

Dynamic Spyware Firewalling

Ian Sin and Jesse Pool

Department of Electrical and Computer Engineering

University of Toronto

{iansin,pool}@eecg.toronto.edu

Abstract

Spyware can most generally be considered software that monitors user behavior and reports collected information to a third-party. This information is valuable because it can be used to understand a consumer market in an effort to better target advertising.

“The spyware problem” presents a significant challenge for security research. Although it entails privacy and integrity concerns, some consider spyware a valid and profitable business model. While there has been some legislative action taken against spyware, its prevalence into our lives has become ubiquitous.

In an effort to defeat spyware, we observe that it creates network connections that are unintended by the user and that these communications significantly deviate from the expected case. We show that spyware activity can be detected by a comparison of network packets between a clean host and one that is infected with spyware. Further, by detecting unintended communication, we are able to define firewall rules on-the-fly which are used to prevent spyware from connecting to the Internet.

To demonstrate the effectiveness of our approach, we analyze the top 20 Internet websites, as reported by Alexa.com, and show that all spyware communication is prevented. Our results indicate that this process is both efficient and accurate.

1 Introduction

There is evidence that millions of computers are infected with spyware today. Like worms, viruses and DDoS, spyware is an Internet threat to which we have yet to find an effective solution. While there has been a recent legislative [1] and corporate [7] push to prevent spyware, these efforts are somewhat hindered due to an inconsistent definition of “the spyware problem.”

In this paper we refer to spyware as software that monitors user behavior and reports collected informa-

tion, or statistics, to a third party [11]. User information may be collected by analyzing browsing patterns, logging key strokes, observing running applications, scanning the hard disk, etc. This activity may be occurring with or without the knowledge or consent of the user and is clearly a privacy and security concern.

Industry has recognized this problem and responded with several solutions. Most notably, tools such as Ad-Aware [2] use signature detection, as well as several other heuristics, to detect spyware. However, these signature-based solutions require regular updating to detect new spyware. Another solution, which is most closely related to our work, is the concept of reverse firewalling which blocks outgoing Internet connections that have not been allowed by the user. However what constitutes spyware is a difficult question which limits the effectiveness of this approach.

Legislative countermeasures to spyware have also been proposed. While these laws are encouraging, spyware is a difficult term to define. What may be spyware to one person may be normal behavior to another. A good example is the Google Toolbar [5]. This software adds Google search capability to the web browser user interface. However, it also notifies Google of each website that is visited. A more serious limitation of spyware law is the global nature of the Internet. While legislation in one country may reduce the prevalence of spyware, it is difficult, if not impossible, to police across international borders.

Spyware is often installed on a computer through the use of free services. Some software packages are offered without cost, but include various forms of spyware. An example of such a case is the popular peer-to-peer software, Kazaa [6]. Generally, spyware is used by companies to better understand their consumers behavior in order to predict buying patterns, and is often used to serve targeted advertising [3]. However, more malicious uses may be to steal account information, or credit card numbers. Installing spyware may also introduce security vulnerabilities into an otherwise secure

operating system [11].

Clearly, it is in our best interest to avoid spyware, but in many cases it is unrealistic to expect a user to know that their system has been compromised. This is especially true because spyware might attempt to avoid detection. Spyware is often packaged with software that is targeted at users who do not know of the risks and are generally naive to its existence. This includes children.

Our observation is that spyware forces a system into making unintended network connections, in order to report behavior, serve targeted advertising or download updates. Based on this observation we assert that spyware activity can be detected and prevented through analysis of the network connections that result from general user activity. By comparing the network activity of a potentially infected host with that of a host that is known to be clean (spyware free), we can define dynamic firewall rules that prevent spyware from connecting to the Internet. We show that these rules are accurate and can be generated on-the-fly.

The contribution of this research is a novel method for detecting spyware in a general way, which is shown to be accurate and effective. We demonstrate that firewall rules can be derived through real-time packet analysis and provide an experimental toolset that can be used to distinguish spyware activity from benign user behavior.

The remainder of this paper is organized as follows. In the next section, we define our objectives. We then present the system architecture and implementation in Sections 3 and 4, respectively. In Section 5 we describe our experiment methodology and report our results in Section 6. Section 7 presents a discussion, which is followed by future work in Section 8 and we conclude in Section 9.

2 Objectives

In this work, we aim to address the spyware problem through a novel approach: dynamic firewalling. The identified objectives are three-fold.

- Design a system that prevents unintended network connections.
- Show that rules can be generated accurately and applied quickly based on packet analysis.
- Show that this system can work well against spyware by preventing user information from being sent to the spyware servers.

3 System Architecture

The proposed architecture is a host-based system made up of two virtual machines, a dynamic rule generator and a controller as shown in Figure 1. A virtual machine (VM) is an abstraction of the underlying hardware and runs its own operating system. Many VMs may share the underlying hardware without interfering with one another. A virtual machine monitor (VMM), which resides on the host-operating system, is used to multiplex and mediate all access requests from virtual machines for computer resources. VMs are isolated from each other and are said to constitute an isolated domain. The proposed system consists of three isolation domains, namely the two VMs and the host operating system, where the rule generator, controller and VMM reside. We believe that it is important to provide isolation of these two critical components (controller and rule generator) to prevent spyware from subverting the system in the future.

Ideally, an operating system running in a virtual machine is used for day-to-day activities like word processing, writing emails, and surfing the Internet. The integrity of this VM is unknown and, as such, it is considered *dirty*. The integrity of a second VM is assumed to be preserved and its state is pristine. This VM is marked *clean* and used for comparison against its dirty counterpart, as we explain in more detail shortly. The dirty VM is blocked from the Internet by default, and relies on the clean operating system to trigger its access to the network. To this end, a *rule generator* observes the activity of the clean VM and dynamically creates *accept* rules in the host operating system. These rules define the allowed network accesses for the dirty virtual machine.

A *controller* executing in the host operating system is currently used to evaluate the validity of this architecture. It coordinates browsing across the two virtual machines. Browsers in the trusted (clean VM) and untrusted (dirty VM) domains are driven to the same webpage in a staggered manner, the clean browser is loaded first followed by the dirty browser.

3.1 Flow of Events

Figure 1 depicts the interaction of system components. Here the controller requests `www.yahoo.com` from each of the clean and dirty virtual machines in turn. The rule generator analyzes packet data from the clean virtual machine and generates rules that allow ac-

in each VM. The plug-in virtualizes the input interface to the browser and implements a server that listens for connections from the controller over TCP/IP. Once a connection is established, the plug-in listens for commands asynchronously. The plug-in executes commands sent by the controller making use of the Document Object Model (DOM) [10] provided by the Firefox framework. The browser plug-in also implements additional features such as returning valid URLs scraped from a webpage for increased user interactivity. It also has the ability to browse autonomously for a user-determined amount of time and depth. The plug-in is implemented using the XML User Interface Language (XUL), JavaScript and Cross-Platform Component Object Model (XPCOM).

4.2 Rule Generator

The dirty virtual machine is blocked from accessing the Internet by default. In order for browser requests to reach their intended host, the correct firewall rules must be in place. The rule generator facilitates the addition of new *accept* rules to the firewall, which allow the dirty virtual machine to access specific subnets on the Internet, when the destination IP address matches an *accept* rule.

To implement dynamic firewalling, the rules are dictated by the clean VM. The rule generator extracts destination IP addresses accessed by the clean virtual machine and uses them to create *iptables* rules on the host operating system. Our implementation uses the Network Address Translation (NAT) table and IP forwarding provided by the Linux kernel. These rules are of the form: `iptables -A OUTPUT -o <interface> -s <dirty ip> -d <destination ip>/<mask> -j ACCEPT`.

In order to facilitate raw packet capture, we use the libpcap packet capture library [8]. The rule generator is written in 308 lines of C code.

4.3 Controller

The controller is used to operate the browsers running on each virtual machine. It coordinates their behavior ensuring that the clean browser navigates before the dirty browser, allowing enough time for the firewall rules to be generated.

Navigation can either be scripted (read from a file) or done interactively from the controller. It provides an intuitive interface that allows a user to navigate and fill forms on any web page.

# Accept Rules	# Good sites blocked	# Spyware thru
97	4	0

Table 1. Results from Top20 websites

Communication with the browser plug-in is implemented with the use of TCP/IP sockets. The controller is written in 213 lines of C code.

5 Methodology

Two virtual machines are setup within VMware [9] and configured with 256MB RAM, a 100 Mbps Network Interface Card and host-only networking. Both VMs run Windows XP Professional SP1 as this is where we find most spyware thrive. We also use Firefox 1.0, where we install our plug-in. The dirty VM is intentionally infected with spyware by installing Kazaa 3.0. This is a commonly used software package for peer-to-peer file sharing. The Kazaa installation includes several spyware components, namely: BTGrab and Need2Find. It also includes adware published by Joltid, which is also blocked by our design.

The evaluation is performed on the Top 20 websites as reported by Alexa.com. We use a tool called *mdiff* which we created to validate our results. *Mdiff* compares the unique IP addresses obtained from the clean trace against those produced by the dirty VM. It also performs a reverse DNS lookup to obtain hostnames when possible. The comparison is performed using a /24 mask in order to produce more coarse grained results. Our observation is that it is unlikely that spyware will connect to the same range of addresses as is needed to make an HTTP request. As our results will show, this is a valid observation and a mask of 24 bits, serves well in most cases. The mask can be defined and tuned by the user. We verify the correctness of the rules that are created by manually analyzing the clean and dirty traces and comparing with *mdiff* results. *Mdiff* is implemented in 317 lines of C code and uses the libpcap for packet capture.

6 Results

This section outlines the results obtained using the above mentioned experimental setup.

#	Clean Trace	Dirty Trace	Host Names
1		63.236.66.5	<i>need2find.com</i>
2	68.142.226.53	68.142.226.53	p22.www.re2.yahoo.com
3		72.246.50.6	<i>yahoo.com</i>
4	72.246.50.7	72.246.50.7	<i>yahoo.com</i>
5	72.246.50.8	72.246.50.8	<i>yahoo.com</i>
6	72.246.50.14		<i>yahoo.com</i>
7		206.116.223.38	d206-116-223-38.bscsia.telus.net
8	216.109.112.136	216.109.112.136	bs1.ads.vip.dcn.yahoo.com

Table 2. Trace of yahoo.com

6.1 Accuracy

Table 2 shows the trace results from a yahoo.com browsing session.¹ Yahoo! is ranked as the most popular website on the Internet according to Alexa.com. We record the unique destination IP addresses in both the clean and dirty traces, and we validate the results manually. All of the IP addresses in the clean trace were confirmed to be valid. For each of these, an iptables rule was created in the host operating system to allow access from the dirty virtual machine to that subnet. We acknowledge that there will be some repetition in the resulting rules due to our use of the 24 bit mask. For example, In row 4, we would generate rule 72.246.50.0/24, the same rule would result from row 5.

Those addresses in the dirty trace that do not match a rule are suspect. However, in row 3, the dirty IP 72.246.50.6 would *not* be flagged as spyware because it has the matching rule 72.246.50.0/24. The reason for the discrepancies (rows 3 and 6) is due to Yahoo! load balancing. Rows 1 and 7 also show suspect IP addresses. These would not be allowed to pass through the firewall as they don't have a matching rule. Manual verification of the packet traces confirms that row 1 is spyware published by Need2Find, while row 7 is adware published by Joltid. These are indeed spyware/adware and we block communication to their respective servers by *not* generating *accept* rules.

The same procedure is followed for the Top 20 websites. In summary, Table 1 shows our aggregate results for the Top 20 websites. Verification using mdiff showed that we have approximately 4% of falsely identified sites as spyware sites and 0% spyware leakage traffic. More precisely, we generated 97 *accept*

¹The italicized text refers to host names that were not resolved through reverse lookup. Instead we manually inspected the packet trace for the DNS request that was related to this IP address.

rules when we should really have generated 101. This means that 4 legitimate IP addresses should have been allowed through from the dirty browser. On closer inspection we note that 3 out of the 4 rules we failed to generate were for dynamic ad content. The remaining rule that was not generated would have been generated if we used an 18 bits netmask instead of 24 bits.

Our results show that a 24 bit mask is accurate. We believe that this could be further improved with more tuning, which we leave for future work.

6.2 Performance

Although no performance metrics are currently available, we have not seen noticeable delays in generating and setting up the *iptables* rules. A back of the envelope calculation tells us that the worst case overhead of such a system is a little more than twice the usual time to get a web page as two round trip times (RTTs) have to be incurred, plus rule generation and setup overheads. Further testing is required to validate this claim. It is also reasonable to assume that simple caching can improve performance by removing the additional round trip time.

7 Discussion

The evaluation of this system shows (as proof of concept) that unintended spyware communications can be accurately blocked using dynamic firewall rules. We demonstrated that our system prevents information leakage to spyware servers and mitigates the often annoying pop-ups that are a result of targeted advertising.

However, our system has several acknowledged limitations. First and foremost, the user experience is non-optimal. The interface is text based and requires specially formatted commands. Moreover, online transactions, like email, e-commerce (buying a book), and

banking are difficult problems as they would entail performing the same action twice - once for the clean and once for the dirty browsers.

While our efforts are aimed at removing spyware, we note that it is not our intention to disrupt legitimate online advertising. Although our results in Section 6.1 demonstrate that the approach taken can incorrectly block some amount of dynamic Internet advertising, we believe that this amount is sufficiently small as to avoid significant disruption.

8 Future Work

While the premise of our proposed architecture is focused on a comparison between clean and dirty network packets, we have not discussed the methods by which a virtual machine can be maintained free of spyware. We intend to investigate how often a VM should be refreshed in order to maintain a pristine state. It is important that the replacement algorithm not be intrusive and results in little overhead.

We also intend to further investigate tuning the rule generator. Specifically, we would like to optimally determine the number of bits used to mask the allowed destination IP address used in rule creation, such that accuracy is acceptable to the end-user. Furthermore, future study will focus on the frequency of rule flushes.

As an extension of our research, we would like to investigate the possibility of using DNS information with the current rule generator to create more accurate firewalling. All the spyware we have studied thus far (180Solutions, Need2Find, BTGrab), and as shown in [11], make DNS requests before contacting their home servers. Preliminary tests have shown that spoofing DNS is possible. This can also be used to redirect all spyware traffic to a *sink* for further analysis.

As mentioned in the discussion (see section 7), work is on-going to improve user experience. It remains our intention to implement a solution that does not deviate from the current usage model of the web browser, or the Internet in general. As such, we are investigating the possibility of including the controller within the dirty virtual machine. This is a significant challenge due to execution in an untrusted/dirty domain as it could possibly be subverted by the spyware.

9 Conclusions

We have designed and implemented a system to address “the spyware problem” in a general way that is

independent of spyware implementation. Our method uses a *clean* baseline to generate firewall rules. These rules are created on-the-fly and are applied dynamically. With the rules in place a user may browse the Internet assured that unintended connections to third-party servers are blocked. Our results show that dynamically applying firewall rules is accurate and effective.

Acknowledgements

We would like to thank David Lie and Stefan Saroiu for their guidance and insightful discussions.

Resources

The project webpage is linked off our respective websites or can be found at <http://utms.sa.utoronto.ca/iansin/projects/webmonkeys>. If you wish to get hold of the source code, please contact us.

References

- [1] Library of Congress Bill Number H.R.2929. Available at <http://thomas.loc.gov/cgi-bin/query/z?c108:H.R.2929:>. 2004.
- [2] Ad-Aware Personal. Available at <http://www.lavasoft-usa.com/software/adaware/>. 2005.
- [3] Benjamin Edelman - Research. Available at <http://www.benedelman.org>. 2005.
- [4] Firefox Web Browser. Available at <http://www.mozilla.com/firefox>. 2005.
- [5] Google Toolbar. Available at <http://toolbar.google.com>. 2005.
- [6] Kazaa P2P Client. Available at <http://www.kazaa.com>. 2005.
- [7] Microsoft Windows AntiSpyware. Available at <http://www.microsoft.com/athome/security/spyware/software/default.msp>. 2005.
- [8] TCPDUMP Public Repository - libpcap library. Available at <http://www.tcpdump.org>. 2005.
- [9] VMware. Available at <http://www.vmware.com>. 2005.
- [10] XULPlanet.com. Available at <http://www.xulplanet.com>. 2005.
- [11] H. M. L. Stefan Saroiu, Steven D. Gribble. Measurement and Analysis of Spyware in a University Environment. In *Proceedings of the ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March 2005.