NetSpy: Automatic Generation of Spyware Signatures for NIDS *

Hao Wang, Somesh Jha and Vinod Ganapathy
Computer Sciences Department, University of Wisconsin-Madison
{hbwang, jha, vg}@cs.wisc.edu

Abstract

We present NetSpy, a tool to automatically generate network-level signatures for spyware. NetSpy determines whether an untrusted program is spyware by correlating user input with network traffic generated by the untrusted program. If classified as spyware, NetSpy also generates a signature characterizing the malicious substrate of the spyware's network behavior. Such a signature can be used by network intrusion detection systems to detect spyware installations in large networks.

In our experiments, NetSpy precisely identified each of the 7 *spyware programs that we considered and generated network-level signatures for them. Of the* 9 *supposedly-benign programs that we considered, NetSpy correctly characterized* 6 *of them as benign. The remaining* 3 *programs showed network behavior that was highly suggestive of spying activity.*

1. Introduction

Spyware is a class of malware that steals private information from users without their knowledge or permission. Popular examples of spyware include keyloggers, programs that monitor web-browsing activity, and Trojans that download and install other malware. Most spyware typically masquerade as programs that provide useful functionality, such as browser plug-ins and extensions, and compromise the privacy of unsuspecting individuals who install them on their computers. Several recent studies show that the threat of spyware is on the rise, with one study reporting that as many as 80% of computers in the US are spyware-infected [9, 19]. Because spyware surreptitiously snoops and reports victim behavior to a malicious remote server, victims often do not notice malicious activity on their machines and do not realize that the spyware program is compromising their privacy. This very characteristic makes it challenging to detect spyware.

Prior techniques for spyware detection fall under two complementary categories: host-based and network-based. Host-based detectors work much like virus scanners, i.e., they scan binary executables for the presence of patterns contained in a database of known spyware patterns. Most commercial tools use simple techniques—such as matching MD5 signatures of executables—which fail to detect variants of spyware. To address this shortcoming, recent work has extended the basic matching approach to a behavior-based approach [17]. In contrast, network-based detectors monitor network traffic for malicious activity suggesting spying behavior. For example, Snort [6] can match outgoing network traffic against signatures of known spyware. These signatures are typically developed manually (e.g., Bleeding Edge Snort [2]).

Both these techniques suffer from two important shortcomings. First, because they use signatures of spyware behavior, either at the binary executable or at the network level, they can only detect known instances of spyware and are thus ineffective at detecting novel spyware instances. Even behavior-based techniques (e.g. Kirda *et al.* [17]) suffer from the shortcoming that they can only detect previously-unseen variants of known spyware behavior. Second, formulating signatures is currently ad hoc and manual with little work on automating the process. For example, Bleeding Edge Snort uses a manual approach to construct spyware signatures for Snort [6]. While there have been recent advances [10, 11] at identifying spyware programs based on suspicious network activity, to the best of our knowledge there is no prior work on automatic generation of network-level signatures for spyware.

This paper focuses on characterizing the network-level behavior of spyware. We present a novel approach to *automatically generate network-level signatures* for spyware. Such signatures can be used with network intrusion detection systems (NIDS), such as Snort or Bro [3], to monitor outgoing traffic, and detect spyware installations within a network. We have developed *NetSpy*, a tool that generates network-level signatures for spyware.

Key to our approach is the observation that spyware monitors victim activity, such as keystrokes and web sites visited, and reports this back to a home server. Thus, out-

*Supported by grants ONR/N00014-01-1-0708, NSF/CCR-0133629, DOE/DE-FG02-93ER25176 and NSF/CNS-0448476.

bound network traffic generated by a spyware program *must* contain some footprint of the victim's behavior that triggered the network activity. NetSpy uses this observation to achieve two goals: (i) determine whether an untrusted program is potentially spyware, and (ii) if the program is spyware, then generate a signature characterizing the malicious substrate of its network behavior. The first goal is achieved by NetSpy's *differential analysis tool* while the second goal is achieved by its *signature generation tool*.

Differential analysis, described in Section 3, is a two-step process. The first step isolates network traffic generated by the untrusted program. The second step analyzes this traffic for spying behavior, i.e., correlation between the contents of network traffic and user-input. If such a correlation exists, then NetSpy deems that the untrusted program is spyware. This network traffic is then fed to the signature generator, described in Section 4, which generates a network-level signature. The main goal of signature generation is to isolate the malicious substrate that remains invariant irrespective of the specific user-input that triggered the spyware program to generate this network traffic. For example, network packets generated by a keylogger will typically contain specific keystrokes that must be filtered out to generate a signature for the keylogger. NetSpy's signature generator uses a variant of the *longest common subsequence* (LCSeq) algorithm [12] to generate succinct and precise spyware signatures for NIDS.

We evaluated NetSpy on 7 known spyware programs and 9 supposedly-benign programs. NetSpy was able to detect all 7 spyware and generate Snort signatures for them. More interestingly, NetSpy detected that *A9 Toolbar*, one of the supposedly-benign programs tested, contains features that may be considered undesirable by many users. A9 Toolbar is an Internet Explorer plug-in that saves URLs that a user has visited on A9's home server, *under the user's account*. Therefore, this requires a user to sign in to A9's web site first. However, NetSpy found that regardless of whether a user has signed into A9, A9 Toolbar always sends URLs that the user has visited to a special server `siteinfo.a9.com`.

In summary, we believe that the following features of NetSpy advance the state-of-the-art in spyware signature generation.

1. **Ability to detect novel spyware.** NetSpy observes the network activity generated by an untrusted program in response to simulated user-input and determines whether the program is possibly spyware. This approach also enables NetSpy to generate signatures for previously-unseen spyware instances.
2. **Network-level signature generation.** If deemed to be spyware, NetSpy generates a signature for the malicious substrate of an untrusted program's network behavior. These signatures can be used by a NIDS that monitors outgoing traffic from a network, thus enabling detection of spyware installations on all machines within the network.
3. **Automation.** NetSpy is fully automatic. When a new program (such as a browser toolbar) is installed on a machine, NetSpy can determine immediately whether the program is potentially spyware and automatically generate Snort signatures for the program.

2. Overview

NetSpy has two goals: (i) to automatically discover possibly malicious network activity generated by novel spyware instances, and (ii) to generate NIDS signatures for this network activity. This section presents a high-level, informal overview of NetSpy, focusing on how an end-user would use NetSpy to generate signatures for spyware. We begin with a running example.

2.1. Browser Accelerator: an example spyware

Browser Accelerator [15] is a spyware program that disguises itself as a plug-in for Internet Explorer. Plug-ins normally enhance the functionality of Internet Explorer by providing additional features. However, Browser Accelerator also monitors user-web-browsing activity, and reports the monitored activity back to a home server.

To illustrate the spying behavior of Browser Accelerator, we first consider the network activity generated by a "clean", i.e., non-spyware-infected, version of Internet Explorer, and compare it with the network activity generated by Internet Explorer with Browser Accelerator installed.

Suppose that an end-user uses Internet Explorer to visit the URL `www.google.com`. On a clean version, this generates two out-bound HTTP requests (Figure 1, rows 1-2), both destined for `www.google.com`. The first request retrieves the root document associated with the URL, while the second retrieves the Google logo image contained within the root document.

With a Browser Accelerator-infected version of Internet Explorer, the same end-user-activity instead generates *seven* HTTP requests (Figure 1, rows 1-4). The first two requests are identical to those observed with the clean version. The third request, destined for a remote server called `data.browseraccelerator.com`, contains the URL `www.google.com`. Four additional requests are destined for another server called `client.browseraccelerator.com`.

In fact, our experiments show that a Browser-Accelerator-infected version of Internet Explorer will generate an almost identical set of extra HTTP requests (Figure 1, rows 3-4), *irrespective* of the URL visited. The only variant is the URL string encoded in the request destined for `data.browseraccelerator.com`, as shown in the box in Figure 1, row 3. For example, the HTTP request, destined

		Destination	HTTP Requests
1	infected, clean	www.google.com	GET /
2	infected, clean	www.google.com	GET /intl/en/images/logo.gif
3	infected	data.browseraccelerator.com	GET /data/track.aspx?...&theurl=http:// $\boxed{\texttt{www.google.com}}$ /
4	infected	client.browseraccelerator.com	*four requests destined to this host are omitted for brevity*

Figure 1. Network traffic generated by Internet Explorer when visiting `www.google.com` in two different settings: rows 1-2 are generated using a clean version; rows 1-4 are generated by a Browser Accelerator-infected version. For brevity, we have only shown relevant portions of the packets.

for `data.browseraccelerator.com`, generated by visiting `www.apple.com` with an infected version will contain the URL string `www.apple.com` instead. The other four requests destined for `client.browseraccelerator.com` are identical across all experiments and therefore are not discussed in detail here.

Observe that for both URLs mentioned above, a Browser Accelerator-infected version of Internet Explorer generates extra network activity that is not observed with a clean version. NetSpy uses this observation to detect spyware programs and generate NIDS signatures for the network activity that they generate. We now discuss how an end-user, such as a system administrator, would use NetSpy to automatically generate NIDS signatures for spyware programs, such as Browser Accelerator.

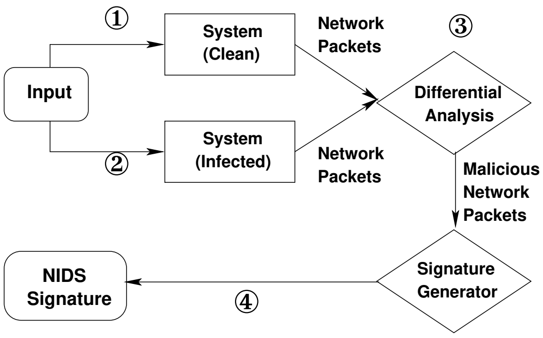

Figure 2. NetSpy Architecture Overview.

2.2 Spyware signature generation using NetSpy

We have designed NetSpy for use in environments where a large number of machines are supported. When a program installed on a particular machine in the network is suspected to be spyware, a system administrator can use NetSpy to determine if it is indeed so, and generate a signature to capture the outbound network-level behavior of the program. A key feature of network-level signature generation is that once a spyware program has been detected on a particular machine in the network, the network-level signature can detect installations of that program on other machines in the network. There are four high-level steps to using NetSpy, as shown in Figure 2.1. We describe each in detail.

Step 1. Collecting reference network statistics. A spyware-infected machine will produce different network activity as compared to a clean machine. Thus, the first step is to collect the network-behavior of a machine that does not have spyware programs installed. The network behavior of such a clean machine serves as a reference point against which all future network-behavior of the machine will be compared and is called the *reference network statistics*. This consists of out-bound network traffic generated by the clean machine in response to a predetermined set of inputs. This step is fully automatic. The only optional manual step is for a system administrator to provide input parameters with which to run NetSpy (however, NetSpy does provide a default set of input parameters).

NetSpy only observes *out-bound network traffic* because our assumption is that spyware monitors user activity, and *reports this back to its home server*. If a system is infected with a spyware, then out-bound network traffic from the infected system *must* differ from that of a clean system. In the rest of this paper, the term *network activity/traffic* is used synonymously with *out-bound network activity/traffic*.

A system administrator first collects reference network statistics by choosing a set of URLs, such as `www.google.com`, as the input with which to run Internet Explorer. For each input URL, Internet Explorer generates one or more HTTP requests. For example, accessing `www.google.com` generates exactly two requests. The first fetches the root document, and the second fetches the Google logo, referenced within the root document. Figure 3 shows an example of reference network statistics collected during an experiment where the input set consists of three URLs. Note that for some URLs, such as `www.apple.com` and `slashdot.org`, there are multiple requests destined for different host addresses.

Although many web sites update their content frequently (e.g., `slashdot.org`), the hosts they access rarely change. Collecting reference network statistics can thus be an infrequent activity (e.g. once a week). Alternately, NetSpy can be configured to retrieve the latest network reference statistics daily, or in extreme cases, each time NetSpy is executed. We leave this as a configurable parameter for NetSpy users. An issue that NetSpy must deal with is the background traffic generated by known benign programs such as email applications and automatic updates. We maintain a *whitelist* to filter out the network traffic generated by benign processes.

Note that our description of this step and the current im-

Input	(Step 1.) Reference Network Statistics		(Step 2 & 3) Infected Network Statistics	
	host	**HTTP Requests**	**host**	**HTTP Requests**
www.apple.com	www.apple.com	GET /main/css/globalprint.css	www.apple.com	GET /main/css/globalprint.css
		GET /home/2006/ticker.rss		GET /home/2006/ticker.rss
	images.apple.com	*other requests omitted for brevity*	images.apple.com	*other requests omitted for brevity*
			data.browser-accelerator.com	GET ...&theurl=http://www.apple.com/ *more requests omitted for brevity*
www.google.com	www.google.com	GET /	www.google.com	GET /
		GET /intl/en/images/logo.gif		GET /intl/en/images/logo.gif
			data.browser-accelerator.com	GET ...&theurl=http://www.google.com/ *more requests omitted for brevity*
slashdot.org	slashdot.org	GET /	slashdot.org	GET /
	images.slashdot.org	GET /topics/topicnintendo.gif	images.slashdot.org	GET /topics/topicnintendo.gif
		other requests omitted for brevity		*other requests omitted for brevity*
			data.browser-accelerator.com	GET ...&theurl=http://slashdot.org/ *more requests omitted for brevity*

Figure 3. Network statistics collected for input set {`www.google.com, www.apple.com, slashdot.org`}.

plementation of NetSpy are tailored to detect spyware programs that infect Internet Explorer. NetSpy currently cannot handle other kinds of spyware (e.g., keyloggers). We plan to extend NetSpy to handle such spyware in the future.

Step 2. Collecting network traffic from an altered machine. NetSpy can determine whether an untrusted program installed on a machine is potentially spyware. To do so, a system administrator runs NetSpy on the machine (with the untrusted program installed) using the same set of inputs from Step 1. NetSpy automatically collects the network traffic generated by the system.

Suppose that Browser Accelerator has been installed on a machine. The system administrator invokes NetSpy with the same set of URLs as used in Step 1, which in turn simulates browsing activity in Internet Explorer. Because of the spying behavior of Browser Accelerator, NetSpy now records extra network activity for each URL visited. For example, it records extra traffic for the input URL `www.apple.com`, as shown in the first shaded row in Figure 3.

A key issue is to determine when to invoke NetSpy. Several possibilities exist. One option is to invoke NetSpy periodically (e.g., overnight) on all machines in the network. Another option is to invoke NetSpy each time a new program is installed on any machine on a network. There are several standard techniques to determine that a new program has been installed. For example, on Windows 2000/XP systems, it is possible to track all browser extensions and plug-ins that have been installed on a system. Thus NetSpy can automatically be invoked when a new extension or plug-in is installed on a system. This paper assumes that techniques to determine when a new program has been installed are readily available to a system administrator. It focuses instead on how NetSpy determines whether an unknown program is spyware and generates a network-level signature for it.

Step 3. Differential analysis. The goals of this step are twofold: (i) to identify network traffic generated by an untrusted program, and (ii) to determine whether the program is spyware. Differential analysis achieves the first goal by comparing network traffic gathered in Step 2 against the reference network statistics, gathered in Step 1. For each captured network packet, NetSpy identifies the hostname that the packet is destined to, and the destination port number.

The second goal is achieved by correlating user-input to network traffic. This correlation provides concrete evidence that a program is indeed spyware, following which a system administrator can use NetSpy's signature generation tool to generate a NIDS signature that characterizes the malicious substrate of the spyware program's network activity.

The shaded rows in Figure 3 show the malicious network activity generated by a Browser Accelerator-infected version of Internet Explorer, as identified by NetSpy. Note that for every input URL entered, there are extra HTTP requests generated by the infected version of Internet Explorer (as compared to the reference network statistics)—a key characteristic of spyware. In addition, because input URLs are correlated to outbound network traffic generated by Browser Accelerator, NetSpy classifies Browser Accelerator as a spyware (Section 5). However, this need not always be the case. For example, because Mirar Toolbar encrypts the data it sends to its home server, the correlation between user input and contents of network traffic is not as clear. Section 3 describes the heuristics that NetSpy currently employs to rank an untrusted program as potential spyware.

Step 4. Generating spyware signatures for NIDS. The final step is to generate a network-level signature for a program identified as spyware (in Step 3). To do so, NetSpy uses the spyware-generated network traffic identified by differential analysis. However, this traffic is typically dependent on user-input. For example, the HTTP requests generated by Browser Accelerator for the URL www.google.com contain the string www.google.com. These requests, if used directly as the signature, would only match traffic generated by Browser Accelerator when the user visits www.google.com. Thus we must identify the invariant portion of network traffic generated by a spyware program. We do so using a modified version of the Longest Common Subsequence (LCSeq) algorithm [12]. The modified algorithm adds a wildcard .* at locations in the output of differential analysis that vary with user input. Doing so produces a regular expression which can then be used directly as a NIDS signature.

HTTP Requests	`GET ...&theurl=`$\boxed{\texttt{http://www.google.com/}}$
	`GET ...&theurl=`$\boxed{\texttt{http://www.apple.com/}}$
	`GET ...&theurl=`$\boxed{\texttt{http://slashdot.org/}}$
LCSeq	`GET ...&theurl=http://.*/`
NIDS Sig.	`alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"Browser Accelerator"; flow:established,to_server; pcre:"GET ...&theurl=http://[.*]"; nocase; classtype:trojan-activity; ... ;)`

Figure 4. Generating NIDS signatures.

We illustrate this on our running example. The first row in Figure 4 contains the three HTTP requests generated by Browser Accelerator for three input URLs, as identified by NetSpy in Step 3. Note that the three requests are almost identical to each other, except for the URL strings (shown in the boxes). The basic LCSeq algorithm, when applied to these requests, finds structure common to these requests. The modified LCSeq algorithm represents portions specific to each request using wild-card entries (shown in the second row of Figure 4), thus creating a regular expression, which can be used to create a signature for use with Snort or Bro. The last row of the figure shows a Snort signature constructed using this regular expression.

3. Differential analysis

Differential analysis examines the network traffic generated by a machine with an untrusted program installed and achieves two goals: (i) it identifies the portion of network traffic generated by the untrusted program, and (ii) it determines whether the program is potentially spyware. The key observation used here is that a spyware program monitors victim activity and reports this back to a home server. Thus, network traffic generated by spyware must be dependent on user-input, i.e., different values for user-input produce different network activity. Differential analysis employs heuristics, described below, to assign a score between 1 and 3 to an untrusted program, where a higher score indicates that the program is more likely to be spyware.

3.1. Identifying network traffic generated by an untrusted program

To isolate network traffic generated by an untrusted program, differential analysis records network traffic generated by a machine with the suspicious program installed and compares that traffic with reference network statistics. Because programs behave differently under different inputs, network traffic generated by a program depends on the input that it is executed with. To identify network traffic generated by an untrusted program, NetSpy compares the traffic generated by running a machine (with the program installed) under an input set $\mathcal{I}$ against the traffic generated by a clean system fed with the same input set.

In particular, suppose that the input set $\mathcal{I}$ consists of the inputs $\{i_1, i_2 \ldots i_n\}$. On our running example, these will be URLs with which Internet Explorer is invoked. NetSpy simulates user activity by feeding each input $i \in \mathcal{I}$ to the program, and records the network traffic generated. We denote the captured traffic as a tuple $< i, \mathcal{P}_i >$, where $\mathcal{P}_i$ itself is a set of tuples, $\mathcal{P}_i = \{< h_1, p_1 >, < h_2, p_2 > \ldots < h_m, p_m >\}$, where each h_i represents a destination host address, and p_i denotes the corresponding set of network packets (e.g., HTTP requests) destined for that host. Note that an input i can trigger the machine to send data to multiple hosts and multiple requests to each of these hosts.

This process is used both to collect reference network statistics and network traffic generated by a machine with an untrusted program installed. Let $\mathcal{O}(\mathcal{I})$ and $\mathcal{O}'(\mathcal{I})$ denote, respectively, the reference network statistics, and the network traffic generated by a machine with an untrusted program when invoked with input set $\mathcal{I}$:

$$\mathcal{O}(\mathcal{I}) = \{< i_1, \mathcal{P}_{i_1} >, < i_2, \mathcal{P}_{i_2} > \ldots < i_n, \mathcal{P}_{i_n} >\}$$
$$\mathcal{O}'(\mathcal{I}) = \{< i_1, \mathcal{P}'_{i_1} >, < i_2, \mathcal{P}'_{i_2} > \ldots < i_n, \mathcal{P}'_{i_n} >\}.$$

The difference between the two sets, $\mathcal{D} = \mathcal{O}'(\mathcal{I}) - \mathcal{O}(\mathcal{I})$, where $\mathcal{D}(i) =< i, \mathcal{P}'_i - \mathcal{P}_i >, i \in \mathcal{I}$, is the network traffic generated by the untrusted program.

Note that we identify network traffic associated with the program under several inputs. As discussed in detail in Section 4, this is necessary to produce a NIDS signature if the program is identified as spyware. A NIDS signature must capture an invariant property of network traffic generated by the spyware program—one that does not depend on specific user-input. To identify this invariant, we collect network traffic under several inputs.

3.2. Classifying an untrusted program as spyware

Having identified network traffic generated by the untrusted program, the next step is to determine if the program is spyware. We assign a score to the program based upon characteristics of the traffic that it generates. The heuristic currently used to assign this score is shown in Figure 5. Specifically, we assign a score of 3 (most likely to be spyware) to a program that (i) sends data to a server previously unseen in reference network statistics, and (ii) generates this data based upon user-input. For example, in the case of Browser Accelerator, we observe that each input URL to Internet Explorer is included as part of a HTTP request sent by Browser Accelerator to `data.browseraccelerator.com`, a server that is not included in reference network statistics.

Input : $\mathcal{S}$: the spyware program to be analyzed; $\mathcal{I}$: the set of inputs $\{i_1, i_2 \ldots i_n\}$; $O(\mathcal{I}) = \{< i, \mathcal{P}_i >, i \in \mathcal{I}\}$, the reference network statistics;
Output : $\mathcal{D}(\mathcal{I})$: the extra network traffic generated by $\mathcal{S}$ for the input $\mathcal{I}$; $\mathcal{R}_S$: the spyware score of the program S

$O'(\mathcal{I}) = \{< i, \mathcal{P}'_i >, i \in \mathcal{I}\}$ // Network traffic with $\mathcal{S}$ installed;
foreach $(i \in \mathcal{I})$ **do**
  $\mathcal{D}$(i) $= < i, \mathcal{P}'_i - \mathcal{P}_i >$;
$\mathcal{R}_S$ = score of 1, 2, or 3 (See Figure 5);
return $(\mathcal{D}(\mathcal{I}), \mathcal{R}_S)$

Algorithm 1: Differential Analysis

A program is assigned a score of 2 if it sends data to a server not included in the reference network statistics, but correlation between network traffic contents and inputs cannot be determined. Out of the 7 spyware programs that we evaluated, NetSpy identified two that fit into this category (see Section 6 for details).

Finally, a program is assigned a score of 1 (i.e., least likely to be spyware) if either (i) it does not produce any network traffic, or (ii) sends data only to servers included in the reference network statistics. For example, most of the benign programs that we tested NetSpy on fall into this category. Algorithm 1 summarizes differential analysis.

4. Signature Generation

Differential analysis assigns a score $\mathcal{R}_S$ (between 1 and 3) to an untrusted program S that it analyzes. If $\mathcal{R}_S$ is above a certain threshold (2 in our current implementation), NetSpy classifies the program as spyware and generates a NIDS signature for this program.

The key to signature generation is to identify the invariant portion of network traffic generated by a spyware program, i.e., we must filter content that is specific to user input. This is because a signature that has content related to specific user input will miss network activity generated by the program on *other* user input. On our running example, Browser Accelerator, each packet sent to the home server `data.browseraccelerator.com` contains a URL that the user has entered. The NIDS signature must be agnostic to the URL and retain the invariant portion of network traffic.

NetSpy uses a variant of the *longest common subsequence* (LCSeq) algorithm [12] to find invariants. LCSeq finds the longest subsequence of characters (i.e., not necessarily contiguous) common to two input strings. We modified LCSeq in two ways. First, to generate signatures for NIDS, the output must be a regular expression that represents network traffic, with portions of the traffic that vary with user input replaced by wild-cards. We do so by introducing *markers* at locations in the input strings where LCSeq identifies differences. These markers are then used to place the `.*` wild-card operator and convert the result into a regular expression. Second, the standard LCSeq algorithm works over *two* input strings. We modified it to work with an arbitrary set of input strings. We do so by iterating over the set of packets and refining a candidate longest common subsequence as we do so. Algorithm 2 shows NetSpy's signature generation algorithm.

Input : $\mathcal{D}(\mathcal{I})$: set of network packets generated by $\mathcal{S}$; $\mathcal{R}_S$: spyware score of S.
Output : Signature_S: NIDS signature for $\mathcal{S}$

if $\mathcal{R}_S <$ threshold **then**
  return EMPTY;
lcs = LCSeq-modified($\mathcal{D}$(1), ..., $\mathcal{D}$(n));
foreach marker introduced by LCSeq-modified **do**
  lcs[index] = `.*`;
Signature_S = lcs;
return Signature_S;

Algorithm 2: Signature Generation

5. Implementation

The NetSpy prototype is currently implemented for Windows 2000/XP, and focuses on generating signatures for spyware that target Internet Explorer. NetSpy is fully automatic. The only (optional) manual task is that of changing a configuration file that contains the input URLs used to collect reference network statistics. It currently produces Snort signatures as output (though it can be adapted to produce signatures in other formats as well). We envision that these signatures can help efforts like Bleeding Edge Snort. The prototype, implemented in C/C++, currently stands at 4700 lines of code, consisting of three principle components: a packet capturing tool, a differential analysis tool, and a NIDS signature generator.

The packet capturing tool records packets generated by Internet Explorer in response to the simulated user activity of visiting set of input URLs. The main components of this tool are a driver, built using the `IWebBrowser2` interface [4], which allows full programmable control over Internet Explorer. For example, the driver uses Internet Explorer to open a URL, reload a page, follow a link on a page and close the browser window. Packets generated by Internet Explorer are captured by NetSpy using two standard network library tools, WinPcap [8] and LibNIDS [5]. WinPcap is an industry-standard, host-based packet-capturing

Score	Spyware?	Network Traffic Characteristics: Unseen Host	Network Traffic Characteristics: Packet Content	Examples
3	Most likely	Yes	Can be correlated to input	Browser Accelerator
2	Likely	Yes	No correlation to input	Mirar Toolbar
1	Least likely	No	Not important	MSN Messenger Toolbar, Weather Bug

Figure 5. Classification of untrusted programs based upon network traffic characteristics.

library for Windows. However, because WinPcap captures packets at the link layer and does not handle network-level protocols such as TCP/IP, NetSpy uses LibNIDS to provide features such as TCP-reassembly that are essential for inspecting packet content.

6. Evaluation

We evaluated the following aspects of NetSpy using several untrusted programs:

1. **False negatives.** To measure false negatives, we ran NetSpy on 7 known spyware programs that target Internet Explorer to determine (i) whether it correctly classified each of them as spyware, and (ii) whether it produced a succinct signature for the malicious substrate of each spyware's network behavior. In our experiments (discussed in detail below and summarized in Figure 6) NetSpy produced no false negatives. In particular, it correctly identified all 7 of the programs as spyware, and produced succinct and precise Snort signatures for them. It is also worth noting that just 2 out of the 7 spyware programs that we tested (namely, 180SearchAssistant and UCmore) have signatures in Bleeding Edge Snort, thus indicating that NetSpy can potentially help such efforts.
2. **False positives.** To measure false positives, we used NetSpy with 9 widely-used toolbars for Internet Explorer that are believed to be benign. NetSpy identified that 6 of these toolbars were indeed benign. As we discuss below, the remaining three toolbars (A9 Toolbar, Google Toolbar, and Yahoo Toolbar) showed network activity that was highly suggestive of spying behavior.

We also updated our Snort signature database with the signatures produced by NetSpy and ran the spyware in a test environment monitored by Snort. With this signature set, Snort was able to detect out-bound network traffic generated by the spyware programs.

6.1. Evaluating NetSpy with known spyware

We discuss a few examples of known spyware programs that NetSpy correctly detected and produced NIDS signatures for. For brevity, in each case we only show the relevant portion of the signature.

180SearchAssistant/Zango is a well-known spyware program that monitors browsing activity and displays pages related pages in a separate Internet Explorer window. For example, if a user visits `www.google.com` and searches for cell-phones, then 180SearchAssistant displays one or more pop-up windows containing cell-phone-related information.

NetSpy revealed that 180SearchAssistant contacted a site called `tvf.180solutions.com` and sent a `POST` message containing the web sites that the user was accessing. NetSpy's signature generator produced the following Snort signature for 180SearchAssistant, where the highlighted portion contains URLs or keywords (such as `mp3`) that the user invokes Internet Explorer with:

```
POST /showme.aspx?keyword=.*&&did=998&...
```

Note, in particular, that NetSpy's signature generator abstracts away the specific keyword to produce a regular expression that represents the malicious substrate of network activity generated by 180SearchAssistant.

AvenueMedia/InternetOptimizer (DyFuCA) is a Browser Helper Object (BHO). [1] It hijacks the error page displayed by Internet Explorer when a user tries to access an invalid URL, and replaces the page with advertisements related to the URL that the user entered. For example, if the user (mistakenly) visits the invalid URL `http://ipod/`, InternetOptimizer generates a page containing links to websites that sell iPod-related products.

NetSpy identified that InternetOptimizer contacteds the server `www.yoogee.com` to obtain advertisements. The corresponding NIDS signature that NetSpy generated was:

```
GET /searchresult/?lt=14&q=.*&cls=wsi12&...
```

To catch such spyware, we also include invalid URLs (e.g., `http://ipod/`) in the default input set used by NetSpy. In practice, we envision that the input set to NetSpy will be updated by users when they observe unusual network activity associated with certain URLs. NetSpy can then determine whether the network activity was in fact generated by a spyware program.

eXactSearchBar is a toolbar for Internet Explorer that, on startup, sends a `POST` message (shown below) to a remote server `checkin.exactsearchbar.com`.

```
POST /checkin2.aspx HTTP/1.0
```

It also automatically downloads updates from the server `files.exactserchbar.com` with messages matching the regular expression:

```
GET /Download/Update/.* HTTP/1.0
```

Finally, when a user visits an invalid URL, eXactSearchBar hijacks Internet Explorer's error page and displays advertisements related to the URL (similar to InternetOptimizer). Specifically, eXactSearchBar contacts the

[1] Browser Helper Objects, Browser Extensions and Toolbars are plugins that add to the functionality provided by Internet Explorer.

	Program Analyzed	Detected Behavior	Type (common to the group)	Signature Generated
Known Spyware Programs	180SearchAssistant/Zango	Monitors URLs visited; download ads/pages related to the URLs	Toolbar	✔
	Browser Accelerator		BHO, Toolbar	✔
	Side Find		BE, BHO	✔
	UCmore		Toolbar	✔
	AvenueMedia/InternetOptimizer	Hijacks Internet Explorer's error pages	BHO	✔
	eXactSearch Toolbar	Downloads/installs updates; Hijacks Internet Explorer's error pages	BHO, Toolbar	✔
	Mirar Toolbar	Monitors URLs visited; use SSL to send data and receive advertisements	Toolbar	✔
Other Programs	A9 Toolbar (No sign on)	Sends tracked URLs to `siteinfo.a9.com`	Toolbar	✔
	A9 Toolbar (Sign on)	Sends tracked URLs to `siteinfo.a9.com` and `client.a9.com`		✔
	Google Toolbar (Tracking off)	No network traffic detected	BHO, Toolbar	
	Google Toolbar (Tracking on)	Monitors URLs visited		✔
	Yahoo Toolbar	Monitors URLs visited	BHO, Toolbar	✔
	MSN Messenger Extension	No network traffic detected	BE	
	MSN Search Toolbar		BHO, Toolbar	
	Sun Java Console		BE, BHO	
	Sunshine Meta Toolbar		Toolbar	
	Weather Bug		Toolbar	
	Yahoo Messenger Extension		BHO	

Figure 6. Experimental results using NetSpy on several programs.

server `www.bestoftheweb.cc` sending requests matching the regular expression:

```
GET /errorpage/?src=404&url=.* HTTP/1.1
```

NetSpy's signature generator produced Snort signatures using each of the three regular expressions shown above. Note that NetSpy created three signatures for eXactSearchBar because it detected traffic going to three distinct hosts.

Mirar toolbar behaves like *180SearchAssistant* in that it displays advertisements related to the URL visited by a user. However, Mirar toolbar uses SSL to encrypt data that it sends to its home server.

While NetSpy identified the extra network activity generated by Mirar, it could not decipher the contents of the packets, because they were encrypted. Thus, NetSpy produced a signature based solely upon the host/port pair of the traffic generated by Mirar toolbar (`64-128-107-140.static.twtelecom.net:443` in our experiments).

One option to correlate network packets with user activity in such cases is to set up an SSL proxy server between the user's computer and the outside network. This proxy server can provide NetSpy with decrypted traffic for analysis.

6.2. Evaluating NetSpy with other programs

We evaluated NetSpy on 9 toolbars and browser extensions (Figure 6) that are widely-believed to be non-spyware programs. The goal was to determine if NetSpy would (erroneously) classify any of these popular programs as spyware.

NetSpy classified 6 of these programs as benign. The remaining three programs, A9 Toolbar, Google Toolbar, and Yahoo Toolbar, were (surprisingly) classified as spyware. Further investigation revealed that these three toolbars exhibited network behavior that was similar to the network behavior of the spyware programs tested. In fact, the End-User-License-Agreements (EULAs) from these three programs state that they track user activity to improve quality of service for their users. It is worth noting that NetSpy made the startling discovery that A9 Toolbar, despite claims in its EULA, contains behavior that may be considered undesirable by many users. A9 Toolbar is an Internet Explorer plug-in that is supposed to save URLs that a user visits on A9's home server, *under the user's account*. This requires the user to first sign in to A9's web site. We conducted an experiment to simulate user browsing activity on a machine with A9 Toolbar installed *but did not sign into A9's web site*. We had expected that A9 Toolbar would not generate any extra traffic. Instead, NetSpy found that A9 Toolbar sends URLs that the user has visited to a special server `siteinfo.a9.com`. When the user does sign in to A9's web site, A9 toolbar *additionally* sends data to `client.a9.com`, which we believe is the server that A9 uses to save URL history on behalf of the user.

We consider the above behavior undesirable because it tracks user activity even when a user is not logged into the service. Google Toolbar also has a similar feature, which when turned on, sends URLs that the user has visited to Google's home server. However, unlike with the A9 Toolbar, NetSpy does not detect extra network activity generated by Google Toolbar when this feature is turned off. This experiment leads us to conclude that it is sometimes difficult to classify a program as spyware. Activity that may be classified as spying behavior by some users may in fact be considered *desirable features* by others, as for example with A9 Toolbar, provided that A9 does not reveal/sell the URL history of each user to third parties.

6.3. Performance

We evaluated the performance of NetSpy to determine the time taken to (i) capture network traffic from a spyware-infected system, (ii) perform differential analysis, and (iii) generate NIDS signatures. All timing measurements were averaged over 10 runs.

Because NetSpy simulates user activity by running Internet Explorer with several URLs, it must wait for the web-page to load completely for each URL. Thus, the time to visit each URL depends on the contents of the web-page visited. For example, visiting `www.google.com` only generates two HTTP requests, and is quick (fewer than 2 seconds, on average), while visiting `www.apple.com` requires longer (10 seconds on average) because the Apple website contains more contents, including images and movies.

6.4. Desirable enhancements to the NetSpy prototype

Based upon our experiments, we have identified several desirable enhancements to NetSpy, that we plan to address in future work.

- NetSpy is currently designed to analyze one program at a time. That is, it must be invoked each time a new program is installed. It may not be as effective when a machine is infected with multiple spyware programs. The key difficulty here is that network traffic may be generated by any one of the multiple spyware programs installed. To enhance NetSpy, we need to associate network packets captured with the program that generated them.
- NetSpy has been implemented to detect spyware programs (specifically plug-ins and toolbars) that infect Internet Explorer. It currently cannot detect spyware programs that run as standalone processes (e.g., keyloggers).
- NetSpy currently observes only HTTP-based network traffic. Spyware programs that use other protocols (such as SMTP and FTP) may evade detection. We plan to extend NetSpy to monitor such traffic as well.

7. Evading NetSpy

There are several fundamental limitations of the approach adopted by NetSpy that a malicious attacker can use to evade detection.

1. A key factor that determines the effectiveness of NetSpy is the coverage of the input set used to collect network traffic. Spyware that generates network traffic only under certain inputs, not covered by NetSpy's set of inputs, will be undetected.

   There are two ways to overcome this shortcoming. The first is to use a crawler to automatically generate input URLs for use by NetSpy. The second is to analyze the (binary executable of the) untrusted program to recover specific URLs that trigger special behavior in the executable. However, we note that binary executable analysis is challenging and this solution may not always be feasible.
2. Time-triggered spyware programs can periodically contact their home server to report user activity. NetSpy, if only invoked during program installation, will miss the network activity generated by this program, thus misclassifying the program as benign.

   One way around is to make NetSpy an "always-on" tool that constantly monitors network activity to identify spyware behavior. NetSpy can also benefit from recent work on the analysis of time-triggered malware [13].
3. NetSpy currently classifies an untrusted program as spyware using the heuristic presented in Table 5. Spyware can evade detection by bypassing this heuristic. For instance, an attacker can use spyware to send data to one of the servers observed in the reference network statistics to evade detection. To retrieve stolen data, the attacker can intercept the network traffic on the link between the victim's network and the destination server.

   We believe that this attack is less feasible because it requires physical access to the link between the victim's network and the destination.

	Pattern-based	Behavior-based
Host-based	Most commercial solutions (e.g., [1, 7])	Kirda *et al.* [17]
Network-based	Bleeding Edge Snort [2] **NetSpy/spyware detection** (Section 4)	Web Tap [10], Siren [11] **NetSpy/spyware characterization** (Section 3)

Figure 7. Related work in spyware detection.

8. Related Work

Spyware-detection techniques fall into two main categories: host-based and network-based. Each of these can be further sub-categorized into pattern-based and behavior-based matching techniques, as shown in Figure 7.

Host-based techniques analyze untrusted binary executables to determine if they are potentially spyware. They work much like commercial virus scanners and search binary executables for known patterns of spyware. Commercial anti-spyware solutions, such as AdAware [1] and Spybot Search & Destroy [7] use simple techniques, such as comparing the MD5-hash of untrusted binary executables against known values to detect spyware. These techniques while fast and accurate—they have near-zero false positives—can only detect known spyware instances and are not resilient even in the face of simple obfuscations.

Recent work by Kirda *et al.* [17] addresses this shortcoming by proposing a *behavior-based approach*. Their work uses static analysis to analyze untrusted binary executables to detect spying behavior, e.g., by searching for the appearance of certain system calls or Internet Explorer API calls. This approach has the advantage of being able to detect *previously-unseen variants* of spyware that exhibit the same spyware behavior. However, it cannot detect novel spyware programs that differ in behavior from existing spyware programs.

BENDER [14] is a host-based malware detection tool designed to identify a class of malware (e.g., worms) that generate network traffic that is not dependent on user activity. Web Tap [10] is another tool that finds anomalous network traffic generated by certain spyware programs that operate without user input. It learns the characteristics of network traffic in a controlled training period and uses this information to find anomalous network traffic. One shortcoming of Web Tap is that it cannot detect spyware that "blends" with normal user activity, such as web surfing [11]. NetSpy is designed to detect spyware that generates network traffic that is dependent on user activity. We believe that NetSpy can complement BENDER and Web Tap in finding previously-unseen malware.

Siren [11] uses a behavior-based approach to detect spyware at the network-level. The key idea here, much like in NetSpy, is that network traffic generated by a spyware-infected system differs from that of a clean system. Siren uses a network-level detector to detect anomalous network traffic generated by a spyware-infected system. The key difference between NetSpy and Siren is NetSpy's ability to generate network-level signatures. In contrast, Siren focuses solely on detection, i.e., determining if a system is spyware-infected.

Automatic signature generation for other malware, such as worms and viruses, is an active research area [16, 18, 20, 21]. Among these, HoneyComb [18] and PAYL [21] use techniques similar to those used by NetSpy to generate NIDS signatures. HoneyComb uses the *Longest Common Substring (LCS)* algorithm to identify common patterns (substrings) within network packets captured by honeypots and generates NIDS signatures using the identified patterns [18]. Note that NetSpy uses *Longest Common Subsequence (LCSeq)* algorithm to create a regular expression suitable for an NIDS. PAYL generates worm signatures by observing both ingress and egress network traffic. In contrast, NetSpy only monitors outbound network traffic.

9. Conclusion

We presented NetSpy, an automatic spyware signature generator. NetSpy identifies if an untrusted program is spyware; if so, it generates network-level signatures that can be used with a NIDS that monitors outgoing network traffic. Experimental results show that NetSpy is effective and that it generates succinct, precise spyware signatures for NIDS.

References

[1] Ad-Aware. http://www.lavasoft.de.

[2] Bleeding Edge of Snort. http://www.bleedingsnort.com/.

[3] Bro Intrusion Detection System. http://bro-ids.org.

[4] IWebBrowser2 Interface. http://msdn.microsoft.com/workshop/browser/webbrowser/reference/ifaces/iwebbrowser2/iwebbrowser2.asp.

[5] LibNIDS 1.17 Win32 Port. http://www.checksum.org/.

[6] Snort. http://www.snort.org.

[7] Spybot Search & Destroy. http://www.safer-networking.org/.

[8] WinPcap. http://www.winpcap.org/.

[9] AOL/NCSA online safety study. http://www.staysafeonline.info/pdf/safety_study_v04.pdf, October 2004.

[10] K. Borders and A. Prakash. Web Tap: Detecting covert web traffic. In *11th ACM Conference on Computer and Communications Security*, October 2004.

[11] K. Borders, X. Zhao, and A. Prakash. Siren: Detecting evasive malware (short paper). In *IEEE Symposium on Security and Privacy*, May 2006.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[13] J. R. Crandall, G. Wasserman, D. Oliveira, Z. Su, S. F. Wu, and F. T. Chong. Temporal search: Detecting hidden malware timebombs with virtual machines. In *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, October 2006.

[14] W. Cui, R. H. Katz, and W. tian Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *ACSAC*, 2005.

[15] Internet Security Systems. ISS x-force database: spyware-7search-browser-accelerator(14221). http://xforce.iss.net/xforce/xfdb/14221.

[16] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, August 2004.

[17] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer. Behavior-based spyware detection. In *15th USENIX Security Symposium*, August 2006.

[18] C. Kreibich and J. Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Hotnets II*, November 2003.

[19] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A crawler-based study of spyware on the web. In *Networked and Distributed System Security Symposium*, February 2006.

[20] J. Newsome, B. Karp, and D. X. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, May 2005.

[21] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *RAID*, September 2005.