Design and Implementation of an Extrusion-based Break-In Detector for Personal Computers

Weidong Cui, Randy H. Katz, Wai-tian Tan
University of California, Berkeley and Hewlett-Packard Laboratories
{wdc,randy}@cs.berkeley.edu, dtan@hpl.hp.com

Abstract

An increasing variety of malware, such as worms, spyware and adware, threatens both personal and business computing. Remotely controlled bot networks of compromised systems are growing quickly. In this paper, we tackle the problem of automated detection of break-ins caused by unknown malware targeting personal computers. We develop a host based system, BINDER (Break-IN DEtectoR), to detect break-ins by capturing user unintended *malicious outbound connections (referred to as* extrusions*). To infer user intent, BINDER correlates outbound connections with user-driven input at the process level under the assumption that user intent is implied by user-driven input. Thus BINDER can detect a large class of unknown malware such as worms, spyware and adware* without *requiring signatures. We have successfully used BINDER to detect real world spyware on daily used computers and email worms on a controlled testbed with very small false positives.*

1 Introduction

An increasing variety of malware like worms, spyware and adware threatens both personal and business computing. Remotely controlled bot networks of compromised systems are growing quickly [28]. Many research efforts [16, 21, 30] and commercial products [25, 37] prevent break-ins by filtering either known malware or unknown malware exploiting known vulnerabilities. To protect computer systems from rapidly evolving malware, these solutions have two requirements. First, a central entity must rapidly generate signatures of new malware after it is detected. Second, distributed computer systems must download and apply these signatures to their local databases before they are attacked. However, these can leave computer systems temporarily unprotected from newly emerging malware. In particular, worms can propagate much more rapidly than humans can respond in terms of generation and distribution of signatures [22]. In this paper, we take a different approach, focusing on fast automated mechanisms for detecting break-ins of new unknown malware after a break-in occurs, as a way of mitigating damage. As a complement to existing prevention schemes, our approach decreases the danger of information leak and protects computers and networks from more severe damage.

A large class of malware makes malicious outbound network connections either for self-propagation (worms) or to disclose user information (spyware/adware). Our key observation is that outbound network connections from a compromised personal computer (used locally by a single user at any time) can be classified into three categories: user intended, user unintended benign, user unintended malicious (referred to as *extrusions*. Extrusion is also defined as unauthorized transfer of digital assets in some other context.) In this paper, we present the architecture, design, and evaluation of BINDER (Break-IN DEtectoR), a host-based system that detects break-ins of new unknown malware on personal computers by capturing their extrusions. The concept of BINDER was presented in a prior short publication [2]. Here we elaborate on the concept, and focus on its implementation and evaluation. To the best of our knowledge, BINDER is the first system to take advantage of user intent for host-based intrusion detection.

Under the assumption that user intent is implied by user-driven input. BINDER can infer user intent by correlating outbound network connections (initiated by the local personal computer) with user-driven input (key strokes and mouse clicks) at the process level. (We do not consider inbound connections (initiated by a remote computer) because most of them are malicious, and firewalls are designed to block such traffic.) Our key assumption is that outbound network connections made by a process that receives user input a short time ago is user intended. BINDER also treats repeated connections as user intended as long as the first one was user intended. By doing this, BINDER can handle the case of automatic refreshing web pages and polling emails. Among user unintended outbound connections, BINDER uses a small whitelist to differentiate benign traffic from malicious traffic. These benign outbound connections include system administration and checking software or security updates. Our Windows prototype of BINDER has 15 whitelist rules. Thus BINDER can detect a large class of malware such as worms, spyware and adware that (1) run as background processes, (2) do not receive any user-driven input, (3) and make outbound network connections.

By testing a Windows prototype both on multiple user systems and in a testbed environment, we demonstrated that BINDER is both effective and non-disruptive, successfully detecting malware without significant false alarms. We admit that, due to limitations of today's operating systems, there are several ways for adversaries to evade BINDER. These include actively subverting BINDER, hiding inside other processes, faking user input, etc.

The remainder of this paper is organized as follows. In Section 2, we compare BINDER with previous work and highlight its contributions. We present the details of BINDER's extrusion detection algorithm in Section 3 and describe its architecture and implementation in Section 4. In Section 5, we demonstrate our evaluation methodology and experimental results. We discuss BINDER's limitations and possible solutions in Section 6 and conclude the paper in Section 7.

2 Related Work

Many research efforts [16, 21, 30] and commercial products [25, 37] have focused on preventing break-ins by filtering known exploits or unknown ones exploiting known vulnerabilities. Complementary to these solutions, BINDER can detect break-ins of unknown new malware before their signatures are widely distributed.

In addition to signature-based filtering of inbound connections, ZoneAlarm's Program Control provides a control over which program is allowed to start outbound connections. It requires users to construct a complete list of legitimate network programs, which is beyond the capability of an average user. Compared with ZoneAlarm, BINDER controls outbound connections based on user intent. This gives BINDER two advantages. First, BINDER can detect compromised programs (like web browser programs) that otherwise are allowed to make outbound connections by ZoneAlarm because they are on its whitelist. Second, BINDER has a much smaller whitelist, which makes it possible to automate the management of whitelist.

Anomaly-based intrusion detection [5, 10, 36] have been studied for detecting unknown malware. The performance of anomaly-based approaches is very limited in practice due to its high false positive rate. BINDER leverages the unique characteristic of personal computers—user intent—to achieve minimal false alarms.

In the past few years, computer worms and spyware [20] have been a menace to both personal computing [27] and large networks [15]. Fast worm detection and containment becomes critical since worms can propagate much more rapidly than human response [22]. Most research efforts [7, 32, 33] have focused on random scanning worms. Instead of targeting at one type of malware, BINDER is simple and works across many kinds of malware such as worms, spyware and adware on personal computers.

There have been research efforts on profiling user behavior for detecting masquerade attacks and the insider threat [4, 11]. Instead of attempting to model user behavior, BINDER leverages a simple assumption that user intent is implied by user input, and outbound network connections made by a process that receives user input a short time ago is user intended. This enables BINDER to detect a large class of malware and achieve minimal false alarms.

3 Algorithm Design

3.1 Overview

Our goal is to automatically detect break-ins of new unknown malware on personal computers. BINDER's design should achieve:

- **Minimal false positives**: This is critical for any automatic intrusion detection system.
- **Generality**: BINDER should work for a large class of malware without the need for signatures, and regardless of how the malware infects the system.
- **Small overhead**: BINDER must *not* use intrusive probing and adversely affect the performance of the host computer.

Our key observation is that outbound network connections from a compromised personal computer can be classified into three categories: user intended, user unintended benign, and user unintended malicious (referred to as *extrusions*). BINDER detects certain kinds of malware by capturing their extrusions. (We will use connections and outbound connections interchangeably unless otherwise specified.)

In BINDER, we assume that user intent is implied by user-driven activities and malware runs as standalone processes. For those that hide under other processes by exploiting techniques proposed in [17], the current BINDER prototype is unable to detect them. We look at this as one of the limitations of today's operating systems. We will discuss more detailed attacks, countermeasures and potential solutions in Section 6.

In the rest of this section, we first demonstrate how user intended connections may be initiated. Then, we describe the extrusion detection algorithm. Finally, we discuss how malware can be detected by this algorithm.

3.2 Inferring User Intent

To study how user intended connections may be initiated, we consider three kinds of events: user events (user input), process events (process start and process finish), and network events (connection request, data arrival and domain name lookup). It is natural for a user input or data arrival event to trigger a new connection in the same process. For example, one clicks a link in IE, then IE will make new connections to download the requested web page. It is also normal to repeat a recent connection in the same process. For example, email clients repeatedly pull emails from the same email server. However, we also need to correlate events between different processes. For example, when a newly launched IE process makes outbound connections to download the default home page, it is its parent process—usually the shell process explorer.exe—that received the user input. In general, a user intended 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.

To verify if a connection is triggered by the *intra-process* rule, we just need to monitor all user and network activities of each single process. However, we need to monitor all possible communications among processes to verify if a connection is triggered by the *inter-process* rule. In our current design, we only consider communications from a parent process to its child process and use the following *parent-process* rule to approximate the *inter-process* rule. In the future we plan to extend BINDER's design to consider more possible communications among processes.

- *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.

3.3 Extrusion Detection Algorithm

The extrusion detection algorithm needs to decide if a connection is user intended and if it is in the whitelist. The whitelist covers three kinds of programs: system daemons, applications automatically checking updates, and network applications automatically started by the operating system. Actual rules are specific to each operating system and may become user specific. We discuss the whitelist for our Windows prototype in Section 4. The main idea is to limit the delay from a triggering event to a connection request event. Note that, for data arrival events, we only consider those of user intended connections. There are three possible delays for a connection request made by process ***P***.

- D_{new}: The delay since the last user input or data arrival event received by the parent process of P before P is created.
- D_{old}: The delay since the last user input or data arrival event received by P.
- D_{prev}: The delay since the last connection request to the same host or IP address made by P.

D_{old} is the reaction time of a process and D_{new} includes the loading time of a process as well. For user intended connections, D_{old} and D_{new} are of the order of seconds while D_{prev} is of the order of minutes. BINDER declares a connection to be an extrusion unless at least one of D_{old}, D_{new} and D_{prev} is within their respective thresholds of D_{new}^{upper}, D_{old}^{upper} and D_{prev}^{upper}. In Section 5.2 we will discuss how to choose these thresholds.

In the design of the extrusion detection algorithm, we assume that BINDER 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 existence of a rule means that any connection to the host made by a process of that application is not considered to be an extrusion.

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

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

After detecting an extrusion, BINDER can either drop the connection or raise an alarm with related information such as the process ID, the image file name, and the connection information. Studying the tradeoff between different reactions is beyond the scope of this paper.

3.4 Detecting Break-Ins

In this section, we discuss BINDER's capability of detecting break-ins of worms, spyware and adware because they generate malicious connections. 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 like Blaster [27] or user-activated like email worms. The latter also requires user input to infect a personal computer.

When the malware runs without user input, BINDER easily captures its first outbound connection as an extrusion. This is because the malware runs as background processes and does not receive any user input. So D_{old}, D_{new} and D_{prev} of the connection do not exist.

When the malware receives user input for its break-in, its connections shortly after the break-in may be masked by user activity. Thus BINDER may not be able to capture these initial extrusions. However, BINDER can detect the break-in later by observing the first non-user triggered connection. Also, BINDER clears user input history after a compromised personal computer is restarted. So even for the malware that received user input for its break-in, BINDER is guaranteed to capture its first connection as an extrusion after the victim system is restarted.

4 System Implementation

4.1 BINDER Architecture

To capture extrusions, BINDER correlates information across three sources: user-driven input, processes, and network traffic. There are four components in BINDER: *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 each operating system. We describe the implementation on Windows operating system in the second part of this section. 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 a user clicks the 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* does 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.

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 an inbound TCP or UDP packet with non-empty payload is received from a normal outbound connections. Note that the direction of a connection is determined by the direction of the first TCP SYN or UDP packet of this connection. 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. By analyzing 2644 DNS lookup times on one of the computers in our real world experiments (see Section 5), we observed that about 8% DNS lookups take more than 2 seconds. A connection request event has five 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 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* implements the extrusion detection algorithm.

4.2 Windows 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 most malware [28]. Given current Windows systems' limitations [17], our Windows prototype does *not* provide a bulletproof solution for break-in detection although it does demonstrate effectiveness of this technique on detecting a large class of existing malware. 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.

The *User Monitor* is implemented with Windows Hooks API [13]. 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.

The *Process Monitor* is implemented based on the built-in Security Auditing on Windows 2000/XP [14]. 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 [18] to parse the security event log and generates *process start* and *process finish* events.

The *Network Monitor* is implemented based on TDIMon [19] and WinDump [34] which requires WinPcap [35]. 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*.

It is straightforward to implement the extrusion detection algorithm based on the information stored in the process data record in the *Extrusion Detector*. Here we focus on the whitelisting mechanism in our Windows implementation. The whitelist in our current implementation has 15 rules. These rules cover three kinds of programs: 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 five 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 six 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 four rules for messenger programs of MSN, Yahoo!, AOL, and ICQ. These programs are allowed to make connections at any time. In the future, we need to include a special rule regarding wireless network status change. For example, an email client on a laptop computer may start sending prewritten emails right after the laptop is connected to the wireless network in a hot spot.

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 [31] may be used to help an average user configure her own whitelist.

5 System Evaluation

We evaluated BINDER on false positives and false negatives in two environments. First, we installed it on six Windows computers used by different volunteers for their daily work, and collected traces over five weeks since September 7th, 2004. Second, in a controlled testbed based on the Click modular router [8] and VMWare Workstation [29], we tested BINDER with the worm Blaster and 22 different email worms collected on a departmental email server over one week since October 7th, 2004.

5.1 Methodology

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,

Table 1. Summary of Collected Traces

User	Machine	OS	Days	User Events	Process Events	Network Apps	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

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. When we evaluate BINDER on false negatives, we check if and how fast it can detect a break-in. The real world experiments are used to evaluate BINDER for both false positives and false negatives, while the experiments in the controlled testbed are only for false negatives.

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 analysis in all experiments.

5.2 Real World Experiments

To evaluate the performance of BINDER in a real world environment, we installed it on six Windows computers used by different volunteers for their daily work, and collected traces over five weeks. We collected traces of user input, process information, and network traffic from the six computers. A summary of the collected traces 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 because it is difficult to convince users to submit their working environment to experimental software. However, from the summary of the collected traces in Table 1, we see that they have good diversity with respect to hardware, operating system, and user behavior. For real world experiments, we discuss parameter selection and then analyze the performance of our approach on false positives and false negatives.

5.2.1 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 tightest possible upper bounds under the condition that the number of false alarms is acceptable. We assume the rules of whitelisting described in Section 4.2 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 10 and 60 seconds, while D_{prev}^{upper} can take values between 600 and 3600 seconds. 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.

5.2.2 False Positives

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

By choosing parameters correctly, we expect to achieve minimal false alarms. From Table 2 we can see that there are at most five 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 2. Parameter selection for D_{old}, D_{new} and D_{prev}

	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

- Incomplete information of inter-process event sharing. For example, four of the five false alarms of User B are caused by this. We observe that PowerPoint calls IE to connect to the same IP address while the parent process of the PowerPoint process is 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.

- Incomplete trace collection. BINDER was accidentally turned off by user in the middle of trace collection. Users were given the control of turning on/off BINDER in case they think BINDER is responsible for bad performance, etc.

A break-down of the false alarms is shown in Table 3. We can see that a better-engineered BINDER can eliminate false alarms associated with the Whitelist and Collection columns in Table 3, resulting in much lower false positives. By extending BINDER to consider more inter-process communications than those between parent-child processes, we can decrease the false alarms associated with the Inter-Process column.

5.2.3 False Negatives

In the real world experiments, among the six computers, one was infected by the adware Gator [24] and CNSMIN [12] and another one was infected by the adware Gator and Spydeleter [23]. In particular, the second computer was compromised by Spydeleter when BINDER was running. BINDER successfully detected the adware Gator and CNSMIN because they do not have any user input in history. In the following, we demonstrate how BINDER detected the break-in of Spydeleter right after it compromised the victim computer.

In Figure 1, 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 that is downloaded 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 would detect 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 its history. So D_{old}, D_{new} and D_{prev} for the connection request made by ftp.exe do not exist. This connection is used to download malicious code. Therefore, BINDER's detection plus some appropriate actions could have stopped the adware from infecting the computer. Note that the three parameters of D_{old}^{upper}, D_{new}^{upper} and D_{prev}^{upper} do not affect BINDER's detection of Spydeleter here.

5.3 Controlled Testbed Experiments

The number of break-ins in our real world experiments is very limited. To evaluate BINDER's performance on false negatives, we need to test BINDER with more real world malware in a controlled testbed. In this section, we describe our controlled testbed and present experimental results on 22 email worms and the worm Blaster.

5.3.1 Controlled Testbed

The key challenge here is to repeat break-ins as real as possible but without unwanted damages. We tackle this problem by building a controlled testbed using the Click modular router [8] and VMWare Workstation [29]. We choose

```
 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 1. A stripped list of events logged during the break-in of adware Spydeleter.

Click for its powerful modules [9] for Network Address and Port Translation (NAPT). In addition, we implement a containment module in Click which can pass, redirect or drop outbound connections according to predefined policies. The advantage of using VMWare is that we can discard an infected system and get a new one just by copying a few files. VMWare also provides host-only private networks.

In the testbed, we have two Linux hosts running VMWare Workstation. On the first host, we have a Windows virtual machine (VM) in a host-only private network. This VM is used for executing malicious code attached in email worms. The Click router on this host includes a containment module and a NAPT module. The containment policy on this router is: (1) allow DNS traffic pass through; (2) redirects all SMTP traffic (to port 25) to another Linux host. The second Linux host has a Linux VM running the eXtremail server [3] in a host-only private network. The email server is configured to accept all relay requests. The Click router on this host also has a NAPT module that guarantees the email server can only receive inbound SMTP connections. Thus, all malicious emails are contained in the email server. This controlled testbed enables us to repeat the whole break-in and propagation process of email worms.

5.3.2 Experiments with Email Worms

We obtained email worms from two sources. First, we set up our own email server and published an email address to USENET groups. This resulted in the email worm W32.Swen.A@mm [26] being sent to us. Second, we were fortunate to convince the system administrators of our department email server to give us 1843 filtered virus email attachments which were collected over the week starting on October 7th, 2004. We used Symantec Norton Antivirus [25] to scan these attachments and recognized 27 unique email worms. Among them, we used 21 email worms because the rest of them were encrypted with a password. Thus, we tested 22 different real world email worms.

For each email worm, we manually start it on the Windows virtual machine that has a file of 10 real email addresses used by authors, let it run for 10 minutes (with user input in history), and then restart the virtual machine to run another 10 minutes (without user input in history). We analyze BINDER's performance using the traces collected during the two 10 minute periods. We choose 10 minutes because they are long enough for email worms to scan hard disk, find email addresses, and send malicious emails to them.

Our results show that BINDER successfully detects break-ins of all 22 email worms in the second 10 minute period by capturing the very first malicious outbound connection. In the rest of this section, we focus on BINDER's performance in the first 10 minute period.

Table 4. The impact of D_{old}^{upper}/D_{new}^{upper} on BINDER's performance of false negatives.

$D_{old}^{upper} = D_{new}^{upper}$ (sec)	10	20	30	40	50	60
Num of email worms detected	22	21	21	19	17	15

According to our discussion on parameter selection, D_{old}^{upper} and D_{new}^{upper} usually take values between 10 and 60 seconds, while D_{prev}^{upper} usually takes values between 600 and 3600 seconds. Since our traces are 10 minute long, the parameter D_{prev}^{upper} does not affect BINDER's performance on false negatives. So we study the impact of D_{old}^{upper} and D_{new}^{upper} on BINDER's performance of false negatives. In Table 4, we show the number of email worms are detected by BINDER when D_{old}^{upper} and D_{new}^{upper} take a same given value between 10 and 60 seconds. We have only one email worm (W32.Swen.A@mm) missed when we take 30 seconds for D_{old}^{upper} and D_{new}^{upper}. This is because the first connection is detected as user intended due to the user input and all following connections repeat the first one.

5.3.3 Experiments with Blaster

We test BINDER with the worm Blaster [27]. In this experiment, we run two Windows XP VMs A and B in a private network. We run BINDER on VM B and run msblast.exe on VM A. Blaster on VM A scans the network, finds VM B and infects it. By analyzing the infection trace collected by BINDER, we see that BINDER detects the first outbound connection made by a process tftp.exe as an extrusion. This is because the process itself and its parent process of cmd.exe does not receive any user input. Thus we can successfully detect Blaster in this case even before the worm itself is transferred over by TFTP.

6 Countermeasures and Solutions

Our limited user study shows that BINDER limits the number of false alarms to at most five over four weeks on each computer. We also show that BINDER successfully detects break-ins of the adware Gator, CNMIN, and Spydeleter, the worm Blaster, and 22 email worms. However, BINDER is far from a complete system, rather its goal is to verify that user intent can be a simple and effective detector of a large class of malware with a very low false positive rate. We devote this section to discussions of potential countermeasures against BINDER if its scheme is known to adversaries. Though we try to investigate all possible attacks against BINDER, we cannot argue that we have considered all of its possible vulnerabilities.

- Direct attack: subvert BINDER on the compromised system;
- Hiding inside other processes: inject malicious code into other processes;
- Faking user input: use APIs provided by the operating system to generate synthesized actions.
- Tricking the user to input: trick users to click on pop-up windows or a transparent overlay window that intercepts all user input.
- Exploiting the whitelist: replace the executables of programs in the whitelist with a tweaked one;
- Exploiting user input in history: When a malicious process is allowed to make one outbound connection due to user input (e.g., a user opens a malicious email attachment), it can evade BINDER's detection by making that connection to a collusive remote site to keep receiving data. This would make BINDER think any new connections made by this process are triggered by those data arrivals.
- Covert Channels: A very tricky countermeasure is to have a legitimate process make connections and use them as a covert channel to leak information. For example, spyware can have an existing IE process 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. A collusive remote server can get private information from the tweaked hyperlink.

Direct attack is a general limit of all end-host software (e.g., antivirus, personal firewalls [37], virus throttles [33] which attempt to limit outgoing attacks). A widespread availability of Trusted Computing-related Virtual Machine-based protection [6] or similar isolation techniques are necessary to turn BINDER or any of these other systems into robust production.

The countermeasures of hiding inside other processes, faking user input, tricking users to input, and exploiting whitelisting are inherent to the limitations of today's operating systems. The effectiveness of BINDER on malware detection implies pressing requirements for next-generation operating systems: isolation among processes, trustworthy user input, reflection on user intent, etc. Possible incomplete solutions for these countermeasures are: monitor corresponding system APIs; verify the integrity of programs listed in the whitelist. Even without a bulletproof solution for today's operating system, we believe a deployed BINDER system can raise the bar for adversaries significantly.

For the countermeasure of exploiting user input in history, a possible solution is to add more constraints on how a user intended connection may be triggered. This requires more research work in the future. For the countermeasure of covert channels, possible solutions are discussed in [1].

7 Conclusions and Future Work

In this paper, we present the design and implementation of BINDER, a host-based system that detects break-ins of worms, spyware and adware on personal computers by capturing their extrusions. The main contributions of this paper are:

- BINDER takes advantage of a unique characteristic of personal computers—user intent. Our evaluations show that user intent is a simple and effective detector for a large class of malware with a very low false positive rate.
- The controlled testbed based on the Click modular router and VMWare enables us to repeat the whole break-in and propagation process of email worms without worrying about unwanted damage.

In the future, we plan to study the advantage of sharing extrusion information among distributed BINDER systems.

References

[1] K. Borders and A. Prakash. Web tap: Detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communication Security*, October 2004.

[2] W. Cui, R. H. Katz, and W. tian Tan. Binder: An extrusion-based break-in detector for personal computers. In *Proceedings of 2005 USENIX Annual Technical Conference*, April 2005.

[3] eXtremail. extremail server. http://www.extremail.com/.

[4] T. Goldring. User profiling for intrusion detection in windows nt. In *Proceedings of the 35th Symposium on the Interface*, 2003.

[5] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[6] Intel. Intel virtualization technology, 2005.

[7] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *2004 IEEE Symposium on Security and Privacy*, May 2004.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[9] E. Kohler, R. Morris, and M. Poletto. Modular components for network address translation. In *Proceedings of OPENARCH'02*, June 2002.

[10] W. Lee and S. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[11] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2002.

[12] T. Micro. ADW_CNSMIN.A. http://www.trendmicro.com/vinfo/virusencyclo /default5.asp?VName=ADW_CNSMIN.A.

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

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

[15] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Magazine of Security and Privacy*, August 2003.

[16] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[17] rattle. Using process infection to bypass windows software firewalls. http://www.phrack.org/show.php?p=62&a=13, 2004.

[18] M. Russinovich and B. Cogswell. Pstools. http://www.sysinternals.com/ntw2k/freeware /pstools.shtml.

[19] M. Russinovich and B. Cogswell. Tdimon. http://www.sysinternals.com/ntw2k/freeware /tdimon.shtml.

[20] S. Saroiu, S. D. Gribble, and H. M. Levy. Measurement and analysis of spyware in a university environment. In *Proceedings of the First Symposium on Networked Systems Design and Implementation*, March 2004.

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

[22] 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.

[23] Suzi. How to get rid of spy deleter. http://netrn.net/spywareblog/archives/2004/03/12 /how-to-get-rid-of-spy-deleter/.

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

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

[26] Symantec. Symantec Security Response - Alphabetical Threat Index. http://securityresponse.symantec.com/avcenter/ venc/auto/index/indexA.html.

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

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

[29] VMWare. Vmware workstation 4.5. http://www.vmware.com/.

[30] 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.

[31] 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.

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

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

[34] WinDump. Windump. http://windump.polito.it/.

[35] WinPcap. Winpcap. http://winpcap.polito.it/.

[36] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[37] ZoneAlarm. http://www.zonelabs.com/.