HPBR detector module. This module ranks every formatted packet and the selectively stored historical formatted packets. When the ranking value is greater than the threshold, alert will be triggered.

5) Profile update module. This module includes the update of these profiles: normal software, normal time slots, blacklist and selectively stored packet history (denoted as MI library). Blacklist and MI library are generated in HPBR detector module. When HPBR detector module completes, these two profiles are also updated.

6) Alert process module. This module processes the responses after an alert is triggered. It will look for the program which sent the corresponding packets. In our implementation, this module will take three measures: alert, log and process termination.

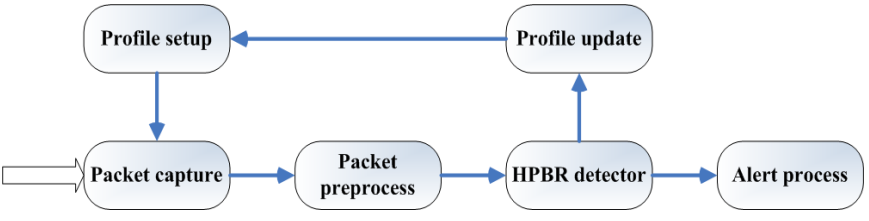

Fig. 2. Architecture of HPBRWD

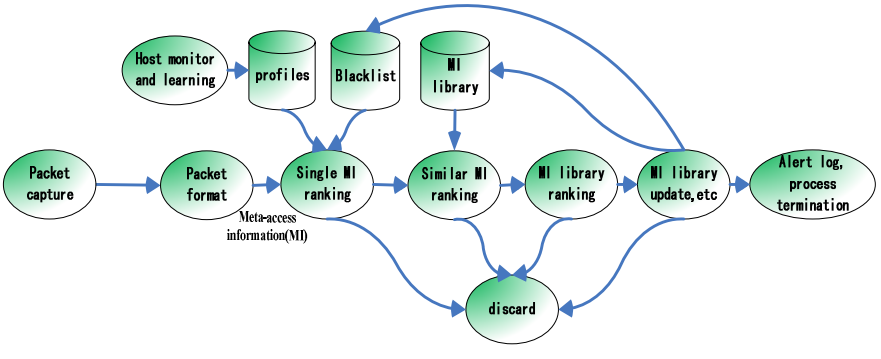

Fig. 3. Flow of HPBRWD

Fig.3 shows the process flow of HPBRWD. The concrete steps are as follows:

1) Creating normal software profile and normal time periods profile through host monitor and learning. The profiles will be stored into database.
2) Capturing and filtering packets according to normal software profiles. After this step, all the traffic coming from the normal software will be filtered. Only the traffic from the abnormal software and the traffic to the host are left.

3) Formatting packets. After this step, packets are formatted to be meta-access information (MI).
4) Ranking single MI. According to profiles created in step 1), we will first rank the single MI. If the single MI is in the blacklist, alert will be triggered at once.
5) Ranking similar MIs. In this step, the current MI will be ranked with MI library. We will find whether there is any MI in the MI library which is similar with current MI. If existing, we will update the rank value of corresponding MI in the MI library.
6) Ranking MI library. The ranking value of whole MI library helps us making decisions on whether to trigger an alert or not. If this ranking value is over the threshold, an alert will be triggered.
7) Updating MI library. When MI is ranked, it will be added to MI library according to MI library updating algorithm. The blacklist will also be updated in this step.
8) Processing alerts. When a worm is detected, measures such as log, alert or process termination will be taken.

4 Key Issues in the Design of HPBRWD

In this section, we discuss some key design issues and our solutions.

4.1 How Do We Carry on HPBR?

To answer this question, we will first give some denotations on data structure used in HPBR, then the definitions of MI and similar MIs. Based on these two definitions, we will introduce the design of HPBR. At last, we will present the algorithm of updating MI library.

Definitions and data structures. Before explaining HPBR, we first list the denotations, data structures and definitions.

Remark 1. Denotations

Table 1. Denotations

Denotation	Description	Denotation	Description
SW	Habitual used software	T	Habitual time accessing internet
B	Blacklist	W	White list
DP_i	Ports used by software *i*	DP	The total ports on one host. $DP = \cup DP_i$
$g(MI_i, DP)$	Weight of destination port of MI_i	$h(MI_i, SW)$	Weight of software sending MI_i
$q(MI_{[0..x-1]}, MI_x)$	Number of history MIs which are similar with MI_x	$k(MI_i, T)$	Weight of time-stamp of MI_i
$w(MI_x, W)$	If MI_x is in W	$B(MI_x, B)$	If MI_x is in B
α	Accelerating coefficient of anomalous software	β	Accelerating coefficient of anomalous destination port

Table 1. (*continued*)

γ	Accelerating coefficient of anomalous similar MIs	θ	Accelerating coefficient of anomalous time-stamp
ε	Max length of MI library	φ	Max number of MIs for a single destination IP
μ	Threshold of triggering alert		

At the same time, we definite some structures corresponding to some of the denotations above.

$$h(MI_i, SW) = \begin{cases} 0, & MI_i \in SW, i \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

$$k(MI_i, T) = \begin{cases} 0, MI_i.pTime \in T \\ 1, \quad \text{otherwise} \end{cases} \tag{2}$$

$$g(MI_i, DP) = \begin{cases} 1, & \sim MI_i.dPort \in DP \\ 0.5, (\sim MI_i.dPort \in DP_i) \cap (MI_i.dPort \in (DP \backslash DP_i)) \\ 0, & MI_i.dPort \in DP_i \end{cases} \tag{3}$$

$$q(A_{[1..y],}MI_x) = \begin{cases} A_i.num + 1, & (x \geq 1) \cap (\exists i \in [1..y], st\ A_i.MI \cong MI_x) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$q(A_{[1..y],}MI_x)_i = \begin{cases} A_i.num + 1, (x \geq 1) \cap (A_i.MI \cong MI_x) \\ 0, \quad \text{therwise} \end{cases} \tag{5}$$

$$w(MI_x, W) = \begin{cases} 0, (MI_x.dPort, MI_x.progName) \in W \\ 1, \quad \text{otherwise} \end{cases} \tag{6}$$

$$b(MI_x, B) = \begin{cases} 0, (MI_x.dPort, MI_x.protocol, MI_x.pSize, MI_x.progName) \in B \\ \delta + 1, \quad \text{otherwise} \end{cases} \tag{7}$$

Remark 2. Definitions

Definition 1: *MI (Meta-access Information).* MI is in fact the formatted packet. We denote it as:
$MI = \{sIP, sPort, dIP, dPort, protocol, pSize, pTime, pProgName\}$. The elements of MI stands for source IP, source port, destination IP, destination port, protocol, size, timestamp of the a packet and the software which sent the packet.

Definition 2: *Similar MIs.* We call two MIs as similar MIs when they meet with the first two principles in section 3.1. If MI_1 and MI_2 are similar, we denote them as $MI_1 \cong MI_2$, to make the judgment of similar MIs easy, we define such functions:

For MI_1 and MI_2:

$$f_1(\mathrm{MI}_1, \mathrm{MI}_2) = \begin{cases} 1, (\mathrm{MI}_1.\mathrm{dPort} = \mathrm{MI}_2.\mathrm{dPort}) \cap (\mathrm{MI}_1.\mathrm{dIP} = \mathrm{MI}_2.\mathrm{sIP}) \\ \qquad \cap (\mathrm{MI}_1.\mathrm{pTime} < \mathrm{MI}_2.\mathrm{pTime}) \\ 0, \qquad \text{otherwise} \end{cases} \tag{8}$$

Here f_1 means whether two MIs have the same destination port and the latter MI's source IP is the same with the former MI's destination IP. If true, we set $f_1 = 1$. In fact, from the view of host, f_1 tells us how to judge the principle 2).

$$f_2(\mathrm{MI_i}) = \begin{cases} 1, \mathrm{MI_i}.\mathrm{dPort} \neq \mathrm{MI_i}.\mathrm{sPort} \\ 0, \qquad \text{otherwise} \end{cases} \tag{9}$$

Here f_2 means that for one MI, whether the destination port equals its source port. If true, we set $f_2 = 1$. In fact, this judgment is from the results of experiments. We found that P2P software always has such feature while computer worms don't have because their propagation process is always multi-thread based. So we add f_2 to eliminate false negatives created by P2P software.

$$f_3(\mathrm{MI}_1, \mathrm{MI}_2) = \begin{cases} 1, & (\mathrm{MI}_1.\mathrm{dPort} = \mathrm{MI}_2.\mathrm{dPort}) \cap \\ & \left(\begin{array}{c} \sim((\mathrm{MI}_1.\mathrm{dPort} \in \mathrm{DP}_{\mathrm{MI}_1.\mathrm{progName}}) \cup \\ (\mathrm{MI}_2.\mathrm{dPort} \in \mathrm{DP}_{\mathrm{MI}_2.\mathrm{progName}})) \end{array} \right) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Here f_3 is to judge whether two MI have the same destination port and this port is not in port list of software sending MI_1 and MI_2. If true, we set $f_3 = 1$. So In fact, f_3 tells us how to judge the principle 1) and 3).

$$f_4(MI_1, MI_2) = \begin{cases} 1, (MI_1.protocol = MI_2.protocol) \cap (MI_1.pTime \neq MI_2.pTime) \\ \qquad \cap (MI_1.dIP \neq MI_2.dIP) \cap (MI_1.dPort \neq MI_2.sPort) \\ \qquad \cap (MI_1.pSize = MI_2.pSize) \\ 0, \qquad otherwise \end{cases} \tag{11}$$

Here f_4 lists the base characters two MIs should have to judge whether a worm exists or not.

$$f(MI_1, MI_2) = \left(f_1(MI_1, MI_2) * f_2(MI_1) * f_2(MI_2) + f_3(MI_1, MI_2)\right) * f_4(MI_1, MI_2) \tag{12}$$

Here f is to judge whether two MIs are suspicious MIs. It is easy to know $\mathrm{MI}_1 \cong \mathrm{MI}_2$ when $f(MI_1, MI_2) \geq 1$. At the same time, $\mathrm{MI}_1 \cong \mathrm{MI}_3$ when $\mathrm{MI}_1 \cong \mathrm{MI}_2$ and $\mathrm{MI}_3 \cong \mathrm{MI}_2$.

Definition 3: *MI library.* We call the set which saves MI history information and statistics information of MI as MI library. MI library is denoted as A and we set $\mathrm{A} = \{\mathrm{a} | \mathrm{a} = (\mathrm{MI_i}, \mathrm{Fvalue}, \mathrm{Num}, \mathrm{IPList})\}$. $|\mathrm{A}|$ means the length of A, Fvalue stands for the ranking value of MI, Num means the number of MI which is the same with $\mathrm{MI_i}$, IPList means the IP list of MI which is the same with $\mathrm{MI_i}$.

Host Packet Behavior Ranking. Based on the denotations and definitions above, for a coming $\mathbf{MI_x}$, we use $\mathbf{F_1(A_{[1..y]}, MI_x)}$ as ranking function:

$$\mathbf{F_1\left(A_{[1..y]}, MI_x\right)} = \begin{cases} \mathbf{q\left(A_{[1..y]}, MI_x\right) * \gamma * \left(1 + \alpha * h(MI_x, SW) + \theta * k(MI_x, T)\right), y \geq 1} \\ \mathbf{0,\ y = 0} \end{cases} \tag{13}$$

Supposing that there exists $\mathrm{MI_x} \cong \mathrm{MI_k}$ in formula (6), then for an $\mathrm{A_i}$ in MI library A, the corresponding rank function is:

$$\mathrm{A_i.Fvalue} = \begin{cases} \sum_{j=1}^{x-1}\left(1 + \alpha * h\left(MI_j, SW\right) + \theta * k\left(MI_j, T\right)\right) * \gamma * q(A_{[1..y]}, MI_j)_i,\ x \geq 2 \\ \sum_{j=1}^{x-1}\left(\left(1 + \alpha * h\left(MI_j, SW\right) + \theta * k\left(MI_j, T\right)\right) * \gamma * q\left(A_{[1..y]}, MI_j\right)_i\right) \\ \qquad + F_1\left(A_{[1..y]}, MI_x\right),\quad x \geq 2, i = k \\ 0, \qquad x = 1 \end{cases} \tag{14}$$

So the ranking function for the whole MI is:

$$F\left(A_{[1..y]}, MI_x\right) = \max\{A_i.\mathrm{Fvalue}\}, x - 1 \geq i \geq 1 \tag{15}$$

When $F\left(A_{[1..y]}, MI_x\right)$ is greater than the threshold μ, an alert will be triggered. At the same time, suppose that $F\left(A_{[1..y]}, MI_x\right) = A_m.\mathrm{Fvalue}$, then when an alert is triggered, we set $A_m.\mathrm{Fvalue} = 0$, but for a new coming MI_{x1}, if $q(A_{[1..y]}, MI_{x1})_m > 0$, then an alert will also be triggered.

MI library updating policy

MI library caches the recently coming MI, and at the same time, from section 4.1, we can see the access to the MI library is also very frequent when performing HPBR. Thus, the length of MI library cannot be infinite because it has a great influence on the performance of HPBR. In our work, to effectively use the MI library, we use Modify_A algorithm to update it.

Algorithm $Modify\_A$:

Input: *MI library A, new coming* MI_x

Output: *none*

1 compute $q(A_{[1..y]}, MI_x)$

2 if $q\left(A_{[1..y]}, MI_x\right) > 0$ *and* $A_i.MI \cong MI_x$ *, then add* $MI_x.dIP$ *to* $\mathrm{A_j}$ *endif;*

3 if $q\left(A_{[1..y]}, MI_x\right) = 0$ *then*

3.1 if $|A| < å$ *then add* $\mathrm{MI_x}$ *to A endif;*

3.2 If $|A| = å$ *then*

3.2.1 Traverse A *and find the minimal from the entire MI library element's rank values;*

3.2.2 If more than one element in MI library has the minimal rank value and the value is 0

Then for every element $\mathrm{A_i}$ *that has the minimal rank value 0*

Compute $min\ \{1 + á * h(A_i.MI, SW) + è * k(A_i.MI, T)\}$

If more than one A_i *meets the condition,*
Then random select one A_i *to delete*
Endif;
Endif;
3.2.3 If only one element has the minimal rank value
Then delete this element
Endif;
3.2.4 If the minimal rank value >0 then
Set |A|=|A| + 1;
Endif;
Endif;
3.3 Add MI_x *to A;*

Modify_A makes the similar MIs have the biggest chance to stay in the MI library, so that we can effetely rank them and detect worms. From Modify_A we can also find that |A| is not certain in all situations. If the minimal rank value is not equal 0, |A| will increase. This is the core of detecting slow worms and the detail description will be introduced in section 4.3.

4.2 How to Setup Normal Software Profile and Time Profile Automatically

For HPBRWD, normal software profile and time profile can help us in three aspects: 1) filtering unnecessary traffic to improve processing efficiency; 2) accelerating detection of computer worms through corresponding accelerating coefficient; 3) eliminating the influence of P2P software. So how to setup these two profiles is one of the key issues of HPBRWD.

Definition 4: *normal software.* If software doesn't have the features like the principle 1) and 2), then we consider it as normal software.

Definition 5: *suspicious software.* If we can't decide whether software has the features like the principle 1) and 2), we consider it as suspicious software.

Definition 6: *infected software.* If software has the features like 1) or 2) or both, we consider it as infected software.

In HPBRWD, the construction of normal software profile is a dynamic process. You can see it in fig 4. The state changes of three kinds of software are also listed in fig 4.

The software which we are using on a host can be divided into two classes: 1) installed software; 2) green software. So to get the software list, we can focus on such two classes.

For installed software, we can check registry to get detail information. Detail to say, all the installed software save their installation information under the registry item "HKEY_LOCAL_MACHINE\software\Microsoft\windows\uninstall\InstallLocation" tells us the name and path of software.

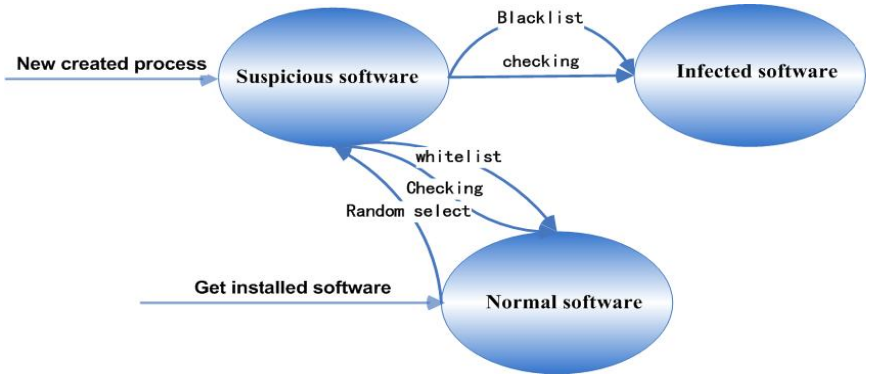

Fig. 4. State changes of the three kinds of software

For Green software, we decide to get its information when it begins to run. We hooked the function *NtResumeThread* to get information of new created process. Thus we can get software which user is running.

For the new created process, we first consider them to be suspicious until we can't find that they have the features like principles 1) and 2) in a given time period. To be suspicious software means all the traffic sending or receiving will be recorded to HPBR.

There are two situations can software A be shifted from suspicious software to normal software: 1) A is in white list; 2) based on HPBR checking, if no principles 1) and 2) are found. When either of these two situations is satisfied, we add this software to Normal software profile.

For those installed software, they will first be classified into normal software, but it doesn't mean that each software in normal software profile is always immune to computer worms, so we randomly select software from normal software profile and change their state to suspicious software every given time slot. So the creation of normal software profile is dynamic.

There are also two situations for software A to change from suspicious software to infected software: 1) A is in blacklist; 2) based on HPBR, an alert is triggered.

For the creation of normal time profile, we first give the definition of normal time:

Definition 7: *normal time.* Normal time means the time slot when a user always accessing internet. Detail to say; if % of normal software is running during time slot A, we call this time slot A as normal time. is a parameter, but in our implementation now, we decide that a time slot will be consider normal time only if there is one normal software running.

Normal time profile is not stable. To create it, we first look up all the process running now to see if there is any in normal software profile, if true, the time slot between now and 5 minutes later are considered to be in normal time profile. When 5 minutes passed, there will be another checking on normal time.

Of course, we can also manually set the normal software profile and normal time profile.

4.3 How Does HPBRWD Eliminate the Influence of Worm-Like P2P Traffic?

In this paper, to eliminate the influence of worm-like P2P traffic is relatively easy. The reason is as follows: traditional P2P traffic identification methods mainly use communication information, while ours is based on host information. We can easily get the P2P software list used in the host. Through hooking *ntDeviceIoControl*, we can easily acquire the correlation between process and port. Source port is the bridge between the packet's information and P2P software self's information, the whole process can be shown in fig5.

When a P2P application sends or receives packets, it will call ntDeviceIoControl function. We have hooked this function and save the corresponding source ports of this application. If this P2P application is in Normal software profile, we will update the source ports information to the normal software profile. At the same time, we use winpcap library to capture the packets of all applications. Source ports are also can be acquired in the packet's information. So according to the source port, we can know whether the packet captured is sent by a normal software or not. So at last, only packets sent by applications which are not in normal software profile will be formatted to MIs.

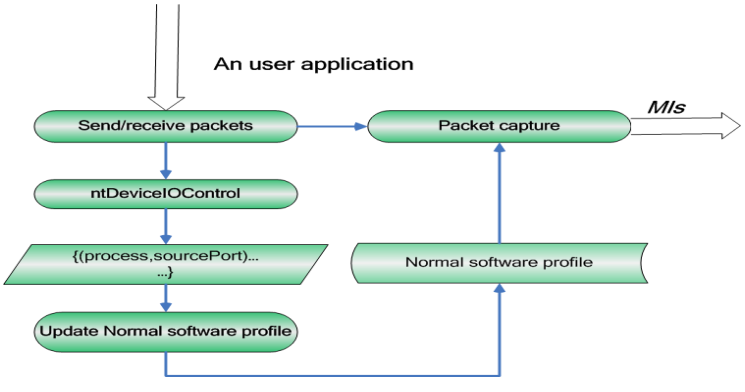

Fig. 5. Process of eliminating P2P traffic

4.4 How Does HPBRWD Detect the Slow Worms?

Slow worm has not become public in the internet, but we think in the near future they will come out. A well-designed slow worm should meet such conditions: 1) they are profit-driven, not only to do damage to the Internet. So they have clear and certain victims; 2) to survive longer, they will not choose fast propagation. They will send exploits in much longer interval. Perhaps they will also learn the communication character before sending exploits. Traditional worm detection methods mainly focus on fast worms and have a bad effect on slow worms because they are based on the characters of fast worms or silent worms.

In our work, we detect slow worms through HPBR. When a slow worm is running on the host, it will first be classified into suspicious software. When a *MI* denoted as MI_x sent by the slow worm is captured, $h(MI_x, SW) > 0$ and$1 + \alpha * h(MI_x, SW) + \theta * k(MI_x, T)) > 0$, so according to $Modify\_A$, MI_x will stay in the *MI library*. Only

if HPBRWD is running, whenever the next *MI* from the slow worm will come, the slow worm will be detected.

4.5 What Are the Considerations for Performance in the HPBRWD?

In HPBRWD, there are three points needed to consider the performance: 1) capturing packets; 2) computing ranking functions in HPBR; 3) updating *MI library*. We will discuss the three aspects as follows:

1) Packet capture. In our work, packet capture module not only captures the packets, but also filters the traffic from the normal software. Although there is not as much traffic in one host as that in one network, with the emergence of P2P software, video and file traffic have become more and more. To improve the performance, we use winpcap to capture the header of packets and install a driver to get the map of port and software process. At the same time, we use a memory hash table to store the map of source port and process so that we can reduce the time of lookup.

2) Updating of *MI library*. When implementing $Modify\_A$, it is time-consuming to look for the minimal ranking value in *MI library*. We use a separate memory structure to record the minimal rank value. After a new *MI* is added to A, the minimal rank value and *MI*'s index in the *MI library* are also updated. In this way, we can greatly reduce the compute time cost.

5 Implementation and Evaluations

5.1 Implementation Introduction

To prove the concept of HPBRWD, we have implemented a prototype using Delphi language on windows2003. To capture the packets, we use the famous winpcap (version 4.0) library to monitor the communication to and from the host (with Delphi). To get the relationship between source port and process, we have hooked ntDeviceIoControl, ntResumeThread and other related functions. In our prototype, all the profiles are stored as plain text file, but after the HPBRWD runs, they will be loaded into the memory and modified during runtime. When HPBRWD ends, these profiles data in memory will be saved to text files again. At the same time, in order to carry on experiments easily, we have dumped the filtered traffic for offline analyzing.

To evaluate HPBRWD, we seek to answer the following questions: 1) Is HPBRWD effective in eliminating the worm-like P2P traffic? 2) Is HPBRWD effective in detecting worms?

5.2 Is HPBRWD Effective in Eliminating the Worm-Like P2P Traffic?

Dataset Setting

Table 2 lists the P2P applications used in this experiment. We select three kinds of typical P2P applications which are widely used: P2P instant messenger, P2P file sharing software and P2P video. All the P2P applications are installed before the experiment begins. Table2 also lists the common operations during using these P2P applications. "Idle" means no manual operations; "Send massive message" means sending message

to all the friends in a group at the same time(in this experiment, we sent a message to a group with 10 friends); "send file" means sending files to a friend; in "download file" and "movie playing", we choose the popular songs and TVs recently.

Table 2. P2P applications used

Type of P2P applications	name	Operations tested
P2P instant messenger	QQ	Send massive message; idle; send file;
P2P file sharing software	Bit torrent, Thunder version 5	Idle; download file
P2P video software	UUsee, PPstream	Idle; movie playing

Algorithms Used and Parameter Setting

To make the effect obvious, we implement a simple connection-rate based worm detection algorithm called CRWD (Connection-Rate based Worm Detection). CRWD is similar with the work in [7], but it is implemented on host not on the network. We use the parameter ϵ as the detection threshold. In this experiment, we set $\epsilon = 4$.

For HPBRWD, we set $(\alpha, \beta, \gamma, \theta, \varepsilon, \varphi, \mu) = (1,1,5,5,256,20,80)$. In fact, after computing, this set of parameters means only if two similar MIs exist, an alert will be triggered. We can see that the threshold of HPBRWD is stricter than CRWD. We want to prove whether HPBRWD is effective in eliminating worm-like P2P traffic.

5.2.1 Result

Table 3 list the false positives created by these P2P applications when using HPBRWD and CRWD. From the table, we can see that there exists worm-like P2P traffic which can create false positives and HPBRWD has successfully eliminating the traffic. Table 3 also lists the destination ports of P2P application which create the false positives. We found that UUsee created more of the worm-like P2P traffic. UUsee is a P2P video application which always sends much maintenance information and request information to other peers.

The reasons of why HPBRWD can eliminate these worm-like P2P traffic are: 1) all the P2P applications have been installed before this experiment, so through automatic

Table 3. Result

software	False positives In HPBRWD	False positives In CRWD	Destination ports of P2P application which create False positives
Bit torrent	No	Yes	137,53,6969
ppstream	No	Yes	7202,7201,8000,53,33366
QQ	No	Yes	8001,8002,8003
UUsee	No	Yes	80,443,11111,444,8665,53,8242,8001,9638, 7775,17024,8565,9600
thunder	No	yes	80,53,3077,8000

normal software profile setup and packets captured, we successfully eliminate the worm-like P2P traffic. We can understand the detail process in section 4.3; 2) these P2P applications do not have too much source ports when used, which makes a high performance in hooking ntDeviceIoControl and updating normal software profile.

5.3 How Effective Is HPBRWD in Detecting Computer Worms?

Dataset Setting

In this experiment, we will use computer worms in the wild: codegreen.a, Sasser.a and Webdav.a. The three worms have different propagation speed, which we can see later.

To carry out this experiment, we choose the five famous computer worms. We executed them manually on our own host, which is in an intranet of our own. The corresponding process names listed above are considered not in the normal software profile because we can't get them from the registry which stores the information of installed software.

In this experiment, we set $(\alpha, \beta, \gamma, \theta, \varepsilon, \varphi, \mu) = (1,1,5,3,256,20,20)$.

Result

Table 4 lists the result of detection using HPBRWD:

Table 4. Detection result

	Codegreen.a	Sasser.a	Webdav.a
Release time	22:11:21.000916	22:47:53.350332	22:20:23.619331
Stop time	22:12:10.470961	22:50:03.340721	22:44:29.096500
Total attack number	928	1954	183
Propagation speed (average)	18.8/s	15/s	7.6/min
First alert time	22:11:21.111759	22:47:53.354992	22:20:52.002402
Time spent on first detection	0.110843	0.004660s	28.383071s
False positives number	0	0	0
False negatives numbers	18	59	0
False negatives ratio	1.94%	2.99%	0%

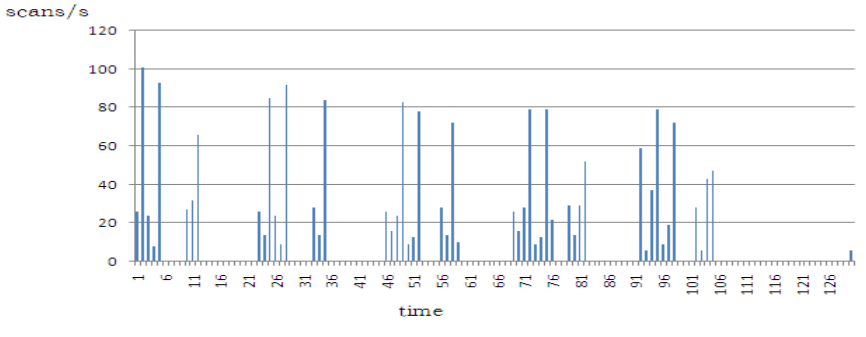

Fig. 6. Speed of sasse.a

We have tested the three worms on win2000 sp4 by manually executing them. Table4 lists the release time and stop time of them. We can see from the table that the three computer worms have different propagation speed. It seems that codegreen.a spread fastest at 18.8/s. Sasser.a was at 5.35/s. But we will find that Sasser.a was detected earliest which only used 0.004660s, while detecting codegreen.a used 0.110843 . The reason is that Sasser.a didn't spread at a stable speed, just as fig. 6.

In fig.6, time means the timestamp at which sasser.a propagated. The "1" time means 22:47:53.350332, while the last time means 22:50:03.340721. We can see that the propagation of sasser.a has the characteristics of periodicity and instability. It even chose to be silent at some timestamps, which will bring challenges to some worm detection algorithm based on fast propagation. The detection result also tells us that HPBRWD is good enough to detect such kind of propagation.

At the same time, we find that HPBRWD can not only detect fast-propagation worms(such as codegreen.a and Sasser.a), but also slow worms. Webdav.a is just such an example. It spead at a speed of 7.6/min and from the detection log, we can see that its speed is stable, which means that it can evade almost all the fast-propagation based worm detection systems. While our HPBRWD is immune to the slow speed and can detect webdav well.

HPBRWD is good at reducing false positives. In the experiment, we can see that false positives are 0. The reason is that the most challenge for behavior-based worm detection is worm-like traffic created by P2P software. We have successfully eliminated them which you can see from the section 5.2. But with the different speed of computer worms, there exists different number of false negatives. It looks like the faster worm speed is the more false negatives HPBRWD has. In our opinion, the reason is that HPBRWD is designed to be a real-time detector, but with worm speed gets faster, the compute cost will also increase, which created some false negatives. This is a problem of both design and implementation. In the future work, we will improve HPBR and its implementation to get a higher performance so as to try to solve this problem.

6 Related Work

Work [3-5] are all automatic worm signatures generation system, but they can't detect unknown worms or polymorphic worms. Work [6] has a try in detecting polymorphic worms, but it cannot detect complex polymorphic and metamorphic worms and it has been proven that polygraph is vulnerable to noise injection [7].

Behavior-based worm detection has been a research hotspot in recent years. Many famous IDS system has added the support for the malicious behavior detection, such as connection rate-based detection [7] and failed connection number based detection [8]. Detection based on netlike association analysis is presented by [9], it analyzes the data similarity when coming out, invalid destination IP and service requests and the similar propagation behavior between hosts. Work [10] collects behavior data from computer and network, and then detects worm through pattern library. In work [11], malicious network behavior is divided to three classes: traffic, responder and connector, and then worm detection is carried out based on these behaviors.

Cliff ChangChun Zou [12] presents Kallman filtering method to evaluate the infection rate of worm. Work [13] [14] studies the frequency of destination IP scanning and source port changing, and then detects worms based on Bayesian analysis and entropy. Recent work [15] has presented a method based on velocity of the number of new connections an infected host makes and control system theory. Work [16] has used both low- and high-interaction honeypot to detect worms. The detection technologies introduced above have a good effect on fast worm detection without many P2P traffic. These technologies are concentrated at the instant or statistics behavior from the network traffic and good detection effect lies in good distinction between normal traffic and malicious traffic. But the emergence of P2P software makes the distinction difficult for the technologies above.

7 Conclusions and Future Work

In this paper, we have presented a hierarchical user habit model. From this model, we found that software and time period is also the important factors to generate network traffic, while they are not paid attention to by traditional user habit model or network behavior based worm detection. An overview of HPBRWD design is also introduced to make the design goal and principles clear. To make the work easy to understand, we introduced several key issues in design and gave our solutions. In this paper, we also explained that host based gave us the advantage to make the worm detection more effective.

At last, several experiments were carried on to test the effect of HPBRWD. Results of the experiments showed that HPBRWD had done a good job.

We plan to optimize the multi-level ranking function in HPBRWD. In the implementation now, it seems a bit complex; for the profile of normal software, in the future work, we want to make the generation more automatic; in section3.1, we have presented an assumption, in the future work, we will solve the problem of injecting into other processes to run the worm code.

Acknowledgement

We would like to thank the High Technology Research and Development Program of China (863 Program) for support of this project with grant 2006AA01Z401 and 2008AA01Z414, we also thank National Natural Science Foundation of China for support of this project with grant 60573136.

References

1. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: Inside the slammer worm. IEEE Security & Privacy l(4), 33–39 (2003)
2. Staniford, S., Moore, D., Paxson, V., Weaver, N.: The top speed of flash worms. In: Paxson, V. (ed.) Proc. of the 2004 ACM Workshop on Rapid Malcode, pp. 33–42. ACM Press, Washington (2004)

3. Kim, H., Karp, B.: Autograph: Toward automated distributed worm signature detection. In: Proceedings of USENIX Security, San Diego,CA (August 2004)
4. Kreibich, C., Crowcroft, J.: Honeycomn-creating intrusion detection signatures using honeypots. In: Proceedings of HotNets, Bostom, MA (November 2003)
5. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of OSDI, San Francisco, CA (December 2004)
6. Newsome, J., Karp, B., Song, D.: Polygraph:Automatically generating signatures for polymorphic worms. In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA (May 2005)
7. Roesch, M.: Snort: Lightweight intrusion detection for networks. In: Proceedings of Conference on system administration (November 1999)
8. Paxson, V.: Bro: a system for detection network intruders in real time. Computer Networks 31 (December 1999)
9. Si-Han, Q., Wei-Ping, W., et al.: A new approach to forecasting Internet worms based on netlike association analysis. Journal On Communications 25(7), 62–70 (2004)
10. Staniford-Chen, S., et al.: GrIDS: A Graph-Based Intrusion Detection System for Large Networks. In: Proceedings of the 19th National Information Systems Security Conference, vol. 1, pp. 361–370 (1996)
11. Dubendorfer, T., Plattner, B.: Host Behavior Based Early Detection of Worm Outbreaks in Internet Backbones. In: Proceedings of 14th IEEE WET ICE/STCA security workshop, pp. 166–171 (2005)
12. Zou, C.C., Gong, W., Towsley, D., et al.: Monitoring and early detection of internet worms[A]. In: Proceedings of the 10th ACM Conference on Computer and Communications Security[C], Washington DC, USA, pp. 190–199. ACM Press, New York (2003)
13. Internet Threat Detection System Using Bayesian Estimation. In: 16th Annul FIRST Conference on Computer Security Incident Handling. 20 Sumeet Singh, Cristian Estanm (2004)
14. Wagner, A., Plattner, B.: Entropy based worm and anomaly detection in fast ip networks. In: WET ICE 2005, pp. 172–177 (2005)
15. Dantu, R., Cangussu, J.W., et al.: Fast worm containment using feedback control. IEEE Transactions On Dependable And Secure Computing 4(2), 119–136 (2007)
16. Portokalidis, G., Bos, H.: SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots. Computer Networks 51(5), 1256–1274 (2007)
17. Xiao, F., Hu, H., et al.: ASG - Automated signature generation for worm-like P2P traffic patterns. In: waim 2008 (2008)