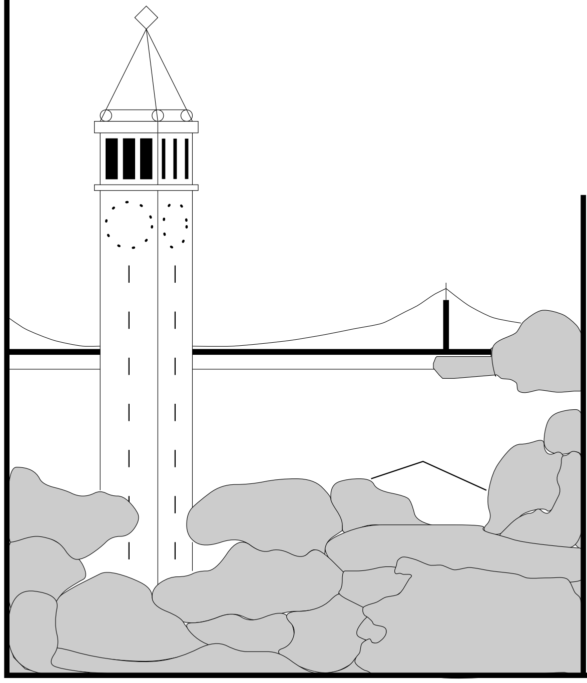
BINDER: An Extrusion-based Break-In Detector for Personal Computers

Weidong Cui[†], *Randy H. Katz*[†], *Wai-tian Tan*[‡]
[†]*Department of Electrical Engineering and Computer Sciences*
University of California, Berkeley, CA
[‡]*Streaming Media Systems Group*
Hewlett-Packard Laboratories, Palo Alto, CA



Report No. UCB/CSD-4-1352

October 2004

Computer Science Division (EECS)
University of California
Berkeley, California 94720

BINDER: An Extrusion-based Break-In Detector for Personal Computers

Weidong Cui[†], Randy H. Katz[†], Wai-tian Tan[‡]
[†]Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA
[‡]Streaming Media Systems Group
Hewlett-Packard Laboratories, Palo Alto, CA

Technical Report UCB/CSD-04-1352

Abstract

In this paper, we tackle the problem of automated detection of break-ins of new unknown threats such as worms, spyware and adware on personal computers. We propose Break-IN DEtectoR (BINDER), a host-based system that detects break-ins by capturing extrusions, stealthy malicious outgoing network traffic sent by them. To capture extrusions, BINDER correlates outgoing network traffic and process information with user activity. This is a unique characteristic of personal computers in contrast to server computers. Since threats tend to run as background processes and thus do not receive any user input, the intuition behind BINDER is that only processes that receive user input are allowed to make connections. We implemented a prototype of BINDER on Windows 2000/XP and evaluated it on 6 computers used by different volunteers for their daily work over 5 weeks. Our results show that BINDER can limit the number of false alarms to at most 5 over 4 weeks on each computer and the false positive rate to less than 0.03%. We also used both real-world and controlled environment to demonstrate Bider's capability for detecting break-ins. We show that BINDER successfully detects all break-ins caused by three adware and four email worms.

1 Introduction

A variety of threats such as worms, spyware and adware, have affected both personal and business computing significantly. Many research efforts [12, 14, 25] and commercial products [17, 33] have focused on preventing break-ins by filtering known exploits or unknown ones exploiting known vulnerabilities. To protect computer systems from new threats, these solutions have two requirements. First, some central entities must rapidly generate signatures of new threats after they are detected. Second, distributed computer systems must download and apply these signatures to their local databases in time. However, these requirements can leave computer systems with obsolete signature database unprotected from newly emerging threats. In particular, worms can propagate much more rapidly than humans can respond in terms of generation and distribution of signatures. An attractive solution is to develop fast mechanisms for detecting break-ins after they occur, but without using pre-defined signatures. Such mechanisms can decrease the danger of information leak and protect computers and networks, and is complementary to existing signature-based schemes.

Many threats send malicious outgoing network traffic either for self-propagation (worms) or for their payloads such as disclosing user information (swore/adware). These malicious network activities usually happen unknown to users on the compromised personal computers. We refer to these *stealthy* malicious outgoing network activities as *extrusions*[1]. The key feature of extrusions is that they are not triggered by user input. In contrast, most normal traffic is initiated by users. Leveraging this anomaly of extrusions, we tackle the problem of automated detection of break-ins of new unknown threats such as worms, spyware and adware on personal computers[2] in contradiction to server computers. In this paper, we present Break-IN DEtectoR (BINDER), a host-based system that detects break-ins on personal computers by capturing extrusions. BINDER can detect certain kinds of break-ins after they occur without priori signatures.

To capture extrusions, BINDER correlates outgoing network connections (initiated by the local computer) and process information with user activities (key strokes and mouse clicks). We do not consider incoming connections (initiated by a remote computer) because most of them are malicious by the nature of personal computers, and firewalls are designed to block such incoming traffic. BINDER classifies outgoing network connections into two categories (*normal* vs. *anomalous*) and treats anomalous outgoing connections as extrusions. Since threats tend to run as background processes and thus do not receive any user input, the intuition behind BINDER is that only processes that receive user input are allowed to make connections.

A key challenge to intrusion detection systems is to avoid false alarms [4]. Since most normal outgoing network traffic from a

[1]Extrusion is also defi ned as unauthorized transfer of digital assets in some other context.

[2]In this paper, a personal computer is a computer that is used locally by a single user at any time.

personal computer is initiated by its user, BINDER uses two simple techniques to solve this problem. By allowing repeated connections to previous hosts, BINDER can handle the case of automatic refreshing web pages which is common for news and sports web sites. BINDER uses whitelisting to cover a small number of system daemons that need unsupervised network access for things like system administration and checking software or security updates.

Given that computers running Windows operating systems are the largest community targeted by malicious attacks, we implement a prototype of BINDER for Windows 2000/XP. BINDER was installed and evaluated on 6 computers used by different volunteers for their daily work over 5 weeks. Our results show that BINDER can limit the number of false alarms to at most 5 over 4 weeks on each computer and the false positive rate to less than 0.03%. We also used both real-world and controlled environment to demonstrate BINDER's capability for detecting break-ins. We show that BINDER successfully detects all break-ins caused by three adware and four email worms.

The remainder of this paper is organized as follows. In Section 2, we explain our goals and assumptions, and present the architecture of BINDER. We describe the extrusion detection algorithm of BINDER in detail in Section 3. We describe the implementation of BINDER in Section 4. We present evaluations and experimental results in Section 5. In Section 6, we discuss open issues of possible countermeasures and solutions. We provide an overview of the related work in Section 7. In Section 8, we conclude the paper and discuss the plan of future work.

2 System Design

We start this section by describing our goals, motivation, and assumptions. Then, we present the architecture of BINDER and discuss the functionality and interface among its components.

2.1 Overview

The goal of our work is to automatically detect break-ins of new unknown exploits on personal computers. Our focus on personal computers is motivated by two observations. First, many threats such as worms, spyware and adware send out malicious network traffic unknown to users after they compromise personal computers. Second, most normal network traffic is initiated by users on personal computers. Leveraging these observations, we develop Break-IN DEtectoR (BINDER), a host-based break-in detection system. BINDER detects break-ins by capturing stealthy outgoing network connections referred to as *extrusions*.

The main objectives we want to achieve for the BINDER design are:

- *Minimal false alarms*: This is the critical base for any intrusion detection system to be useful in practice.
- *Generality*: BINDER should work for a large class of threats without the need for signatures beforehand.
- *Security with open design*: We want to design BINDER so threats cannot bypass it by knowing its scheme.
- *Small overhead*: BINDER must *not* use intrusive probing and affect the performance of the computers it sits on.

Patterns of network traffic and system calls have been used for intrusion detection [6, 8, 9]. To the best of our knowledge, BINDER is the first system to take advantage of a unique characteristic of personal computers: user-driven activities. By trusting the user input, BINDER simply detects break-in extrusions by determining they are unrelated to user actions. In Section 6, we discuss how BINDER can verify a user input is not faked or tricked by break-ins.

A natural approach for BINDER to take is to correlate network traffic with user input directly. However, a "smart" threat can bypass it by monitoring user input and sending traffic at appropriate times. To avoid this, BINDER also uses process information to limit the correlation. The intuition behind it is that only processes that receive user input are allowed to make connections.

2.2 BINDER Architecture

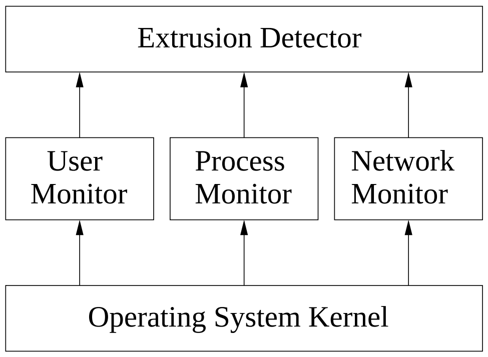


Figure 1: BINDER Architecture

As a host-based break-in detection system, BINDER correlates information across three sources: user input, processes, and network traffic. The BINDER architecture is shown in Figure 1. There are 4 components in a BINDER system: *User Monitor*, *Process Monitor*, *Network Monitor*, and *Extrusion Detector*. The first three components *independently* collect information from the operating system (OS) *passively* in *real time* and report user, process, and network events to the *Extrusion Detector*. APIs for real-time monitoring are specific to every operating system. In Section 4, we describe the implementation on Windows operating system. In the following, we explain the functionality and interface of these components that are general to all operating systems.

The *User Monitor* is responsible for monitoring user input and reporting user events to *Extrusion Detector*. It reports a *user input* event when observing a user clicks mouse or hits a key.

A *user input* event has two components: the time when it happens and the ID of the process that receives this user input. This mapping between a user input and a process is provided by the operating system. So the *User Monitor* do not rely on the *Process Monitor* for process information. Since a *user input* event has only the time information and the *Extrusion Detector* only stores the last *user input* event, BINDER avoids leaking user privacy information.

When a process is created or stopped, the *Process Monitor* correspondingly reports to *Extrusion Detector* two types of process events: *process start* and *process finish*. A *process start* event includes the time, the ID of the process itself, its image file name, and the ID of the parent process. A *process finish* event has only the time and the process ID.

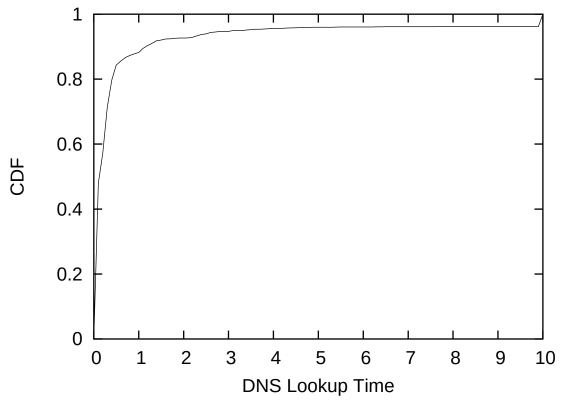


Figure 2: CDF of DNS lookup time on an experimental computer

The *Network Monitor* audits network traffic and reports network events. For the interest of detecting extrusions, it reports three types of network events: *connection request*, *data arrival* and *domain name lookup*. For *connection request* events, the *Network Monitor* checks TCP SYN packets and UDP packets. A *data arrival* event is reported when a TCP or UDP packet with non-empty payload is received. The *Network Monitor* also parses DNS lookup packets. It associates a successful DNS lookup with a following connection request to the same remote IP address as returned in the lookup. This is important because DNS lookup may take significant time between a user input and the corresponding connection request. The Cumulative Distribution Function (CDF) of 2644 DNS lookup times on one of the experimental computers is shown in Figure 2. We can see that about 8% DNS lookups take more than 2 seconds. A *connection request* event has 5 components: the time, the process ID, the local transport port number, the remote IP address and the remote transport port number. Note that the time is the starting time of its DNS lookup if it has any or the connection itself. The mapping between network traffic and processes is provided by the operating system. A *data arrival* event has the same components as a *connection request* event except that its time is the time when the data packet is received. A *domain name lookup* event has the time, the domain name for lookup, and a list of IP addresses mapping to it.

Except for domain name lookup results that are shared among all processes, the *Extrusion Detector* organizes events based on processes and maintains a data record for each process. A process data record has the following members: the process ID, the image file name, the parent process ID, the time of the last *user input* event, the time of the last *data arrival* event, and all the previous normal connections. When a *process start* event is received, a process data record is created with the process ID, the image file name and the parent process ID. The time of the last *user input* event is updated when a *user input* event of the process is reported. Similarly, the time of the last *data arrival* is updated when a *data arrival* event is received. A process data record is closed when its corresponding *process finish* event is received. All process records are cleared up when the system is shutdown. The size of the event database is small because the number of simultaneous processes on a personal computer is usually less than 100. Based on all the information of user, process and network events, the *Extrusion Detector* detects extrusions. The extrusion detection algorithm is presented in the next section.

3 Extrusion Detection

In this section, we describe the extrusion detection algorithm. Instead of searching for conditions that can detect extrusions directly, we look for the cases where normal connections may be generated. This is in concert with our objectives of minimizing false alarms and detecting a large class of threats (see Section 2.1). By covering all normal cases, we can first control false alarms. Then, we can detect any threat that generates connections in a way that does not match any normal case.

We first motivate the design of the detection algorithm using an example. Then, we introduce the algorithm with a focus on how it can detect normal connections correctly. Finally, we discuss what threats can be detected by this algorithm.

3.1 Motivating Examples

To motivate the design of the detection algorithm, we use an example of a user visiting a news web site. Let us assume that the user opens an `Internet Explorer` (`IE`) window, goes to a news web site, then leaves the window idle for answering a phone call. A list of events generated by these user actions is shown in Figure 3. In this example, new connections are triggered in four different ways.

- Case I: When the user opens `IE` by double-clicking its icon on `My Desktop` in Windows, the shell process `explorer.exe` (PID=1664) of Windows receives the user input (Event 1-2), and then starts the `IE` process (Event 3). After the domain name (`www.cs.berkeley.edu`) of the default homepage is resolved (Event 4), the `IE` process makes a connection to it to download the homepage (Event 5). This connection

```
 1 10/01/2004 09:32:33, PID=1664 (user input)
 2 10/01/2004 09:32:33, PID=1664 (user input)
 3 10/01/2004 09:32:35, PID=2573, PPID=1664, NAME="C:\...\iexplore.exe" (process start)
 4 10/01/2004 09:32:39, HOST=www.cs.berkeley.edu, IP=169.229.60.105 (domain name lookup)
 5 10/01/2004 09:32:39, PID=2573, LPORT=5354, RIP=169.229.60.105, RPORT=80 (connection request)
 6 10/01/2004 09:32:40, PID=2573, LPORT=5354, RIP=169.229.60.105, RPORT=80 (data arrival)
 7 10/01/2004 09:32:40, PID=2573, LPORT=5354, RIP=169.229.60.105, RPORT=80 (data arrival)
 8 10/01/2004 09:32:43, PID=2573 (user input)
 9 10/01/2004 09:32:45, HOST=news.yahoo.com, IP=66.218.75.230 (domain name lookup)
10 10/01/2004 09:32:45, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (connection request)
11 10/01/2004 09:32:47, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
12 10/01/2004 09:32:47, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
13 10/01/2004 09:32:48, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
14 10/01/2004 09:32:48, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
15 10/01/2004 09:32:50, HOST=us.ard.yahoo.com, IP=216.136.232.142,216.136.232.143 (domain name lookup)
16 10/01/2004 09:32:50, PID=2573, LPORT=5359, RIP=216.136.232.142, RPORT=80 (connection request)
17 10/01/2004 09:32:51, HOST=us.ent4.yimg.com, IP=192.35.210.205,192.35.210.199 (domain name lookup)
18 10/01/2004 09:32:51, PID=2573, LPORT=5360, RIP=192.35.210.205, RPORT=80 (connection request)
19 10/01/2004 09:32:52, PID=2573, LPORT=5359, RIP=216.136.232.142, RPORT=80 (data arrival)
20 10/01/2004 09:32:52, PID=2573, LPORT=5360, RIP=192.35.210.205, RPORT=80 (data arrival)
21 10/01/2004 09:32:53, PID=2573, LPORT=5359, RIP=216.136.232.142, RPORT=80 (data arrival)
22 10/01/2004 09:32:53, PID=2573, LPORT=5360, RIP=192.35.210.205, RPORT=80 (data arrival)
23 10/01/2004 09:43:01, HOST=news.yahoo.com, IP=66.218.75.230 (domain name lookup)
24 10/01/2004 09:43:01, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (connection request)
25 10/01/2004 09:43:02, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
26 10/01/2004 09:43:02, PID=2573, LPORT=5357, RIP=66.218.75.230, RPORT=80 (data arrival)
   ......
```

Figure 3: An example of events generated by browsing a news web site.

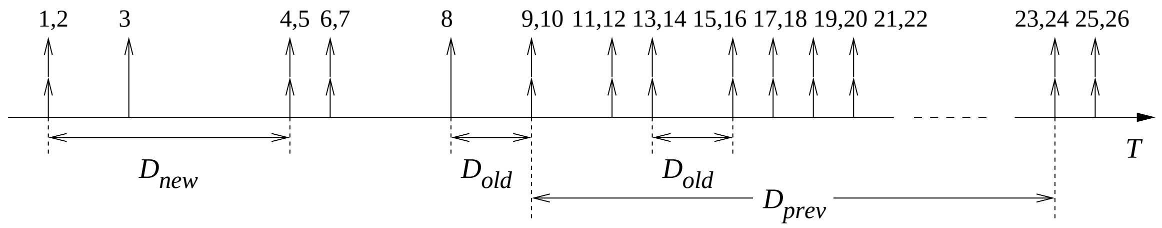


Figure 4: A time diagram of events in Figure 3.

of `IE` is triggered by the *user input* events of its parent process of `explorer.exe`.

- Case II: After the user clicks a bookmark of `news.yahoo.com` in the `IE` window (Event 8), the domain name is resolved as `66.218.75.230` (Event 9). Then the `IE` process makes a connection to it to download the HTML file (Event 10). This connection is triggered by the *user input* event of the same process.

- Case III: After receiving the HTML file in 4 packets (Event 11-14), `IE` goes to retrieve two image files from `us.ard.yahoo.com` and `us.ent4.yimg.com`. Then `IE` makes connections to them (Event 16,18) after the domain names are resolved (Event 15,17). These two connections are triggered by the *data arrival* events of the same process.

- Case IV: According to a setting in the web page, `IE` starts refreshing the web page for updated news 10 minutes later. This connection (Event 24) repeats the previous connection (Event 10).

It is natural for a *user input* or *data arrival* event to trigger a new connection in the same process (Case II and III). However, Case I implies that it is necessary to correlate events among processes. There are a small number of system daemons that need unsupervised network access for things like system administration and software update. In BINDER, we handle these with whitelisting. In general, a normal connection must be triggered by one of the rules below

- *Intra-process* rule: A connection of a process may be triggered by a *user input*, *data arrival* or *connection request* event of the same process.

- *Inter-process* rule: A connection of a process may be triggered by a *user input* or *data arrival* event of another process.
- *Whitelisting* rule: A connection of a process may be triggered according to a rule in the whitelist.

To verify if a connection is triggered by the *intra-process* rule, we just need the single process information stored in the process data record in the *Extrusion Detector*. However, we need to monitor all possible inter-process communications (in a general concept rather than the one used in process programming) to verify if a connection is triggered by the *inter-process* rule. This contradicts our objective of small overhead. Instead, we use the following two rules to approximate it.

- *Parent-process* rule: A connection of a process may be triggered by a *user input* or *data arrival* event received by its parent process before it is created.
- *Web-browser* rule: A connection of a web browser process may be triggered by a *user input* or *data arrival* event of other processes.

The *web-browser* rule cannot be covered by the *parent-process* rule because, when a user clicks a hyperlink in a window of a process, the corresponding web page is loaded by an existing web browser process if there is any. The advantages of this approximation are that the two rules only rely on the process information stored in the *Extrusion Detector* and they cover a dominant fraction of connections triggered by the *inter-process* rule.

Our evaluations in Section 5 show that the *intra-process*, *parent-process* and *web-browser* rule along with whitelisting have a very good coverage (99.97%) of how normal connections may be triggered.

3.2 Detection Algorithm

The detection algorithm is based on the *intra-process*, *parent-process* and *web-browser* rule as well as whitelisting. Since whitelisting is OS dependent, we discuss the whitelist of our Windows implementation in Section 4. The main idea of the algorithm is to limit the delay from a triggering event to a *connection request* event. There are three possible delays for a *connection request* though some of them may not exist. In Figure 4, we show them in a time diagram of events in Figure 3. For a *connection request* made by process P, we define the three delays as follows

- D_{new}: The delay since the last *user input* or *data arrival* event received by the parent process of P before P is created. It is the delay from Event 1 to 5 in Figure 4.
- D_{old}: The delay since the last *user input* or *data arrival* event received by P. In Figure 4, it is the delay from Event 8 to 10, from Event 14 to 16, and from Event 14 to 18. Note that the triggering event can be from any process if P is a web browser according to the *web browser* rule.
- D_{prev}: The delay since the last *connection request* to the same host or IP address made by P. It is the delay from Event 10 to 24 in Figure 4.

As D_{old} is the reaction time of a process, D_{new} includes the loading time of a process as well. For normal connections, D_{old} and D_{new} are in the order of seconds while D_{prev} is in the order of minutes. Depending on how a normal connection is triggered, it must have at least one of the three delays fall into a normal range. This is the basic idea behind the detection algorithm. The extrusion detection algorithm needs 3 parameters for the upper bound of the delays (defined as D_{new}^{upper}, D_{old}^{upper}, and D_{prev}^{upper}).

In the design of the extrusion detection algorithm, we assume the *Extrusion Detector* can learn rules from previous false alarms. Each rule includes an application name (the image file name of a process) and a remote host name. The rule means any connection to the host made by a process of the application is always normal.

Given a *connection request*, the detection algorithm works as follows:

- If it is in the rule set of previous false alarms, then it is normal;
- If it is in the whitelist, then it is normal;
- If D_{prev} exists and is less than D_{prev}^{upper}, then it is normal;
- If D_{new} exists and is less than D_{new}^{upper}, then it is normal;
- If D_{old} exists and is less than D_{old}^{upper}, then it is normal;
- Otherwise, it is anomalous.

The order of comparison is chosen for optimizing parameters to decrease false negatives. We check the rules of previous false alarms and whitelist first because they are used to minimize false positives. We check D_{prev} before D_{new} and D_{old} because it is larger than the latter two in orders and has less impact on false negatives. Since D_{old}^{upper} controls if a connection is normal after a process is created, it has more impact on false negatives. By checking D_{old} in the end, we can avoid the cases that have large D_{old} values but meet one of the first four conditions. Therefore, we can choose D_{old}^{upper} with a smaller value without increasing false alarms.

After detecting an extrusion, the *Extrusion Detector* can raise an alarm with related information such as the process ID, the image file name, and the connection information. Further possible actions responding to threats are discussed for future work in Section 8.

3.3 Detecting Threats

We have focused on designing the extrusion detection algorithm to detect normal connections correctly. In this section, we discuss why connections made by a large class of threats can be detected as extrusions by the algorithm.

The goal of our work is to detect worms, spyware and adware on personal computers. Unlike worms, spyware and adware cannot propagate themselves and thus require user input to infect a computer system. Worms can be classified as self-activated worms such as Blaster [21] and user-activated worms such as email worms. While break-ins of user-activated worms must have user input, those of self-activated worms must have a vulnerable process receive malicious data. Most threats also infect a system in such a way that the malicious processes will be started automatically when the system is restarted. Therefore, when a threat breaks into a personal computer, the break-in can be split into two phases by the time of the first restart of the victim computer. The difference of these two phases is how malicious processes of a threat are started. In the first phase, malicious processes are started either by an existing infected process or by a user accidentally with user input or data arrival in history. In the second phase, malicious processes are started by the operating system without any user input or data arrival in history. Moreover, threats tend to run as background processes to avoid being detected or shutdown by computer users. A feature of background processes is that they do not receive any user input.

In the second phase of a break-in, BINDER can detect the break-in by capturing the first connection made by its malicious processes as an extrusion. This is because their parent processes do not receive any user input before they are created. And they do not receive any user input after they are created. Therefore, D_{old}, D_{new} and D_{prev} do not exist for such a connection. Thus BINDER can be guaranteed to detect break-ins of worms, spyware and adware after the victim computer is restarted. We will prove this by showing that BINDER successfully detects three adware in a real-world environment and some well-known email worms in a controlled environment.

In the first phase of a break-in, connections made by malicious processes may not be initially detected as extrusions. This is because, before they are started, their parent processes may have received user input (e.g., a user opens a virus attachment in an email client program) or data arrival (e.g., a vulnerable process receives malicious traffic). However, BINDER can detect a break-in by observing even a single extrusion it makes. In Section 5, we will show that BINDER successfully detects the adware Spydeleter [3] and three email worms Beagle [20], NetSky [23] and Swen [24] in the first phase of their break-ins.

In Section 6, we discuss possible countermeasures a break-in can take to bypass BINDER in both phases. For each countermeasure, we also discuss possible containment approaches.

4 System Implementation

We implement a prototype of BINDER for Windows 2000/XP. This is because computers running Windows operating systems are the largest group attacked by the most threats [16]. The prototype demonstrates the feasibility of BINDER and enables us to evaluate it with real users in real-world environment. Though this prototype is implemented in the application space, we assume a BINDER system runs in the kernel space if it is adopted in practice. In this section, we first present the implementation of the *User Monitor*, *Process Monitor* and *Network Monitor*. Then, we describe the whitelist used in the *Extrusion Detector*.

4.1 User Monitor

The function of the *User Monitor* is to report *user input* events including the time and process ID. The *User Monitor* is implemented based on Windows Hooks API [29]. It uses three hook procedures, `KeyboardProc`, `MouseProc` and `CBTProc`. `KeyboardProc` is used to capture keyboard events while `MouseProc` is used to capture mouse events. `MouseProc` can provide the information of which window will receive a mouse event. Since `KeyboardProc` cannot provide the same information for a keyboard event, we use `CBTProc` to capture events when a window is about to receive the keyboard focus. After determining which window will receive a *user input* event, the *User Monitor* uses procedure `GetWindowThreadProcessId` to get the process ID of the window.

4.2 Process Monitor

The objective of the *Process Monitor* is to report *process start* and *process finish* events. The *Process Monitor* is implemented based on the built-in Security Auditing on Windows 2000/XP [30]. By turning on the local security policy of auditing process tracking (`Computer Configuration/Windows Settings/Security Settings/Local Policies/Audit Policy/Audit process tracking`), the Windows operating system can audit detailed tracking information for process start and finish events. The *Process Monitor* uses `psloglist` [13] to parse the security event log and generates *process start* and *process finish* events.

4.3 Network Monitor

The *Network Monitor* aims to report *connection request*, *data arrival* and *domain name lookup* events. It is built on `TDIMon` [18] and `WinDump` [31] which requires `WinPcap` [32]. `TDIMon` monitors activity at the Transport Driver Interface (TDI) level of networking operations in the operating system kernel. It can capture all network events associated with process information. Since `TDIMon` does not capture complete DNS packets, The *Network Monitor* uses `WinDump` for this purpose. Based on the information collected by `TDIMon` and DNS packets captured by `WinDump`, the *Network Monitor* reports network events to the *Extrusion Detector*.

4.4 Extrusion Detector

The detection algorithm of the *Extrusion Detector* was described in Section 3. Here we focus on the whitelist mechanism in our Windows implementation. The whitelist in our current implementation has 15 rules. These rules can be classified into three categories: system daemons, software updates and network applications automatically started by Windows. A rule for system daemons has only a program name. Processes of the program are allowed to make connections at any time. In our current implementation, we have 5 system daemons including `System`, `spoolsv.exe`, `svchost.exe`, `services.exe` and `lsass.exe`. A rule for software updates has both a program name and an update web site. Processes of the program are allowed to connect to the update web site at any time. In this category, we now have 6 rules that covers Symantec, Sygate, ZoneAlarm, Real Player, Microsoft Office, and Mozilla. For network applications automatically started by Windows when it starts, we currently have 4 rules for messenger programs of MSN, Yahoo!, AOL, and ICQ. These programs are allowed to make connections at any time.

If BINDER is deployed, managing the whitelist for an average user is very important. Rules for system daemons usually do not change until the operating systems are upgraded. Since the number of softwares that require regular updates is small and do not change very often, the rules for software updates can be updated by some central entity adopting BINDER. Though rules in the last category have to be configured individually for each system, we believe some central entity can provide help by maintaining a list of applications that fall into this category. A mechanism similar to PeerPressure [26] may be used to help an average user configure her own whitelist.

5 Evaluations

We evaluated BINDER in two environments. First, we installed it on 6 Windows computers used by different volunteers for their daily work, and collected traces over 5 weeks since September 7th, 2004. Second, we tested BINDER on a sample set of well-known email worms in a controlled testbed using VMWare [19].

5.1 Methodology

To evaluate BINDER with different values for the three parameters D_{old}^{upper}, D_{new}^{upper} and D_{prev}^{upper}, we used offline, trace-based experiments. We collected traces of user input, process information, and network traffic from the 6 computers that installed the prototype of BINDER. A summary of the collected trace is shown in Table 1. On one hand, these computers were used for daily work, so the traces are real-world. On the other hand, our experimental population is small due to the difficulty of getting user commitment for a long term study on their daily work computers. However, from the summary of the collected traces in Table 1, we can see that they have good diversity with respect to hardware, operating system, and user behavior.

The most important design objective of BINDER is to minimize false alarms while maximizing detected extrusions. In our experiments, we used the number of false alarms rather than the false positive rate to evaluate BINDER. This is because users who respond to alarms are more sensitive to the absolute number than a relative rate. When BINDER detects extrusions, it is based on connections. However, when we count the number of false alarms, we do not use the number of misclassified normal connections directly. This is because a false alarm covers a series of consecutive connection requests. Therefore, for misclassified normal connections, we split them into groups and count each group as one false alarm.

5.2 Parameter Selection

In this section, we discuss how to choose values for the three parameters D_{old}^{upper}, D_{new}^{upper} and D_{prev}^{upper}. The goal of parameter selection is to make the parameters as small as possible under the condition that the number of false alarms is acceptable. We assume the rules of whitelisting described in Section 4.4 are fixed. The performance metric is the number of false alarms. Based on the real-world traces, we calculate D_{old}, D_{new} and D_{prev} for all *connection request* events for every user. Then we take the 90th, 95th and 99th percentile for all three parameters and calculate the number of false alarms for each percentile. The results are shown in Table 2.

From Table 2 we can see that D_{old}^{upper}, D_{new}^{upper} and D_{prev}^{upper} must be different for different users because they are dependent on computer speed and user pattern. Thus, they should be selected on a per-user base. We can also see that the performance of taking the 90th percentile is not acceptable and the improvement from taking the 95th percentile to the 99th percentile is small. Therefore, the parameters can be selected by choosing some value in the 95th percentile according to user's preference. For conservative users, we should choose smaller values. The percentiles can be obtained by training BINDER over a period of virus-free time. Without training, these parameters can also be chosen based on user's preference. D_{old}^{upper} and D_{new}^{upper} can take values between 30 and 60 seconds, while D_{prev}^{upper} can take values between 1200 and 3600 seconds.

In the real-world experiments of detecting break-ins, we take the 95th percentile values for the two infected computers. In the controlled experiments, We take 30 seconds, 30 seconds and 800 seconds for D_{old}^{upper}, D_{new}^{upper} and D_{prev}^{upper}, respectively. Note that D_{old}^{upper} can be greater than D_{new}^{upper} because the reaction time from a *user input* or *data arrival* event to a *connection request* event is dependent on the instant running condition of a computer.

5.3 False Alarms

By choosing parameters correctly, we expect to achieve minimal false alarms. From Table 2 we can see that there are at most 5 false alarms for each computer by choosing the 99th percentile. The false positive rate is 0.03%. We manually check these remaining false alarms and find that they are caused by one of the three reasons:

Table 1: Summary of Collected Traces

User	Machine	OS	Trace (Days)	# of User Events	# of Process Events	# of Network Apps	# of TCP Conns
A	Desktop	WinXP	27	35270	5048	33	33480
B	Desktop	WinXP	26	80497	12502	35	15450
C	Desktop	WinXP	23	24781	7487	55	36077
D	Laptop	Win2K	23	99928	8345	28	9784
E	Laptop	WinXP	13	8630	2448	21	10210
F	Laptop	WinXP	12	20490	5402	20	7592

Table 2: Parameter selection for D_{old}, D_{new} and D_{prev}

User	90% (s)				95% (s)				99% (s)			
User	D_{old}	D_{new}	D_{prev}	# of FAs	D_{old}	D_{new}	D_{prev}	# of FAs	D_{old}	D_{new}	D_{prev}	# of FAs
A	18	11	142	15	33	15	752	5	79	21	4973	3
B	15	12	64	23	28	21	260	7	79	22	3329	5
C	14	14	28	20	25	15	134	5	74	33	2272	1
D	16	81	213	3	33	81	715	3	85	81	4611	2
E	19	12	539	5	32	14	541	4	93	90	4216	3
F	14	8	80	10	27	13	265	5	79	31	3633	2

Table 3: Break-down of false alarms according to causes.

User	Inter-Process	Whitelist	Collection	Total
A	2	1	0	3
B	4	1	0	5
C	1	0	0	1
D	0	1	1	2
E	1	1	1	3
F	0	1	1	2

- Incomplete information of inter-process event sharing. 4 of the 5 false alarms of User B are caused by this. We observe that `PowerPoint` follows `IE` to connect to the same IP address while the parent process of the `PowerPoint` process is the windows shell. We hypothesize this is due to the usage of Windows API `ShellExecute`.

- Incomplete whitelisting. For example, connections made by Windows Application Layer Gateway Service are treated as extrusions. This is easy to fix if a BINDER system is well engineered.

- Incomplete trace collection. BINDER was accidentally turned off in the middle of trace collection.

A break-down of the false alarms is shown in Table 3. We can see that BINDER can achieve minimal false alarms if it is well engineered and executed.

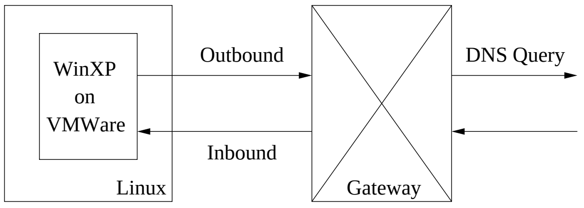


Figure 5: A controlled testbed for email worm detection

5.4 Controlled Testbed

To test BINDER with a large class of *real* threats, we set up a controlled testbed. It is very difficult to run this kind of experiments. We overcame several obstacles to make it possible.

We were targeting at email worms because they are a big threat to personal computers. However, we found it is very difficult to obtain real email worms. First, we were not successful to use Google to find virus email attachments. Second, the email server in our institution is managed very well. So we cannot get virus emails from our own email accounts. We finally used several channels to get virus emails. We set up our own mail server and published an email address to Usenet groups. This helped us get the email worm Swen [24] which exploits the email addresses published in Usenet groups. We also asked for help from other colleagues in the research community. A colleague forwarded a Beagle [20] virus email to us. In the end, we were lucky to convince the administrators in our institution to give us more than 1000 unique virus email attachments they captured over a week. We are in the process of taking an extensive experiments on these $1000+$ virus email attachments.

For every email worm, we need a clean Window operating system to test with. It would be very slow if we had to reinstall Windows for every test. We used VMWare [19] to avoid this problem. The advantage of using VMWare is that we can discard an infected system and get a new one just by copying a few files.

When we tested with real email worms, we need to be extremely careful about containing them. However, if the testbed is completely isolated from the Internet, email worms will not be able to make connection requests due to failures of domain name lookup. To solve this problem, we allowed DNS traffic to get through the gateway. However, we can only test if BINDER can detect the first connection made by a break-in email worm as an extrusion because its behavior is unpredictable after it due to the fact that it can not connect to the outside.

The testbed is shown in Figure 5. It consists of a Linux computer and a network gateway. We run Windows XP on VMWare [19] that runs on the Linux computer. The network gateway is used to block outgoing traffic except for DNS traffic.

5.5 Detecting Break-Ins

We have shown that BINDER can limit false alarms very well. Here we discuss how BINDER detects break-ins. We use experimental results from both the real-world and controlled environment.

In the real-world experiments, among the 6 computers, one is infected by adware Gator [2] and CNSMIN [1] and another one is infected by adware Gator and Spydeleter [3]. In particular, the second computer was compromised by Spydeleter after BINDER was installed.

In a controlled environment, we test BINDER on a sample set of well-known email worms including Beagle [20], NetSky [23], Mydoom [22], and Swen [24]. in a controlled environment.

As we discussed in Section 3.3, BINDER can be guaranteed to detect break-ins after a victim computer is restarted but may not detect those whose behavior is similar to normal processes right after their break-ins. In both the real-world and controlled environment, BINDER successfully detected break-ins of all three adware and four email worms after the victim computer was restarted. In the rest of this section, we focus on the problem of detecting break-ins right after they happen.

For a connection request to be treated as normal, one of the three delays D_{old}, D_{new} and D_{prev} must exist and is less than the upper bound. Due to our limited access to real malicious code, we cannot prove what percentage of today's threats violate these conditions during a computer break-in. In the following, we show that BINDER can detect the break-ins of the adware Spydeleter in the real-world environment and the email worm Beagle, NetSky and Swen in the controlled environment.

In Figure 6, we show a stripped list of events logged during the break-in of the adware Spydeleter. Note that all IP addresses are anonymized. Two related processes not shown in the list are a process of `explorer.exe` with PID 240 and a process of `svchost.exe` with PID 960. After `IE` is opened, a user connects to a site with IP $12.34.56.78$. The web page has code to exploit a vulnerability in `mshta.exe` which processes `.HTA` files. After `.mshta.exe` is infected by the malicious `.HTA` file it downloads from $87.65.43.21$, it starts a series of processes of `ntvdm.exe` which provides an environment for a 16-bit process to execute on a 32-bit platform. Then, a process of `ntvdm.exe` starts a process of `ftp.exe` which makes a connection request to $44.33.22.11$.

Since the prototype of BINDER does not have complete information for verifying if a connection is triggered according to the *inter-process* rule (see Section 3), the connection made by `mshta.exe` is detected as an extrusion. This is because its parent process is `svchost.exe` rather than `iexplore.exe`, though it is the latter process that triggers its creation. If BINDER had complete information for inter-process event sharing, it detects the connection request made by `ftp.exe` as an extrusion. This is because both the process of `ftp.exe` and its parent process of `ntvdm.exe` does not have any *user input* or *data arrival* event in history. So all of D_{old}, D_{new} and D_{prev} do not exist. The connection made by `ftp.exe` is used to download malicious code. Therefore, BINDER's detection plus some appropriate actions could have stopped the adware from infecting the computer.

As we discussed in the previous section, we can only test if BINDER can detect the first connection made by a break-in email worm as an extrusion because its behavior is unpredictable after it. Among the four email worms we test, BINDER successfully detects the first connection request made by Beagle, NetSky and Swen but not MyDoom.

- Beagle: The real worm code `bawindo.exe` is embedded in the attachment of `joke.com`. When the attachment is opened, `joke.com` is executed with email client `Outlook Express` as its parent process. Then `joke.com` executes `bawindo.exe`. When the process of `bawindo.exe` makes a connection request, BINDER detects it as an extrusion. This is because both this process and its parent process do not receive any *user input* or *data arrival* events, though its grand-parent process of `Outlook Express` receives user input when the attachment is opened.

- Swen: Similar to the Beagle case, the worm code is embedded in the attachment. So it is detected by BINDER.

- NetSky: The worm code itself is the attachment. After the attachment is opened, the worm code is executed. However, it makes its first connection 90 seconds later. So D_{new} of this connection request is 90 seconds, which is larger than D_{new}^{upper} as 30 seconds.

- MyDoom: The worm code itself is the attachment. After the attachment is opened by a user, the malicious process makes its first connection request immediately. So its

```
 1 10/02/2004 14:40:10, PID=2368, PPID=240, NAME="C:\...\iexplore.exe" (process start)
 2 10/02/2004 14:40:15, PID=2368 (user input)
 3 10/02/2004 14:40:24, PID=2368 (user input)
 4 10/02/2004 14:40:24, PID=2368, LPORT=1054, RIP=12.34.56.78, RPORT=80 (connection request)
 5 10/02/2004 14:40:24, PID=2368, LPORT=1054, RIP=12.34.56.78, RPORT=80 (data arrival)
   ......
 6 10/02/2004 14:40:28, PID=2552, PPID=960, NAME="C:\...\mshta.exe" (process start)
 7 10/02/2004 14:40:29, PID=2552, LPORT=1066, RIP=87.65.43.21, RPORT=80 (connection request)
 7 10/02/2004 14:40:29, PID=2552, LPORT=1066, RIP=87.65.43.21, RPORT=80 (data arrival)
   ......
 8 10/02/2004 14:40:34, PID=2896, PPID=2552, NAME="C:\...\ntvdm.exe" (process start)
 9 10/02/2004 14:40:35, PID=2988, PPID=2896, NAME="C:\...\ ftp.exe" (process start)
10 10/02/2004 14:40:35, PID=2988, LPORT=1068, RIP=44.33.22.11, RPORT=21 (connection request)
   ......
```

Figure 6: A stripped list of events logged during the break-in of adware Spydeleter.

D_{new} is very small. BINDER treats it as a normal connection. This happens because the user herself executes the code. So BINDER cannot tell if it is the user's attention to run it and let it make network connections. However, BINDER may detect it afterwords. In the next section, we discuss possible solutions for extending BINDER to handle this problem.

6 Attacking BINDER

We have shown that BINDER can detect break-ins of existing worms, spyware and adware if they send malicious outgoing traffic unknown to users. We devote this section to discussions of potential countermeasures against BINDER if its scheme is known to attackers. We focus on malicious false negatives which allow break-ins to send malicious traffic without being detected. For each countermeasure, we will also discuss possible containment approaches. Though we try to investigate all possible attacks against BINDER, we cannot argue that we have considered all possible vulnerabilities.

- Direct attacks: As a host-based system targeting at detecting break-ins after they have occurred, BINDER faces the danger of being stopped by break-ins. However, we expect a BINDER system from prototype at application level to deployed service within the kernel. Thus deactivation of BINDER itself is a clear sign of break-ins.

- Faking user input: BINDER is built based on the assumption that user input can be trusted. A possible countermeasure is to fake user input by using APIs provided by the operating system. For example, the `SendInput` function in Windows synthesizes keystrokes, mouse motions, and button clicks. A possible solution is to monitor the related APIs and ignore or treat user input events generated by them differently. A tool called APISpy32 [7] provides evidence for the feasibility of monitoring system APIs. If a trustworthy computing base [10] is deployed, it becomes easier to differentiate hard and soft user input events.

- Tricking the user to generate input: Another way for break-ins to obtain user input events is to trick a user to click on its window. This kind of pop-up windows are similar to extrusions in the sense that they are not triggered by user input! We can apply the same idea of detecting extrusions to detect pop-ups. If a process opens its first window without receiving any user input or its parent process receiving any user input before it is created, then it is a pop-up window. Any user input in this window will be ignored for triggering network connections.

- Hiding under processes: In modern operating systems, processes become more complex. Many tasks are executed at the thread level. A possible countermeasure is to let a break-in install itself as a DLL library file, which can be automatically loaded by Windows shell process `explorer.exe`. Under the current design of BINDER, this malicious thread can make connections whenever the process of `explorer.exe` receives user input. A possible solution is to extend BINDER to monitor activities on the thread level. In fact, every window in Windows operating system is associated with a process and a thread. Thus a malicious thread in a process cannot make connections when another thread in the process receives user input.

- Covert Channels: A very tricky countermeasure is to have a legitimate process or thread make connections and use it as a covert channel to leak information. For example, spyware can have an existing `IE` process or thread download a web page of a tweaked hyperlink by using some API provided by Windows shell right after a user clicks on the `IE` window of the same process or thread. A collusive remote server can get private information from the tweaked hyperlink. A possible solution to this problem is to monitor related APIs in such a way that we can tell the tweaked hyperlink was provided by a malicious process or thread.

- Using a user input to hide: During a break-in, if the malicious process is started by a user (as the case of MyDoom), it can use a countermeasure to bypass BINDER before the victim computer is restarted. Though the malicious process may not receive any user input after it is created, it just needs to maintain a connection to a collusive remote site to keep receiving some data packets regularly. Then BINDER would think any new connections made by it are triggered by those data packets. A possible solution is not to allow a process to make new connections if it has not received any user input for some period of time. We are now studying how long that period should be based on our real-world traces.

We believe, by extending the current design of BINDER, we can avoid the countermeasures above. We plan to study the following solutions:

- Restrict conditions on how a normal connection may be triggered;
- Extend the *Process Monitor* to *Thread Monitor*;
- Apply the idea of extrusion detection to pop-up window detection;
- Add a *System API Monitor* to BINDER.

7 Related Work

Many research efforts [12, 14, 25] and commercial products [17, 33] have focused on preventing break-ins by filtering known exploits or unknown ones exploiting known vulnerabilities. Bro [12] and Snort [14] are both Network-based Intrusion Detection Systems, which monitor network traffic and drop malicious traffic of known exploits. Shield [25] is a solution for preventing vulnerability-specific, exploit-generic attacks by using network filters built on known vulnerabilities. Commercial security products as Symantec Norton AntiVirus [17] and ZoneAlarm [33] are Host-based Intrusion Detections Systems. They monitor not only network traffic but also applications and file system on the monitored host. Complementary to these solutions, BINDER targets at detecting break-ins of unknown new exploits before their signatures are widely distributed.

ZoneAlarm's Program Control provides a static control over which program is allowed to send outgoing traffic. It requires users to construct a complete list of legitimate network programs, which is beyond the capability of an average user. In contrast, BINDER uses whitelisting to cover exceptions only. Most rules used in BINDER can be decided before it is installed.

Anomaly-based intrusion detection have been studied for detecting unknown exploits. [6] uses short sequences of system calls executed by running processes to detect anomalies caused by intrusions. [8] proposes a data mining framework for constructing feature and training models of network traffic for intrusion detection. Recent work of [9] correlates simultaneous anomalous behaviors from different perspectives to improve the accuracy of intrusion detection. The performance of anomaly-based approaches is very limited in practice due to its high false positive rate. BINDER leverages the observation that most normal traffic is triggered by users to achieve minimal false alarms.

In the past few years, computer worms have been a big threat to both personal computing [21] and large networks [11]. Fast worm detection and containment becomes critical since worms can propagate much more rapidly than human response [15]. Most research efforts have focused on random scanning worms. [28] is a rate-limiting solution that throttles the rate of new connections made by a compromised host. [27] proposes a solution to contain random scanning worms in an enterprise environment at very high speed by leveraging hardware acceleration. BINDER can detect a large class of worms that infect personal computers.

[5] uses extrusion detection to stop spams. They leverage the anomaly that spam relayed by a compromise system is unknown to users and often sent to non-existing email addresses.

8 Conclusions and Future Work

In this paper, we propose Break-IN DEtectoR (BINDER), a host-based system that detects break-ins of worms, spyware and adware on personal computers by capturing extrusions. The main contributions of this paper are:

- BINDER takes advantage of a unique characteristic of personal computers: user-driven activities. By trusting the user input, BINDER simply detects break-in extrusions by determining they are unrelated to user actions. This implies a new direction for tackling the problem of intrusion detection on personal computers.
- We have shown that BINDER limits false alarms very well and can detect a large class of threats. BINDER has very small overhead since it monitors the system passively for selective events and the size of its event database is in the order of the number of simultaneous processes.
- We tested BINDER on a sample set of real email worms on a controlled testbed. Our experience on conducting this kind of experiments is useful for other researchers.

In the future, we plan to extend the design of BINDER to minimize malicious false negatives. We also plan to study the advantage of sharing extrusion information among distributed BINDER systems. Finally, we want to set up a formal model for the user, process and network events and use it to solve the problem of detecting break-ins with user input in history.

Acknowledgments

We are grateful to Minghua Chen, Yanmei Li, Mukund Seshadri, Rui Xu, Fang Yu, Haibo Zeng, Jianhui Zhang and Wei Zheng for their generous help of allowing us to install and evaluate BINDER on their computers. We are thankful to Dan Ellis, Jaeyeon Jung, Jon Kuroda and Zhenmin Li for sharing virus emails with us. We would like to thank Chris Karlof, Yaping Li and Zhi-Li Zhang for their insightful comments on a draft of this paper. Special thanks go to David Wagner and Li Yin for their helpful discussion and valuable feedback.

References

[1] Adware.CNSMIN, http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=ADW_CNSMIN.A.

[2] Adware.Gator, http://securityresponse.symantec.com/avcenter/venc/data/adware.gator.html.

[3] Adware.Spydeleter, http://netrn.net/spywareblog/archives/2004/03/12/how-to-get-rid-of-spy-deleter/.

[4] S. Axelsson, "The base-rate fallacy and its implications for the difficulty of intrusion detection," in *Proceedings of the 6th ACM Conference on Computer and Communication Security*, 1999.

[5] R. Clayton, "Stopping spam by extrusion detection," in *Proceedings of the First Conference on Email and Anti-Spam*, Mountain View, CA, USA, July 2004.

[6] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[7] Y. Kaplan, "API spying techniques for Windows 9X, NT and 2000," http://www.internals.com/articles/apispy/apispy.htm.

[8] W. Lee and S. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security*, vol. 3, no. 4, November 2000.

[9] Z. Li, J. Taylor, E. Partridge, Y. Zhou, W. Yurcik, C. Abad, J. J. Barlow, and J. Rosendale, "Uclog: A unified, correlated logging architecture for intrusion detection," in *the 12th International Conference on Telecommunication Systems - Modeling and Analysis (ICTSM)*, 2004.

[10] Microsoft, "Next-generation secury computing base," http://www.microsoft.com/resources/ngscb/default.mspx.

[11] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," August 2003.

[12] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[13] PsTools, http://www.sysinternals.com/ntw2k/freeware/pstools.shtml.

[14] Snort, The Open Source Network Intrusion Detection System, http://www.snort.org/.

[15] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *Proceedings of the 11th Usenix Security Symposium*, August 2002.

[16] Symantec Internet Security Threat Report, http://enterprisesecurity.symantec.com/content.cfm?articleid=1539, September 2004.

[17] Symantec Norton Antivirus, http://www.symantec.com/.

[18] TDIMon, http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml.

[19] VMWare Inc., http://www.vmware.com/.

[20] W32.Beagle.AR@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.beagle.ar@mm.html.

[21] W32.Blaster.Worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html.

[22] W32.MyDoom.V@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.v@mm.html.

[23] W32.NetSky.P@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.netsky.p@mm.html.

[24] W32.Swen.A@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.swen.a@mm.html.

[25] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier, "Shield: Vulnerability-driven network filters for preventing known vulnerability exploits," in *Proceedings of ACM SIGCOMM*, August 2004.

[26] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Usenix OSDI*, San Francisco, CA, December 2004.

[27] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *Proceedings of the 13th Usenix Security Symposium*, August 2004.

[28] M. M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," HP Labs Bristol, Tech. Rep. Technical Report HPL-2002-172, 2002.

[29] Windows Hooks API, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/hooks.asp.

[30] Windows Security Auditing, http://www.microsoft.com/technet/security/prodtech/win2000/secwin2k/09detect.mspx.

[31] WinDump, http://windump.polito.it/.

[32] WinPcap, http://winpcap.polito.it/.

[33] ZoneAlarm, http://www.zonelabs.com/.