Profiling Self-Propagating Worms via Behavioral Footprinting

Xuxian Jiang
Dept. of Information and Software Engineering
George Mason University
Fairfax, VA 22030, USA
xjiang@ise.gmu.edu

Dongyan Xu
CERIAS and Dept. of Computer Science
Purdue University
West Lafayette, IN 47907, USA
dxu@cs.purdue.edu

ABSTRACT

This paper proposes behavioral footprinting, a new dimension of worm profiling based on worm infection sessions. A worm's infection session contains a number of steps (e.g., for probing, exploitation, and replication) that are exhibited in certain order in every successful worm infection. Behavioral footprinting complements content-based signature by enriching a worm's profile, which will be used in worm identification, an important task in post worm attack investigation and recovery. We propose an algorithm to extract a worm's behavioral footprint from the worm's traffic traces. Our evaluation with a number of real worms and their variants confirms the existence of worms' behavioral footprints and demonstrates their effectiveness in worm identification.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General – Security and protection (e.g., firewalls)

General Terms

Security

Keywords

Worms, Worm Profiling, Content Signature, Behavioral Footprinting

1. INTRODUCTION

In Internet worm defense, it is useful to create a complete, multi-facet profile for each worm. Such worm profiles can be used for worm identification, which is important to hold specific worms (and possibly worm authors) accountable for detected worm attacks and to recover from the damages inflicted by a specific worm. Worm identification is different from worm detection, in that the latter answers the question "is there a worm attack?" while the former answers the question "which worm is it?". A well established dimension of worm profiling is content-based fingerprinting [16, 17, 19, 22], which characterizes a worm by extracting the most representative content sequence(s). Though effective, the content-based dimension alone does not create a complete worm profile.

In this paper, we propose a new dimension of worm profiling called *behavioral footprinting*, which profiles worms based on their infection sessions. A worm's infection session contains a number of steps that are performed in certain order in every successful infection. Our approach is based on the key observation that the logic in a worm's implementation is different from that of the service or software being exploited by the worm. Moreover, it differs from other worms by exhibiting its "personalities" in terms of the target vulnerability, exploitation means, replication scheme, and payload features. For example, in its infection session, the MSBlaster worm [3] first exploits an *RPC-DCOM* vulnerability (MS03-026) with one TCP connection. Once successful, it creates a new port-binding shell service (*4444/TCP*) in the victim. It then replicates itself by using the *tftp* protocol. Unlike the MSBlaster worm, the Lion worm [2] first creates two TCP connections to exploit a *BIND* vulnerability (CA-2001-02). Once successful, the victim connects back to the infecter and replicates itself by using a new encapsulated HTTP connection. It does not create port-binding shell in the victim. As such, each worm has a unique behavioral footprint that differs from the normal service behavior as well as from that of other worms.

We have experimented with more than 10 real worms and their variants and extracted unique behavioral footprint for each of them[1]. Our experiments also show the effectiveness of behavioral footprints for worm identification from real-world traffic traces. Being orthogonal to the content-based dimension of worm profiling, behavioral footprint is by nature unaffected by attacks against content-based signatures such as content mutation [25]. The rest of this paper is organized as follows: Section 2 makes a case for worm behavioral footprinting. Section 3 describes behavioral footprint representation and extraction. Experimental results are presented in Section 4. Possible attacks and suggested solutions are described in Section 5. Section 6 discusses related work and Section 7 concludes this paper.

WORM'06, November 3, 2006, Alexandria, Virginia, USA.

[1]We in this work focus on worms that exploit vulnerable servers without any human intervention. We are currently evaluating other types of worms, such as mass-mailing, P2P, and IM worms.

2. A CASE FOR BEHAVIORAL FOOTPRINTING

2.1 Worm Infection Sessions

The infection session of a self-propagating worm (between the infecter and the victim) spans the following three phases:

(1) Target selection and probing Using an address scanning strategy (e.g. random or biased scanning), a worm attempts to pick a victim for infection. For example, an ICMP echo request packet or a TCP *SYN* probe can be used to infer the reachability of a victim. Additional packets may be used to obtain the version of a vulnerable service. Note that this phase may *not* exist for non-scanning worms (e.g. those that carry a pre-computed target list).

(2) Exploitation Once the worm receives a positive response from the victim, a number of malicious packets [2] may be sent to exploit the specific vulnerability. Successful exploitation will result in the execution of crafted code by the victim. This code is usually implemented differently in different worms.

(3) Replication If the exploitation is successful, the replication phase follows to transmit a worm replica to the victim. Once the replica is installed in the victim, the infection session is completed.

As an example, the MSBlaster worm [3] exploits an RPC-DCOM vulnerability (MS03-026). Figure 1 shows the steps in one infection session of the MSBlaster worm:

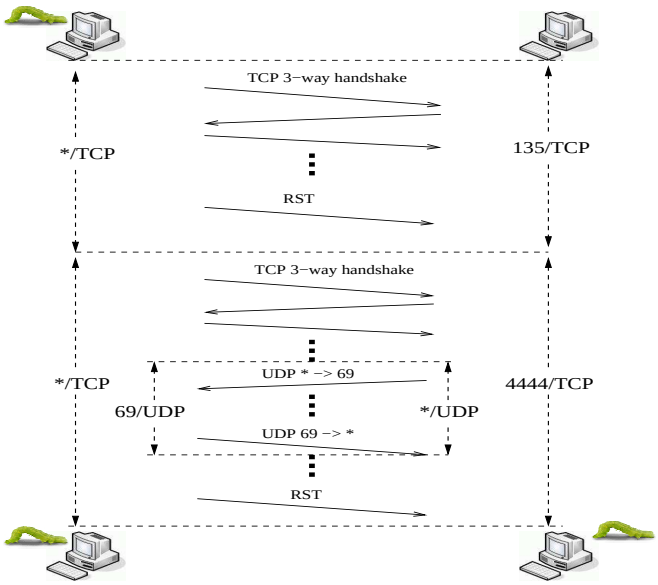

Figure 1: An Infection Session of the MSBlaster Worm

- A three-way TCP handshake on port 135 is used to check the reachability of the victim.
- Upon the establishment of the TCP connection, the worm sends a number of malicious packets to exploit the RPC-DCOM vulnerability using specially crafted attack code. If successful, the attack code will be executed by the victim. As a result, a new shell service will be started on TCP port 4444.
- The new shell service on $4444/TCP$ is immediately contacted by the worm to send instructions on how to download the worm replica, i.e., *msblast.exe*. The *tftp* protocol is used for the downloading.

The same infection session is repeated during the propagation of the MSBlaster worm. Other worms have their own infection sessions.

2.2 Behavioral Footprints for Worm Profiling

We are motivated to use the steps and their ordering during an infection session to profile the behavior of a worm.

First, there are intrinsic differences between a worm infection session and a normal access session to the vulnerable service. During the exploitation phase of a worm infection session, a worm will attempt to misuse a vulnerable service in a way that deviates from the normal access. In fact, several recent works such as Shield [26] rely on such difference to derive vulnerability-specific models. The replication phase of a worm infection session should not happen during a normal service access. Moreover, the temporal order of infection steps taken by a worm reflects the intrinsic dependencies that must be followed to ensure a successful infection.

Second, worm infection sessions reflect their respective implementations. Even for worms exploiting the same vulnerable service, their sequences of infection steps are different. This is because of the fact that the worms' implementations tend to have different exploitation means, replication idiosyncrasies, and carried-on payloads.

As a result, a worm's infection steps and their ordering during each infection session become valuable identity-revealing information. We are motivated to extract such information, which we call *behavioral footprint*, to profile the worm and use it for worm identification in post-attack analysis. This new dimension of worm profiling enriches the worm profile by complementing the existing content-based fingerprinting dimension. These two dimensions combined are expected to generate more complete worm profiles.

3. BEHAVIORAL FOOTPRINT REPRESENTATION AND EXTRACTION

In this section, we first define the representation of a worm's behavioral footprint. We then present an algorithm to extract a worm's behavioral footprint based on collected network traces that contain worm infection sessions.

3.1 Representation

We first break one infection session into different infection phases, each of which contains a number of flows (e.g. TCP, UDP, or ICMP connections). We identify each flow and present a sequence of flow-level actions as elements in the worm's behavioral footprint. For example, the behavioral footprint of the MSBlaster worm, based on its infection session in Figure 1, can be represented as $S_1\overleftarrow{S_1^A}A_1 \cdots R_1 S_2 \overleftarrow{S_2^A} A_2 \cdots \overleftarrow{U_1} U_1 \cdots R_2$, where

[2]There are some worms such as Slammer that might blindly send exploiting packets to any probed hosts.

$$
\begin{aligned}
S_1 &: < TCP, 4581/infecter, 135/victim, SYN > \\
\overleftarrow{S_1^A} &: < TCP, 135/victim, 4581/infecter, SYN, ACK > \\
A_1 &: < TCP, 4581/infecter, 135/victim, ACK > \\
R_1 &: < TCP, 4581/infecter, 135/victim, RST > \\
S_2 &: < TCP, 4599/infecter, 4444/victim, SYN > \\
\overleftarrow{S_2^A} &: < TCP, 4444/victim, 4599/infecter, SYN, ACK > \\
A_2 &: < TCP, 4599/infecter, 4444/victim, ACK > \\
\overleftarrow{U_1} &: < UDP, 1552/victim, 69/infecter > \\
U_1 &: < UDP, 69/infecter, 1552/victim > \\
R_2 &: < TCP, 4599/infecter, 4444/victim, RST >
\end{aligned}
$$

Each letter in the above behavioral footprint denotes either a TCP flow with different control bits (SYN, ACK, RST), an UDP flow (U), or an ICMP flow (I). The subscripts denote different flows. For example, $\overleftarrow{S_1^A}$ represents the second step (SYN and ACK bits set) in a normal three-way TCP handshaking procedure. The arrow sign is used to mark the traffic flow direction and can be omitted when there is no ambiguity. For convenience, a well-known *subsequence* can be denoted by a single letter. For example, a standard TCP 3-way handshake sequence (e.g., $S_i\overleftarrow{S_i^A}A_i$, $i = 1, 2$, in previous sequence) can simply be denoted as C_i.

Each letter in the behavioral footprint is further specified as a tuple with varying number of fields. In this example, the letter representing a TCP flow has four fields $< TCP, source\_port, dest\_port,$ TCP control bits $>$; while the letter representing a UDP flow has three fields $< UDP,$ $source\_port, dest\_port >$. We note that additional fields can be added to carry other "context" information such as the packet length, content signature, or timing information relative to the previous step. The design goal of such an extensible representation is to make it easier to integrate other worm profiling dimensions. Particularly, the content-based signature of a worm can be added, indicating the occurrence of a specific content signature during that corresponding infection step.

In addition, as a worm infection step might involve a non-deterministic port, a special *wildcard* field is introduced. Using the MSBlaster worm as an example, the source ports (e.g., the port 4581, 4599, 1552 in S_1, S_2, $\overleftarrow{U_1}$, respectively) vary in different infection sessions while the destination ports are the same (e.g., the port 135, 4444, 69 in S_1, S_2, $\overleftarrow{U_1}$, respectively). The special wildcard can be used for the source port field. As another example, the Witty worm has a fixed UDP source port 4000 and a random destination port. In this case, the wildcard is used to represent the destination port field. We also recognize that although a worm infection session usually involves only two nodes (infecter and victim), a *coordinated* worm infection might involve more than two nodes (e.g., downloading the worm replica from a third node). In this case, the wildcard field can be used to represent the infecter.

3.2 Extraction

A worm's behavioral footprint can be obtained in the following way: the raw infection session traces of the worm are first collected (e.g., using a honeyfarm [12]). The raw traces are then processed by an algorithm, which extracts the worm's behavioral footprint common in the traces. In this section, we present a pairwise alignment algorithm for the extraction.

Our algorithm is based on Needleman-Wunsch algorithm [9], which has been extensively used in bioinformatics research to find certain patterns in large sequences of strings such as DNA, RNA, and protein sequences. Note that any type of protein is a sequence of amino acid sub-units and there are only 20 different amino acids, which constitute the "alphabet" for protein sequence analysis. Similarly, if we consider all possible infection steps in a worm infection session as the alphabet, the behavioral footprint of a worm can be represented as a sequence of letters in the alphabet.

Given two infection traces $\mathscr{F}_1 = x_1x_2\cdots x_n$ and $\mathscr{F}_2 = y_1y_2\cdots y_m$, our algorithm achieves an optimal alignment between them. Based on a pre-defined scoring matrix (e.g., a match yields 1 while a mismatch yields 0), the alignment algorithm inserts gaps, if necessary, to achieve maximum alignment of the two sequences. The maximum alignment is defined as the sum of terms for each aligned pair of letters $< x_i, y_j >$ within the sequences (representing similarity $s(x_i, y_j)$), plus terms for each gap (representing a penalty p). The similarity and gap penalty are defined as a part of the scoring matrix and can be specific to different scenarios.

Based on our algorithm, a matrix $\mathscr{M}$, indexed by i and j with one index for each sequence, is iteratively constructed. The cell $\mathscr{M}(i, j)$ is the score of the best alignment between the initial segment $x_1x_2\cdots x_i$ of x up to x_i and the initial segment $y_1y_2\cdots y_j$ of y up to y_j. Initially, $\mathscr{M}(0, 0) = 0$, $\mathscr{M}(i, 0) = -ip$, $\mathscr{M}(0, j) = -jp$. The matrix is then iteratively filled from top-left cells to bottom-right cells based on Eqn.(1).

$$
\mathscr{M}(i,j) = \max \begin{cases} \mathscr{M}(i-1, j-1) + s(x_i, y_j), & i \geq 1, j \geq 1 \\ \mathscr{M}(i-1, j) - p, & i \geq 1 \\ \mathscr{M}(i, j-1) - p, & j \geq 1 \end{cases} \tag{1}
$$

Each case represents an option how current $\mathscr{M}(i, j)$ cell is derived from one of the other three cells (above-left $[i - 1, j - 1]$, above $[i - 1, j]$, or left $[i, j - 1]$). Once all values are calculated, the choices taken at each cell starting from the bottom rightmost one are *traced* back so that an optimal global alignment is derived. An example alignment applying our algorithm to a synthesized Welchia worm variant is shown in Figure 2.

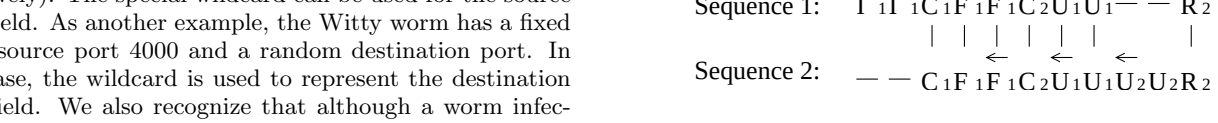

Figure 2: A pairwise alignment of two raw infection traces of the same worm. The choices made during the alignment are shown as "-" and "|". The "-" in the top sequence used as index i for $\mathscr{M}$ corresponds to the choice "above" $[i-1, j]$, the "-" in the bottom sequence used as index j for $\mathscr{M}$ represents "left" choice $[i, j-1]$, while the "|" in the middle shows the option "above-left" $[i - 1, j - 1]$.

Our experiments (Section 4) show that the alignment al-

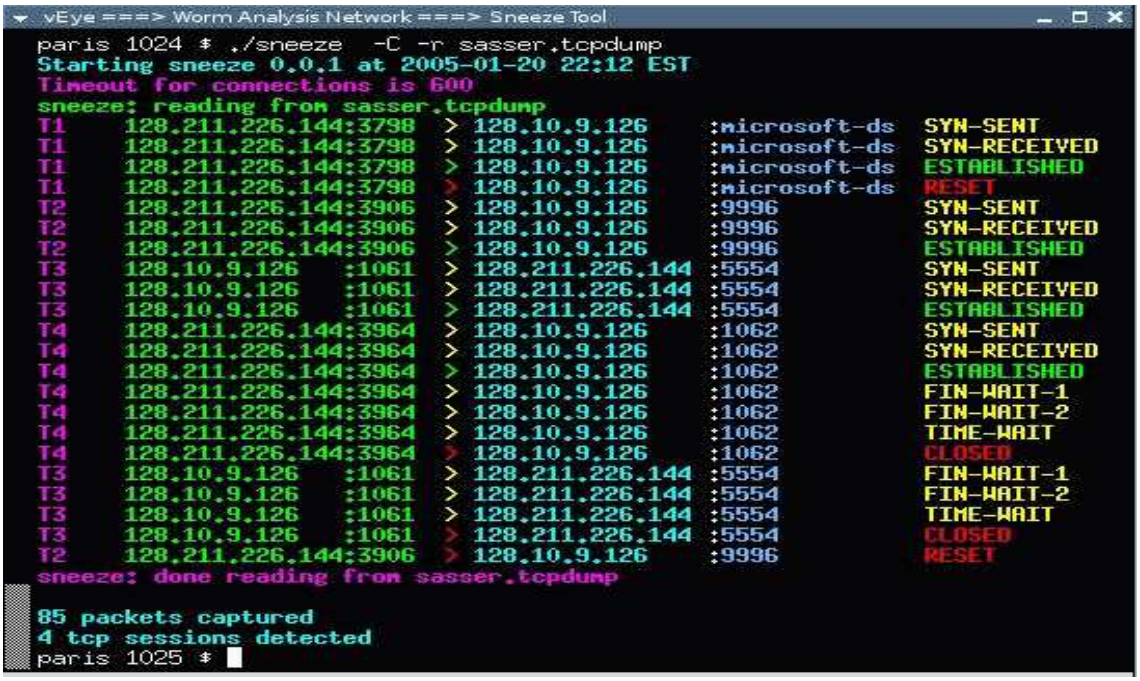

Figure 3: A Sample Output of Our Sneeze Tool

gorithm is highly effective in extracting behavioral footprints for all the worms (Table 1) we have experimented with. For more advanced worms that exhibit behavior polymorphism, a counterpart of content polymorphism, we have designed a tree-based algorithm that takes multiple infection traces as input and the details are presented in [13].

4. EVALUATION

In this section, we first demonstrate the existence of behavioral footprints of worms. We then apply behavioral footprints to worm identification during a post attack investigation based on real network traces.

4.1 Existence of Behavioral Footprints

We have accumulated a number of worm infection traces from two earlier projects: Collapsar [12] and vGround [14]. Each infection trace is collected as a separate *tcpdump* log file and is processed by *sneeze*, a tool we have developed for behavioral footprint extraction and worm identification. Sneeze extracts all TCP/UDP/ICMP flows in a complete infection session [3]. If necessary, it also performs packet reordering and reassembly. After the extraction, the TCP/UDP /ICMP flows are ordered based on time-stamps. The duration and payload size of each flow are also calculated by sneeze. An example output generated by sneeze after analyzing a successful Sasser worm [4] infection session is shown in Figure 3.

When analyzing different TCP segments in the same TCP flow, sneeze is able to track relevant TCP states. Specifically, TCP control packets with SYN, ACK, FIN, or RST bit set are recorded in the final representation. The TCP data packets (though ACK bit turned on) are usually ignored. However, as discussed in Section 3.1, unique payload content, or certain vulnerability-specific information can also be integrated here to enrich the accuracy and completeness of worm profiles.

Our pairwise alignment algorithm (Section 3) is then applied to extract behavioral footprints. The results are shown in Table 1. Each letter in the table represents either a TCP flow, a UDP flow, or an ICMP flow. The letter C_i represents the standard three-way TCP connection handshake process. Note that the same letter in different rows of the table contains different field values (e.g., destination port numbers) and are omitted for brevity.

We are able to reliably extract behavioral footprints for all worms investigated. The Welchia worm is similar to the MSBlaster worm except that an initial ICMP probing packet is generated before actual exploitation and the second TCP connection ($\overleftarrow{C_2}$) is initiated from the victim with connect-back attack code. Though the MSBlaster worm and the Welchia worm exploit the *same* vulnerability, their behavioral footprints are *different*. The Enbiei worm exhibits a footprint similar to that of the MSBlaster worm but has a different binary file and payload. The Sasser worm uses the *ftp* protocol ($\overleftarrow{C_3}$) to download a worm replica. Within the ftp session, a *PORT* primitive is initiated to start an embedded ftp-data session (C_4).

Table 1 also shows behavioral footprints of several historical worms. The Ramen worm is a multi-vector worm with three infection vectors (IVs): LPRng (CVE-2000-0917), wu-ftpd (CVE-2000-0573), and nfs-utils (CVE-2000-0666). For all three IV-specific behavioral footprints, the Ramen worm starts a TCP control packet with SYN and FIN bits (S_1^F) set, source port 21, and destination port 21 to probe victims.

[3]The current sneeze prototype assumes that each infection session contains an exploitation using one infection vector.

Name	*Infection Vector*	*Behavioral Footprints Derived*	Platforms
MSBlaster	RPC-DCOM vulnerability (MS03-026)	$C_1R_1C_2\overleftarrow{U_1}U_1R_2$	Windows
Welchia	RPC-DCOM vulnerability (MS03-026)	$I_1\overleftarrow{I_1}C_1F_1\overleftarrow{F_1}\overleftarrow{C_2}\overleftarrow{U_1}U_1\overleftarrow{U_2}U_2R_2$	Windows
Enbiei	RPC-DCOM vulnerability (MS03-026)	$C_1R_1C_2\overleftarrow{U_1}U_1R_2$	Windows
Sasser	LSASS vulnerability (MS04-011)	$C_1R_1C_2\overleftarrow{C_3}C4F_4\overleftarrow{F_4}F_3\overleftarrow{F_3}R_2$	Windows
Ramen	LPRng vulnerability (CVE-2000-0917)	$S_1^F\overleftarrow{S_1}R_1C_2F_2\overleftarrow{F_2}C_3\overleftarrow{C_4}F_4\overleftarrow{F_4}$	Linux
	WU-FTPD vulnerability (CVE-2000-0573)	$S_1^F\overleftarrow{S_1}R_1C_2R_2C_3R_3$	
	NFS-UTILS vulnerability (CVE-2000-0666)	$S_1^F\overleftarrow{S_1}R_1U_1\overleftarrow{U_1}U_2C_2\overleftarrow{C_3}F_3\overleftarrow{F_3}R_2$	
Lion	BIND vulnerability (CA-2001-02)	$C_1F_1\overleftarrow{F_1}C_2\overleftarrow{C_3}F_3\overleftarrow{F_3}R_2$	Linux
Slapper	OpenSSL vulnerability (CA-2002-23)	$C_1F_1\overleftarrow{F_1}C_2\overleftarrow{F_2}\prod_{i=3}^{22}C_iC_{23}C_{24}$	Linux
SARS	Samba vulnerability (CAN-2003-0201)	$U_1\overleftarrow{U_1}U_2\overleftarrow{U_2}C_1F_1C_2F_2\overleftarrow{F_2}C_3\overleftarrow{C_4}\overleftarrow{F_4}F_4R_3$	Linux/BSD

Table 1: Worms and Their Behavioral Footprints

4.2 Behavioral Footprints for Worm Identification

In this section, we demonstrate the effectiveness of behavioral footprints in worm profiling and identification. To this end, we perform a post-attack worm investigation using a 7-hour network trace ($80M$ bytes) collected from a live honeypot system that was successfully infected by 3 different worms.

As a comparison, we apply the popular content-based intrusion detection system *snort* side-by-side with our approach. The signature database used in snort has been updated to contain the latest content signatures for known intrusions. In the meantime we extend *sneeze* to recognize worm infection sessions based on extracted behavioral footprints. The results from snort and sneeze experiments are shown in Table 2 and Figure 4, respectively.

Table 2 shows that snort performs well in detecting ongoing attacks (e.g., RPC DCOM buffer overflow attacks) and reports numerous alerts such as "ICMP PING CyberKit 2.2 Windows". However, these alerts are separately raised even though they may be involved in the same worm infection session. Figure 4 shows the sneeze result. Sneeze naturally identifies 3 *successful* worm infections and also reports 2 *unsuccessful* worm infections. Further manual analysis shows that one unsuccessful worm infection has erroneously generated a *wrong* address (*192.168.1.59*) to download the worm replica while another unsuccessful infection has a flawed exploitation in binding the command shell service. As the *tftp* protocol is used for all these worms, we compare both outputs in this aspect. Table 2 reports four alerts with messages "TFTP GET" while Figure 4 further reports that one *tftp* is related to the Enbiei worm, one *tftp* is related to the MSBlaster worm, and the other two *tftp*s are attributed to the Welchia worm, which uses one *tftp* session to download the file $DLLHOST.exe$ (the worm payload) and the other *tftp* session for $SVCHOST.exe$ (a *tftpd* daemon).

Based on the above comparison, we observe that snort inspects every incoming/outgoing packet and raises an alert if a specific content-based signature is detected. However, it is not intended to correlate multiple alerts to identify the same worm infection session. Sneeze instead recognizes a worm's infection session and therefore is capable of identifying each individual worm.

Our other experiments [13] also show that snort becomes unable to identify worms when their contents are mutated or encrypted to evade content-based signatures. Sneeze is by nature unaffected by these attacks. In the next Section, we discuss some advanced attacks and possible countermeasures.

5. ATTACKS AND IMPROVEMENTS

Behavior substitution attack The goal of our algorithm is to achieve an optimal alignment between worm infection traces. An advanced worm could intentionally introduce *substitutable* subsequences to corrupt the alignment process while still achieving its goal of infection and propagation. This is a feasible attack. However, if we consider each substitution as a potential mutation, this attack is reminiscent of the classic challenge faced by biologists of how to optimally align gene sequences under possible mutations; and the solutions lie in enhancing the underlying scoring matrix. As a possible improvement, instead of simply returning either 0 or 1 when comparing two flows, the scoring matrix can be defined to return a normalized factor, indicating how likely these two flows are substitutable for each other.

Behavior-camouflaging attack In this attack, a worm author might attempt to hide true infection steps by injecting "noises" into the infection sequences. These noises, if not removed, may eventually pollute the final behavioral footprint generated. We could manually examine each infection session and only use those true infection steps for footprint analysis. However, this may be tedious and error-prone. How to automatically and accurately extract behavioral footprints from tainted sequences is our on-going research problem [4].

6. RELATED WORK

Our approach falls into the broader category of using attack behavior [1, 10, 11, 23] for intrusion profiling and identification. For example, [11] profiles the behavior of vulnerable programs and use them as a reference for detecting potential intrusions against systems. GrIDS [23] builds an activity graph based on machines activities and uses it to detect distributed intrusions. Assuming the existence of behavioral signatures of worm attacks, [10] focuses on the modeling of inter-machine communication patterns for worm

[4]A number of techniques might be helpful, such as protocol compliance analysis, semantic-aware tracking, as well as various data mining techniques.

	Snort Signature	*# Alerts*	*# Sources*	*# Dests*
1	NETBIOS DCERPC ISystemActivator path overflow attempt little endian	539	12	201
2	NETBIOS SMB-DS Session Setup And X request unicode username overflow attempt	15	1	1
3	NETBIOS SMB-DS DCERPC NTLMSSP asn1 overflow attempt	14	2	1
4	ICMP Source Quench	28	28	1
5	ICMP redirect host	27	1	1
6	TFTP Get	24	1	4
7	ICMP Large ICMP Packet	3	2	2
8	ICMP PING CyberKit 2.2 Windows	307551	33	153549
9	ICMP Destination Unreachable Communication Administratively Prohibited	156	2	1
10	SCAN UPnP service discover attempt	30	1	1
11	NETBIOS SMB-DS IPC$ share unicode access	6	3	1

Table 2: Intrusion Alerts by Snort

TIME	DUR.	SRC:PORT <-> DST:PORT	SERVICE	BYTES	SIGNATURE
Thu Nov 27 02:48:57 2003	8 s	81.168.168.127:4030 -> 128.10.9.127:135	135/TCP	1896 bytes	
Thu Nov 27 02:49:05 2003	25 s	81.168.168.127:4043 -> 128.10.9.127:4444	krb524/TCP	671 bytes	Enbiei Worm [sid]
Thu Nov 27 02:49:05 2003	21 s	128.10.9.127:1036 <-> 81.168.168.127:69	tftp/UDP	12134 bytes	
Thu Nov 27 04:58:24 2003	10 s	204.188.17.242:42948 -> 128.10.9.127:135	135/TCP	1908 bytes	MSBlaster/Enbiei Worm? [sid]
Thu Nov 27 04:58:35 2003	25 s	204.188.17.242:43362 -> 128.10.9.127:4444	krb524/TCP	349 bytes	
Thu Nov 27 06:08:24 2003	7 s	208.29.230.14:56429 -> 128.10.9.127:135	135/TCP	1572 bytes	MSBlaster/Enbiei Worm? [sid]
Thu Nov 27 06:08:30 2003	0 s	208.29.230.14:57021 -> 128.10.9.127:4444	krb524/TCP	78 bytes	
Thu Nov 27 07:23:46 2003	2 s	66.66.221.210:4581 -> 128.10.9.127:135	135/TCP	1896 bytes	MSBlaster Worm [sid]
Thu Nov 27 07:23:49 2003	17 s	66.66.221.210:4599 -> 128.10.9.127:4444	krb524/TCP	666 bytes	
Thu Nov 27 07:23:49 2003	11 s	128.10.9.127:1552 <-> 66.66.221.210:69	tftp/UDP	6372 bytes	
Thu Nov 27 08:03:55 2003	0 s	128.10.16.220 -> 128.10.9.127	ICMP	64 bytes	
Thu Nov 27 08:03:55 2003	0 s	128.10.9.127 -> 128.10.16.220	ICMP	64 bytes	
Thu Nov 27 08:03:55 2003	0 s	128.10.16.220:1567 -> 128.10.9.127:135	135/TCP	436 bytes	
Thu Nov 27 08:03:55 2003	4 s	128.10.9.127:3912 -> 128.10.16.220:707	707/TCP	1686 bytes	Welchia Worm [sid]
Thu Nov 27 08:03:56 2003	0 s	128.10.9.127:3953 <-> 128.10.16.220:69	tftp/UDP	40 bytes	
Thu Nov 27 08:03:56 2003	0 s	128.10.9.127:3953 <-> 128.10.16.220:1605	1605/UDP	20196 bytes	
Thu Nov 27 08:03:56 2003	0 s	128.10.9.127:3954 <-> 128.10.16.220:69	tftp/UDP	40 bytes	
Thu Nov 27 08:03:56 2003	0 s	128.10.9.127:3954 <-> 128.10.16.220:1606	1606/UDP	10488 bytes	

Figure 4: Worm Profiling and Identification by Sneeze

activity detection. However, it does not address the problems of formal representation and extraction of behavioral signatures, which are the main focus of our work.

Another related technique, anomaly detection [8, 15, 18, 27], leverages the insight that worms are likely to exhibit anomalous behavior such as port scanning [15] and failed connection attempts, which are different from the normal behavior. Although such approach has been demonstrated effective in detecting worm infection (i.e. "is there a worm infection?"), it is not intended to identify worms (i.e. "which worm is this?").

Content-based fingerprinting [16, 17, 19, 22] is an extensively studied dimension of worm characterization by extracting the most representative worm-identifying content sequence(s). Previously, it was often a manual process. A number of systems [16, 17, 22, 19] have recently been developed to *automate* the task of extracting representative content sequences. Our approach complements content-based fingerprinting by capturing a worm's behavior signature. The two approaches can be naturally integrated to create a more complete, multi-facet worm profile.

Other related approaches include vulnerability-specific characterization [5, 26] and semantic-aware taintedness tracking [6, 7, 20, 21, 24]. Shield [26] proposes the notion of vulnerability-specific signature and uses it to accurately filter out attack flows. TaintCheck [20], Minos [7], Vigilante [6], and others [21, 24] enable the detection of unknown attacks by associating a tag to untrusted information sources and reporting an alert if a tainted instruction is executed. These schemes are also applicable to the detection of unknown attacks. While being able to detect the occurrence of an exploitation, they do not aim at characterizing the *entire* worm infection session where exploitation is only one phase of the session.

7. CONCLUSION

We have presented a new dimension – behavioral footprinting – to profile self-propagating worms. Orthogonal and complementary to existing dimensions, behavioral footprinting characterizes worm infection steps and their order in every worm infection session. A pairwise alignment algorithm is proposed to extract a worm's behavioral footprint from raw traffic traces. Our experiments with real-world worms confirm the existence of behavioral footprints and demonstrate their effectiveness in worm identification.

Acknowledgments

We thank the anonymous reviewers for their valuable feedbacks and suggestions. We also thank Dr. Pei Cao and Xiaoxin Wu for their comments on an earlier version of this paper. This work was supported in part by a gift from Microsoft Research and grants from the National Science Foundation (OCI-0438246, OCI-0504261, CNS-0546173).

8. REFERENCES

[1] Snort. *http://www.snort.org*.

[2] Lion Worms. *http://www.sans.org/y2k/lion.htm*, 2001.

[3] MSBlaster Worms. *http://www.cert.org/advisories/CA-2003-20.html*, 2003.

[4] Sasser Worms. *http://www.microsoft.com/security/incident/sasser.asp*, 2004.

[5] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards Automatic Generation of Vulnerability-Based Signatures. *Proceedings of the 27th IEEE Symposium on Security and Privacy*, May 2006.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-End Containment of Internet Worms. *Proceedings of ACM SOSP 2005*, Oct. 2005.

[7] J. R. Crandall and F. T. Chong. Minos: Control Data Attack Prevention Orthogonal to Memory Model. *Proceedings of 37th International Symposium on Microarchitecture*, Oct. 2004.

[8] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. *Proceedings of RAID 2004*, Sept. 2004.

[9] R. Durbin, S. Eddy, and A. Krogh. *Biological Sequence Analysis*. Cambridge University Press, ISBN: 0521629713, 1998.

[10] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia. A Behavioral Approach To Worm Detection. *Invited talk in ACM WORM 2004*, Oct. 2004.

[11] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Learning Program Behavior Profiles for Intrusion Detection. *Proceedings of the 1999 Workshop on Intrusion Detection and Network Monitoring*, Apr. 1999.

[12] X. Jiang and D. Xu. Collapsar: A VM-Based Architecture for Network Attack Detention Center. *Proceedings of the 13th USENIX Security Symposium*, Aug. 2004.

[13] X. Jiang and D. Xu. Behavioral Footprinting: a New Dimension to Characterize Self-Propagating Worms. *Department of Computer Science Technical Report CSD TR 05-027, Purdue University*, Jan. 2005.

[14] X. Jiang, D. Xu, H. J. Wang, and E. H. Spafford. Virtual Playgrounds for Worm Behavior Investigation. *Proceedings of RAID 2005*, Sept. 2005.

[15] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. *Proceedings of the 25th IEEE Symposium on Security and Privacy*, May 2004.

[16] H. A. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. *Proceedings of the 13th Usenix Security Symposium*, Aug. 2004.

[17] C. Kreibich and J. Crowcroft. Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. *ACM SIGCOMM Computer Communication Review*, Jan. 2004.

[18] M. E. Locasto, J. J. Parekh, A. D. Keromytis, and S. J. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. *Proccedings of the 6th Annual IEEE Information Assurance Workshop*, June 2005.

[19] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. *Proceedings of the 26th IEEE Symposium on Security and Privacy*, May 2005.

[20] J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. *Proceedings of NDSS 2005*, Feb. 2005.

[21] M. Rinard, C. Cadar, D. Dumitran, D. Roy, and T. Leu. A Dynamic Technique for Eliminating Buffer Overflow Vulnerabilities (and Other Memory Errors). *Proceedings of ACSAC*, Dec. 2004.

[22] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. *Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design & Implementation*, Dec. 2004.

[23] S. Staniford and et al. The Design of GrIDS: A Graph-Based Intrusion Detection System. *UCD Technical Report CSE-99-2*, Jan. 1999.

[24] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure Program Execution via Dynamic Information Flow Tracking. *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[25] G. Vigna, W. Robertson, and D. Balzarotti. Testing Intrusion Detection Signatures Using Mutant Exploits. *Proceedings of the 11th ACM Conference on Computer and Communication Security*, Oct. 2004.

[26] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. *Proceedings of ACM SIGCOMM 2004*, Sept. 2004.

[27] K. Wang and S. J. Stolfo. Anomalous Payload-based Network Intrusion Detection. *Proceedings of RAID 2004*, Sept. 2004.