Analyzing Malware Detection Efficiency with Multiple Anti-Malware Programs

Jose Andre Morales
Software Engineering Institute
Carnegie Mellon University
Email: jamorales@cert.org

Shouhuai Xu
Department of Computer Science
University of Texas at San Antonio
Email: shxu@cs.utsa.edu

Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio
Email: ravi.sandhu@utsa.edu

ABSTRACT

Commercial anti-malware programs have become mainstream security products and are widely deployed. In practice, perhaps due to economic factors, users may only deploy a single anti-malware program. It has been proven that there is no universally effective anti-malware program, which effectively bases malware defense on the *implicit* assumption that at least the popular anti-malware programs can provide sufficient security. This assumption has been neither justified nor examined in a systematic fashion. In this paper, we propose a methodological framework for examining this assumption. We define an anti-malware program to be *competent* when it detects and cleans all malware present on a system. Our initial experiments demonstrate that a single anti-malware program is not sufficient. It is challenging to figure out how many anti-malware programs are needed in order to achieve competence in the majority of malware scenarios, although our experimental results can serve as a good starting point toward ultimately answering the question.

I INTRODUCTION

Commercial anti-malware program is the most popular defense tool used by desktops, laptops, and mobile devices. Perhaps because of economic reasons, end users have the (implicit) expectation that these programs will provide comprehensive protection, specifically the effective detection and treatment of malware. As a consequence, end users seemingly trust an anti-malware program to safeguard their system and assume malware is automatically and effectively treated when malware is detected. Unfortunately, there has been no systematic study to (in)validate the underlying assumption: a single anti-malware program is indeed sufficient to defend against malware even though it is widely perceived that no single anti-malware program can provide 100% detection and treatment effectiveness. If one anti-malware program is seemingly insufficient, then how many anti-malware programs are needed in order to protect a computer against malware? Motivated by this question, in this paper we make the following contributions. First, we propose a methodological framework for examining the detection and treatment effectiveness of anti-malware programs. We define an anti-malware program to be *competent* if it detects and cleans all malware present on a system. Second, we conduct two experiments, each dealing with three anti-malware programs installed together on a system. Our results revealed, in several cases, that malware was still detected on a system after performing detection and treatment with multiple anti-malware programs. Experimental results (i) reaffirm the widely accepted belief that one anti-malware program is not sufficient to defend against malware, and (ii) suggest that having multiple programs, or one program with multiple detection engines, installed on a system does not necessarily guarantee complete protection. Our experimental results show that competence was mostly achieved in simple scenarios with anti-malware programs installed before malware penetration. The results also reveal several issues in an anti-malware program's self-defense mechanism against attacks that disallow proper installation and/or execution in a compromised system. We found detection effectiveness varies based the execution order of multiple anti-malware programs.

The main focus of the paper is to *qualitatively* reaffirm the widely accepted belief that one anti-malware program is not sufficient, while making an initial effort to address the quantitative question — how many anti-malware programs are needed — by forecasting a suitable amount based on our experimental results.

Related Work. Cohen's formal study of computer viruses shows that it is in general undecidable to determine whether a given piece of code is a computer virus [7]. This argument certainly applies to modern malware. The implication of Cohen's impossibility result is that we need to keep developing new detection tools for emerging new computer viruses/malware. Adleman showed that a certain class of computer virus infections can be disinfected [1] but others cannot. Adleman's impossibility re-

ISBN: 978-1-62561-001-0

©ASE 2012

sult justifies the importance of studying detection and treatment effectiveness of specific anti-malware programs with respect to specific types of malware. In this paper, we consider the detection accuracy and treatment effectiveness of multiple anti-malware programs against diverse malware.

A large body of work related to our research exists in the literature, which presents a broad set of approaches to tackling the problem of detecting malware. A sample of this body of work is [5,6,8,9,11,12, 14, 15, 17, 19–21]. However, these investigations only emphasize detection accuracy. There also have been attempts at standardizing the testing and evaluation of anti-malware programs [2–4, 10, 13, 16, 18], which intend to address general notions such as reliable and transparent testing with standardized output. Unfortunately, there has been no well accepted evaluation methodology and the issue of treatment effectiveness has not been considered until now.

Online services, such as `virustotal.com`, are used to assess the signature-based detection accuracy of several anti-malware programs on submitted files. We execute samples in a realistic environment which facilitates infection across the system including files and processes. This facilitates both signature and behavior-based detection, which are seemingly a more realistic evaluation of an anti-malware program's competence than a signature-based static scan against a single submitted sample. In this paper, we aim to verify an anti-malware program's detection accuracy and treatment effectiveness by examining and observing multiple anti-malware programs which are installed together and perform detection scanning in a sequential manner. If a malware is detected, it implies that the previous program(s) was/were not competent due to failing in either the detection or the treatment of malware.

The rest of the paper is organized as follows: Section II presents our methodological framework for experiments to evaluate the effectiveness of anti-malware programs. Section III describes our experiments and results. Section IV discusses conclusions and future work.

II METHODOLOGY OF EXAMINING THE EFFECTIVENESS OF ANTI-MALWARE PROGRAMS

Now we present a methodological framework for addressing the question — how many commercial anti-malware programs are needed in order to protect an end user's computer against malware? Unfortunately, there is a universal solution as what has been proven [7]. As such, we necessarily have to pursue particular detection solutions to particular malware programs while bearing in mind that false positives and false negatives are possible.

Specifically, let $\{C_1(\cdot), \ldots, C_n(\cdot)\}$ be a set of available anti-malware programs (or functions). Suppose each C_i, $1 \leq i \leq n$, takes as input a computer object S (e.g., a single computer file or directory, one or multiple computer processes, or a whole system in a clean or infected state) during each instance execution of C_i. Therefore, $C_i(S)$ is the output after running $C_i(\cdot)$ against S, which may include (i) whether S was infected[1], and (ii) S' where $S' = S$ means that S was deemed as not infected or deemed as infected but not treated (possibly because C_i did not know the proper treatment), and $S' \neq S$ means S was deemed as infected and was treated with outcome S'.

For a given set of anti-malware programs $\{C_1(\cdot), \ldots, C_n(\cdot)\}$, there are $n!$ permutations on their sequential scanning order. With respect to a given object S and a given specific scanning order denoted (without loss of generality) by $(C_1(\cdot), \ldots, C_n(\cdot))$, our methodology is illustrated in Fig. 1 and elaborated as follows.

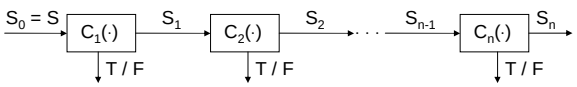

Fig. 1: Experimental methodology

More specifically, we install $(C_1(\cdot), \ldots, C_n(\cdot))$ and initially disable their scanning ability. We then enable and run $C_1(S_0)$, where $S_0 = S$, to obtain an indicator whether S_0 was infected and S_1 (which may or may not be equal to S_0). Then we disable the scanning ability of C_1 and enable and run $C_2(S_1)$, where S_1 is output by C_1, to obtain an indicator whether or not S_1 was infected and S_2 (which may or may not be equal to S_1). For general i, we enable the scanning ability and run $C_i(S_{i-1})$ to obtain an indicator whether or not S_{i-1} was infected and S_i (which may or may not be equal to S_{i-1}). Finally, we enable and

[1]This can include identification of which malware infected S. We omit such identification for the sake of conciseness.

ISBN: 978-1-62561-001-0

©ASE 2012

run $C_n(S_{n-1})$ to obtain an indicator whether S_{n-1} was infected and S_n.

Let us define predicate $\mathsf{DT}(C_i(S_{i-1}))$, which outputs $T = true$ if C_i has detected at least one object as being malware in input S_{i-1} and $F = false$ if no malware was detected. In other words, this predicate serves as the indicator of C_i's detection capability. Let us also define predicate $\mathsf{SDT}(C_i(S))$, which outputs $T = true$ if and only if C_i has successfully detected and treated all malware present in an input S. In other words, this predicate serves as the indicator of C_i treatment effectiveness against detected malware.

We say anti-malware program C_1 is *competent*, denoted by $\mathsf{SDT}(C_1(S)) = T$, if for *every* input S it holds that

$$(\mathsf{DT}(C_1(S_0))=T)\wedge(\mathsf{DT}(C_2(S_1))=F)\wedge\ldots\wedge(\mathsf{DT}(C_n(S_{n-1})=F)$$

where $S_0 = S$. This means that $\mathsf{SDT}(C_1(S)) = T$ if and only if malware was detected in $S_0 = S$ but was not detected in S_i for $1 \leq i \leq n$, where S_i is the output by C_i on input S_{i-1}. Intuitively, for any scanning permutation $(C_1(\cdot), \ldots, C_n(\cdot))$ of the n anti-malware programs, $\mathsf{SDT}(C_1(S)) = T$ for all S means that C_1 is competent. If malware is detected in S_i for some $i > 0$, then C_1 is not competent. If there is no C_i for some $1 \leq i \leq n$ that is competent (i.e., no such program in all the $n!$ possible permutations), then we need multiple anti-malware programs. In this case, we can further define the minimal number of needed programs. For example, consider also the scanning permutation $(C_1(\cdot), \ldots, C_n(\cdot))$, even if $\mathsf{SDT}(C_1(S)) = F$, $C_1(\cdot)$ and $C_2(\cdot)$ together are competent when

$$(\mathsf{DT}(C_1(S_0))=T)\wedge(\mathsf{DT}(C_2(S_1))=T)\wedge\ldots\wedge(\mathsf{DT}(C_n(S_{n-1})=F)$$

where $\mathsf{DT}(C_i(S_{i-1})) = F$ for $3 \leq i \leq n$. The above discussion can be extended to cases of the $n-1$ anti-malware programs as competent.

Our experimental study considers the special case of $n = 3$. In our given methodology, a false negative can impact our methodology in the following manner: Assume we have $(C_1(S_0), C_2(S_1), C_3(S_2))$ and $\mathsf{SDT}(C_1(S_0)) = True$. This implies both $C_2(S_1)$ and $C_3(S_2)$ returned $False$. It is possible for $C_2(S_1)$ and $C_3(S_2)$ to produce a false negative on some malware that was not initially detected by $C_1(S_0)$. In this case, $\mathsf{SDT}(C_1(S_0)) = T$ is incorrect as the result should be F for all three C_i.

Our methodology works properly in cases where false negative maximally occurs in $n-1$ anti-malware programs meaning that there are at lease one C_i, which detect the malware that was missed by all the other programs in the set. If every anti-malware program in a set n produces a false negative, then the result is unreliable. In general, to determine whether or not C_i has possibly produced a false negative on an input S_n, the output S_{n+1} must be submitted to C_{i+1} for detection and treatment of malware. If sufficient number of C_i's report no presence of malware starting with a given $S = S_0$ then it may be possible to conclude that S is in fact malware-free. Setting a threshold value for the number of anti-malware programs that are needed in order to draw this conclusion is a core issue of the framework.

III EVALUATION EXPERIMENTS AND RESULTS

Guided by our methodology while considering the feasibility of experiments, we consider two sets with each consisting of three anti-malware programs. For a given set of anti-malware programs, denoted by $\{C_1(\cdot), C_2(\cdot), C_3(\cdot)\}$, we consider all possible scanning permutations, resulting in $3! = 6$ cases. For a specific order $(C_1(\cdot), C_2(\cdot), C_3(\cdot))$, C_1 detects and treats malware possibly present in a given $S = S_0$, then we submit S_1, namely the output of C_1, as input to $C_2(\cdot)$. If C_2 detects the presence of a malware in S_1, then C_1 failed in detection and/or treatment of the malware present in S_0. Similarly, in order to determine if C_2 effectively detected and treated malware present in S_1, we submit S_2, which is the output of C_2, as input to $C_3(\cdot)$. If $C_3(\cdot)$ detects the presence of a malware in S_2, then C_2 failed in detection and/or treatment of malware present in S_1. The complete scanning sequence must be performed even in cases where a C_i does not detect any malware. This is because C_i could have produced a false negative, which may not be noticed until after the rest of the scanning sequence is completed. In the above experiments, all three programs of a set are installed together. Once installation is completed, their scanning abilities are disabled. When performing a scanning permutation, only the scanning ability of C_i is enabled. Once scanning completes, the scanning ability of C_i is again disabled. This is to ensure that only one C_i is capable of malware detection at any given time during our tests. In this approach, C_2 verifies C_1 scanning results and C_3 verifies C_2. C_3 is not verified, thus we consider C_1 and C_2 results verified as true, and C_3 results true but unverified. Our analysis will focus mostly on C_1 and C_2 results.

ISBN: 978-1-62561-001-0

©ASE 2012

1 EXPERIMENT DESIGN

Sets of anti-malware programs. Testing was performed using two sets of three anti-malware programs each, labeled as EAZ and KGB respectively as shown in Table 1. Their scan results, documented in log files, were the basis of determining if malware was detected and treated. The 6 programs were chosen and grouped into sets based on the facility of being installed together. Each anti-malware program was a free trial version, which was installed and fully updated before each experiment. All instructions given by the program during installation, detection and treatment were followed. In cases where an anti-malware program asked the user to choose treatment, we chose the closest equivalent treatment options in this order: disinfection, quarantine, deletion. As mentioned above, during the experiments, we performed detection database updates for every tested anti-malware program after initial installation and before each experiment to guarantee the latest detection information was in use.

1st Anti-malware set - EAZ	**2nd Anti-malware set - KGB**
ESET Smart Security	Kaspersky Internet Security
AVG Internet Security	G-Data Internet Security
ZoneAlarm Extreme Security	BitDefender Total Security

Table 1: Anti-malware sets used in Experiments 1 and 2

Experiment implementation. All experiments were performed on VMWare Workstation running a snapshot containing a clean and fully updated install of Windows 7, 32 bit operating system. As a preliminary step, each anti-malware program used in testing was installed and performed a full system scan in this clean state. None of the programs detected any malicious objects, this was done to ensure an initial malware free testing environment. We implemented two experiments (with respect to each of the two sets of anti-malware programs) as elaborated below.

Experiment 1. All three anti-malware programs of a set were installed together and their scanning abilities disabled in a system with a malware-free state followed by the execution of one known malware sample for 3 minutes. Each anti-malware program was allowed to perform detection and treatment in its default manner followed by a user requested full system scan. Once completed, the program's scanning ability was disabled and the next anti-malware program in the set was enabled and requested to perform a full system scan. This same step was taken with the final anti-malware program in the set. Once each program completed its detection and treatment of malware, we recorded if the program detected any malware on the system along with the total number of detected malicious objects. The steps are highlighted in Figure 2.

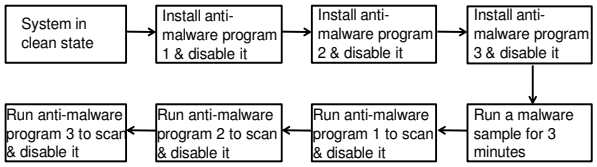

Fig. 2: Experiment 1 steps

Experiment 2. One known malware sample was executed for 3 minutes with the system initially in a clean state. After 3 minutes all three anti-malware programs of a set were installed with their scanning ability disabled and each allowed to perform detection and treatment in its default manner followed by a user requested full system scan. Once completed, the program's scanning ability was disabled and the the scanning ability of the next anti-malware program in the set was enabled and requested to perform a full system scan. This same step was taken with the final anti-malware program in the set. Recording malware detection for each anti-malware program was also done in the same fashion as in Experiment 1. The steps are highlighted in Figure 3.

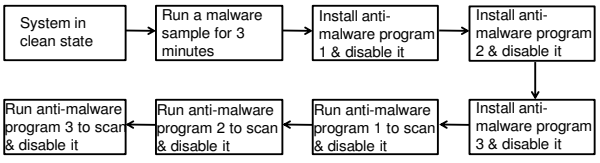

Fig. 3: Experiment 2 steps

Malware samples used in the experiments. For Experiments 1 and 2, a test set consisting of 500 known malware samples were used. The malware used in the experiments were randomly chosen from 974 samples downloaded between August 2010 and November 2010 from the GFI SandBox malware repository. The specific download date was chosen to not be recent so as to give the anti-malware program maintainers time to incorporate detection and treatment of the malware samples. The sets consisted of network worms, peer-to-peer worms, email

©ASE 2012
ISBN: 978-1-62561-001-0

viruses, rootkits, bots, password stealers, malware downloaders, and backdoors. Several malware samples included in the set such as bots, backdoors, and malware downloaders initially infect a system with one malware which may then download and/or install several other malware programs. This produces a compromised system with several malicious objects consisting of many different malware types.

We infected our system by executing one malware sample per experiment and allowing the sample to run for three minutes. This infection approach was used to evaluate an anti-malware program's effectiveness and resilience in a compromised system containing a possibly high number of malicious objects. Our approach created a realistic infected environment closely emulating current malware trends. Most anti-malware programs perform multiple diurnal detection database updates, thus we assume that given the test set download date of 2010, sufficient time had passed for the anti-malware program to detect, inspect, and create detection signatures and/or rules for the malware samples used in the experiments.

2 EXPERIMENT 1 RESULTS

Scanning permutation	$\mathsf{DT}(C_1(\cdot))$	$\mathsf{DT}(C_2(\cdot))$	$\mathsf{DT}(C_3(\cdot))$
EAZ	500	13	0
EZA	500	8	4
ZEA	500	0	0
ZAE	500	0	0
AEZ	500	6	0
AZE	500	7	2
KGB	500	0	0
KBG	500	0	0
GBK	500	0	3
GKB	500	6	0
BKG	500	7	1
BGK	500	2	4

Table 2: Experiment 1 results for $\mathsf{DT}(C_n)$

The detection results (DT) of Experiment 1 are listed in Table 2 and the competence results (SDT) are listed in Table 3. The first column of each table is the scanning permutation being evaluated. In Table 2, columns 2,3, and 4 report the total number of samples in which $DT(C_n(\cdot)) = T$ for $C_1(\cdot)$, $C_2(\cdot)$, and $C_3(\cdot)$ respectively. For example, in Table 2, the first permutation listed is EAZ with $DT(C_1(\cdot)) = 500$ means that of the tested 500 samples in our set, $DT(C_1(\cdot))$, which in this case is $DT(E(\cdot))$, detected at least one object as malicious in every sample in the set. $DT(C_2(\cdot)) = 13$ indicating in only 13 of 500 samples did $DT(C_2(\cdot))$, which is $DT(A(\cdot))$, detect an object as malicious. $DT(C_3(\cdot)) = 0$, means that $DT(Z(\cdot))$ did not detect a malicious object in any of the 500 samples.

Scanning permutation	$\mathsf{SDT}(C_1(\cdot))$	$\mathsf{SDT}(C_1 \wedge C_2)$	$\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$
EAZ	487	13	0
EZA	488	8	4
ZEA	500	0	0
ZAE	500	0	0
AEZ	494	6	0
AZE	493	5	2
KGB	500	0	0
KBG	500	0	0
GBK	497	0	3
GKB	494	6	0
BKG	493	6	1
BGK	494	2	4

Table 3: Experiment 1 results for $\mathsf{SDT}(C_1 \wedge ... \wedge C_n)$

In Table 3, columns 2, 3, and 4 report the total number of times $\mathsf{SDT}(C_1 \wedge ... \wedge C_3) = T$ for a given permutation, representing the total number of samples for which competence was achieved. More specifically for each permutation in column 1, column 2 reports the total number of samples in our data set which competence was achieved by $\mathsf{SDT}(C_1)$, column 3 reports competence for $\mathsf{SDT}(C_1 \wedge C_2)$, and column 4 for $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$. For example, in Table 3, the first permutation listed is EAZ with $\mathsf{SDT}(C_1) = 487$ which means competence was achieved by $\mathsf{SDT}(E)$ in 487 out of 500 tested samples. Competence was established by assuring these 487 samples were not detected by C_2 = A and C_3 = Z thus validating the complete detection and treatment of the 487 samples by C_1 = E. Continuing with the example of permutation EAZ, $\mathsf{SDT}(C_1 \wedge C_2) = 13$, which means 13 samples were detected and treated by C_1 and C_2 and not detected at all by C_3. The implication is these 13 samples were detected by C_1 and possibly not treated correctly leading to their re-detection by C_2. These 13 samples not being detected by C_3 implies C_2 succeeded in detection and treatment thus competence was achieved only after being detected and treated by both C_1 and C_2. The final result for EAZ is $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3) = 0$ which implies C_3 did not detect any malicious objects for any of the 500 tested samples. Since there is no fourth anti-malware

©ASE 2012
ISBN: 978-1-62561-001-0

program in our permutations to compare competence results against $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$, we must accept these results as true but unverified, as opposed to $\mathsf{SDT}(C_1)$ and $\mathsf{SDT}(C_1 \wedge C_2)$, where the competence results have been verified true. The approach described here for determining DT and SDT values were equally applied to the result listed in Table 4 and Table 5.

The $\mathsf{DT}(C_1(\cdot))$ values in all the permutations match the number of samples in our malware data set. During this experiment, since the scanning ability of the first anti-malware program of a permutation was enabled prior to a malware being executed, accessing the malware file caused the anti-malware program, in almost every case, to immediately detect and treat the sample which often resulted in automatic deletion. A result of $\mathsf{DT}(C_2(\cdot)) > 0$ was reported for 7 of the tested permutations. In each of these cases the samples were detected by C_1 but possibly not effectively treated resulting in some malicious object left active on the system. As a result C_2 detected and treated the remaining active malicious objects. A result of $\mathsf{DT}(C_3(\cdot)) > 0$ was reported for 5 of the tested permutations. In each of these cases the sample was detected by both C_1 and C_2 but was not treated correctly by either one or the other or both thus facilitating C_3 to detect the remaining malicious objects.

The $\mathsf{SDT}(C_1)$ results were very high with 4 scanning permutations where C_1 achieved competence in all 500 samples. C_1 in the rest of the permutations performed well with the lowest being C_1 = E (ESET) in permutation EAZ with 487 samples where ESET was competent. The results for $\mathsf{SDT}(C_1 \wedge C_2)$ and $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$ successfully detected and treated the unaccounted samples from $\mathsf{SDT}(C_1)$. The permutation EZA returned $\mathsf{SDT}(C_1) = 488$, $\mathsf{SDT}(C_1 \wedge C_2) = 8$, and $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3) = 4$. The four samples detected by C_3 were detected and not correctly treated by C_1 and were not detected at all by C_2, this detection trend also occurred in permutations AZE, GBK, BKG, and BGK.

Overall, the permutations performed relatively well in Experiment 1 with the vast majority of samples being detected and treated by C_1 producing high level of competence, although there were 8 permutations with a combined total of 61 samples where multiple anti-malware programs were required to achieve competence. Of these 61, there were a combined 14 samples in 5 permutations which required the complete permutation of 3 programs to seemingly achieve competence. These results confirm that, in general, having one anti-malware program installed in a clean state may greatly increases detection accuracy and treatment effectiveness. One caveat to this notion is the competence of an anti-malware program is, at minimum, partially dependent on having detection databases up to date which was done during this experiment. It is unclear if consumers follow due diligence and keep their detection databases updated which may decrease their anti-malware program's competence.

3 EXPERIMENT 2 RESULTS

The detection results (DT) of Experiment 2 are listed in Table 4 and the competence results (SDT) are listed in Table 5.

Scanning permutation	$\mathsf{DT}(C_1(\cdot))$	$\mathsf{DT}(C_2(\cdot))$	$\mathsf{DT}(C_3(\cdot))$
EAZ	372	192	128
EZA	406	216	161
ZEA	154	261	102
ZAE	146	302	251
AEZ	328	104	31
AZE	295	98	126
KGB	472	64	21
KBG	461	43	9
GBK	389	104	57
GKB	371	179	102
BKG	302	112	31
BGK	299	183	92

Table 4: Experiment 2 results for $\mathsf{DT}(C_n)$

In Experiment 2, the DT results were much higher than the results of Experiment 1 for C_1, C_2, and C_3. For every permutation the summation of the totals for $DT(C_1(\cdot)) + DT(C_2(\cdot)) + DT(C_3(\cdot))$ surpassed 500 resulting from multiple objects infected by each sample during the 3 minute period before the scan by C_1 was performed. This led to the same sample being detected by multiple anti-malware programs in a given permutation. Note, if all 500 samples in our set were detected by all three programs in a given permutation, the maximal result for DT would be $DT(C_1(\cdot))+DT(C_2(\cdot))+DT(C_3(\cdot)) = 1500$. In 10 out of 12 permutations the highest number of detections occurred in C_1. The top 3 detection totals for C_1 were permutations KGB (472), KBG (461), and EZA (406). The top three detection to-

ISBN: 978-1-62561-001-0

©ASE 2012

tals for $DT(C_1(\cdot))+DT(C_2(\cdot))$ were EZA (622), EAZ (564), and GKB (550). For $DT(C_1(\cdot))+DT(C_2(\cdot))+DT(C_3(\cdot))$, the top three detection totals were EZA (783), ZAE (699), and EAZ (692). Permutation EZA is particularly interesting due its high detection rate in all three measurements which implies this permutation requires multiple engines to achieve effective detection for several different types of malware. The permutations KGB and KBG only appear in the top detection for C_1 = K implying Kaspersky is capable of effectively detecting several malicious objects infected by our broad malware sample set. Further proof of this is evident by the totals of $DT(C_1(\cdot)) + DT(C_2(\cdot))$, and $DT(C_1(\cdot)) + DT(C_2(\cdot)) + DT(C_3(\cdot))$ for KGB and KBG which are 64, 21 and 43, 9 respectively.

Scanning permutation	$\mathsf{SDT}(C_1(\cdot))$	$\mathsf{SDT}(C_1 \wedge C_2)$	$\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$
EAZ	180	64	128
EZA	190	43	161
ZEA	86	157	102
ZAE	106	71	251
AEZ	251	98	31
AZE	207	49	126
KGB	403	57	21
KBG	412	38	9
GBK	302	76	57
GKB	239	146	102
BKG	298	104	31
BGK	287	116	92

Table 5: Experiment 2 results for $\mathsf{SDT}(C_1 \wedge ... \wedge C_n)$

Given the very high DT results in this experiment, the SDT totals were much lower with no permutation capable of achieving competence for all 500 tested malware samples. The top 3 competence totals for C_1 were permutations KBG (412), KGB (403), and GBK (302). The top 3 summations of competence totals for $C_1 \wedge C_2$ were permutations KGB (460), KBG (450), and BGK (403). For $C_1 \wedge C_2 \wedge C_3$, the top 3 permutations for the summation of competence totals were BGK (495), GKB (487), and KGB (481). Kaspersky again had the best result with C_1 = K in KBG detecting 82.4% of the samples. The permutation KGB was able to detect 92.0% of the samples after completing $C_1 \wedge C_2$. The permutation BGK, which only appeared in the top 3 of $C_1 \wedge C_2 \wedge C_3$ came closest to achieving competence for all 500 samples, detecting 99.0%, with 5 samples undetected.

A key factor in disallowing better competence results, especially in C_1, and $C_1 \wedge C_2$ was anti-malware programs either failing to install or failing to perform a scan in a compromised system. During Experiment 2, there were 384 instances of installation failure and 297 instances of an installed anti-malware program unable to scan the system. A total of 681 instances occurred where a detection scan was not performed by one or more members of a permutation set. The permutations with the most failures overall were ZEA, ZAE, and AZE, with ZoneAlarm having the most failed installs overall and AVG having the most failed scan attempts. KGB, KBG, an GKB were the most resilient with the least amount of failures. Using ZEA as an example, a typical failure scenario would occur with ZoneAlarm crashing during install and not completing. In this instance, the totals of both $DT(C_1(\cdot))$ and $SDT(C_1)$ were not incremented and the test continued with the remaining permutation components EA. ESET would install correctly but would fail to perform a detection scan, in this case again both $DT(C_2(\cdot))$ and $SDT(C_1 \wedge C_2)$ totals were not incremented and the test finished with the final permutation component A. AVG would install and scan the system successfully and accordingly the totals for $DT(C_3(\cdot))$ and $SDT(C_1 \wedge C_2 \wedge C_3)$ were incremented as needed. This general approach of not incrementing DT and SDT totals and continuing with the test was followed in all instances where a failure, as described above, occurred.

No anti-malware program being found competent in Experiment 2 along with the observed failures is a troubling result. One could conjecture a cross section of consumers tend to buy or update their chosen anti-malware program only after suspicion of malware present on their system. In these cases the results of Experiment 2 do not put detection accuracy and treatment effectiveness in the user's favor. It is unclear if this is the result of malware able to compromise the system and the anti-malware program, or the program's lacking a self-defense mechanism to protect themselves in s compromised environment or a mix of both. One could conjecture the last option is possibly most likely. Experiment 2 illustrates the need to enhance the resilience of anti-malware programs when installed and updated, along with competence when scanning a malware infected system.

4 DISCUSSION

Experiment 1 evaluated 2 permutation sets of 3 anti-malware programs each. For each set a maximal 3! =

ISBN: 978-1-62561-001-0

©ASE 2012

6 permutations were evaluated for an experiment total of 12 permutations. Each permutation set was tested against a malware data set of 500 samples resulting in 1,500 individual anti-malware program test per permutation, 9,000 tests per permutation set, and 18,000 tests for Experiment 1. The same testing occurred in Experiment 2 for a grand total of 36,000 individual anti-malware program tests carried out in this research across 24 total permutations. Of these 24, only 4 permutations, 16% overall, were found competent for $\mathsf{SDT}(C_1)$ for all 500 samples, all occurring in Experiment 1. For $\mathsf{SDT}(C_1 \wedge C_2)$, only 3 permutations, 12.5% overall, in Experiment 1, achieved competence for all 500 samples. Competence for $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$ was achieved for all 500 samples by 5 permutations, 20.8% overall, again occurring only in Experiment 1. Overall, only 7 permutations, 29.2%, all occurring in Experiment 1 were verified to have achieved competence with all 500 samples though permutation BGK in Experiment 2 did come close with 495. Experiment 1 produced no failures, every anti-malware program in every permutation installed, updated, and scanned correctly. The high amount of competence results in Experiment 1 suggest anti-malware programs properly detect and treat malware when installed in a clean state and allowed to passively monitor the machine. In Experiment 2, many problems were encountered during anti-malware program installation and scanning which may have facilitated failed competence attempts. This implies that anti-malware programs may not be resilient to infected states and may fail when being installed or executed in a compromised environment. Installing and executing the anti-malware program from a boot CD, USB or some other external media, may reduce problems and provide better detection and treatment, but this might be an unrealistic approach as one can conjecture that average users may prefer the faster and more convenient method of downloading an installable anti-malware program or the ISO image of a bootable anti-malware CD from some online source and attempt to install and execute within the infected system.

Turnover rates. Table 6 lists the turnover rate, as a percentage, from DT to SDT for C_1, and $C_1 \wedge C_2$ for Experiments 1 and 2. The turnover rate for $C_1 \wedge C_2 \wedge C_3$ was intentionally omitted since those results are accepted as true but unverified, as opposed to the rest of the results which are verified as true. In cases where competence was achieved for all 500 samples in C_1 or $C_1 \wedge C_2$, a value of N/A was recorded for the subsequent C_n results. The turnover rate is calculated by dividing the competence result $SDT(C_1 \wedge ... \wedge C_n)$ by its corresponding detection result. For example, in Table 6, for Experiment 1, the first permutation is EAZ. According to Table 2 $DT(C_1(.)) = 500$ with $C_1 = \text{E}$, in Table 3, the corresponding competence value is $SDT(C_1) = 487$. This signifies ESET detected 500 samples as being malicious but only effectively treated 487 samples, thus ESET's turnover ratio in this specific instance is $487/500 = 97.4\%$. The turnover rate is an important measure of an anti-malware program's treatment effectiveness of detected malicious objects. The higher the number of samples for which competence was achieved for a specific anti-malware program, the higher that program's turnover rate will be. A low turnover rate indicates an anti-malware program may be very effective in malware detection but less effective in treatment. On the other hand, a high turnover rate indicates an anti-malware program is highly effective in treatment leading to full eradication of detected malware from a system.

	C_1	$C_1 \wedge C_2$	$C_1 \wedge C_2 \wedge C_3$
Experiment 1			
EAZ	97.4	100	N/A
EZA	97.6	100	N/A
ZEA	100	N/A	N/A
ZAE	100	N/A	N/A
AEZ	98.8	100	N/A
AZE	98.6	71.4	N/A
KGB	100	N/A	N/A
KBG	100	N/A	N/A
GBK	99.4	0	N/A
GKB	98.8	100	N/A
BKG	98.6	85.7	N/A
BGK	98.8	100	N/A
Experiment 2			
EAZ	48.4	33.3	N/A
EZA	46.8	20	N/A
ZEA	55.8	60.1	N/A
ZAE	72.6	23.5	N/A
AEZ	76.5	94.2	N/A
AZE	70.2	50	N/A
KGB	85.8	89	N/A
KBG	89.3	88.4	N/A
GBK	77.6	73	N/A
GKB	64.4	81.5	N/A
BKG	98.7	92.8	N/A
BGK	96	63.4	N/A

Table 6: Detection to Competence Turnover Rate

In Experiment 1, for C_1, the turnover rates were very high with the lowest being 97.4% by ESET

©ASE 2012
ISBN: 978-1-62561-001-0

in permutation EAZ. There were 4 instances of a 100% turnover rate. For $C_1 \wedge C_2$, 5 instances of 100% turnover rate occurred, but the lowest rate was 71.4% by ZoneAlarm in permutation AZE. The average turnover rates for C_1 and $C_1 \wedge C_2$ in Experiment 1 were very high at 99% and 82.1% respectively. The consistently high turnover rates in Experiment 1 imply malware infection occurring after an anti-malware program is running on a system have a much lower chance of survival due to highly effective treatment leading to complete eradication.

In Experiment 2, for C_1, the highest turnover rate was 98.7% in permutation BKG, the lowest rate was 46.8% in permutation EZA. For $C_1 \wedge C_2$, the highest turnover rate was 94.2% by permutation AEZ and the lowest rate was 23.5% by permutation ZAE. There was no 100% turnover rate in Experiment 2. The average turnover rates for C_1 and $C_1 \wedge C_2$ in Experiment 2 were 73.5% and 64.1% respectively. This range of turnover rates implies effective treatment is difficult in a compromised environment and is also highly subjective based on an individual anti-malware program's capabilities.

The turnover rates can be used to measure the false negative production since the rate is reflective of competence which is based on detection and treatment effectiveness. Given that a 100% turnover rate is equivalent to competence in all 500 tested samples, we measure false negative production for a given permutation with a given $C_1...C_n$ by subtracting the turnover rate from 100. For example, in Table 6, Experiment 1, the turnover rate for permutation EAZ with $C_1 = 97.4\%$, thus the false negative rate is $100 - 97.4 = 2.6\%$. For Experiment 1, in C_1 there were 4 permutations (ZEA, ZAE, KGB, KBG) with no false negatives, the permutation EAZ produced the highest false negative rate of 2.6%. As for $C_1 \wedge C_2$, there were 5 permutations (EAZ, EZA, AEZ, GKB, BGK) with no false negatives and the lowest false negative was permutation AZE with 28.6%. In Experiment 2, for C_1, the lowest false negative rate was permutation BKG with 1.6% and the highest was EZA with 53.2%. As for $C_1 \wedge C_2$, the lowest false negative rate was permutation AEZ with 5.8% and the highest false negative rate was EZA with 80%. In both experiments, the false negative rates seem relatively high, with 1.6% perhaps being tolerable, while the other rates above 2% (more than 10 samples) are a clear sign that even in Experiment 1, where the anti-malware programs seemingly have the upper hand, false negatives can be higher than expected.

An interesting observation from the turnover rates is the order of anti-malware programs may have an impact on their detection capabilities. For example, in Experiment 2, ZoneAlarms's turnover rate varied from 72.6% to 20% and in Experiment 1, Kaspersky's rate varied from 100% to 85.7%. Other cases produced turnover rates with less than 1% difference for various sequence position. The fact that some anti-malware programs have varying detection rates when in a different sequence position may imply improved performance can be achieved when various anti-malware programs are used in specific sequence and merits further investigation.

Experiment observations. From the results of Experiments 1 and 2, we can observe the following:

1. Competence seems readily achievable with the anti-malware program running before malware is executed on the system.

2. Competence was mostly achieved with signature based detection upon invoking a mouse over of the malware sample.

3. The combined detection power of multiple engines increases competence though, as the results suggest, there are cases where even 3 engines may not be enough to achieve competence and completely eradicate malware from a system.

4. An infected system seems to facilitate the instability of an anti-malware program.

5. An anti-malware program installed in an infected system may have a higher risk of false negative production.

6. The tested anti-malware programs seem to lack a self-defense mechanism possibly facilitating malware to attack it resulting in compromise when being installed or performing a scan in an infected system.

7. Malware running in the system may block access to resources required by an anti-malware program causing a crash when attempting to run.

An initial assessment at how many anti-malware programs to deploy. The results suggest having one anti-malware program installed on a system may be considered competent in some but not

©ASE 2012
ISBN: 978-1-62561-001-0

all malware scenarios. How many anti-malware programs should be deployed on a system to provide adequate protection from malware remains a non-trivial question but we can forecast possible answers based on our results. We assess the suitable number of anti-malware programs for low, medium and high protection on a system.

Based on Experiment 1 results we can assess that in general, at least 2 anti-malware programs should be present for low protection, 3 for medium protection and possibly 4 or 5 for high protection. There were many competent instances after scanning with $\mathsf{SDT}(C_1 \wedge C_2)$ indicating 2 anti-malware programs may provide minimal protection. Most of the anti-malware program sequences were competent after $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$ thus three programs may provide medium protection since there were a few malware found which could infect further many other files and processes. Forecasting high protection with 4 or 5 anti-malware programs may suffice for the minimal number of malware left from the first three. Based on Experiment 2 results it is clear that 3 anti-malware programs may not suffice for low protection. In $\mathsf{SDT}(C_1 \wedge C_2 \wedge C_3)$, none of the instances seem to achieve competence and almost all reported high DT amounts. Given these high infection amounts, a suitable amount of anti-malware programs, such as 4, 5, or more, to provide even low protection may be needed but currently would be too many and unrealistic for practical purposes.

Assuming that not all consumers follow due diligence in keeping detection databases up to date and some anti-malware programs do not run harmoniously together, making an effective deployment of an adequate number of anti-malware programs to achieve competence is a difficult task with no simple solution. A better approach maybe one anti-malware program using the detection techniques of several other anti-malware programs to improve detection and treatment effectiveness. Possibly for the foreseeable future, users are seemingly left with inadequate protection choices resulting in a likely facilitation of malware infection.

Limitations.

1. We limited our permutation sets to 3 anti-malware programs each which forcibly disallows the verification of SDT for C_3, thus leaving our verified results to permutations of two anti-malware programs.
2. Selecting malware samples from a small time frame of upload dates may reduce diversity and create unintended bias in testing.

In future testing, we will address the above item 1 by incorporating more computing power to run tests and verify results with longer permutation sets. We will address the above item 2 by populating our test set with several samples from a broader time frame to reduce any possible bias.

IV CONCLUSION AND FUTURE WORK

We reported our initial study on anti-malware program competence to establish detection efficiency using multiple anti-malware programs. We introduced the notion of an anti-malware program achieving *competence* when it detects and cleans all malware present on a system. We presented a general framework and reported some initial results based on multiple runs of two experiments. In several results, a single anti-malware program was not competent in various malware scenarios, which reaffirms the widely accepted belief that one anti-malware program is insufficient for complete malware protection. Furthermore, when attempting to achieve competence in a malware infected system with multiple anti-malware programs running, malware was still detected. Based on experiment results, we forecast that employing a minimum of 5 anti-malware programs on a system may be required to achieve competence in a broad and diverse range of malware scenarios, which may prove too many and unrealistic for practical use. A possible remedy is to use one anti-malware program that employs the detection techniques of several anti-malware programs, although achieving this may be impractical given the nature of open market competitiveness. Our future work includes extensive experiments with several malware samples to create longer permutations of multiple anti-malware programs. We will also craft new experiments of detection and disinfection of malware under a diverse set of realistic end user scenarios.

ACKNOWLEDGEMENTS. This work was performed when the first author was at the Institute for Cyber Security, University of Texas at San Antonio. This work was supported in part by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of these agencies or the State of Texas.

©ASE 2012
ISBN: 978-1-62561-001-0

References

[1] L. Adleman. An abstract theory of computer viruses. In *Proceedings on Advances in cryptology*, pages 354–374, 1990.

[2] http://www.anti-malware-test.com.

[3] The fundamental principles of testing. Technical report, Anti-Malware Testing Standards Organization, 2008.

[4] Whole product testing guidelines. Technical report, Anti-Malware Testing Standards Organization, 2010.

[5] M. Christodorescu and S. Jha. Testing malware detectors. *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 34–44, 2004. ACM Press, New York, NY, USA. http://doi.acm.org/10.1145/1007512.1007518.

[6] M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, editors. *Malware Detection*. Springer, 2007.

[7] F. Cohen. Computer viruses: theory and experiments. *Comput. Secur.*, 6:22–35, February 1987.

[8] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 43–53, New York, NY, USA, 2004. ACM Press.

[9] E. Filiol. *Computer Viruses: from Theory to Applications*. IRIS International series, Springer Verlag, 2005. ISBN 2-287-23939-1.

[10] S. Gordon and R. Ford. Real world anti-virus product reviews and evaluations - the current state of affairs. In *Proceedings of the 1996 National Information Systems Security Conference*, 1996.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[12] C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *18th Usenix Security Symposium*, 2009.

[13] A. Marx. Guideline to anti-malware software testing. In *European Institute for Computer Anti-Virus Research (EICAR) 2000 Best Paper Proceedings*, 2000. pp.218-253.

[14] J. A. Morales, P. J. Clarke, Y. Deng, and B.G. Kibria. Identification of file infecting viruses through detection of self-reference replication. *Journal in Computer Virology Special EICAR conference invited paper issue*, 2008.

[15] J. A. Morales, P. J. Clarke, B.M. Kibria, and Y. Deng. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology*, September 2006.

[16] J. A. Morales, R. Sandhu, and S. Xu. Evaluating detection and treatment effectiveness of commercial anti-malware programs. In *5th International Conference on Malicious and Unwanted Software, MALWARE 2010*, pages 35–42, 2010.

[17] R. Moskovitch, Y. Elovici, and L. Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Comput. Stat. Data Anal.*, 52(9):4544–4566, 2008.

[18] I. Muttik and J. Vignoles. Rebuilding anti-malware testing for the future. In *Virus Bulletin Conference*, 2008.

[19] J. C. Rabek, R. I. Khazan, S. M. Lewandowski, and R. K. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 76–82, New York, NY, USA, 2003. ACM Press.

[20] P. Szor. *The Art of Computer Virus Research and Defense*. Symantec Press and Addison-Wesley, 2005. ISBN 9-780321-304544.

[21] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127, New York, NY, USA, 2007. ACM.

©ASE 2012
ISBN: 978-1-62561-001-0